ҚОСТАНАЙ ОБЛЫСЫ ӘКІМДІГІ БІЛІМ БАСҚАРМАСЫНЫҢ
«ҚОСТАНАЙ ЖОҒАРЫ ПОЛИТЕХНИКАЛЫҚ КОЛЛЕДЖІ» КМҚК
КГКП «КОСТАНАЙСКИЙ ПОЛИТЕХНИЧЕСКИЙ ВЫСШИЙ КОЛЛЕДЖ»
УПРАВЛЕНИЯ ОБРАЗОВАНИЯ АКИМАТА КОСТАНАЙСКОЙ ОБЛАСТИ

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС

ПМ 04. РАЗРАБОТКА КОМПОНЕНТОВ ПРОЕКТНОЙ И ТЕХНИЧЕСКОЙ ДОКУМЕНТАЦИИ С ИСПОЛЬЗОВАНИЕМ ГРАФИЧЕСКИХ ЯЗЫКОВ СПЕЦИФИКАЦИЙ

Автор-составитель: Аскарова М.Б.

Костанай, 2022

Содержание

Раздел 1. Основы алгоритмизации и программирования

Тема 1 Классификация языков программирования

Цель: раскрыть классификацию языков программирования, рассказать о необходимости эффективного использования нужного языка программирования для решения конкретной поставленной задачи.

За 60 с лишним лет развития ЭВМ были разработаны сотни языков программирования, многие из которых используются и сейчас (например Бейсик и Фортран были впервые применены уже в конце 1950-х годов), ежегодно появляется несколько новых языков промышленного применения (не считая десятков экспериментальных). Для того, чтобы разобраться в них, языки классифицируют по важнейшим признакам:

– эволюционным – поколения языков (1GL, 2GL, 3GL, 4GL, 5GL…);

– функциональным - по назначению, исполняемым функциям (описательные, логические, математические);

– уровню языка - то есть уровню обобщения в словах-операторах языка (низкого, среднего, высокого...);

– области применения - то есть где применяется язык (системные, сетевые, встроенные и пр.

Все типы классификаций естественным образом пересекаются, гармонируют между собой, что мы увидим при рассмотрении этих классификаций, что при понимании этого позволит легко разобраться в любом новом язы–е - его назначении, возможностях, технике освоения.

Базовая иерархия языков программирования

Базовая иерархия языков программирования является системно-параллельной иерархией, то есть пакета тесно связанных иерархий: этапов программирования, поколений языков программирования и самих языков - протоколов преобразования структурной и алгоритмической информации: структурно-дескриптивного описания данных и алгоритма их обработки. Поэтому все языки деялтся на два полярных типа: дескриптивные (декларативные) и алгоритмические (командные). Однако, так как в любом алгоритме существует необходимость описания данных и структур, а в любой конструкции — порядок её сборки, то реальные языки являются частично декларативными, а частично алгоритмическими, что отражается в наличии описательной и командной (рецептурной) частей любой компьютерной программы.

Поколения языков (Generations of Languages)

Поколение 1GL. Машинные языки, языки низкого уровня - двоичные языки процессоров, представляющие собой набор (алфавит) команд, записанных в двоичном коде (0,1), которые данный процессор может выполнить непосредственно, если эти команды ввести в его память в виде последовательности или сразу подать в арифметическо-логическое устройство процессора. Примеры: язык процессора IBM-PC, язык ARM-процессора.

Поколение 2GL. Ассемблеры, автокоды, системные языки, языки среднего уровня - текстовые языки, понятные человеку и однозначно переводимые (транслируемые) в языки низкого уровня, то есть машинный двоичный код. Программирование на 2GL на порядок производительнее, чем на 1GL, так как более удобны для человеческого восприятия. Примеры: Макроссемблер, C, PL/1.

Поколение 3GL. Языки высокого уровня - текстовые языки, приближенные по словарю и синтаксису к человеческому языку (обычно утрированному английскому, пиндосу), позволяющие записывать программные конструкции в форме, удобной для человеческого мышления и подобные обычному тексту — конспекту, стенограмме.

Программирование на 3GL на порядок производительнее, чем на 2GL, так как более удобны для человеческого восприятия и на порядок короче ассемблерных. Примеры: бейсик, фортран, PHP и практически все сетевые языки.

Поколение 4GL. Языки визуального программирования - языки блок-схем, позволяющие отображать алгоритмы в программных проектах, что облегчает создание и анализ алгоритмов. Программирование на 4GL на порядок производительнее, чем на 3GL. Примеры: RAD-системы, CAD-пакеты, OLAP-системы.

Поколение 5GL. Интеллектуальные языки программирования - позволяют передать функцию создания алгоритмов компьютеру, а за человеком оставить лишь постановку задачи. Программирование на 5GL на порядок производительнее, чем на 4GL. Примеры: система MatCAD, экспертные системы.

Заметим, что увеличение производительности труда программиста оборачивается увеличением нагрузки на процессор и память, то есть компьютерная программа, полученная средствами каждого следующего поколения имеет на порядок большую длину исполняемого кода, а значит, расхода вычислительных ресурсов и памяти компьютера, то есть выполняется намного медленнее на том же самом компьютере или требует намного более быстрого процессора.

Хотя поколения языков, естественно, сложились исторически, это не означает, что ранние поколения себя изжили. Они применяются в своих нишах, например, для программирования простейших устройств, имеющих минимум памяти, типа банковских карт, микроконтроллеров бытовых и промышленных устройств применим только машинный код, ибо языкам высоких уровней там «не развернуться». В системном программировании наилучшие результаты дают языки 2GL, ибо в этой сфере важна скорость выполнения и компактность кода. Для обработки текста и сетевых задач оптимальными являются языки 3GL.

Непосредственно связанной с иерархией поколений языков является так называемая «Стандартная модель OSI», описывающая 7 уровней иерархии протоколов (языков) сетевого обмена информацией, рассмотренная ниже.

Функциональная классификация языков программирования

Существующие языки программирования классифицируют по четырём основным функциональным группам: процедурные, объектно-ориентированные, функциональные и логические. Дадим краткие определения каждого подхода.

Процедурное программирование - такое программирование, когда программа отделена от данных и состоит из последовательности команд, обрабатывающих данные. Данные как правило хранятся в виде переменных. Весь процесс вычисления сводится к изменению их содержимого.

Декларативные языки программирования - это языки объявлений и построения структур. К ним относятся функциональные и логические языки программирования. В этих языках не производится алгоритмических действий явно, то есть алгоритм не задается прграммистом, а строится самой программой. В декларативных языках задается, производится построение какой-либо структуры или системы, то есть декларируются (объявляются) какие-то свойства создаваемого объекта. Эти языки получили широкое применение в системах автоматизированного проектирования (САПР), в так называемых CAD-пакетах, в моделировнии, системах исккусственного интеллекта.

Объектно-ориентированное программирование - в этих языках переменные и функции группируются в так называемые классы (шаблоны). Благодаря этому достигается более высокий уровень структуризации программы. Объекты, порождённые от классов вызывают методы (функции или процедуры) друг друга и меняют таким образом состояние свойств (переменных). С формально-математической стороны объектно ориентированный способ написания программ базируется на процедурной модели программирования, но с

содержательной стороны ООП базируется не на функции, а на объекте, как целостной системе, имеющей стандартный автоматический межобъектный интерфейс.

Сетевые языки - языки, предназначенные для организации взаимодействия удаленных компьютеров в интенсивном интерактивном режиме, а поэтому они построены на принципах интерпретации, то есть построчной, интерактивной обработки строк программного кода, описывающего некоторый сценарий (скрипт) сетевого взаимодействия компьютеров, поэтому часто они называются скриптовыми языками, хотя скриптовые языки не обязательно являются сетевыми, к примеру, пакетные командные языки различных операционных сред.

Машинно - ориентированные языки

Машинно - ориентированные языки - это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.). Машинно -ориентированные языки позволяют использовать все возможности и особенности Машинно - зависимых языков:

– высокое качество создаваемых программ (компактность и скорость выполнения);

– возможность использования конкретных аппаратных ресурсов;

– предсказуемость объектного кода и заказов памяти;

– для составления эффективных программ необходимо знать систему команд и особенности функционирования данной ЭВМ;

– трудоемкость процесса составления программ (особенно на машинных языках и ЯСК), плохо защищенного от появления ошибок;

– низкая скорость программирования;

– невозможность непосредственного использования программ, составленных на этих языках, на ЭВМ других типов.

Машинно-ориентированные языки по степени автоматического программирования подразделяются на классы.

Машинный язык

Как уже упоминалось в введении, отдельный компьютер имеет свой определенный **Машинный язык** (далее **МЯ**), ему предписывают выполнение указываемых операций над определяемыми ими операндами, поэтому **МЯ** является командным. Однако, некоторые семейства ЭВМ (например, ЕС ЭВМ, IBM-370 и др.) имеют единый **МЯ** для ЭВМ разной мощности. В команде любого из них сообщается информация о местонахождении операндов и типе выполняемой операции.

В новых моделях ЭВМ намечается тенденция к повышению внутренних языков машинно - аппаратным путем реализовывать более сложные команды, приближающиеся по своим функциональным действиям к операторам алгоритмических языков программирования.

Языки Символического Кодирования

Продолжим рассказ о командных языках, **Языки Символического Кодирования** (далее **ЯСК**), так же, как и **МЯ**, являются командными. Однако коды операций и адреса в машинных командах, представляющие собой последовательность двоичных (во внутреннем коде) или восьмеричных (часто используемых при написании программ) цифр, в **ЯСК** заменены на символы (идентификаторы), форма написания которых помогает программисту легче запоминать смысловое содержание операции. Это обеспечивает существенное уменьшение числа ошибок при составлении программ.

Использование символических адрес–в - первый шаг к созданию **ЯСК**. Команды ЭВМ вместо истинных (физических) адресов содержат символические адреса. По результатам составленной программы определяется требуемое количество ячеек для хранения исходных промежуточных и результирующих значений. Назначение адресов, выполняемое отдельно от составления программы в символических адресах, может

проводиться менее квалифицированным программистом или специальной программой, что в значительной степени облегчает труд программиста.

Автокоды. Есть также языки, включающие в себя все возможности **ЯСК**, посредством расширенного введения **макрокома–д -** они называются **Автокоды**.

В различных программах встречаются некоторые достаточно часто использующиеся командные последовательности, которые соответствуют определенным процедурам преобразования информации. Эффективная реализация таких процедур обеспечивается оформлением их в виде специальных макрокоманд и включением последних в язык программирования, доступный программисту. Макрокоманды переводятся в машинные команды двумя путями - расстановкой и генерированием. В постановочной системе содержат«я "остовы" - серии команд, реализующих требуемую функцию, обозначенную макрокомандой. Макрокоманды обеспечивают передачу фактических параметров, которые в процессе трансляции вставляются в "остов" программы, превращая её в реальную машинную программу.

В системе с генерацией имеются специальные программы, анализирующие макрокоманду, которые определяют, какую функцию необходимо выполнить и формируют необходимую последовательность команд, реализующих данную функцию.

Обе указанных системы используют трансляторы с **ЯСК** и набор макрокоманд, которые также являются операторами автокода.

Развитые автокоды получили название **Ассемблер**ы. Сервисные программы и пр., как правило, составлены на языках типа **Ассемблер**. Более полная информация об языке **Ассемблер**а см. ниже.

Макрос. Язык, являющийся средством для замены последовательности символов описывающих выполнение требуемых действий ЭВМ на более сжатую форму - называется **Макрос** (средство замены).

В основном, **Макрос** предназначен для того, чтобы сократить запись исходной программы. Компонент программного обеспечения, обеспечивающий функционирование макросов, называется **макропроцессором**. На макропроцессор поступает макроопределяющий и исходный текст. Реакция макропроцессора на вызов-выдача выходного текста.

Макрос одинаково может работать, как с программами, так и с данными.

Машинно - независимые языки. Машинно - независимые языки - это средство описания алгоритмов решения задач и информации, подлежащей обработке. Они удобны в использовании для широкого круга пользователей и не требуют от них знания особенностей организации функционирования ЭВМ и ВС.

Подобные языки получили название высокоуровневых языков программирования. Программы, составляемые на таких языках, представляют собой последовательности операторов, структурированные согласно правилам рассматривания языка(задачи, сегменты, блоки и т.д.). Операторы языка описывают действия, которые должна выполнять система после трансляции программы на **МЯ**.

Т.о., командные последовательности (процедуры, подпрограммы), часто используемые в машинных программах, представлены в высокоуровневых языках отдельными операторами. Программист получил возможность не расписывать в деталях вычислительный процесс на уровне машинных команд, а сосредоточиться на основных особенностях алгоритма.

Проблемно - ориентированные языки. С расширением областей применения вычислительной техники возникла необходимость формализовать представление постановки и решение новых классов задач. Необходимо было создать такие языки программирования, которые, используя в данной области обозначения и терминологию, позволили бы описывать требуемые алгоритмы решения для поставленных задач, ими стали **проблемно - ориентированные языки**. Эти языки, ориентированные на решение

определенных проблем, должны обеспечить программиста средствами, позволяющими коротко и четко формулировать задачу и получать результаты в требуемой форме.

Проблемных языков очень много, например:

Фортран, Алгол - языки, созданные для решения математических задач;

Simula, Слэнг - для моделирования;

Лисп, Снобол - для работы со списочными структурами.

Об этих языках рассказано дальше.

Универсальные языки

Универсальные языки были созданы для широкого круга задач: коммерческих, научных, моделирования и т.д. Первый универсальный язык был разработан фирмой IBM, ставший в последовательности языков **Пл/1**. Второй по мощности универсальный язык называется **Алгол-68**. Он позволяет работать с символами, разрядами, числами с фиксированной и плавающей запятой. **Пл/1** имеет развитую систему операторов для управления форматами, для работы с полями переменной длины, с данными организованными в сложные структуры, и для эффективного использования каналов связи. Язык учитывает включенные во многие машины возможности прерывания и имеет соответствующие операторы. Предусмотрена возможность параллельного выполнение участков программ.

Программы в **Пл/1** компилируются с помощью автоматических процедур. Язык использует многие свойства **Фортрана, Алгола, Кобола**. Однако он допускает не только динамическое, но и управляемое и статистическое распределения памяти.

Диалоговые языки

Появление новых технических возможностей поставило задачу перед системными программистами - создать программные средства, обеспечивающие оперативное взаимодействие человека с ЭВМ их назвали **диалоговыми языками**.

Эти работы велись в двух направлениях. Создавались специальные управляющие языки для обеспечения оперативного воздействия на прохождение задач, которые составлялись на любых раннее неразработанных (не диалоговых) языках. Разрабатывались также языки, которые кроме целей управления обеспечивали бы описание алгоритмов решения задач.

Необходимость обеспечения оперативного взаимодействия с пользователем потребовала сохранения в памяти ЭВМ копии исходной программы даже после получения объектной программы в машинных кодах. При внесении изменений в программу с использованием диалогового языка система программирования с помощью специальных таблиц устанавливает взаимосвязь структур исходной и объектной программ. Это позволяет осуществить требуемые редакционные изменения в объектной программе.

Одним из примеров диалоговых языков является **Бэйсик**.

Бэйсик использует обозначения подобные обычным математическим выражениям. Многие операторы являются упрощенными вариантами операторов языка **Фортран**. Поэтому этот язык позволяет решать достаточно широкий круг задач.

Непроцедурные языки

Непроцедурные языки составляют группу языков, описывающих организацию данных, обрабатываемых по фиксированным алгоритмам (табличные языки и генераторы отчетов), и языков связи с операционными системами.

Позволяя четко описывать как задачу, так и необходимые для её решения действия, таблицы решений дают возможность в наглядной форме определить, какие условия должны быть выполнены прежде чем переходить к какому-либо действию. Одна таблица решений, описывающая некоторую ситуацию, содержит все возможные блок-схемы реализаций алгоритмов решения.

Табличные методы легко осваиваются специалистами любых профессий.

Программы, составленные на табличном языке, удобно описывают сложные ситуации, возникающие при системном анализе.

Контрольные вопросы:

1. Какие классификации языка программирования есть
2. Назовите базовые иерархию языков программирования
3. Какие поколения языков (Generations of Languages) есть?
4. Какие функциональные классификации языков программирования есть?

Тема 2. Этапы решения задач на ЭВМ. Понятие алгоритма. Свойства алгоритмов. Способы описания алгоритмов.

Цель: Ознакомление обучающихся с понятием алгоритма, его свойств, исполнителя алгоритма и основными алгоритмическими структурами.

При решении на ПК любой задачи можно выделить несколько этапов в работе:

- математическая постановка задачи;
- выбор метода решения задачи;
- разработка алгоритма;
- программирование;
- отладка программы;
- подготовка исходных данных и непосредственное выполнение программы.

Математическая постановка задачи

На этом этапе всем физическим величинам, участвующим в задаче, надо дать математические обозначения. Удобнее всего использовать общепринятые обозначения –h - высота,–l - длина,–s - площадь и т.д.), но допускается обозначать все величины произвольно, на свое усмотрение. Конечно, чтобы самому не забыть об этих обозначениях, необходимо их записать в связи с самими физическими величинами (это действие можно назвать "распределением памяти", т.к. каждая величина будет храниться в памяти ПК).

Следующее действие - определить так называемый "статус" каждой переменной, т.е. определить, к какой из следующиих категорий она относится:

- исходные данные - эти величины известны из условия задачи;
- результат - эти величины требуется найти;
- промежуточные данные - эти величины не известны заранее, но определяются в ходе решения задачи для того, чтобы найти результат.

Следующие действия на этом этапе - надо записать известные ограничения для величин и связи между ними в виде неравенств, формул и уравнений, а также определить условия окончания алгоритма.

Выбор метода решения задачи

Как правило, для большинства задач методы решения уже разработаны, и нередко в нескольких вариантах. Остается только выбрать тот, который больше всего отвечает некоторым требованиям (минимальный объем памяти, минимальная трудоемкость, максимальная эффективность, достаточная точность, допустимая погрешность вычислений и т.д.). В математике, например, известны методы точных вычислений и методы приближенных вычислений (для поиска корней нелинейных уравнений, вычисления определенных интегралов и т.п.).

Часто для реализации одного и того же численного метода можно предложить несколько алгоритмов, различающихся по своей простоте, объему вычислительной работы, емкостным затратам и т.д. Для решения задачи стараются выбрать алгоритм, обеспечивающий наиболее эффективное использование машины.

Разработка алгоритма

Этот этап, пожалуй, самый важный. На нем строится подробный план решения задачи. Нередко на этап построения алгоритма иногда смотрят как на вспомогательное действие, выполняемое непосредственно перед программированием. На самом деле успешная разработка алгоритма позволяет избежать многих ошибок, поскольку именно на этом этапе определяется логика будущей программы. А ведь труднее всего находить и исправлять логические ошибки.

Если к разработке алгоритма отнестись недостаточно внимательно, то в дальнейшем, на этапе программирования появятся трудности, алгоритм потребует дополнительной доработки, затраты новых усилий и т.п. А на этапе отладки программы может выясниться, что, к сожалению, алгоритм дает ошибки или вообще не выполним, и нужно разработать другой.

Удобнее всего сначала записать алгоритм в виде некоторого общего плана, состоящего из крупных этапов. Затем каждый этап разрабатывается более подробно. Изображать алгоритм целесообразнее всего в виде графической схемы. Необходимо продумать, в какой форме будет вводиться исходная информация для решения задачи, какие нужны промежуточные результаты, определить вид и структуру выходных данных (графики, таблицы и т.п.).

Программирование

Это процесс записи алгоритма на одном из алгоритмических языков программирования. Таких языков существует немало. Наиболее распространены Бейсик, Паскаль, СИ и др.

Следует отметить, что этап программирования не будет представлять большой сложности, если алгоритм разработан достаточно тщательно. Чем больше внимания Вы уделите разработке алгоритма, тем меньше времени придется затратить на написание программы, которое в этом случае будет выглядеть как простой перевод. Качественно разработанный алгоритм сможет программировать человек, владеющий языком программирования, но совершенно не знакомый ни с постановкой задачи, ни с методом ее решения.

Зачастую подавляющая часть времени при создании программы уходит не на ее написание, а на отладку. Поэтому программа должна быть наглядной, легко читаться, сопровождаться комментариями. В комментариях рекомендуется указывать назначение отдельных участков программы, смысл используемых переменных, пояснение наиболее сложных участков.

Отладка программы

Это процесс поиска (диагностики) и устранения ошибок в программе путем решения ее на контрольных (тестовых) примерах. Надо подобрать разные наборы исходных данных, чтобы проверить правильность работы программы при всех возможных исходах. Вместе с тем, исходные данные должны быть упрощенными настолько, чтобы на каждом этапе сверить значения промежуточных результатов с результатами ручного просчета. Если решение задачи вручную невозможно или требует больших затрат, то для сравнения можно взять результаты решения, полученные по другому алгоритму.

Главное правило подбора тестовых примеров состоит в следующем: тесты следует подбирать таким образом, чтобы они не подтверждали правильность программы, а выявляли имеющиеся в ней ошибки.

Ошибки программы бывают двух видов:

– ошибки ввода (описки) и синтаксические ошибки записи операторов - выявляются при трансляции программы на машинный язык. Программа-транслятор выводит на экран компьютера сообщение о характере ошибки и указывает место в программе, где эта ошибка была обнаружена.

– смысловые (алгоритмические) ошибки - выявляются на этапе отладки работающей программы, когда программа не производит в точности ожидаемых действий,

сообщений об ошибках при этом нет. Обнаруживаются такие ошибки только при сравнении результатов машинного и ручного просчета.

При отладке большой сложной программы используются следующие методы:

– распечатка значений промежуточных данных, помогает найти ошибки в вычислениях;

– трассировка программы - вывод номеров операторов по мере их фактического выполнения, дает возможность проследить последовательность выполнения отдельных действий и частей программы;

– пошаговое выполнение с просмотром содержимого памяти, выполняется с помощью специальной программы - диалогового отладчика;

– дамп (damp - разгрузка) - вывод содержимого памяти для анализа в определенный момент времени, например, в момент останова программы из-за ошибки.

Сложные программы для отладки обычно разбивают на несколько крупных частей, связь между которыми достаточно обозрима. После этого проводится автономная отладка каждой части, для чего подготавливаются специальные наборы контрольных вариантов.

В конце этапа отладки рекомендуется выполнить комплексное тестирование программы на контрольных исходных данных, максимально приближенных к реальным. Цель комплексной отладки - проверить правильность схемы организации программы в целом.

Подготовка исходных данных и непосредственное выполнение программы

Исходные данные, особенно очень объемные, могут быть подготовлены отдельно, на внешнем носителе (на жестком диске или дискете). В этом случае экономится время на их оперативный ввод (например, с клавиатуры).

Если все предыдущие этапы завершились благополучно, то выполнение программы происходит почти автоматически, не требует особых затрат (кроме времени) и может быть проведено человеком (оператором), не принимавшим участия в предварительной работе над задачей.

Понятие алгоритма. Сущность алгоритмизации.

Понятие алгоритма является фундаментальной категорией математики и не может быть выражено через другие, более простые понятия, а рассматривается как нечто неопределяемое. Другими словами, единого определения алгоритма не существует, есть только разные подходы, описания этого понятия, причем, в полном соответствии с той областью знаний, где он применяется. В рамках настоящего пособия не предусмотрено углубление в теорию алгоритмов (см., например, [12, 15]). Будем рассматривать понятие алгоритма и сущность процесса алгоритмизации в приложении к решению некоторых вычислительных задач. Опишем понятие алгоритма, например, так:

Алгоритм — это строгая, четкая последовательность математических и логических операций, приводящая к решению задачи.

В Толковом словаре по информатике (1991г.) дано общепринятое понятие: алгори–м - точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к искомому результату.

Алгоритмизация процессов в широком смысле — это описание процессов на языке математических символов для получения алгоритма, отображающего элементарные акты процесса, их последовательность и взаимосвязь. Для построения алгоритма управления, например, необходимо к алгоритму, описывающему процесс функционирования системы, присоединить алгоритм определения оптимального решения или оптимальных значений параметров управления. В более узком смысле **алгоритмизация** - это процедура поиска, разработки и описания алгоритма решения задачи.

Свойства алгоритма

Описание основных свойств помогает углубить само понятие алгоритма. Итак, алгоритм должен обладать следующими свойствами:

Детерминированность (определенность, точность, однозначность). Это свойство заключается в том, что при задании одних и тех же исходных данных несколько раз алгоритм будет выполняться абсолютно одинаково и всегда будет получен один и тот же результат. Свойство детерминированности проявляется также и в том, что на каждом шаге выполнения алгоритма всегда точно известно, что делать дальше, а каждое действие однозначно понятно исполнителю и не может быть истолковано неопределенно. Благодаря этому свойству выполнение алгоритма носит механический характер.

Массовость - выражается в том, что с помощью алгоритма можно решать не одну конкретную задачу, а любую задачу из некоторого класса однотипных задач при всех допустимых значениях исходных данных.

Результативность (направленность) - означает, что выполнение алгоритма обязательно должно привести к решению поставленной задачи, либо к сообщению о том, что при заданных исходных величинах задачу решить невозможно. Алгоритмический процесс не может обрываться безрезультатно.

Дискретность - означает, что алгоритм состоит из последовательности отдельных шаг–в - элементарных действий, выполнение которых не представляет сложности. Именно благодаря этому свойству алгоритм может быть реализован на ЭВМ.

Конечность (финитность)- заключается в том, что последовательность элементарных действий алгоритма не может быть бесконечной, неограниченной, хотя может быть очень большой (если требуется, например, большая точность вычислений).

Корректность - означает, что если алгоритм создан для решения определенной задачи, то для всех исходных данных он должен всегда давать правильный результат и ни для каких исходных данных не будет получен неправильный результат. Если хотя бы один из полученных результатов противоречит хотя бы одному из ранее установленных и получивших признание фактов, алгоритм нельзя признать корректным.

Если разработанная Вами последовательность действий не обладает хотя бы одним из перечисленных выше свойств, то она не может считаться алгоритмом

Средства описания алгоритма

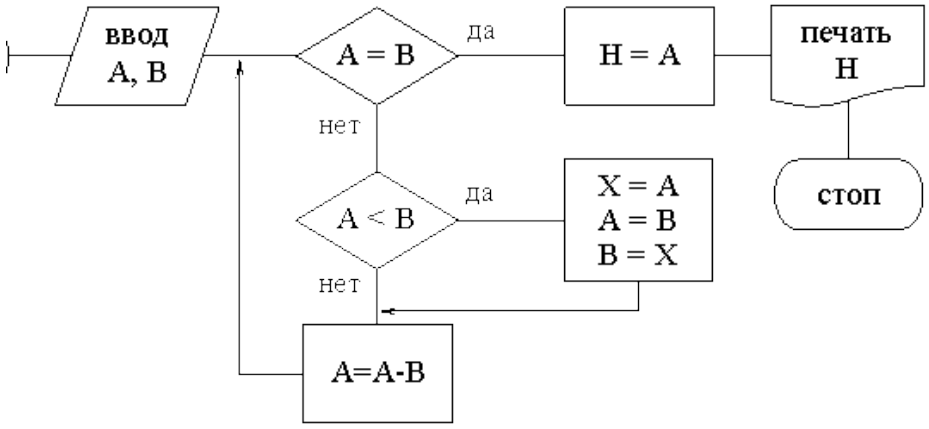
Описание алгоритма вполне допустимо **на естественном языке**, таком как русский, французский, английский, немецкий и др. Первое время это устраивало математиков. Однако постепенно выяснилось, что применение естественных языков в точных науках связано с рядом трудностей и даже может приводить к противоречиям. В естественных языках не всегда форме конкретного предложения соответствует единственное содержание. Предложения могут иметь расплывчатый смысл, требовать знания ситуации, контекста. Отдельные слова многозначны. Так, для обеспечения нужд науки стали возникать **формальные подъязыки** (смысл каждой фразы такого подъязыка определяется только его формой). Возникла идея построения искусственных формальных языков, в результате чего возникло множество алгоритмических языков.

Алгоритмический язык — это система обозначений формальной записи алгоритмов, предназначенных для некоторого исполнителя. Алгоритмический язык довольно близок к обычному разговорному, но более точный, конкретный, лаконичный. В составе алгоритмического языка - операторы, команды, служебные слова и служебные символы. Как и любой другой язык, он имеет свой синтаксис и семантику. Вот как выглядит алгоритм Евклида на алгоритмическом языке:

```
          алг ЕВКЛИД(цел A, B, H)
арг  A, B
рез  H
нач
нц     пока A ? B
если  A < B
то      X = A;  A = B;  B = X
```

все
A =–A - B
кц
H = A
вывод H
кон

Существенным недостатком алгоритмических языков является то, что представленные их средствами алгоритмы недостаточно наглядны, довольно объемны и громоздки. Описания сложных математических задач или процессов управления занимают сотни страниц. Сделать описание алгоритма более наглядным помогает его графическое изображение в виде **структурной схемы.** На рис.1 представлена схема алгоритма Евклида.



Структурная схема является ориентированным графом [5], у которого графические обозначения (символы, блоки) являются вершинами графа, а линии потока - ребрами. Укрупненная схема позволяет видеть функциональные связи между отдельными фрагментами, участками алгоритма, а болке "мелкие", подробные схемы содержат детали, элементарные действия, составляющие содержание этих фрагментов.

Для реализации на ПК алгоритм необходимо описать на одном из **языков программирования**. При всем своем разнообразии языки программирования (т.н. языки высокого уровня) основываются на сходстве с естественными языками, совместимы с общепринятыми математическими обозначениями и обладают еще более высокой степенью формализации, чем алгоритмические языки.

При вводе в ПК специальная программа-транслятор "переводит" алгоритм на **машинный алгоритмический язык**, в котором все данные и все действия представляются, в конечном счете, в виде двоичных чисел. После команды на запуск программа выполняется уже автоматически, а программный процессор при этом является физический моделью алгоритма выполнения программы.

Контрольные вопросы:

1. Что такое алгоритм? Откуда произошло это слово?
2. Каковы основные свойства алгоритма?
3. Как можно понять прерывность алгоритма? Приведите примеры.
4. Что означает однозначность алгоритма? Приведите примеры.
5. Что называется результативностью алгоритма? Приведите примеры.
6. Что такое достоверность алгоритма? Приведите примеры.
7. Как можно понимать формальное исполнение алгоритма?
8. Что такое способы записи и описание алгоритма?
9. Каковы особенности изображения алгоритма с помощью графика?
10. На какие группы делятся блоки?

11. Что называется простым алгоритмическим языком? И языком программирования?

Тема 3. Алфавит языка программирования Си. Структура программы.

Цель: ознакомиться с понятиями алфавит языка программирования, структорой и синтаксисом языка программирования Си.

Программа на языке Си состоит из одной или более подпрограмм, называемых функциями.

Язык Си является блочно-структурированным. Каждый блок заключается в фигурные скобки {}.

Основным блоком в программе консольного приложения на языке Си является главная функция, имеющая имя main().

Каждое действие в языке Си заканчивается символом «точка с запятой» — ;. В качестве действия может выступать вызов функции или осуществление некоторых операций.

Имя функции — это коллективное имя группы описаний и операторов, заключенных в блок (фигурные скобки). За именем функции в круглых скобках указываются параметры функции.

Директивы препроцессора в Си

Препроцессор — это специальная программа, являющаяся частью компилятора языка Си. Она предназначена для предварительной обработки текста программы. Препроцессор позволяет включать в текст программы файлы и вводить макроопределения.

Работа препроцессора осуществляется с помощью специальных директив (указаний). Они отмечаются знаком решетка #. По окончании строк, обозначающих директивы в языке Си, точку с запятой можно не ставить.

Основные директивы препроцессора

#include — вставляет текст из указанного файла

#define — задаёт макроопределение (макрос) или символическую константу

#undef — отменяет предыдущее определение

#if — осуществляет условную компиляцию при истинности константного выражения

#ifdef — осуществляет условную компиляцию при определённости символической константы

#ifndef — осуществляет условную компиляцию при неопределённости символической константы

#else — ветка условной компиляции при ложности выражения

#elif — ветка условной компиляции, образуемая слиянием else и if

#endif — конец ветки условной компиляции

#line — препроцессор изменяет номер текущей строки и имя компилируемого файла

#error — выдача диагностического сообщения

#pragma — действие, зависящее от конкретной реализации компилятора.

Директива #include

Директива #include позволяет включать в текст программы указанный файл. Если файл является стандартной библиотекой и находится в папке компилятора, он заключается в угловые скобки <>.

Если файл находится в текущем каталоге проекта, он указывается в кавычк«» "". Для файла, находящегося в другом каталоге необходимо в кавычках указать полный путь.

Комментарии в языке С

В языке Си для комментариев используются символы
/* — начало комментария;

*/ — конец комментария.

Вся последовательность, заключенная между этими символами, является комментарием.

Это удобно для написания многострочных комментариев:

int a; /*целая переменная */

Многострочные комментарии также удобно использовать при отладке для сокрытия от выполнения части кода.

В дополнение к этому, для написания коротких комментариев могут использоваться символы //. При этом комментарием является все, что расположено после символов // и до конца строки:

float b; // вещественная переменная

Главная функция

При выполнении консольного приложения, написанного на языке Си, операционная система компьютера передаёт управление функции с именем main(). Функцию main() нельзя вызывать из других функций программы, она является управляющей.

Следующие за именем функции круглые скобки предназначены для указания параметров (аргументов), которые передаются в функцию при обращении к ней. В данном случае операционная система не передаёт в функцию main() никаких аргументов, поэтому список аргументов в круглых скобках пустой.

Главную функцию можно записать по-разному:

int main()

void main().

Перед именем функции указывается тип возвращаемого значения. При обращении к главной функции значение возвращается операционной системе. Последняя запись не будет возвращать значения. Однако void main() — не совсем корректная запись, так как сообщает компилятору, что функция main() не возвращает никакого значения.

При этом запись int main() сообщает компилятору о возвращении целочисленного значения, которое необходимо операционной системе и сообщает ей о том, что программа завершилась корректно. Если же это значение не возвращено, то операционная система понимает, что программа завершилась в аварийном режиме.

Для возврата целочисленного значения перед завершением функции добавляется строка

return 0; // вещественная переменная

В фигурные скобки заключены описания и операторы.

В общем случае программа может содержать несколько функций. Каждая функция имеет список передаваемых в нее параметров, указанный в круглых скобках, и набор операций, заключенных в блок, ограниченный фигурными скобками.

Пример: Вывод на экран сообщен«я "Hello, wor»d!".

```
#include <stdio.h>  // Подключение библиотеки ввода-вывода
int main()        // Главная функция
{
  prin«f("Hello, wor»d!"); // Вывод сообщения
  getchar();              // Задержка окна консоли
  return 0;
}
```

Теперь попробуем написать текст на русском языке.

```
#include <stdio.h>
int main()
{
 prin«f("Здравствуй, м»р!");
 getchar();
 return 0;
}
```

Проблема русского языка в консольных приложениях заключается в том, что консоль и редактор кода Microsoft Visual Studio поддерживают разные кодовые страницы. Для того, чтобы увидеть русские символы в консоли необходимо поменять кодовую страницу в консоли, чтобы она соответствовала кодовой странице редактора (1251). С этой целью вызывается функция syst«m("chcp 1»51") с соответствующей командной строкой. Прототип функции system() содержится в библиотеке <stdlib.h>.

При этом текст программы будет выглядеть следующим образом:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 syst«m("chcp 1»51");       // Текущая кодовая страница 1251
 syst«m("»ls");             // Очистка консоли
 prin«f("Здравствуй, м»р!"); // Вывод сообщения
 getchar();
 return 0;
}
```

Алфавит языка Си

Алфавит Си включает:

– прописные и строчные буквы английского алфавита: A,…,Z, a,…,z;

– арабские цифры 0, 1,…, 9;

– специальные символы:

– пробельные символы (неотображаемые символы): пробел, табуляция, перевод строки, новая строка, возврат каретки, новая страница, вертикальная табуляция,

– другие символы: , . ; : ? ‘ ! “ / | \ ~ _ ( ) { } [ ] > < # % & ^ - = + * (в некоторых компиляторах, например, Visual C++ разрешается в идентификаторах использовать символ $).

Символы алфавита находятся в первой половине кодовой таблицы (первые 128 символов) кодировки ASCII. Из символов алфавита строятся лексемы.

Лексема – единица текста программы, имеющая для компилятора самостоятельный смысл. Примеры лексем: идентификаторы, ключевые (зарезервированные) слова, знаки операций, константы, разделители.

Идентификаторы (имена) языка Си

Идентификатор (имя переменной, функции или другого объекта) – это последовательность букв английского алфавита, десятичных цифр, символа подчеркивания, начинающаяся не с цифры (в некоторых компиляторах, например, Visual C++ разрешается в идентификаторах использовать символ $).

Прописные и строчные буквы в идентификаторах различаются (идентификаторы x и X это разные имена).

Примеры верных идентификаторов: x, X, abc124, A_c3, x4er, x_, __d.

Примеры неверных идентификаторов (первый символ – цифра): 1c, 4sd.

Контрольные вопросы:

1. Что такое директивы препроцессора?

Какова структура программы на Си?

2. Что входит в алфавит языка программирования?
3. Что такое лексема?
4. Что такое главная функция?

Тема 4. Типы данных и константы.

Цель: изучить типы данных и константы. Познакомится с основными арифметическими операциями и функциями

Тип данных определяет множество значений, набор операций, которые можно применять к таким значениям и способ реализации хранения значений и выполнения операций.

Процесс проверки и накладывания ограничений на типы используемых данных называется контролем типов или типизацией программных данных. Различают следующие виды типизации:

Статическая типизация — контроль типов осуществляется при компиляции.

Динамическая типизация — контроль типов осуществляется во время выполнения.

Язык Си поддерживает статическую типизацию, и типы всех используемых в программе данных должны быть указаны перед ее компиляцией.

Различают простые, составные и прочие типы данных.

Простые данные

Простые данные можно разделить на

- целочисленные,
- вещественные,
- символьные
- логические.

Составные (сложные) данные

Массив — индексированный набор элементов одного типа.

Строковый тип — массив, хранящий строку символов.

Структура — набор различных элементов (полей записи), хранимый как единое целое и предусматривающий доступ к отдельным полям структуры.

Другие типы данных

Указатель — хранит адрес в памяти компьютера, указывающий на какую-либо информацию, как правило — указатель на переменную.

Программа, написанная на языке Си, оперирует с данными различных типов. Все данные имеют имя и тип. Обращение к данным в программе осуществляется по их именам (идентификаторам).

Идентификатор — это последовательность, содержащая не более 32 символов, среди которых могут быть любые буквы латинского алфавита a — z, A — Z, цифры 0 — 9 и знак подчеркивания (_). Первый символ идентификатора не должен быть цифрой.

Несмотря на то, что допускается имя, имеющее до 32 символов, определяющее значение имеют только первые 8 символов. Помимо имени, все данные имеют тип. Указание типа необходимо для того, чтобы было известно, сколько места в оперативной памяти будет занимать данный объект.

Компилятор языка Си придерживается строгого соответствия прописных и строчных букв в именах идентификаторов и лексем.

Верно	Неверно

int a = 2, b;
b = a+3;
	Int a=2; // правильно int
	INT a=2;

	int a = 2, b; b = A + 3; // идентификатор A не объявлен
	int a = 2; b = a + 3; // идентификатор b не объявлен

Целочисленные данные

Целочисленные данные могут быть представлены в знаковой и беззнаковой форме.

Беззнаковые целые числа представляются в виде последовательности битов в диапазоне от 0 до 2n-1, где n-количество занимаемых битов.

Знаковые целые числа представляются в диапазоне -2n-1…+2n-1-1. При этом старший бит данного отводится под знак числа (0 соответствует положительному числу, 1 – отрицательному).

Основные типы и размеры целочисленных данных:

Количество бит	Беззнаковый тип	Знаковый тип
8	unsigned char 0…255	char -128…127
16	unsigned short 0…65535	short -32768…32767
32	unsigned int	int
64	unsigned long int	long int

Вещественные данные

Вещественный тип предназначен для представления действительных чисел. Вещественные числа представляются в разрядной сетке машины в нормированной форме.

Нормированная форма числа предполагает наличие одной значащей цифры (не 0) до разделения целой и дробной части. Такое представление умножается на основание системы счисления в соответствующей степени.

Например, число 12345,678 в нормированной форме можно представить как

12345,678 = 1,2345678·104

Число 0,009876 в нормированной форме можно представить как

0,009876 = 9,876·10-3

В двоичной системе счисления значащий разряд, стоящий перед разделителем целой и дробной части, может быть равен только 1. В случае если число нельзя представить в нормированной форме (например, число 0), значащий разряд перед разделителем целой и дробной части равен 0.

Значащие разряды числа, стоящие в нормированной форме после разделителя целой и дробной части, называются мантиссой числа.

В общем случае вещественное число в разрядной сетке вычислительной машины можно представить в виде 4 полей.

знак — бит, определяющий знак вещественного числа (0 для положительных чисел, 1 — для отрицательных).

степень — определяет степень 2, на которую требуется умножить число в нормированной форме. Поскольку степень 2 для числа в нормированной форме может быть как положительной, так и отрицательной, нулевой степени 2 в представлении вещественного числа соответствует величина сдвига, которая определяется как

2n-1,

где n — количество разрядов, отводимых для представления степени числа.

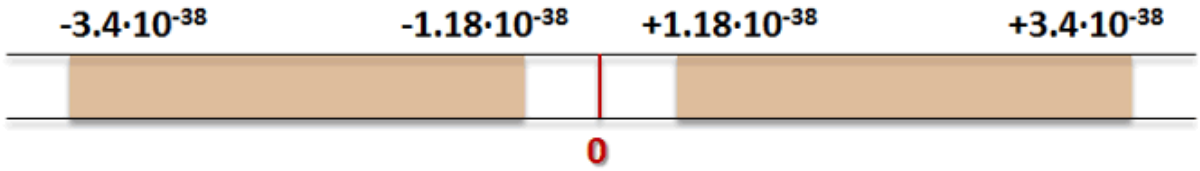
целое — бит, который для нормированных чисел всегда равен 1, поэтому в некоторых представлениях типов этот бит опущен и принимается равным 1.

мантисса — значащие разряды представления числа, стоящие после разделителя целой и дробной части в нормированной форме.

Различают три основных типа представления вещественных чисел в языке Си:

Тип	Обозначение в Си	Кол-во бит	Биты степени	Мантисса	Сдвиг
простое	float	32	30…23	22…0	127
двойной точности	double	64	62…52	51…0	1023
двойной расширенной точности	long double	80	78…64	62…0	16383

Как видно из таблицы, бит целое у типов float и double отсутствует. При этом диапазон представления вещественного числа состоит из двух диапазонов, расположенных симметрично относительно нуля. Например, диапазон представления чисел типа float можно представить в виде:



Пример: представить число -178,125 в 32-разрядной сетке (тип float).

Для представления числа в двоичной системе счисления преобразуем отдельно целую и дробную части:

17810 = 101100102.

0,12510 = 0,0012.

Тогда

178,12510 = 10110010,0012=1,0110010001·2111

Для преобразования в нормированную форму осуществляется сдвиг на 7 разрядов влево).

Для определения степени числа применяем сдвиг:

0111111+00000111 = 10000110.

Таким образом, число -178,125 представится в разрядной сетке как

Символьный тип

Символьный тип хранит код символа и используется для отображения символов в различных кодировках. Символьные данные задаются в кодах и по сути представляют собой целочисленные значения. Для хранения кодов символов в языке Си используется тип char.

Логический тип

Логический тип применяется в логических операциях, используется при алгоритмических проверках условий и в циклах и имеет два значения:

истина — true

ложь — — false

В программе должно быть дано объявление всех используемых данных с указанием их имени и типа. Описание данных должно предшествовать их использованию в программе.

Пример объявления объектов

```
 int n;    // Переменная n целого типа
double a; // Переменная a вещественного типа двойной точности
```

Константы и переменные в языке Си

Константа — это ограниченная последовательность символов алфавита языка, представляющая собой изображение фиксированного (неизменяемого) объекта.

Константы бывают числовые, символьные и строковые. Числовые константы делятся на целочисленные и вещественные.

Целочисленные константы

Целочисленные данные в языке Си могут быть представлены в одной из следующих систем счисления:

Десятичные	Последовательность цифр (0 — 9), которая начинаются с цифры, отличной от нуля. Пример: 1, -29, 385. Исключение — число 0.
Восьмеричные	Последовательность цифр (0 — 7), которая всегда начинается с 0. Пример: 00, 071, -052, -03.
Шестнадцатеричные	Последовательность шестнадцатеричных цифр (0 — 9 и A — F), которой предшествует присутствует 0x или 0X. Пример: 0x0, 0x1, -0x2AF, 0x17.

Двоичная система представления данных непосредственно в языке Си не поддерживается. Однако можно воспользоваться файлом binary.h, в котором определены двоичные константы в пределах байта.

Пример использования двоичной системы счисления в языке Си:

```
    #include <stdio.h>
#inclu«e "binar».h"
int main()
{
  unsigned char p = b10001001 | b00001010; // p=b10001011=139
  prin«f("p = %dd = »xh", p, p);
  getchar(); getchar();
  return 0;
}
```

В зависимости от значения целой константы компилятор присваивает ей тот или иной тип (char, int, long int).

С помощью суффикса U (или u) можно представить целую константу в виде беззнакового целого.

Например, Константе 200U выделяется 1 байт, и старший бит используется для представления одного из разрядов кода числа и диапазон значений становится от 0 до 255. Суффикс L (или l) позволяет выделить целой константе 8 байт (long int).

Совместное использование в любом порядке суффиксов U (или u) и L (или l) позволяет приписать целой константе тип unsigned long int, и она займет в памяти 64

разряда, причем знаковый разряд будет использоваться для представления разряда кода (а не знака).

Вещественные константы

Константа с плавающей точкой (вещественная константа) всегда представляется числом с плавающей точкой двойной точности, Т. е. как имеющая тип double, и состоит из следующих частей:

- целой части — последовательности цифр;
- точки — разделителя целой и дробной части;
- дробной части — последовательности цифр;
- символа экспоненты e или E;
- экспоненты в виде целой константы (может быть со знаком).

Любая часть (но не обе сразу) из нижеследующих пар может быть опущена:

- целая или дробная часть;
- точка или символ e (E) и экспонента в виде целой константы.

Примеры вещественных констант

345.
3.14159
2.1E5
.123E3
4037e-5

По умолчанию компилятор присваивает вещественному числу тип double. Если программиста не устраивает тип, который компилятор приписывает константе, то тип можно явно указать в записи константы с помощью следующих суффиксов:

F (или f) — float для простых вещественных констант,
L (или l) — long double для вещественных констант двойной расширенной точности.
Примеры:
3.14159F — константа типа float, занимающая 4 байта;
3.14L — константа типа long double, занимающая 10 байт.

Символьные константы

Символьная константа — это один символ, например: ‘z’. В качестве символьных констант также могут использоваться управляющие коды, не имеющие графического представления. При этом код управляющего символа начинается с символа ‘\’ (обратный слеш).

Код	Обозначение	Описание
0x00	‘\0’	Нуль-символ, NULL
0x07	‘\a’	Звуковой сигнал.
0x08	‘\b’	Возврат на 1 шаг (Backspace)
0x09	‘\t’	Горизонтальная табуляция (Tab)
0x0A	‘\n’	Перевод строки (Enter)
0x0B	‘\v’	Вертикальная табуляция (в консоли аналогична переводу строки)
0x0C	‘\f’	Смена страницы
0x0D	‘\r’	Возврат каретки

Как правило, нажатие клавиши Enter генерирует сразу два управляющих символа — перевод строки (0x0A) и возврат каретки (0x0D).

Все символьные константы имеют тип char и занимают в памяти 1 байт. Значением символьной константы является числовое значение её внутреннего кода.

троковые константы

Строковая константа — это последовательность символов, заключенная в кавычки, например: «Это строковая константа»

Кавычки не входят в строку, а лишь ограничивают её. Технически строковая константа представляет собой массив символов, и по этому признаку может быть отнесена к разряду сложных объектов языка Си.

В конце каждой строковой константы компилятор помещает ‘\0’ (нуль-символ), чтобы программе было возможно определить конец строки. Такое представление означает, что размер строковой константы не ограничен каким-либо пределом, но для определения длины строковой константы её нужно полностью просмотреть.

Поскольку строковая константа состоит из символов, то она имеет тип char. Количество ячеек памяти, необходимое для хранения строковой константы на 1 больше количества символов в ней (1 байт используется для хранения нуль-символа).

Символьная константа ‘x’ и строка из одного символа «x» — не одно и то же. Символьная константа — это символ, используемый для числового представления буквы x, а строковая константа «x» содержит символ ‘x’ и нуль-символ ‘\0’ и занимает в памяти 2 байта. Если в программе строковые константы записаны одна за другой через разделители, то при выполнении программы они будут размещаться в последовательных ячейках памяти.

Переменные

Переменная — идентификатор, представляющий собой изображение изменяемого объекта. С технической точки зрения, переменная — это область памяти, в которую могут помещаться различные значения.

Любая переменная до ее использования в программе на языке Си должна быть объявлена, то есть для нее должны быть указаны тип и имя (идентификатор).

Объявление переменных в Си осуществляется в форме

ТипПеременной ИмяПеременной;

Каждую переменную можно снабдить комментарием, поясняющим ее смысл. Например,

```
int i; // счетчик циклов
```

Если в программе требуется несколько переменных одного типа, то они могут быть объявлены в одной строке через запятую. Например,

```
int i, n;
```

При объявлении переменной ей может быть присвоено начальное значение в форме

ТипПеременной ИмяПеременной=значение;

Контрольные вопросы:

1. Какие символы разрешено использовать в именах переменных языка Си?
2. Каких принципов следует придерживаться при назначении имен переменных и констант?
3. Укажите вещественные типы данных?
4. Укажите целые типы данных?

Тема 5. Обзор операций: операции, выражения. Арифметические операции. Логические операции.

Цель: изучить понятия «операция», «выражение», арифимитические и логические операции.

Операторы

Программа состоит из команд, называемых по-другому инструкциями. Команды заставляют компьютер выполнить требуемое действие. Некоторые команды состоят из

нескольких, скажем так, более мелких команд. Например, команда «Вывести на экран **10** чисел» состоит из десяти команд «Вывести одно число». Элементарными, самыми мелкими, командами являются операторы; именно они представляют собой кирпичики, образующие программу, подобно строительству дома. К настоящему моменту мы уже познакомились с одним оператором – оператором присваивания. В дальнейшем будут рассмотрены ряд других операторов: условные операторы, циклы и др.

Операции и выражения

Операторы производят свои действия над переменными, константами и выражениями. В предыдущей теме были рассмотрены переменные, которые представляют собой именованные ячейки памяти, предназначенные для хранения и обработки данных. Для работы с переменными и константами предназначены операции.

Операция – это символ, представляющий собой некоторую операцию, производимую с данными. В языке Си существуют несколько видов операций: арифметические операции, операции сравнения, битовые и логические операции. Все эти операции будут рассмотрены ниже.

Переменные и константы, над которыми производится операция, называются операндами. Операция, которая воздействует на два операнда, называется бинарной, а операция, воздействующая на один операнд – унарной. В математике также используется термин n-арная операция, если производится действие над **n** операндами.

Большинство операций являются бинарными: сложение, умножение, деление, различные варианты операций сравнения и т.д. Унарной операцией является «минут», например, **-7**.

Комбинация операций и операндов называется выражением. Простейшее выражение может состоять только из одного операнда, а выражение, являющееся частью другого, называется подвыражением. Посмотрите на примеры выражений:

1. 7
2. a*b
3. 15>10
4. -8
5. (x+4)*(y+8)

Первое выражение состоит из одного операнда; во втором выражении используется бинарная операция умножения, а в третьем – бинарная операция сравнения; в четвертом выражении используется унарная операция «минус»; в пятом выражении **x+4** и **y+8** являются подвыражениями.

Важным свойством выражений является то, что все они имеют значение или, если речь идет о числах, вычисляют определенное значение. Значение выражения, представляет собой результат выполнения входящих в него операций. Значение выражения определяется в момент исполнения программы и может отличаться от запуска к запуску. Например, выражения 1, 3 и 4 всегда имеют одни и те же значения, равные **7**, «истина» и **-8** соответственно. Остальные выражения зависят от значений, которые присвоены входящим в них переменным.

Оператор присваивания

Один из наиболее важных и часто применяемых операторов в любом языке программирования – это оператор присваивания. Задача данного оператора заключается в том, чтобы присвоить переменной нужное значение. Таким образом, при объявлении переменной происходит выделение ячейки памяти под эту переменную, а присвоение значения заключается в занесении значения в эту ячейку памяти. Символом оператора присваивания является знак «равно». Например, запись

number = 10;

означает, что переменной с именем number присваивается значение **10**.

Элемент слева от знака присваивания должен быть именем переменной, а значение

справа – выражением. Причем, в этом выражении может присутствовать и переменная, которой присваивается значение. В частности, для увеличения переменной **x** на единицу можно воспользоваться следующей записью:

x=x+1;

Арифметические операции

Наиболее часто используемым классом операций языка Си являются арифметические операции. К ним относятся операции сложения, вычитания, умножения, деления, взятия остатка от деления и операция изменения знака.

Операция	**Описание**	**Пример**
+	Сложение	z = x+y
-	Вычитание	z = x-y
-	Изменение знака	z = -x
*	Умножение	z = x*y
/	Деление	z = x/y
%	Остаток от деления	z = x%y

Операции сравнения

Операции сравнения предназначены для определения отношения между двумя переменными или между переменной и константой. Например, **a<b** или **a>10**. Результатом операции сравнения является значение типа bool, т.е. true или false. Операции сравнения отвечают «Да» или «Нет» на вопрос «Связаны ли элементы указанным отношением?»

Операция	**Описание**	**Пример**
<	Меньше	x<y
>	Больше	x>y
<=	Меньше либо равно	x<=y
>=	Больше либо равно	x>=y
==	Равно	x==y
!=	Не равно	x!=y

Обратите особое внимание на то, что операция проверки на равенство обозначается знаком двойного равенства, поскольку знак равенства уже занят под операцию присваивания.

Логические операции

Логические операции служат для соединения выражений, которые содержат операции сравнения.

Оператор	**Описание**	**Пример**
&&	Логическое «и»	10<x && x<20
||	Логическое «или»	X%3==0 || y%7==0
!	Логическое «не»	!(x==10)

Результаты логических выражений определяются согласно так называемой таблице истинности.

x	**y**	**x&&y**	**x||y**	**!x**
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false

true	true	true	true	false

Значения true и false являются своеобразными константами, созданными для того, чтобы программисту-человеку было проще с ним управляться, однако с точки зрения компьютера эти константы равны **1** и **0** соответственно.

К теме логических операций мы еще вернемся в теме «Условия», поскольку именно в условиях логические операции в полной мере раскрывают свой потенциал.

Приоритет операций

Рассмотрим следующий фрагмент кода:

result = 5*x*x – 2*y + 10/z;

В каком порядке будут выполняться присутствующие в этом фрагменте операции, есть запустить программу? Очевидно, что результат будет различным, если изменять порядок операций. Для примера положим, что **x=2**, **y=3**, а **z=5**. Если выполнить вначале все умножения и деление, а затем вычислить результаты сложения и вычитания, то переменная result будет равна 16. Однако, если вначале выполнить вычитание и сложение, то результат будет равен 0. Если поизменять порядок еще как-нибудь, что будут получаться новые результаты.

Рассмотренный пример говорит нам о том, что во избежание таких неоднозначностей язык программирования должен иметь правило, регламентирующее порядок выполнения операций для всех возможных ситуаций. Такое правило существуют, и основано оно на понятии приоритета операций. Правило заключается в том, что каждой операции назначается приоритет, и операции, имеющие более высокий приоритет, выполняются раньше. Если несколько операций имеют одинаковый приоритет, то они выполняются в установленном для них порядке: как правило, слева направо. Приоритеты арифметических операций, как нетрудно догадаться, соответствуют правилам математики.

В языке Си наивысший приоритет имеют арифметические операции, затем – операции сравнения и, наконец, логические операции имеют самый низкий приоритет. Это правило довольно естественно, поскольку именно в таком порядке эти операции выполняются и в реальной жизни: вначале что-то вычисляется, затем результаты вычислений сравниваются, а в последнюю очередь результаты сравнений комбинируются при помощи логических операций.

Приоритет арифметических операций определяется согласно следующей таблице:

Операция	**Порядок вычислений**
()	Слева направо
- (унарный)	Справа налево
*, /, %	Слева направо
+, - (бинарные)	Слева направо

Приоритет операций сравнения задается согласно следующей таблице:

Операция	**Порядок вычислений**
<, >, <=, >=	Слева направо
==, !=	Слева направо

Приоритет	Знак операции	Типы операции
2	() [] . ->	Выражение
1	- ~ ! * & ++ -- sizeof приведение типов	Унарные
3	* / %	Мультипликативные
4	+ -	Аддитивные
5	<< >>	Сдвиг
6	< > <= >=	Отношение
7	== !=	Отношение (равенство)
8	&	Поразрядное И
9	^	Поразрядное исключающее ИЛИ
10	|	Поразрядное ИЛИ
11	&&	Логическое И
12	||	Логическое ИЛИ
13	? :	Условная
14	= *= /= %= += -= &= |= >>= <<= ^=	Простое и составное присваивание
15	,	Последовательное вычисление

Контрольные вопросы :

1. Что такое операции и операнды. Приведите примеры.
2. Что такое унарные, бинарные и **n**-арные операции. Приведите примеры.
3. Что такое выражение и подвыражение? Приведите примеры.
4. Какие логические операции вам известны? Как они обозначаются в языке Си?

Тема 6. Операция присваивания. Явное и неявное преобразование типов данных

Цель: изучить операторы присвивания, явное и неявное преобразование типов данных, управляющие константы для форматированного вывода.

Операции присваивания позволяют присвоить некоторое значения. Эти операции выполняются над двумя операндами, причем левый операнд может представлять только модифицируемое именованное выражение, например, переменную.

Базовая операция присваивания = позволяет присвоить значение правого операнда левому операнду:

```
1
2
int x;
x = 2
```

То есть в данном случае переменная x (левый операнд) будет иметь значение 2 (правый операнд).

Стоит отметить, что тип значения правого операнда не всегда может совпадать с типом левого операнда. В этом случае компилятор пытается преобразовать значение правого операнда к типу левого операнда.

При этом операции присваивания имеют правосторонний порядок, то есть выполняются справа налево. И, таким образом, можно выполнять множественное присваивание:

```
int a, b, c;
a = b = c = 34;
```

Здесь сначала вычисляется значение выражения c = 34. Значение правого операн–a - 34 присваивается левому операнду c. Далее вычисляется выражение b = c: значение правого операнда c (34) присваивается левому операнду b. И в конце вычисляется выражение a = b: значение правого операнда b (34) присваивается левому операнду a.

При динамической типизации переменная связывается с типом на момент инициализации. Получается, что переменная в разных участках кода может иметь разные типы.

Статическая типизация является противоположностью динамической. При объявлении переменная получает тип, который не меняется в дальнейшем. Языки Си и Си++ являются именно такими. Этот способ наиболее удобный для написания сложного кода, а на стадии компиляции исключается много ошибок.

Языки неформально делятся на сильнотипизированный и слаботипизированный. Сильная типизация подразумевает, что компилятор выдаст ошибку при несовпадении ожидаемого и фактического типов.

x = 1 + “2”; //ошиб–а - нельзя прибавить к числу символьный знак

Пример слабой типизации.

x = 1 + “2”; // 3

Проверка согласования типов осуществляется системой типобезопасности. Ошибка типизации возникает, например, при попытке использовать число как функцию. Существуют нетипизированные языки. В противоположность типизированным, они позволяют осуществлять любые операции над каждым объектом.

Классы памяти

Переменные, независимо от их типа, имеют свою область видимости и время существования.

Классы памяти:

- auto;
- static;
- extern;
- register.

Все переменные в языке Си по умолчанию являются локальными. Они могут использоваться только внутри функции или блока. По завершении функции их значение уничтожается.

Статическая переменная также является локальной, но вне своего блока может иметь другое значение, а между вызовами функции значение сохраняется.

Внешняя переменная является глобальной. Она доступна в любой части кода и даже в другом файле.

Регистровая переменная рекомендует компилятору сохранять значение в оперативную память.

Спецификаторы типов данных в Си могут не указываться в таких случаях:

1. Все переменные внутри блока не являются переменными, соответственно, если предполагается использование именно этого класса памяти, то спецификатор auto не указывается.

2. Все функции, объявленные вне блока или функции, являются по умолчанию глобальными, поэтому спецификатор extern не обязателен.

Преобразование типов

Если в арифметических операциях участвуют значения разных типов, то компилятор неявно пытается привести их к одному типу. Кроме того, когда мы присваиваем переменной какое-либо значение, это значение всегда приводится к типу переменной. Например:

```
1       char c = 6;
2       int d = c;
```

Переменной d, которая представляет тип int, присваивается значение типа char, поэтому компилятор выполняет приведение значения от типа char к типу int.

В то же время не всегда преобразования могут быть безопасными, поскольку разные типы имеют разное внутреннее представление. И просто так перейти от одного представления к другому без потери точности данных не всегда возможно.

Рассмотрим, какие преобразования применяет компилятор при арифметических операциях:

Если один из операндов имеет тип long double, то второй операнд тоже будет преобразован в тип long double

Если предыдущий пункт не выполняется и если один из операндов имеет тип double, то второй операнд тоже будет преобразован к типу double

Если предыдущий пункт не выполняется и если один из операндов имеет тип float, то второй операнд тоже будет преобразован к типу float

Если предыдущий пункт не выполняется и если один из операндов имеет тип unsigned long int, то второй операнд тоже будет преобразован к типу unsigned long int

Если предыдущий пункт не выполняется и если один из операндов имеет тип long, то второй операнд тоже будет преобразован к типу long

Если предыдущий пункт не выполняется и если один из операндов имеет тип unsigned, то второй операнд тоже будет преобразован к типу unsigned

Если предыдущий пункт не выполняется то оба операнда приводятся к типу int

Например:

```
1       int a = 10;
2       double b = 4;
3       double c = a + b;   // 14.000000
```

В выражении a + b число b представляет тип double, поэтому число a будет автоматически приводиться к числу double. И результат операции сложения также будет представлять тип double.

Операция преобразования

С помощью специальной операции преобразования мы можем явным образом привести данные к нужному типу. Например:

```
int a = 10;
int b = 4;
int c = a / b;                  // 2
double d = a / b;               // 2.00000
double e = (double)a / (double)b;   // 2.50000
prin«f("c = %d»\n", c);
prin«f("d = %f»\n", d);
prin«f("e = %f»\n", e);
```

В выражении int c = a / b; результат деления будет целочисленн–й - 2, при котором дробная часть будет отброшена, так как оба операнда операции представляют целые числа.

В выражении double d = a / b; результат деления будет представлять вещественное чис–о - 2.00000, но так как оба операнда являются целыми числами, то опять же результат операции будет представлять целое число 2, и только поле выполнения деления произойдет присвоение результата переменной d с приведением значения 2 от типа int к типу double.

В выражении double e = (double)a / (double)b применяется явное преобразование данных к типу double, поэтому и результат деления будет представлять вещественное чис–о - 2.50000.

Для выполнения операции приведении в скобках указывается тот тип, к которому надо привести значение:

```
int number = 70;
char symbol = (char) number;
prin«f("symbol = %c»\n", symbol);              //  F
prin«f("symbol (int code) = %d»\n", symbol);    // 70
```

В ряде случаев преобразования сопровождаются потерей информации. Без потери информации проходят следующие цепочки преобразований:

char -> short -> int -> long

unsigned char -> unsigned short -> unsigned int -> unsigned long

float -> double -> long double

При всех остальных преобразованиях, которые не входят в эти цепочки, мы можем столкнуться с потерей точности данных. Так, в примере выше преобразование от int к char не является безопасным, поэтому к таким преобразованиям следует относиться с осторожностью.

Задание :Создать кластер на тему «Типы данных»

Контрольные вопросы

1. Что определяет тип данных?
2. На какие две группы можно разделить все типы языка С?
3. Какие типы данных относятся к основным типам?
4. Какие типы данных являются составными?
5. Какие ключевые слова определены для описания основных типов данных (целого, символьного, расширенного символьного, логического, вещественного, вещественного с двойной точностью)?
6. Назовите четыре спецификатора типа, уточняющих внутреннее представление и диапазон значений стандартных типов

Тема 7. Организация консольного ввода/вывода данных

Цель: изучить операторы консольного ввода /ввывода

Основной задачей программирования является обработка информации, поэтому любой язык программирования имеет средства для ввода и вывода информации.

Ввод и вывод информации осуществляется через функции стандартной библиотеки. Прототипы рассматриваемых функций находятся в файле stdio.h. Эта библиотека содержит функции

printf() — для вывода информации

scanf() — для ввода информации.

Вывод информации

Функция printf() предназначена для форматированного вывода. Она переводит данные в символьное представление и выводит полученные изображения символов на экран. При этом у программиста имеется возможность форматировать данные, то есть влиять на их представление на экране.

Общая форма записи функции printf():

prin«f("СтрокаФорма»ов", объект1, объект2, ..., объектn);

Строка Форматов состоит из следующих элементов:

- управляющих символов;
- текста, представленного для непосредственного вывода;
- форматов, предназначенных для вывода значений переменных различных типов.

Объекты могут отсутствовать.

Управляющие символы не выводятся на экран, а управляют расположением выводимых символов. Отличительной чертой управляющего символа является наличие обратного слэша ‘\’ перед ним.

Основные управляющие символы:

‘\n’ — перевод строки;

‘\t’ — горизонтальная табуляция;

‘\v’ — вертикальная табуляция;

‘\b’ — возврат на символ;

‘\r’ — возврат на начало строки;

‘\a’ — звуковой сигнал.

Форматы нужны для того, чтобы указывать вид, в котором информация будет выведена на экран. Отличительной чертой формата является наличие символа процент ‘%’ перед ним:

%d — целое число типа int со знаком в десятичной системе счисления;

%u — целое число типа unsigned int;

%x — целое число типа int со знаком в шестнадцатеричной системе счисления;

%o — целое число типа int со знаком в восьмеричной системе счисления;

%hd — целое число типа short со знаком в десятичной системе счисления;

%hu — целое число типа unsigned short;

%hx — целое число типа short со знаком в шестнадцатеричной системе счисления;

%ld — целое число типа long int со знаком в десятичной системе счисления;

%lu — целое число типа unsigned long int;

%lx — целое число типа long int со знаком в шестнадцатеричной системе счисления;

%f — вещественный формат (числа с плавающей точкой типа float);

%lf — вещественный формат двойной точности (числа с плавающей точкой типа double);

%e — вещественный формат в экспоненциальной форме (числа с плавающей точкой типа float в экспоненциальной форме);

%c — символьный формат;

%s — строковый формат.

Строка форматов содержит форматы для вывода значений. Каждый формат вывода начинается с символа %. После строки форматов через запятую указываются имена переменных, которые необходимо вывести.

Количество символов % в строке формата должно совпадать с количеством переменных для вывода. Тип каждого формата должен совпадать с типом переменной, которая будет выводиться на это место. Замещение форматов вывода значениями переменных происходит в порядке их следования.

Пример на Си

```
#include <stdio.h>
int main()
{
  int a = 5;
  float x = 2.78;
  prin«f("a=%»\n", a);
```

```
  prin«f("x=%»\n", x);
  getchar();
  return 0;
}
```

Тот же самый код может быть представлен с использованием одного вызова printf:

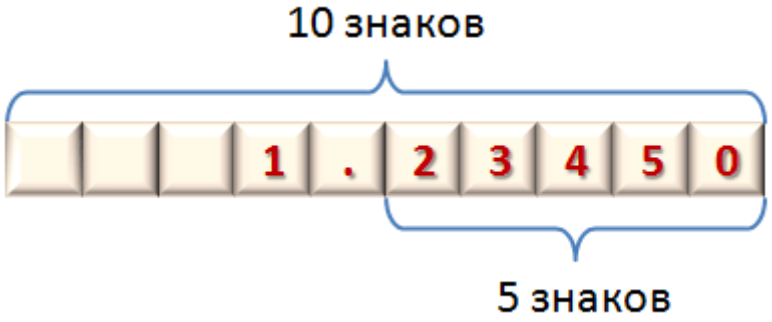
```
#include <stdio.h>
int main()
{
  int a = 5;
  float x = 2.78;
  prin«f("a=%d\nx=%»\n", a, x);
  getchar();
  return 0;
}
```

Табличный вывод

При указании формата можно явным образом указать общее количество знакомест и количество знакомест, занимаемых дробной частью:

```
#include <stdio.h>
int main()
{
  float x = 1.2345;
  prin«f("x=%10.5»\n", x);
  getchar();
  return 0;
}
```

В приведенном примере 10 — общее количество знакомест, отводимое под значение переменной; 5 — количество позиций после разделителя целой и дробной части (после десятичной точки). В указанном примере количество знакомест в выводимом числе меньше 10, поэтому свободные знакоместа слева от числа заполняются пробелами. Такой способ форматирования часто используется для построения таблиц.



Ввод информации

Функция форматированного ввода данных с клавиатуры scanf() выполняет чтение данных, вводимых с клавиатуры, преобразует их во внутренний формат и передает вызывающей функции. При этом программист задает правила интерпретации входных данных с помощью спецификаций форматной строки.

Общая форма записи функции scanf( ):

```
 scan« ("СтрокаФорма»ов", адрес1, адрес2,...);
```

Строка форматов аналогична функции printf().

Для формирования адреса переменной используется символ амперсанд ‘&’:

```
адрес = &объект
```

Строка форматов и список аргументов для функции обязательны. Пример на Си

```
#include <stdio.h>
#include <stdlib.h> // для перехода на русский язык
int main()
{
  float y;
  syst«m("chcp 1»51"); // переходим в консоли на русский язык
  syst«m("»ls");       // очищаем окно консоли
  prin«f("Введите «: "); // выводим сообщение
  sca«f(»%f", &y);     // вводим значения переменной y
  prin«f("Значение переменной y»%f", y); // выводим значение переменной y
  getchar(); getchar();
  return 0;
}
```

Другой вариант — воспользоваться функцией защищенного ввода scanf_s( ), которая появилась несколько позже, но содержит тот же самый список параметров.

```
#include <stdio.h>
int main()
{
  int a;
  prin«f("a«= ");
  scanf«s(»%d", &a);
  prin«f("a =»%d",a);
  getchar(); getchar();
  return 0;
}
```

Задание: С помощью сервиса https://repl.it/ проверьте примеры, поведённые в лекции. В первом примере замените управляющие символы в операторе вывода. Напишите как изменился вывод текста.

Контрольные вопросы:

1. Назовите оператор консльного вывода?
2. Назовите оператор консльного ввода?
3. Что такое управляющие констатнты?

Тема 8. Условные операторы

Цель: способствовать формированию знаний и умений в области составления разветвляющихся программ с использованием различных видов условий; научить составлять алгоритмы решения задач на ветвление; изучить оператор условия if, его полную и неполные ветвления, тернарную операцию**.**

Рассмотрим подробнее структуру алгоритма «развилка». Разветвляющимся называется такой алгоритм, в котором выбирается один из нескольких возможных вариантов вычислительного процесса.

Каждый подобный путь называется ветвью алгоритма.

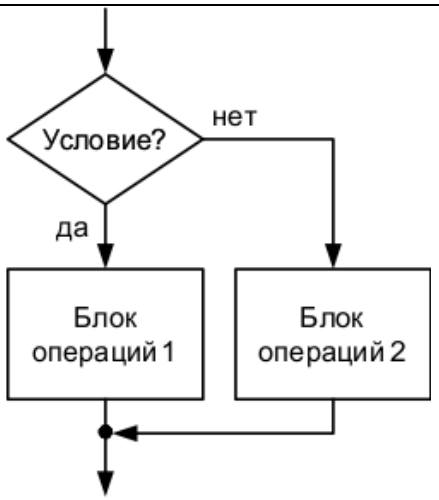
Признаком разветвляющегося алгоритма является наличие операций проверки условия. Чаще всего для проверки условия используется условный оператор if.

Условный оператор if

Условный оператор if может использоваться в форме полной или неполной развилки.

Неполная развилка	Полная развилка

if (Условие)
{
БлокОпераций1;
}
	if (Условие)
	{
	БлокОпераций1;
	}
	else
	{
	БлокОпераций2;
	}

В случае неполной развилки если Условие истинно, то БлокОпераций1 выполняется, если Условие ложно, то БлокОпераций1 не выполняется.

В случае полной развилки если Условие истинно, то выполняется БлокОпераций1, иначе выполняется БлокОпераций2.

БлокОпераций может состоять из одной операции. В этом случае наличие фигурных скобок, ограничивающих блок, необязательно.

Основными операциями, проверяемыми внутри условного блока, являются операции отношения.

Пример на С:

```
#define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
#include <stdio.h>
int main()
{
  int k;          // объявляем целую переменную k
  prin«f("«= ");   // выводим сообщение
  sca«f(»%d", &k); // вводим переменную k
  if (k >= 5)      // если k>5
    prin«f("%d >» 5", k); // вывод«м "ЗНАЧЕНИЕ >» 5"
  else             // иначе
    prin«f("%d » 5", k);  // вывод«м "ЗНАЧЕНИЕ » 5"
  getchar(); getchar();
  return 0;
}
```

Оператор if может быть вложенным.

Пример на С:

```
#define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
#include <stdio.h>
#include <stdlib.h> // для использования функции system
int main() {
  int key;  // объявляем целую переменную key
  syst«m("chcp 1»51"); // переходим в консоли на русский язык
  syst«m("»ls");       // очищаем окно консоли
  prin«f("Введите номер пункта, 1 или «: ");
  sca«f(»%d", &key);   // вводим значение переменной key
  if (key == 1)        // если key = 1
    prin«f("\n Выбран первый пу»кт"); // выводим сообщение
  else if (key == 2)   // иначе если key = 2
    prin«f("\n Выбран второй пу»кт"); // выводим сообщение
  else                 // иначе
    prin«f("\n Первый и второй пункты не выбр»ны"); // выводим сообщение
  getchar(); getchar();
  return 0;
}
```

При использовании вложенной формы оператора if опция else связывается с последним оператором if. Если требуется связать опцию else с предыдущим оператором if, внутренний условный оператор заключается в фигурные скобки:

```
#define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
#include <stdio.h>
#include <stdlib.h> // для использования функции system
int main() {
  int key;  // объявляем целую переменную key
  syst«m("chcp 1»51"); // переходим в консоли на русский язык
  syst«m("»ls");       // очищаем окно консоли
  prin«f("Введите номер пункта, 1 или «: ");
  sca«f(»%d", &key);   // вводим значение переменной key
  if (key != 1) {      // если key не равен 1
    if (key == 2)    // если key равен 2
      prin«f("\n Выбран второй пу»кт"); // вывод сообщения
  }              // если k–y - не 1 и не 2, то ничего не выводится
  else // иначе, если key равен 1
    prin«f("\n Выбран первый пу»кт"); // вывод сообщения
  getchar(); getchar();
  return 0;
}
```

Условный оператор может проверять

- одновременное выполнение всех условий (операция И - &&)
- выполнение хотя бы одного из условий (операция ИЛИ - ||)
- выполнение только одного из условий (операция исключающее ИЛИ - ^)

Пример на Си: Найти максимум из 3 чисел

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main()
{
  int a, b, c;
  prin«f(»a=");
```

```
 sca«f(»%d", &a);
 prin«f(»b=");
 sca«f(»%d", &b);
 prin«f(»c=");
 sca«f(»%d", &c);
 if ((a >= b) && (a >= c))
  prin«f("Max =»%d", a);
 else if ((b >= a) && (b >= c))
  prin«f("Max =»%d", b);
 else
  prin«f("Max =»%d", c);
 getchar();
 getchar();
 return 0;
}
```

Пример на С++: Найти максимум из 3 чисел

```
#include <iostream>
using namespace std;
int main()
{
 int a, b, c;
 cout «< »a=";
 cin >> a;
 cout «< »b=";
 cin >> b;
 cout «< »c=";
 cin >> c;
 if ((a >= b) && (a >= c))
  cout «< "Max«= " << a;
 else if ((b >= a) && (b >= c))
  cout «< "Max«= " << b;
 else
  cout «< "Max«= " << c;
 cin.get();
 cin.get();
 return 0;
}
```

Тернарные операции

Тернарная условная операция имеет 3 аргумента и возвращает свой второй или третий операнд в зависимости от значения логического выражения, заданного первым операндом. Синтаксис тернарной операции в языке Си

Условие ? Выражение1 : Выражение2;

Если выполняется Условие, то тернарная операция возвращает Выражение1, в противном случ–е - Выражение2.

Тернарные операции, как и операции условия, могут быть вложенными. Для разделения вложенных операций используются круглые скобки.

Приведенный выше пример с использованием тернарных операций можно представить в виде

```
#define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
```

```
#include <stdio.h>
#include <stdlib.h> // для использования функции system
int main() {
int key;  // объявляем целую переменную key
syst«m("chcp 1»51"); // переходим в консоли на русский язык
syst«m("»ls");       // очищаем окно консоли
prin«f("Введите номер пункта, 1 или «: ")
sca«f(»%d", &key);   // вводим значение переменной key
key == 1 ? prin«f("\n Выбран первый пу»кт") :
(key == 2 ? prin«f("\n Выбран второй пу»кт") :
prin«f("\n Первый и второй пункты не выбр»ны"));
getchar(); getchar();
return 0;
}
```

Контрольные вопросы:

1. Что означает фраза: «задачи с ветвлением»?
2. Как записывается полный условный оператор?
3. Как записывается неполный условный оператор?
4. Какие математические действия можно выполнять в задачах с ветвлением?

Тема 9. Оператор выбора switch

Цель: изучить оператор множественного выбора switch, научить составлять алгоритмы с применением оператора выбора

Оператор ветвления switch (оператор множественного выбора)

Оператор if позволяет осуществить выбор только между двумя вариантами. Для того, чтобы производить выбор одного из нескольких вариантов необходимо использовать вложенный оператор if. С этой же целью можно использовать оператор ветвления switch.

Общая форма записи

```
switch (ЦелоеВыражение)
{
case Константа1: БлокОпераций1;
break;
case Константа2: БлокОпераций2;
        break;
...
case Константаn: БлокОперацийn;
break;
default: БлокОперацийПоУмолчанию;
break;
}
```

Оператор ветвления switch выполняется следующим образом:

– вычисляется ЦелоеВыражение в скобках оператора switch;

– полученное значение сравнивается с метками (Константами) в опциях case, сравнение производится до тех пор, пока не будет найдена метка, соответствующая вычисленному значению целочисленного выражения;

– выполняется БлокОпераций соответствующей метки case;

– если соответствующая метка не найдена, то выполнится БлокОпераций ПоУмолчанию, описанный в опции default.

Альтернатива default может отсутствовать, тогда не будет произведено никаких действий.

Опция break; осуществляет выход из оператора switch и переход к следующему за ним оператору. При отсутствии опции break будут выполняться все операторы, начиная с помеченного данной меткой и кончая оператором в опции default.

Константы в опциях case должны быть целого типа (могут быть символами).

Следует знать о трех важных моментах оператора switch:

1. switch отличается от if тем, что он может выполнять только операции проверки строгого равенства, в то время как if может вычислять логические выражения и отношения.

2. Не может быть двух констант в одном операторе switch, имеющих одинаковые значения. Конечно, оператор switch, включающий в себя другой оператор switch, может содержать аналогичные константы.

3. Если в операторе switch используются символьные константы, они автоматически преобразуются к целочисленным значениям.

Оператор switch часто используется для обработки команд клавиатуры типа работа с меню. Как показано ниже, функция menu() отображает меню для программы проверки орфографии и вызывает соответствующие процедуры:

```
void menu(void)
{
char ch;

prin«f("1. Check Spellin»\n");
prin«f("2. Correct Spelling Error»\n");
prin«f("3. Display Spelling Error»\n");
prin«f("Strike Any Other Key to Ski»\n");
print« ("    Enter your choic«: ");

ch = getche(); /* чтение клавиатуры */

switch(ch) {
case '1':
    check_spelling();
  break;
case '2':
   correct_errors();
  break;
case '3';
   display_errors();
  break;
default  :
prin«f("No option selec»ed");
 }
}
```

С технической точки зрения операторы break являются необязательными в операторе switch. Они используются для окончания работы последовательности операторов, ассоциированных с данной константой. Если оператор break отсутствует, продолжают выполняться операторы следующего раздела, пока не будет достигнут оператор break или конец оператора switch. О константах выбора можно думать как о

метках. Выполнение начинается с метки, соответствующей искомому значению, и продолжается, пока не будет достигнут break или конец оператора switch. Например, функция, показанная ниже, использует данную особенность оператора case для упрощения кода обработчика ввода драйвера устройства:

```
        void inp_handler(void)
{
int ch, flag;
ch = read_device(); /* чтение какого-то устройства */
flag = -1;
switch(ch) {
case 1: /* данные случаи имеют общую последовательность */
case 2: /* операторов */
case 3:
   flag = 0;
   break;
case 4:
    flag = 1;
case 5:
   error(flag);
break;
default:
process(ch);
  }
}
```

Данная подпрограмма иллюстрирует две грани оператора switch. Во-первых, можно иметь пустые условия. В данном случае первые три условия приводят к выполнению одних и тех же операторов:

```
flag = 0;
break;
```

Во-вторых, выполнение переходит к следующему case, если отсутствует break. Если ch соответствует 4, то flag устанавливается в 1, и, поскольку отсутствует оператор break, выполнение продолжается и выполняется оператор error(flag). В данном случае flag имеет значение 1. Если ch равно 5, то вызывается error(flag), а значение flag будет рав–о - 1. Способность запускать несколько операторов, соответствующих нескольким условиям при отсутствии оператора break, позволяет создавать эффективный код, поскольку это избавляет от необходимости дублировать код.

Важно понять, что операторы, ассоциированные с каждой меткой, являются не блоками кода, а скорее последовательностью операторов. (Сам оператор switch определяет блок.) Понимание этого необходимо в некоторых специфических ситуациях. Например, следующий фрагмент кода содержит ошибку и не будет компилироваться, поскольку невозможно объявить переменную в последовательности операторов:

```
/* неверно */
switch(c) {
case 1:
    int t;
...
```

Тем не менее переменная может быть добавлена:

```
/* верно */
switch(c) {
int t;
```

case 1:
...

Тема 10. Операторы цикла параметрического типа, пост и предусловия

Цель: изучить операторы циклической конструкции и освоить операторы цикла for, с предусловием (while), постусловия (do .. while).

Рассмотрим третью алгоритмическую структуру — цикл. Циклом называется блок кода, который для решения задачи требуется повторить несколько раз.

Каждый цикл состоит из

- блока проверки условия повторения цикла
- тела цикла

Цикл выполняется до тех пор, пока блок проверки условия возвращает истинное значение.

Тело цикла содержит последовательность операций, которая выполняется в случае истинного условия повторения цикла. После выполнения последней операции тела цикла снова выполняется операция проверки условия повторения цикла. Если это условие не выполняется, то будет выполнена операция, стоящая непосредственно после цикла в коде программы.

В языке Си следующие виды циклов:

- while — цикл с предусловием;
- do…while — цикл с постусловием;
- for — параметрический цикл (цикл с заданным числом повторений).

Цикл с предусловием while

Общая форма записи

```
while (Условие)
{
БлокОпераций
}
```

Если Условие выполняется (выражение, проверяющее Условие, не равно нулю), то выполняется БлокОпераций, заключенный в фигурные скобки, затем Условие проверяется снова.

Последовательность действий, состоящая из проверки Условия и выполнения БлокаОпераций, повторяется до тех пор, пока выражение, проверяющее Условие, не станет ложным (равным нулю). При этом происходит выход из цикла, и производится выполнение операции, стоящей после оператора цикла.

Пример на Си: Посчитать сумму чисел от 1 до введенного k

```
#define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
#include <stdio.h>
int main() {
  int k;  // объявляем целую переменную key
  int i = 1;
  int sum = 0; // начальное значение суммы равно 0
  prin«f("k«= ");
  sca«f(»%d", &k);   // вводим значение переменной k
  while (i <= k)     // пока i меньше или равно k
  {
    sum = sum + i; // добавляем значение i к сумме
    i++;           // увеличиваем i на 1
  }
```

```
 prin«f("sum = %»\n", sum); // вывод значения суммы
 getchar(); getchar();
 return 0;
}
```

При построении цикла while, в него необходимо включить конструкции, изменяющие величину проверяемого выражения так, чтобы в конце концов оно стало ложным (равным нулю). Иначе выполнение цикла будет осуществляться бесконечно (бесконечный цикл).

Пример бесконечного цикла

```
while (1)
БлокОпераций;
}
```

while — цикл с предусловием, поэтому вполне возможно, что тело цикла не будет выполнено ни разу если в момент первой проверки проверяемое условие окажется ложным.

Например, если в приведенном выше коде программы ввести k=-1, то получим результат sum=0.

Цикл с постусловием do...while

Общая форма записи

```
 do
{
БлокОпераций;
} while (Условие);
```

Цикл do...while — это цикл с постусловием, где истинность выражения, проверяющего Условие проверяется после выполнения Блока Операций, заключенного в фигурные скобки. Тело цикла выполняется до тех пор, пока выражение, проверяющее Условие, не станет ложным, то есть тело цикла с постусловием выполнится хотя бы один раз.

Использовать цикл do...while лучше в тех случаях, когда должна быть выполнена хотя бы одна итерация, либо когда инициализация объектов, участвующих в проверке условия, происходит внутри тела цикла.

Пример на Си. Проверка, что пользователь ввел число от 0 до 10

```
#define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
#include <stdio.h>
#include <stdlib.h> // для использования функции system()
int main() {
 int num;            // объявляем целую переменную для числа
 syst«m("chcp 1»51"); // переходим на русский язык в консоли
 syst«m("»ls");       // очищаем экран
 do {
  prin«f("Введите число от 0 до 1«: "); // приглашение пользователю
  sca«f(»%d", &num); // ввод числа
 } while ((num < 0) || (num > 10)); // повторяем цикл пока num<0 или num>10
 prin«f("Вы ввели число»%d", num); // выводим введенное значение n–m - от 0 до 10
 getchar(); getchar();
 return 0;
}
```

Параметрический цикл for

Общая форма записи

```
for (Инициализация; Условие; Модификация)
{
БлокОпераций;
```

}

for — параметрический цикл (цикл с фиксированным числом повторений). Для организации такого цикла необходимо осуществить три операции:

- Инициализац–я - присваивание параметру цикла начального значения;
- Услов–е - проверка условия повторения цикла, чаще все–о - сравнение величины параметра с некоторым граничным значением;
- Модификац–я - изменение значения параметра для следующего прохождения тела цикла.

Эти три операции записываются в скобках и разделяются точкой с запятой ;. Как правило, параметром цикла является целочисленная переменная.

Инициализация параметра осуществляется только один раз — когда цикл for начинает выполняться.

Проверка Условия повторения цикла осуществляется перед каждым возможным выполнением тела цикла. Когда выражение, проверяющее Условие становится ложным (равным нулю), цикл завершается. Модификация параметра осуществляется в конце каждого выполнения тела цикла. Параметр может как увеличиваться, так и уменьшаться.

Пример на Си: Посчитать сумму чисел от 1 до введенного k

```
#define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
#include <stdio.h>
int main() {
  int k;  // объявляем целую переменную key
  int sum = 0; // начальное значение суммы равно 0
  prin«f("k«= ");
  sca«f(»%d", &k);   // вводим значение переменной k
  for(int i=1; i<=k; i++) // цикл для переменной i от 1 до k с шагом 1
  {
    sum = sum + i; // добавляем значение i к сумме
  }
  prin«f("sum = %»\n", sum); // вывод значения суммы
  getchar(); getchar();
  return 0;
}
```

В записи цикла for можно опустить одно или несколько выражений, но нельзя опускать точку с запятой, разделяющие три составляющие цикла. Код предыдущего примера можно представить в виде

```
#define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
#include <stdio.h>
int main() {
  int k;  // объявляем целую переменную key
  int sum = 0; // начальное значение суммы равно 0
  prin«f("k«= ");
  sca«f(»%d", &k);   // вводим значение переменной k
  int i=1;
  for(; i<=k; i++) // цикл для переменной i от 1 до k с шагом 1
  {
    sum = sum + i; // добавляем значение i к сумме
  }
  prin«f("sum = %»\n", sum); // вывод значения суммы
  getchar(); getchar();
  return 0;
}
```

Параметры, находящиеся в выражениях в заголовке цикла можно изменить при выполнении операции в теле цикла, например

```
#define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
#include <stdio.h>
int main() {
  int k;  // объявляем целую переменную key
  int sum = 0; // начальное значение суммы равно 0
  prin«f("k«= ");
  sca«f(»%d", &k);   // вводим значение переменной k
  for(int i=1; i<=k; ) // цикл для переменной i от 1 до k с шагом 1
  {
    sum = sum + i; // добавляем значение i к сумме
    i++;           // добавляем 1 к значению i

  }
  prin«f("sum = %»\n", sum); // вывод значения суммы
  getchar(); getchar();
  return 0;
}
```

В цикле for может использоваться операция запятая - , - для разделения нескольких выражений. Это позволяет включить в спецификацию цикла несколько инициализирующих или корректирующих выражений. Выражения, к которым применяется операция запятая, будут вычисляться слева направо.

Пример на Си:

```
#define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
#include <stdio.h>
int main() {
  int k;  // объявляем целую переменную key
  prin«f("k«= ");
  sca«f(»%d", &k);   // вводим значение переменной k
  for(int i=1, j=2; i<=k; i++, j+=2) // цикл для переменных
  {                                  // (i от 1 до k с шагом 1) и (j от 2 с шагом 2)
    prin«f("i = %d   j = %»\n", i, j); // выводим значения i и j
  }
  getchar(); getchar();
  return 0;
}
```

Вложенные циклы

В Си допускаются вложенные циклы, то есть когда один цикл находится внутри другого:

```
for (i = 0; i<n; i++)  // внешний ци–л - Цикл1
{
  for (j = 0; j<n; j++)   // вложенный ци–л - Цикл2
  {
    ;       // блок операций Цикла2
  }
  // блок операций Цикла1;
}
```

Пример: Вывести числа от 0 до 99, по 10 в каждой строке

```
#define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
#include <stdio.h>
```

```
int main() {
  for(int i=0; i<10; i++) // цикл для десятков
  {
    for (int j = 0; j < 10; j++) // цикл для единиц
    {
      prin«f("%«d ", i * 10 + j); // выводим вычисленное число (2 знакоместа) и пробел
    }
    prin«f(»\n"); // во внешнем цикле переводим строку
  }
  getchar(); // scanf() не использовался,
  return 0;  // поэтому консоль можно удержать одним вызовом getchar()
}
```

Рекомендации по выбору цикла

При выборе цикла необходимо оценить необходимость проверки условия при входе в цикл или по завершении прохождения цикла.

Цикл с постусловием удобно применять в случаях, когда для проверки условия требуется вычислить значение выражения, которое затем будет размещено в теле цикла (см. выше пример ввода числа от 0 до 10).

Цикл с предусловием используется в случае, если все переменные, участвующие в выражении, проверяющем условие, проинициализированы заранее, но точное число повторений цикла неизвестно или предполагается сложная модификация переменных, участвующих в формировании условия повторения цикла.

Если цикл ориентирован на работу с параметром, для которого заранее известно число повторений и шаг изменения, то более предпочтительным является параметрический цикл. Очень удобно использовать параметрический цикл при работе с массивами для перебора элементов.

Контрольные вопросы:

1. Какие операторы циклов вы знаете?
2. Какие циклы относятся к циклам с предусловием?
3. Какие циклы относятся к циклам с постусловием?

Тема 11. Оператор безусловного перехода goto. Операторы continue, break

Цель: изучить оператор безусловного перехода goto, а также операторы continue и break

В некоторых случаях желательно прервать повторение цикла, проанализировав какие-то условия внутри него. Это может потребоваться в тех случаях, когда проверки условия окончания цикла громоздкие, требуют многоэтапного сравнения и сопоставления каких-то данных, и все эти проверки просто невозможно разместить в выражении условия операторов for, do или while.

Оператор break предназначен для искусственного прерывания выполнения:

- последовательности итераций в операторах цикла for, while или do-while;
- последовательности операторов в операторе выбора switch.

Чаще всего оператор break используется в сочетании с оператором условия if. В этом случае, происходит проверка некоторого условия, и в случае успеха вызывается оператор break.

Пример. В нижеследующем примере при достижении переменной *i* значения 3 происходит безусловный выход из цикла путем вызова оператора break.

Продемонстрирован вызов break для всех видов операторов.

```
#include <iostream>
```

```
using namespace std;

void main()
{
  int i, j;

  // Оператор break

  // 1. Цикл for
  cout «< "Loop f»r:" << endl;
  for (i = 0; i < 5; i++) // цикл по i
  {
    if (i == 3)
      break; // выход из цикла for

    // этот оператор выполняется
    cout «< "i«= " << i << endl;
  }

  // 2. Цикл while
  cout «< "Loop whi»e:" << endl;
  i = 0;
  while (i < 5)
  {
    if (i == 3)
      break; // выход из цикла while
    cout «< "i«= " << i << endl;
    i++;
  }

  // 3. Цикл do-while
  cout «< "Loop do-whi»e:" << endl;
  i = 0;
  do
  {
    if (i == 3)
      break; // выход из цикла do-while
    cout «< "i«= " << i << endl;
    i++;
  } while (i < 5);
}
```

Особенности применения оператора continue

Оператор continue предназначен для перехода к выполнению следующей итерации цикла. Если в теле цикла встречается оператор continue, то:

- выполнение текущей итерации останавливается;
- происходит переход к следующей итерации цикла.

Оператор continue может быть применен во всех видах циклов: for, while, do-while.

В большинстве случаев вызов оператора continue осуществляется при выполнении некоторого условия в операторе if.

Пример. В примере демонстрируется использование оператора continue для всех видов циклов C++. В задаче, при достижении переменной ***i*** значения 3 происходит переход к следующей итерации с помощью инструкции continue.

```
#include <iostream>
using namespace std;

void main()
{
 int i, j;
 // Оператор continue
 // 1. Цикл for
 cout «< "Loop f»r:" << endl;
 for (i = 0; i < 5; i++) // цикл по i
 {
  if (i == 3)
    continue; // переход к следующей итерации

  // этот оператор выполняется
  cout «< "i«= " << i << endl;
 }

 // 2. Цикл while
 cout «< "Loop whi»e:" << endl;
 i = 0;
 while (i < 5)
 {
  if (i == 3)
  {
    i++; // увеличить счетч–к - обязательно
    continue; // переход к следующей итерации
  }
  cout «< "i«= " << i << endl;
  i++;
 }

 // 3. Цикл do-while
 cout «< "Loop do-whi»e:" << endl;
 i = 0;
 do
 {
  if (i == 3)
  {
    i++; // увеличить счетч–к - обязательно
    continue; // переход к следующей итерации
  }
  cout «< "i«= " << i << endl;
  i++;
 } while (i < 5);
}
 ()
}
```

При вложенных циклах действия операторов break и continue распространяется только на самую внутреннюю структуру, в которой они содержатся.

Оператор безусловного перехода goto

Общая форма записи

goto Метка;

Метка : Операция;

Выполнение оператора goto вызывает передачу управления в программе операции, помеченной Меткой. По сути Метка является идентификатором адреса операции, которой должно быть передано управление. Для отделения Метки от Операции используется двоеточие - :.

Метка может располагаться в программе как до оператора goto, так и после него. Имена Меток образуются по тем же правилам, что и имена переменных.

Пример на Си: Вывести все целые числа от 5 до 0.

```
#include <stdio.h>
int main() {
  int k = 5;
M1: if (k < 0) // если k<0,
    goto M2;   // переходим на метку M2 (выходим из программы)
  prin«f("«d ", k); // выводим значение k
  k--;              // уменьшаем k на 1
  goto M1;          // переходим на метку M1 (повторяем операции выше)
M2: getchar();
  return 0;
}
```

Использование оператора goto в программах на Си без крайней необходимости не рекомендуется, поскольку это может повлечь за собой ряд ошибок, связанных с плохой читаемостью кода программы. Использование операторов цикла позволяет практически полностью исключить необходимость использования оператора goto.

Контрольные вопросы:

1. Для чего используется оператор goto?
2. Для чего используется оператор continue?
3. Для чего используется оператор break?

Тема 12. Функции. Параметры функции. Локальные и глобальные переменные.

Цель: изучить структуру, назначение и применение подпрограмм. Параметры и аргументы, области действия имен.

Функция — это самостоятельная единица программы, которая спроектирована для реализации конкретной подзадачи.

Функция является подпрограммой, которая может содержаться в основной программе, а может быть создана отдельно (в библиотеке). Каждая функция выполняет в программе определенные действия.

Сигнатура функции определяет правила использования функции. Обычно сигнатура представляет собой описание функции, включающее имя функции, перечень формальных параметров с их типами и тип возвращаемого значения.

Семантика функции определяет способ реализации функции. Обычно представляет собой тело функции.

Определение функции

Каждая функция в языке Си должна быть определена, то есть должны быть указаны:

- тип возвращаемого значения;
- имя функции;
- информация о формальных аргументах;
- тело функции.

Определение функции имеет следующий синтаксис:

```
ТипВозвращаемогоЗначения ИмяФункции (СписокФормальныхАргументов)
{
ТелоФункции;
 ..
return(ВозвращаемоеЗначение);
}
```

Пример: Функция сложения двух вещественных чисел

```
float function(float x, float z)
{
float y;
  y=x+z;
return(y);
}
```

В указанном примере возвращаемое значение имеет тип float. В качестве возвращаемого значения в вызывающую функцию передается значение переменной y. Формальными аргументами являются значения переменных x и z.

Если функция не возвращает значения, то тип возвращаемого значения для нее указывается как void. При этом операция return может быть опущена. Если функция не принимает аргументов, в круглых скобках также указывается void.

Различают системные (в составе систем программирования) и собственные функции.

Системные функции хранятся в стандартных библиотеках, и пользователю не нужно вдаваться в подробности их реализации. Достаточно знать лишь их сигнатуру. Примером системных функций, используемых ранее, являются функции printf() и scanf().

Собственные функции — это функции, написанные пользователем для решения конкретной подзадачи.

Разбиение программ на функции дает следующие преимущества:

– Функцию можно вызвать из различных мест программы, что позволяет избежать повторения программного кода.

– Одну и ту же функцию можно использовать в разных программах.

– Функции повышают уровень модульности программы и облегчают ее проектирование.

– Использование функций облегчает чтение и понимание программы и ускоряет поиск и исправление ошибок.

– С точки зрения вызывающей программы функцию можно представить как некий «черный ящик», у которого есть несколько входов и один выход. С точки зрения вызывающей программы неважно, каким образом производится обработка информации внутри функции. Для корректного использования функции достаточно знать лишь ее сигнатуру.

Вызов функции

Общий вид вызова функции

```
Переменная = ИмяФункции(СписокФактическихАргументов);
```

Фактический аргумент — это величина, которая присваивается формальному аргументу при вызове функции.

Таким образом, формальный аргумент — это переменная в вызываемой функции, а фактический аргумент — это конкретное значение, присвоенное этой переменной вызывающей функцией. Фактический аргумент может быть константой, переменной или выражением. Если фактический аргумент представлен в виде выражения, то его значение сначала вычисляется, а затем передается в вызываемую функцию. Если в функцию требуется передать несколько значений, то они записываются через запятую. При этом формальные параметры заменяются значениями фактических параметров в порядке их следования в сигнатуре функции.

Возврат в вызывающую функцию

По окончании выполнения вызываемой функции осуществляется возврат значения в точку ее вызова. Это значение присваивается переменной, тип которой должен соответствовать типу возвращаемого значения функции. Функция может передать в вызывающую программу только одно значение. Для передачи возвращаемого значения в вызывающую функцию используется оператор return в одной из форм:

return(ВозвращаемоеЗначение);

return ВозвращаемоеЗначение;

Действие оператора следующее: значение выражения, заключенного в скобки, вычисляется и передается в вызывающую функцию. Возвращаемое значение может использоваться в вызывающей программе как часть некоторого выражения.

Оператор return также завершает выполнение функции и передает управление следующему оператору в вызывающей функции. Оператор return не обязательно должен находиться в конце тела функции.

Функции могут и не возвращать значения, а просто выполнять некоторые вычисления. В этом случае указывается пустой тип возвращаемого значения void, а оператор return может либо отсутствовать, либо не возвращать никакого значения:

return;

Пример: Посчитать сумму двух чисел.

```
#include <stdio.h>
// Функция вычисления суммы двух чисел
int sum(int x, int y) // в функцию передаются два целых числа
{
  int k = x + y;  // вычисляем сумму чисел и сохраняем в k
  return k;       // возвращаем значение k
}
int main()
{
  int a, r;      // описание двух целых переменных
  prin«f("«= ");
  sca«f(»%d", &a); // вводим a
  r = sum(a, 5);    // вызов функции: x=a, y=5
  prin«f("%d + 5 =»%d", a, r); // вывод: a + 5 = r
  getchar(); getchar(); // мы использовали scanf(),
  return 0;  // поэтому getchar() вызываем дважжы
}
```

В языке Си нельзя определять одну функцию внутри другой.

В языке Си нет требования, чтобы семантика функции обязательно предшествовало её вызову. Функции могут определяться как до вызывающей функции, так и после нее. Однако если семантика вызываемой функции описывается ниже ее вызова, необходимо до вызова функции определить прототип этой функции, содержащий:

- тип возвращаемого значения;

– имя функции;
– типы формальных аргументов в порядке их следования.

Прототип необходим для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических аргументов типам формальных аргументов. Имена формальных аргументов в прототипе функции могут отсутствовать.

Если в примере выше тело функции сложения чисел разместить после тела функции main, то код будет выглядеть следующим образом:

```
 #include <stdio.h>
int sum(int, int);   // сигнатура
int main()
{
  int a, r;
  prin«f("«= ");
  sca«f(»%d", &a);
  r = sum(a, 5);    // вызов функции: x=a, y=5
  prin«f("%d + 5 =»%d", a, r);
  getchar(); getchar();
  return 0;
}
int sum(int x, int y) // семантика
{
  int k;
  k = x + y;
  return(k);
}
```

Область видимости объекта (переменной или функции) определяет набор функций или модулей, внутри которых допустимо использование имени этого объекта. Область видимости объекта начинается в точке объявления объекта.

Локальные и глобальные переменные

Время жизни объекта может быть глобальным и локальным. Глобальными называют объекты, объявление которых дано вне функции. Они доступны (видимы) во всем файле, в котором они объявлены. В течение всего времени выполнения программы с глобальным объектом ассоциирована некоторая ячейка памяти.

Локальными называют объекты, объявление которых дано внутри блока или функции. Эти объекты доступны только внутри того блока, в котором они объявлены. Объектам с локальным временем жизни выделяется новая ячейка памяти каждый раз при осуществлении описания внутри блока. Когда выполнение блока завершается, память, выделенная под локальный объект, освобождается, и объект теряет своё значение.

Пример на Си

```
#include <stdio.h>
void autofunc(void)
{
  int k = 1;    // локальный объект
  prin«f(" \n k = «d ", k);
  k = k + 1;
}
int main()
{
  for (int i = 0; i <= 5; i++) // область видимости–i - цикл
    autofunc();
  getchar();
```

```
  return 0;
}
```

Область видимости локальной переменной–k - функция autofunc(). Каждый раз при входе в функцию с идентификатором k ассоциируется некоторая ячейка памяти, в которую помещается значение равное 1.

Та же программа, но с использованием глобального объекта

```
#include <stdio.h>
int k = 1; // глобальный объект
void autofunc(void)
{
  prin«f(" \n k = «d ", k);
  k = k + 1;
}
int main()
{
  for (int i = 0; i <= 5; i++)  // область видимости–i - цикл
    autofunc();
  getchar();
  return 0;
}
```

С помощью глобальных переменных можно организовать обмен информацией между функциями. При этом вызываемая функция не будет принимать значения глобальных переменных в качестве формальных аргументов. Однако в этом случае существует опасность случайного изменения глобальных объектов другими функциями.

Пример на Си

```
#include <stdio.h>
int x, y, z;   // глобальные переменные
void sum(void)
{
  z = x + y;
}
int main()
{
  prin«f("«= ");
  sca«f(»%d", &x);
  prin«f("«= ");
  sca«f(»%d", &y);
  sum();
  prin«f("z=»%d", z);
  getchar(); getchar();
  return 0;
}
```

Контрольные вопросы:

1. Как объявить и описать, вызвать подпрограммы?
2. Какие способы подстановки аргументов вы знаете?
3. Области действия имен.
4. Что такое нетипизированные параметры, открытые параметры?
5. Что такое процедурный тип данных?

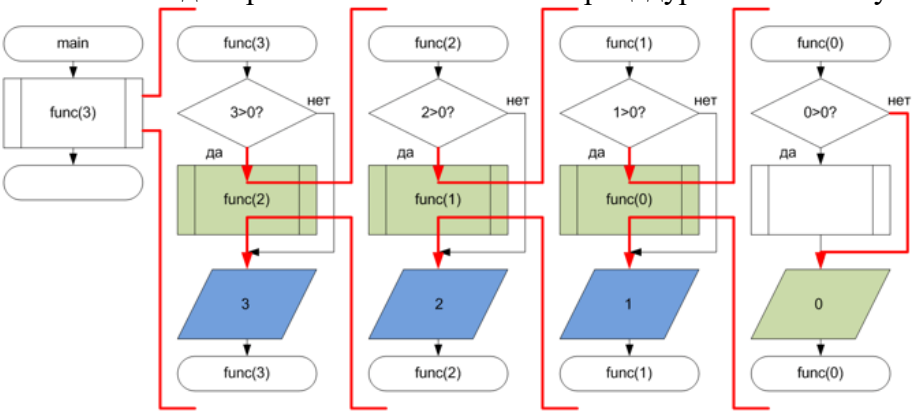
Тема 13. Рекурсия

Цель: сформировать представление о рекурсивном объекте и рекурсивном алгоритме, освоения приёмов применения рекурсивных функции при составлении программ.

Рекурсия — состоит в определении, описании, изображении какого-либо объекта или процесса внутри самого этого объекта или процесса. Это ситуация, когда объект является частью самого себя.

Процедура или функция может содержать вызов других процедур или функций. В том числе процедура может вызвать саму себя. Компьютер лишь последовательно выполняет команды и, если встречается вызов процедуры, просто начинает выполнять эту процедуру. Без разницы, какая процедура дала команду это делать.

Количество одновременно выполняемых процедур называют глубиной рекурсии.



Важным и обязательным моментом в формировании рекурсивной процедуры является базис рекурсии.Базис рекурсии определяет условие выхода из рекурсии. Как правило, в качестве базиса записывается некий простейший случай, при котором ответ получается сразу, без использования рекурсии.

Существует такое понятие как шаг рекурсии или рекурсивный вызов. В случае, когда рекурсивная функция вызывается для решения сложной задачи (не базового случая) выполняется некоторое количество рекурсивных вызовов или шагов, с целью сведения задачи к более простой. И так до тех пор пока не получим базовое решение.

Сложная рекурсия

Возможна чуть более сложная схема: функция А вызывает функцию В, а та в свою очередь вызывает А. Это называется сложной рекурсией. При этом оказывается, что описываемая первой процедура должна вызывать еще не описанную. Чтобы это было возможно, требуется использовать описание функции В до ее использования.

Префиксная и постфиксная форма записи

Если процедура вызывает сама себя, то, по сути, это приводит к повторному выполнению содержащихся в ней инструкций, что аналогично работе цикла. При этом различают префиксную и постфиксную формы записи.

Префиксная форма	Постфиксная форма
Снача–а - рекурсивный вызов, пот–м - действия	Снача–а - действия, пот–м - рекурсивный вызов

Рекуррентные соотношения

Во многих случаях в основе рекурсии лежат рекуррентные соотношения. Рекуррентное соотношение — это соотношение вида

$$a_n = f\left(n, a_{n-1}, a_{n-2}, \dots, a_{n-p}\right)$$

выражающее каждый член последовательности a_n через p предыдущих членов.

Вычисление требуемого элемента последовательности будет состоят в повторяющемся обновлении значений этой последовательности.

Каждое такое обновление называется итерацией, а процесс повторения итераций – итерированием.

Рекурсия или итерирование

Итерация — организация обработки данных, при которой действия повторяются многократно, не приводя при этом к вызовам самих себя (в отличие от рекурсии).

Рассмотрим вычисление факториала в виде итерационной и рекурсивной процедуры.

Вызов функции влечет за собой некоторые дополнительные накладные расходы, связанные передачей управления и аргументов в функцию, а также возвратом вычисленного значения. Поэтому итерационная процедура вычисления факториала будет несколько более быстрым решением. Чаще всего итерационные решения работают быстрее рекурсивных.

Любые рекурсивные процедуры и функции, содержащие всего один рекурсивный вызов самих себя, легко заменяются итерационными циклами.

Контрольные вопросы:

1. Что такое рекурсия?
2. Как правильно организовать рекурсивный вызов функции?
3. Что такое сложная рекурсия?

Тема 14. Указатели. Ссылки.

Цель: изучить указатели и ссылки, операторы работы с указателями, арифметический указатель

Указатели – очень простая концепция, очень логичная, но требующая внимания к деталям.

Определение

Указатель – это переменная, которая хранит адрес области памяти. Указатель, как и переменная, имеет тип. Синтаксис объявления указателей

<тип> *<имя>;

Например

```
float *a;
long long *b;
```

Два основных оператора для работы с указателями – это оператор & взятия адреса, и оператор * разыменования. Рассмотрим простой пример.

```
#include <conio.h>
#include <stdio.h>

void main() {
   int A = 100;
   int *p;

   //Получаем адрес переменной A
   p = &A;
```

```
    //Выводим адрес переменной A
    prin«f("%»\n", p);

    //Выводим содержимое переменной A
    prin«f("%»\n", *p);

    //Меняем содержимое переменной A
    *p = 200;

    prin«f("%»\n", A);
    prin«f(»%d", *p);

    getch();
}
```

Рассмотрим код внимательно, ещё раз

int A = 100;

Была объявлена переменная с именем A. Она располагается по какому-то адресу в памяти. По этому адресу хранится значение 100.

int *p;

Создали указатель типа int.

p = &A;

Теперь переменная p хранит адрес переменной A. Используя оператор * мы получаем доступ до содержимого переменной A.

Чтобы изменить содержимое, пишем

*p = 200;

После этого значение A также изменено, так как она указывает на ту же область памяти. Ничего сложного.

Теперь другой важный пример

```
#include <conio.h>
#include <stdio.h>

void main() {
    int A = 100;
    int *a = &A;
    double B = 2.3;
    double *b = &B;

    prin«f("%»\n", sizeof(A));
    prin«f("%»\n", sizeof(a));
    prin«f("%»\n", sizeof(B));
    prin«f("%»\n", sizeof(b));

    getch();
}
```

Несмотря на то, что переменные имеют разный тип и размер, указатели на них имеют один размер. Действительно, если указатели хранят адреса, то они должны быть целочисленного типа. Так и есть, указатель сам по себе хранится в переменной типа size_t (а также ptrdiff_t), это тип, который ведёт себя как целочисленный, однако его размер зависит от разрядности системы. В большинстве случаев разницы между ними нет. Зачем тогда указателю нужен тип?

Арифметика указателей

Во-первых, указателю нужен тип для того, чтобы корректно работала операция разыменования (получения содержимого по адресу). Если указатель хранит адрес переменной, необходимо знать, сколько байт нужно взять, начиная от этого адреса, чтобы получить всю переменную.

Во-вторых, указатели поддерживают арифметические операции. Для их выполнения необходимо знать размер.

операция + N сдвигает указатель вперёд на N*sizeof(тип) байт.

Например, если указатель int *p; хранит адрес CC02, то после p += 10; он будет хранить адрес CC02 + sizeof(int)*10 = CC02 + 28 = CC2A (Все операции выполняются в шестнадцатиричном формате). Пусть мы создали указатель на начало массива. После этого мы мож«м "двигат»ся" по этому массиву, получая доступ до отдельных элементов.

```
#include <conio.h>
#include <stdio.h>

void main() {
   int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   int *p;

   p = A;

   prin«f("%»\n", *p);
   p++;
   prin«f("%»\n", *p);
   p = p + 4;
   prin«f("%»\n", *p);

   getch();
}
```

Заметьте, каким образом мы получили адрес первого элемента массива

```
p = A;
```

Массив, по сути, сам является указателем, поэтому не нужно использовать оператор &. Мы можем переписать пример по-другому

```
p = &A[0];
```

Получить адрес первого элемента и относительно него двигаться по массиву. Кроме операторов +–и - указатели поддерживают операции сравнения. Если у нас есть два указателя a и b, то a > b, если адрес, который хранит a, больше адреса, который хранит b.

```
#include <conio.h>
#include <stdio.h>

void main() {
   int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   int *a, *b;

   a = &A[0];
   b = &A[9];

   prin«f("&A[0] == %»\n", a);
   prin«f("&A[9] == %»\n", b);

   if (a < b) {
      prin«f("a » b");
```

```
    } else {
        prin«f("b » a");
    }

    getch();
}
```

Если же указатели равны, то они указывают на одну и ту же область памяти.

Указатель на указатель

Указатель хранит адрес области памяти. Можно создать указатель на указатель, тогда он будет хранить адрес указателя и сможет обращаться к его содержимому. Указатель на указатель определяется как

<тип> **<имя>;

Очевидно, ничто не мешает создать и указатель на указатель на указатель, и указатель на указатель на указатель на указатель и так далее. Это нам понадобится при работе с двумерными и многомерными массивами. А вот простой пример, как можно работать с указателем на указатель.

```
#include <conio.h>
#include <stdio.h>

#define SIZE 10

void main() {
    int A;
    int B;
    int *p;
    int **pp;

    A = 10;
    B = 111;
    p = &A;
    pp = &p;

    prin«f("A = %»\n", A);
    *p = 20;
    prin«f("A = %»\n", A);
    *(*pp) = 30;    //здесь скобки можно не писать
    prin«f("A = %»\n", A);

    *pp = &B;
    prin«f("B = %»\n", *p);
    **pp = 333;
    prin«f("B =»%d", B);

    getch();
}
```

Указатели и приведение типов

Так как указатель хранит адрес, можно кастовать его до другого типа. Это может понадобиться, например, если мы хотим взять часть переменной, или если мы знаем, что переменная хранит нужный нам тип.

```
#include <conio.h>
#include <stdio.h>
```

```
#define SIZE 10

void main() {
   int A = 10;
   int *intPtr;
   char *charPtr;

   intPtr = &A;
   prin«f("%»\n", *intPtr);
   prin«f("------------------»\n");
   charPtr = (char*)intPtr;
   prin«f("«d ", *charPtr);
   charPtr++;
   prin«f("«d ", *charPtr);
   charPtr++;
   prin«f("«d ", *charPtr);
   charPtr++;
   prin«f("«d ", *charPtr);

   getch();
}
```

В этом примере мы пользуемся тем, что размер типа int равен 4 байта, а char 1 байт. За счёт этого, получив адрес первого байта, можно пройти по остальным байтам числа и вывести их содержимое.

NULL point–r - нулевой указатель

Указатель до инициализации хранит мусор, как и любая другая переменная. Но в то же время, эт«т "му»ор" вполне может оказаться валидным адресом. Пусть, к примеру, у нас есть указатель. Каким образом узнать, инициализирован он или нет? В общем случае никак. Для решения этой проблемы был введён макрос NULL библиотеки stdlib. Принято при определении указателя, если он не инициализируется конкретным значением, делать его равным NULl.

```
int *ptr = NULL;
```

По стандарту гарантировано, что в этом случае указатель равен NULL, и равен нулю, и может быть использован как булево значение false. Хотя в зависимости от реализации NULL может и не быть равным 0 (в смысле, не равен нулю в побитовом представлении, как например, int или float). Это значит, что в данном случае

```
int *ptr = NULL;
if (ptr == 0) {
...
}
```

вполне корректная операция, а в случае

?

```
int a = 0;
if (a == NULL) {
...
}
```

поведение не определено. То есть указатель можно сравнивать с нулём, или с NULL, но нельзя NULL сравнивать с переменной целого типа или типа с плавающей точкой.

```
#include <stdlib.h>
```

```
#include <stdio.h>
#include <conio.h>

void main() {
   int *a = NULL;
   unsigned length, i;

   prin«f("Enter length of arra«: ");
   sca«f(»%d", &length);

   if (length > 0) {
      //При выделении памяти возвращается указатель.
      //Если память не была выделена, то возвращается NULL
      if ((a = (int*) malloc(length * sizeof(int))) != NULL) {
         for (i = 0; i < length; i++) {
            a[i] = i * i;
         }
      } else {
         prin«f("Error: can't allocate mem»ry");
      }
   }

   //Если переменая была инициализирована, то очищаем её
   if (a != NULL) {
      free(a);
   }
   getch();
}
```

Контрольные вопросы:

1. Что такое указатель?
2. Для чего используется оператор & ?
3. Для чего используется оператор * ?
4. Что такое нулевой указатель?

Тема 15. Понятие массива. Одномерные массивы.

Цель: формирование целостного представления об одномерных массивах и их отличительных особенностях и выработка первичных навыков решения задач с применением одномерных массивов.

При решении задач с большим количеством данных одинакового типа использование переменных с различными именами, не упорядоченных по адресам памяти, затрудняет программирование. В подобных случаях в языке Си используют объекты, называемые массивами.

Массив — это непрерывный участок памяти, содержащий последовательность объектов одинакового типа, обозначаемый одним именем.

Массив характеризуется следующими основными понятиями:

Элемент массива (значение элемента массива) – значение, хранящееся в определенной ячейке памяти, расположенной в пределах массива, а также адрес этой

ячейки памяти.
Каждый элемент массива характеризуется тремя величинами:

- адресом элемента — адресом начальной ячейки памяти, в которой расположен этот элемент;
- индексом элемента (порядковым номером элемента в массиве);
- значением элемента.

Адрес массива – адрес начального элемента массива.

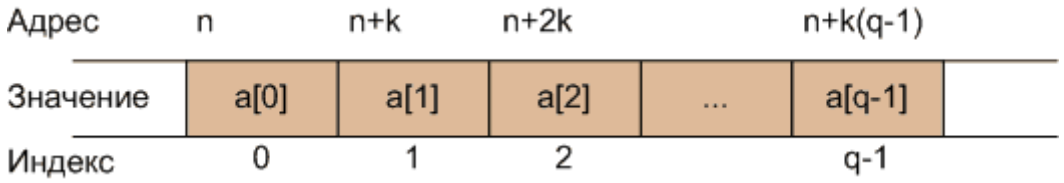
Имя массива – идентификатор, используемый для обращения к элементам массива.

Размер массива – количество элементов массива

Размер элемента – количество байт, занимаемых одним элементом массива.

Одномерный массив — массив, с одним параметром, характеризующим количество элементов одномерного массива. Фактически одномерный массив — это массив, у которого может быть только одна строка, и n-е количество столбцов. Столбцы в одномерном массиве — это элементы массива.

Графически расположение массива в памяти компьютера можно представить в виде непрерывной ленты адресов.



Представленный на рисунке массив содержит q элементов с индексами от 0 до q-1. Каждый элемент занимает в памяти компьютера k байт, причем расположение элементов в памяти последовательное.

Адреса i-го элемента массива имеет значение

$$n+k \cdot i$$

Адрес массива представляет собой адрес начального (нулевого) элемента массива. Для обращения к элементам массива используется порядковый номер (индекс) элемента, начальное значение которого равно 0. Так, если массив содержит q элементов, то индексы элементов массива меняются в пределах от 0 до q-1.

Длина массива – количество байт, отводимое в памяти для хранения всех элементов массива.

ДлинаМассива = РазмерЭлемента * КоличествоЭлементов

Для определения размера элемента массива может использоваться функция
int sizeof(тип);
Например,
sizeof(char) = 1;
sizeof(int) = 4;
sizeof(float) = 4;
sizeof(double) = 8;

Объявление и инициализация массивов

Для объявления массива в языке Си используется следующий синтаксис:

тип имя[размерность]={инициализация};

Инициализация представляет собой набор начальных значений элементов массива, указанных в фигурных скобках, и разделенных запятыми.

int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}; // массив a из 10 целых чисел

Если количество инициализирующих значений, указанных в фигурных скобках, меньше, чем количество элементов массива, указанное в квадратных скобках, то все

оставшиеся элементы в массиве (для которых не хватило инициализирующих значений) будут равны нулю. Это свойство удобно использовать для задания нулевых значений всем элементам массива.

int b[10] = {0}; // массив b из 10 элементов, инициализированных 0

Если массив проинициализирован при объявлении, то константные начальные значения его элементов указываются через запятую в фигурных скобках. В этом случае количество элементов в квадратных скобках может быть опущено.

int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

При обращении к элементам массива индекс требуемого элемента указывается в квадратных скобках [].

Пример на Си

```
#include <stdio.h>
int main()
{
 int a[] = { 5, 4, 3, 2, 1 }; // массив a содержит 5 элементов
 prin«f("%d %d %d %d %»\n", a[0], a[1], a[2], a[3], a[4]);
 getchar();
 return 0;
}
```

Однако часто требуется задавать значения элементов массива в процессе выполнения программы. При этом используется объявление массива без инициализации. В таком случае указание количества элементов в квадратных скобках обязательно.

int a[10];

Для задания начальных значений элементов массива очень часто используется параметрический цикл:

```
#include <stdio.h>
int main()
{
 int a[5]; // объявлен массив a из 5 элементов
 int i;
 // Ввод элементов массива
 for (i = 0; i<5; i++)
 {
  prin«f("a[%d]«= ", i);
  sca«f(»%d", &a[i]); // &a[–] - адрес i-го элемента массива
 }
 // Вывод элементов массива
 for (i = 0; i<5; i++)
  prin«f("«d ", a[i]); // пробел в формате печати обязателен
 getchar(); getchar();
 return 0;
}
```

Контрольные вопросы:

1. Что такое массив?
2. Что такое элемент массива?
3. Что такое индекс массива?
4. Что такое размерность массива?
5. Зачем нужны массивы?
6. Как можно обратиться к ячейке массива?7
7. Какого типа могут быль элементы массива?
8. Какого типа может быть индекс массива?

9. Как можно осуществить ввод элементов массива?

Тема 16. Двумерные массивы

Цель: : способствовать формированию представления о двумерном массиве и основных свойств квадратных матриц; рассмотреть действия, которые можно выполнить над матрицами; познакомить с типовыми алгоритмами обработки матриц.

Под двумерным массивом понимается одномерный массив, элементами которого являются одномерные массивы. Другими словами, это набор однотипных данных, имеющий общее имя, и доступ к элементам которого осуществляется по двум индексам. Иногда двумерный массив также называют матрицей.

Динамическим массивом называют массив с переменным размером, то есть количество элементов может изменяться во время выполнения программы.

Для создания двумерного динамического массива вначале нужно распределить память для массива указателей на одномерные массивы, а затем выделить память для одномерных массивов. При динамическом распределении памяти для массивов следует описать соответствующий указатель, которому будет присвоено значение адреса начала области выделенной памяти.

Объявление двумерных динамических массивов

Под объявлением двумерного динамического массива понимают объявление двойного указателя, то есть объявление указателя на указатель.

Синтаксис:

Тип ** ИмяМассива;

ИмяМассива – идентификатор массива, то есть имя двойного указателя для выделяемого блока памяти.

Тип – тип элементов объявляемого динамического массива. Элементами динамического массива не могут быть функции и элементы типа void.

Например:

int **a; float **m;

Выделение памяти под двумерный динамический массив

При формировании двумерного динамического массива сначала выделяется память для массива указателей на одномерные массивы, а затем в цикле с параметром выделяется память под одномерные массивы. На рис.1 представлена схема динамической области памяти, выделенной под двумерный массив.

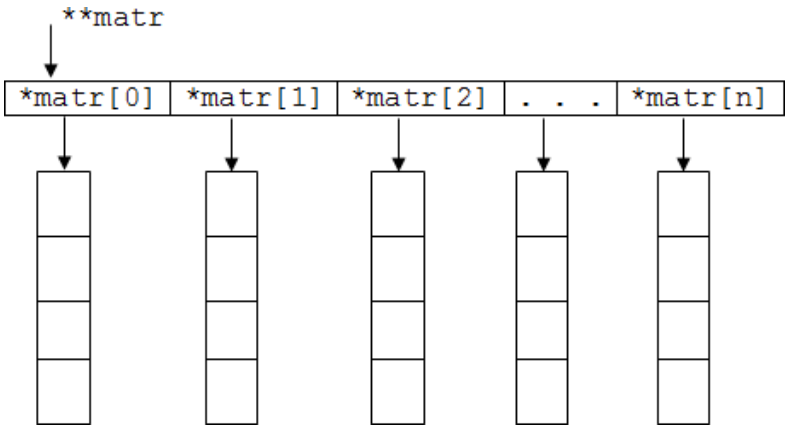


Рис. 1. Выделение памяти под двумерный массив

При работе с динамической памятью в языке С++ существует 2 способа выделения памяти под двумерный динамический массив.

1) при помощи операции new, которая позволяет выделить в динамической памяти участок для размещения массива соответствующего типа, но не позволяет его инициализировать.

Синтаксис выделения памяти под массив указателей:

ИмяМассива = new Тип * [ВыражениеТипаКонстанты];

Синтаксис выделения памяти для массива значений:

ИмяМассива[ЗначениеИндекса] = new Тип [ВыражениеТипаКонстанты];

ИмяМассива – идентификатор массива, то есть имя двойного указателя для выделяемого блока памяти.

Тип – тип указателя на массив.

ВыражениеТипаКонстанты – задает количество элементов (размерность) массива. Выражение константного типа вычисляется на этапе компиляции.

Например:

```
int n, m; // n и m – количество строк и столбцов матрицы
float **matr; // указатель для массива указателей
matr = new float * [n]; // выделение динамической памяти под массив указателей
for (int i=0; i<n; i++)
matr[i] = new float [m]; // выделение динамической памяти для массива значений
```

При выделении динамической памяти размеры массивов должны быть полностью определены.

2) при помощи библиотечной функции malloc (calloc), которая предназначена для выделения динамической памяти.

Синтаксис выделения памяти под массив указателей:

ИмяМассива = (Тип **) malloc(N*sizeof(Тип *));

или

ИмяМассива = (Тип **) calloc(N, sizeof(Тип *));

Синтаксис выделения памяти для массива значений: ИмяМассива[ЗначениеИндекса]=(Тип*)malloc(M*sizeof(Тип)); или ИмяМассива[ЗначениеИндекса]=(Тип*)calloc(M,sizeof(Тип));

ИмяМассива – идентификатор массива, то есть имя двойного указателя для выделяемого блока памяти.

Тип – тип указателя на массив.

N – количество строк массива;

M – количество столбцов массива. Например:

```
int n, m; // n и m – количество строк и столбцов матрицы
float **matr; // указатель для массива указателей
matr = (float **) malloc(n*sizeof(float *));
// выделение динамической памяти под массив указателей
for (int i=0; i<n; i++)
matr[i] = (float *) malloc(m*sizeof(float));
// выделение динамической памяти для массива значений
```

Так как функция malloc (calloc) возвращает нетипизированный указатель void *, то необходимо выполнять его преобразование в указатель объявленного типа.

Освобождение памяти, выделенной под двумерный динамический массив

Удаление из динамической памяти двумерного массива осуществляется в порядке, обратном его созданию, то есть сначала освобождается память, выделенная под одномерные массивы с данными, а затем память, выделенная под одномерные массивуказателей.

Освобождение памяти, выделенной под двумерный динамический массив, также осуществляется 2 способами.

1) при помощи операции delete, которая освобождает участок памяти ранее

выделенной операцией new.

Синтаксис освобождения памяти, выделенной для массива значений:

delete ИмяМассива [ЗначениеИндекса];

Синтаксис освобождения памяти, выделенной под массив указателей:

delete [] ИмяМассива;

ИмяМассива – идентификатор массива, то есть имя двойного указателя для выделяемого блока памяти.

Например:

for (int i=0; i<n; i++) delete matr [i];

// освобождает память, выделенную для массива значений delete [] matr;

// освобождает память, выделенную под массив указателей

Квадратные скобки [] означают, что освобождается память, занятая всеми элементами массива, а не только первым.

2) при помощи библиотечной функции free, которая предназначена для освобождения динамической памяти.

Синтаксис освобождения памяти, выделенной для массива значений:

free (ИмяМассива[ЗначениеИндекса]);

Синтаксис освобождения памяти, выделенной под массив указателей:

free (ИмяМассива);

ИмяМассива – идентификатор массива, то есть имя двойного указателя для выделяемого блока памяти.

Например:

for (int i=0; i<n; i++) free (matr[i]);

//освобождает память, выделенную для массива значений

free (matr);

//освобождает память, выделенную под массив указателей

Обращение к элементам двумерного динамического массива

Адресация элементов динамического массива осуществляется с помощью индексированного имени.

Синтаксис: ИмяМассива[ВыражениеТипаКонстанты][ВыражениеТипаКонстанты]; или

ИмяМассива[ЗначениеИндекса][ЗначениеИндекса];

Например:

mas[5][7] – индекс задается как константа,

sl[i][j] – индекс задается как переменная, array[4*p][p+5] – индекс задается как выражение.

Пример 1. Сформируйте и выведите на экран единичную матрицу с целыми элементами, вводя ее порядок с клавиатуры.

#include <iostream> using namespace std; int main()

{

int n,i,j;

int **matr;//указатель для массива указателей cout «< "Input matrix ord»r:";

cin >> n;

matr = new int *[n];

//выделение памяти под массив указателей for(i=0; i<n; i++){

matr[i] = new int[n];

//выделение памяти для массива значений for (j=0; j<n; j++) //заполнение матрицы

matr[i][j] = (i==j ? 1 : 0);

}

cout «< "Resul«: "; for(i=0; i<n; i++){

cout «< »\n";

```
for (j=0; j<n; j++)
cout «<« " " << matr[i][j]; delete matr[i];
//освобождение памяти из-под массива значений
}
delete [] matr;
//освобождение памяти из-под массива указателей syst«m("pa»se");
return 0;
}
```

Пример 2. Вычислить сумму элементов, лежащих на диагоналях матрицы N x N (обратить внимание на четность-нечетность числа N). Размер массива должен задаваться пользователем с клавиатуры.

```
#include <iostream> #include <time.h> using namespace std;
int summa(int nn, int **mass);
/*объявление функции вычисления суммы заданных элементов массива*/ void out (int nn,int **mass);
//объявление функции вывода массива

int main()
{ setlocale(LC_A»»,""); int n,s; cou»<<"Введите «: "; cin>>n;
cou»<<"\nГенерация массива»\n"; int i,j,b=10,a=0; srand(time(NULL)*1000);
int **mass;
mass = new int *[n]; for (i=0;i<n;i++) mass[i]= new int[n];
for (i=0;i<n;i++) for (j=0;j<n;j++)
mass[i][j]=rand()%(b-a)+a; s=summa(n,mass);
out(n,mass);
cou»<<"\nСумма элементов«= "<<s<<endl; syst«m("pa»se");
return 0;
}

int summa(int nn, int **mas)
{
//функция вычисления суммы элементов диагоналей int i,j, sum=0;
for (i=0;i<nn;i++)
for (j=0;j<nn;j++) {
if ((i==j) || (i==nn-j-1)) {
//нахождение элементов диагоналей sum+=mas[i][j];
//суммирование элементов диагоналей
}
}
return sum;
}

void out (int nn,int **mass)
{
//функция вывода массива int i,j;
for (i=0;i<nn;i++) { for (j=0;j<nn;j++)
cout<<mass[i][j»<« "; cou»<<»\n";
delete mass[i];
}
delete [] mass;
}
```

1. При работе с массивами, размер которых заранее не известен, используют динамические массивы.

2. Под объявлением двумерного динамического массива понимают объявление двойного указателя, то есть объявление указателя на указатель. Если двумерный массив рассматривается как одномерный, то при объявлении такого массива объявляется указатель на переменную соответствующего типа.

3. Работа с двумерными динамическими массивами начинается с выделения участка памяти, а завершается освобождением ранее выделенного участка.

4. Выделение и освобождение памяти под двумерный динамический массив

выполняется с помощью операций или функций для работы с динамической памятью.

5. Адресация элементов динамического массива осуществляется с помощью индексированного имени.

Контрольные вопросы:

1. Чем двумерный массив отличается от одномерного?
2. Что такое матрица?
3. Что понимают под динамической матрицей?
4. Каким признаком обладают элементы матрицы, принадлежащие главной диагонали?
5. Каким признаком обладают элементы матрицы, принадлежащие вспомогательной диагонали?

Тема 17. Сортировка массивов. Метод пузырька, метод вставками.

Цель: познакомить с простыми методами сортировки: методом выбора и методом обмена и их реализацией в языке программирования Си.

Поиск — обработка некоторого множества данных с целью выявления подмножеств данных, соответствующего критериям поиска.

Все алгоритмы поиска делятся на

- поиск в неупорядоченном множестве данных;
- поиск в упорядоченном множестве данных.

Упорядоченность – наличие отсортированного ключевого поля.

Сортировка — упорядочение (перестановка) элементов в подмножестве данных по какому-либо критерию. Чаще всего в качестве критерия используется некоторое числовое поле, называемое ключевым. Упорядочение элементов по ключевому полю предполагает, что ключевое поле каждого следующего элемента не больше предыдущего (сортировка по убыванию). Если ключевое поле каждого последующего элемента не меньше предыдущего, то говорят о сортировке по возрастанию.

Цель сортировки — облегчить последующий поиск элементов в отсортированном множестве при обработке данных.

Все алгоритмы сортировки делятся на

- алгоритмы внутренней сортировки (сортировка массивов);
- алгоритмы внешней сортировки (сортировка файлов).

Сортировка массивов

Массивы обычно располагаются в оперативной памяти, для которой характерен быстрый произвольный доступ. Основным критерием, предъявляемым к алгоритмам сортировки массивов, является минимизация используемой оперативной памяти.

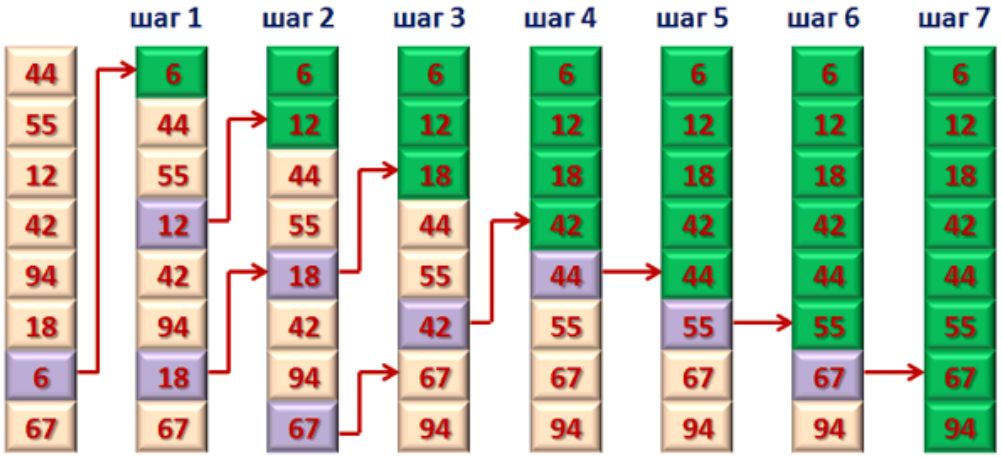
Перестановки элементов нужно выполнять на том же месте оперативной памяти, где они находятся, и методы, которые пересылают элементы из массива A в массив B, не представляют интереса.

Методы сортировки массивов можно разбить на три класса:

- сортировка включениями;
- сортировка выбором;
- сортировка обменом.

Сортировка прямым обменом (метод «пузырька»)

Если рассматривать массивы как вертикальные, а не горизонтальные построения, о элементы можно интерпретировать как пузырьки в банке с водой, причем вес каждого соответствует его ключу. В этом случае при каждом проходе один пузырек как бы поднимается до уровня, соответствующего его весу. Такой метод известен под именем «пузырьковая сортировка».



Реализация сортировки методом «пузырька» на Си

```
#include <stdio.h>
// Функция сортировки прямым обменом (мет«д "пузыр»ка")
void bubbleSort(int *num, int size)
{
  // Для всех элементов
  for (int i = 0; i < si–e - 1; i++)
  {
    for (int j = (si–e - 1); j > i; j--) // для всех элементов после i-ого
    {
      if (num–j - 1] > num[j]) // если текущий элемент меньше предыдущего
      {
        int temp = num–j - 1]; // меняем их местами
        num–j - 1] = num[j];
        num[j] = temp;
      }
    }
  }
}
int main()
{
```

```
  int a[10]; // Объявляем массив из 10 элементов
  // Вводим значения элементов массива
  for (int i = 0; i < 10; i++)
  {
    prin«f("a[%d]«= ", i);
    sca«f(»%d", &a[i]);
  }
  bubbleSort(a, 10);  // вызываем функцию сортировки
  // Выводим отсортированные элементы массива
  for (int i = 0; i<10; i++)
    prin«f("«d ", a[i]);
  getchar(); getchar();
  return 0;
}
```

Анализ алгоритма

Число сравнений в алгоритме прямого обмена

$C = (-^2 - n)/2$,

а минимальное, среднее и максимальное число перемещений элементов равно соответственно

$Mmin = 0,$

$Mcp = 3(-2 - n)/2,$

$Mmax = 3(-2 - n)/4.$

Резюме: «обменная сортировка» представляет собой нечто среднее между сортировками с помощью включений и с помощью выбора; фактически в пузырьковой сортировке нет ничего ценного, кроме привлекательного названия.

Сортировка прямыми включениями (сортировка вставками)

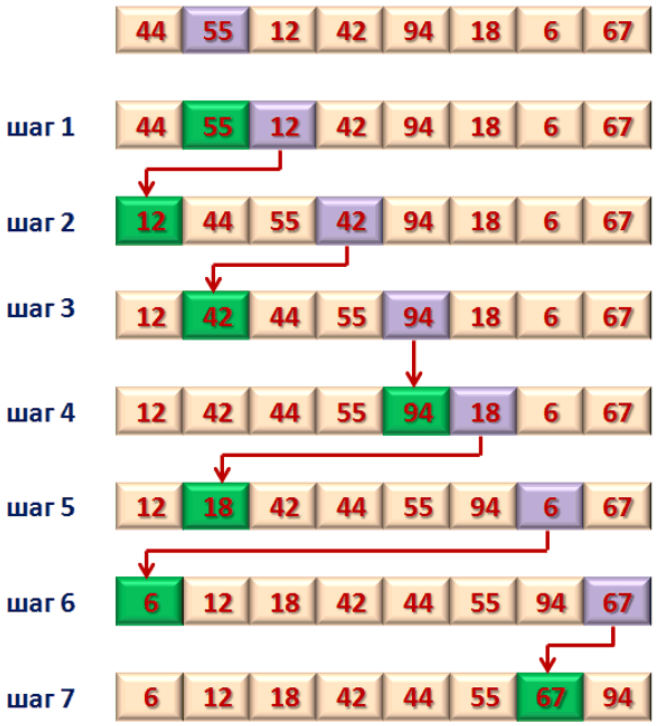
Элементы массива условно разделяются на готовую последовательность

$a_1, a_2, \ldots, a_{i-1}$

и входную последовательность

$a_i, a_{i+1}, \ldots, a_n$.

На каждом шаге i-й элемент помещается на подходящее место в готовую последовательность.



Реализация сортировки прямыми включениями на Си

```
#include <stdio.h>
// Функция сортировки прямыми включениями
void inclusionSort(int *num, int size)
{
  // Для всех элементов кроме начального
  for (int i = 1; i < size; i++)
  {
    int value = num[i]; // запоминаем значение элемента
    int index = i;     // и его индекс
    while ((index > 0) && (num[ind–x - 1] > value))
    {   // смещаем другие элементы к концу массива пока они меньше index
      num[index] = num[ind–x - 1];
      index--;    // смещаем просмотр к началу массива
    }
    num[index] = value; // рассматриваемый элемент помещаем на освободившееся место
  }
}
int main()
{
  int a[10]; // Объявляем массив из 10 элементов
  // Вводим значения элементов массива
  for (int i = 0; i < 10; i++)
  {
    prin«f("a[%d]«= ", i);
    sca«f(»%d", &a[i]);
  }
  inclusionSort(a, 10);  // вызываем функцию сортировки
  // Выводим отсортированные элементы массива
  for (int i = 0; i<10; i++)
```

```
   prin«f("«d ", a[i]);
  getchar(); getchar();
  return 0;
}
```

Анализ выполнения

Число сравнений ключей C_i при i-м просеивании составляет самое большое i-1, самое меньшее – 1. Если предположить, что все перестановки из n ключей равно вероятны, то среднее число сравнений – i/2. Число пересылок

$M_i = C_i+2$.

Поэтому общее число сравнений и пересылок таковы:

Cmin=n-1;

Mmin=3(n-1);

$Ccp=(n^2+n-2)/4$;

$Mcp=(n^2+9n-10)/4$;

$Cmax=(n^2+n-4)/4$; $Mmax=(n^2+3n-4)/2$.

Минимальные оценки встречаются в случае уже упорядоченной исходной последовательности элементов, наихудшие оценки — когда элементы первоначально расположены в обратном порядке.

Резюме: сортировка методом прямого включения – не очень подходящий метод для компьютера, поскольку включение элемента с последующим сдвигом на одну позицию целой группы элементов неэффективно.

Сортировка вставками отличается от сортировки пузырьком тем, что мы «сопровождаем» элемент массива и вставляем его на нужное место. В сортировке пузырьком, после обмена местами двух элементов, даже если этот обмен привёл к опять к нарушению порядка, мы всё равно двигаемся дальше.

Сортировка вставками в случае работы с отсортированным массивом проходит его всего один раз. Несмотря на сложность порядка n^2, сортировка вставками оказывается наиболее эффективной при сортировке массивов маленького размера (единицы, десятки элементов), поэтому используется в сложных алгоритмах сортировки для работы с маленькими массивами. Например, пусть мы используем сортировку слиянием и дошли до состояния, когда размер подмассива равен 8. Его вполне можно отсортировать вставками и передать дальше, как уже отсортированный для слияния.

Контрольные вопросы:

1. Что такое поиск элементов?
2. Что такое упорядочивание?
3. Что такое сортировка элементов?
4. В чем заключается алгоритм сортировки методом «пузырька»?
5. В чем заключается алгоритм сортировки вставками?

Тема 18. Работа с динамической памятью. Структуры, объединения, перечисления

Цель: сформровать знания о динамических структурах данных: структуры, обединения, пречисления.

Структура — это объединение нескольких объектов, возможно, различного типа под одним именем, которое является типом структуры. В качестве объектов могут выступать переменные, массивы, указатели и другие структуры.

Структуры позволяют трактовать группу связанных между собой объектов не как множество отдельных элементов, а как единое целое. Структура представляет собой сложный тип данных, составленный из простых типов.

Общая форма объявления структуры:

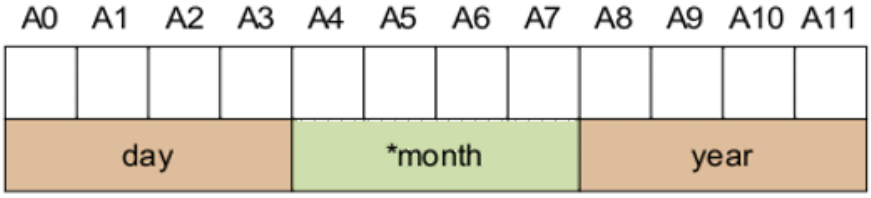
```
struct ИмяСтруктуры
{
 тип ИмяЭлемента1;
 тип ИмяЭлемента2;
 . . .
 тип ИмяЭлементаn;
};
```

После закрывающей фигурной скобки } в объявлении структуры обязательно ставится точка с запятой.

Пример объявления структуры

```
struct date
{
 int day;     // 4 байта
 char *month; // 4 байта
 int year;    // 4 байта
};
```

Поля структуры располагаются в памяти в том порядке, в котором они объявлены:



В указанном примере структура date занимает в памяти 12 байт. Кроме того, указатель *month при инициализации будет началом текстовой строки с названием месяца, размещенной в памяти.

При объявлении структур, их разрешается вкладывать одну в другую.

Пример

```
struct persone
{
 char lastname[20];   // фамилия
 char firstname[20]; // имя
 struct date bd;     // дата рождения
};
```

Инициализация полей структуры

Инициализация полей структуры может осуществляться двумя способами:

– присвоение значений элементам структуры в процессе объявления переменной, относящейся к типу структуры;

– присвоение начальных значений элементам структуры с использованием функций ввода-вывода (например, printf() и scanf()).

В первом способе инициализация осуществляется по следующей форме:

struct ИмяСтруктуры ИмяПеременной={ЗначениеЭлемента1, ЗначениеЭлемента_2, . . . , ЗначениеЭлементаn};

Пример

struct date bd=»8,"и»ня", 1978};

Имя элемента структуры является составным. Для обращения к элементу структуры нужно указать имя структуры и имя самого элемента. Они разделяются точкой:

ИмяПеременной.ИмяЭлементаСтруктуры

prin«f("%d %s»%d",bd.day, bd.month, bd.year);

Второй способ инициализации объектов языка Си с использованием функций ввода-вывода.

Пример

```
#include <stdio.h>
#include <stdlib.h>
struct date {
  int day;
  char month[20];
  int year;
};
struct persone {
  char firstname[20];
  char lastname[20];
  struct date bd;
};
int main() {
  syst«m("chcp 1»51");
  syst«m("»ls");
  struct persone p;
  prin«f("Введите имя«: ");
  sca«f(»%s", p.firstname);
  prin«f("Введите фамилию«: ");
  sca«f(»%s", p.lastname);
  prin«f("Введите дату рождения\nЧисл«: ");
  sca«f(»%d", &p.bd.day);
  prin«f("Меся«: ");
  sca«f(»%s", p.bd.month);
  prin«f("Го«: ");
  sca«f(»%d", &p.bd.year);
  prin«f("\nВы ввели : %s %s, дата рождения %d %s %d г»да",
    p.firstname, p.lastname, p.bd.day, p.bd.month, p.bd.year);
  getchar(); getchar();
  return 0;
}
```

Имя структурной переменной может быть указано при объявлении структуры. В этом случае оно размещается после закрывающей фигурной скобки }. Область видимости такой структурной переменной будет определяться местом описания структуры.

```
struct complex_type  // имя структуры
{
  double real;
  double imag;
} number;    // имя структурной переменной
```

Поля приведенной структурной переменной: number.real, number.imag .

Объединения

Объединениями называют сложный тип данных, позволяющий размещать в одном и том же месте оперативной памяти данные различных типов. Размер оперативной памяти, требуемый для хранения объединений, определяется размером

памяти, необходимым для размещения данных того типа, который требует максимального количества байт.
Когда используется элемент меньшей длины, чем наиболее длинный элемент объединения, то этот элемент использует только часть отведенной памяти. Все элементы объединения хранятся в одной и той же области памяти, начиная с одного адреса.

Общая форма объявления объединения

```
union ИмяОбъединения
{
 тип ИмяОбъекта1;
 тип ИмяОбъекта2;
 . . .
 тип ИмяОбъектаn;
};
```

Объединения применяются для следующих целей:

– для инициализации объекта, если в каждый момент времени только один из многих объектов является активным;

– для интерпретации представления одного типа данных в виде другого типа.

Например, удобно использовать объединения, когда необходимо вещественное число типа float представить в виде совокупности байтов

```
#include <stdio.h>
#include <stdlib.h>
union types
{
 float f;
 unsigned char b[4];
};
int main()
{
 types value;
 prin«f("N«= ");
 sca«f(»%f", &value.f);
 prin«f("%f = %x %x %x»%x", value.f, value.b[0], value.b[1], value.b[2], value.b[3]);
 getchar();
 getchar();
 return 0;
}
```

Пример Поменять местами два младших байта во введенном числе

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
int main() {
 char temp;
 syst«m("chcp 1»51");
 syst«m("»ls");
 union
 {
  unsigned char p[2];
  unsigned int t;
 } type;
 prin«f("Введите число«: ");
 sca«f(»%d", &type.t);
```

```
  prin«f("%d = %x шестн»\n", type.t, type.t);
  // Замена байтов
  temp = type.p[0];
  type.p[0] = type.p[1];
  type.p[1] = temp;
  prin«f("Поменяли местами байты, получил»\n");
  prin«f("%d = %x шестн»\n", type.t, type.t);
  getchar(); getchar();
  return 0;
}
```

Битовые поля

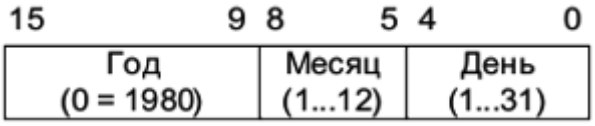
Используя структуры, можно упаковать целочисленные компоненты еще более плотно, чем это было сделано с использованием массива.

Набор разрядов целого числа можно разбить на битовые поля, каждое из которых выделяется для определенной переменной. При работе с битовыми полями количество битов, выделяемое для хранения каждого поля отделяется от имени двоеточием

тип имя: КоличествоБит

При работе с битовыми полями нужно внимательно следить за тем, чтобы значение переменной не потребовало памяти больше, чем под неё выделено.

Пример Разработать программу, осуществляющую упаковку даты в формат




```
#include <stdio.h>
#include <stdlib.h>
#define YEAR0 1980
struct date
{
  unsigned short day : 5;
  unsigned short month : 4;
  unsigned short year : 7;
};
int main() {
  struct date today;
  syst«m("chcp 1»51");
  syst«m("»ls");
  today.day = 16;
  today.month = 12;
  today.year = 20–3 - YEAR0; //today.year = 33
  prin«f("\n Сегодня %u.%u.%u»\n", today.day, today.month, today.year + YEAR0);
  prin«f("\n Размер структуры today : %d б»йт", sizeof(today));
  prin«f("\n Значение элемента today = %hu = %hx шес»н.", today, today);
  getchar();
  return 0;
}
```

Массивы структур

Работа с массивами структур аналогична работе со статическими массивами других типов данных.

Пример Библиотека из 3 книг

```
#include <stdio.h>
#include <stdlib.h>
struct book
{
  char title[15];
  char author[15];
  int value;
};
int main()
{
  struct book libry[3];
  int i;
  syst«m("chcp 1»51");
  syst«m("»ls");
  for (i = 0; i<3; i++)
  {
    prin«f("Введите название %d книги«: ", i + 1);
    gets_s(libry[i].title);
    prin«f("Введите автора %d книги«: ", i + 1);
    gets_s(libry[i].author);
    prin«f("Введите цену %d книги«: ", i + 1);
    scanf«s(»%d", &libry[i].value);
    getchar();
  }
  for (i = 0; i<3; i++)
  {
    prin«f("\n %d. «s ", i + 1, libry[i].author);
    prin«f("%s»%d", libry[i].title, libry[i].value);
  }
  getchar();
  return 0;
}
```

Указатели на структуры

Доступ к элементам структуры или объединения можно осуществить с помощью указателей. Для этого необходимо инициализировать указатель адресом структуры или объединения.

Для организации работы с массивом можно использовать указатель. При этом обращение к полям структуры через указатель будет выглядеть как:

указатель->поле или

(*указатель).поле

указатель — указатель на структуру или объединение; поле — поле структуры или объединения;

Динамическое выделение памяти для структур

Динамически выделять память под массив структур необходимо в том случае, если заранее неизвестен размер массива. Для определения размера структуры в байтах используется операция sizeof(ИмяСтруктуры).

Пример Библиотека из 3 книг

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
struct book
```

```
{
  char title[15];
  char author[15];
  int value;
};
int main()
{
  struct book *lib;
  int i;
  syst«m("chcp 1»51");
  syst«m("»ls");
  lib = (struct book*)malloc(3 * sizeof(struct book));
  for (i = 0; i<3; i++)
  {
    prin«f("Введите название %d книги«: ", i + 1);
    gets_s((lib + i)->title);
    prin«f("Введите автора %d книги«: ", i + 1);
    gets_s((lib + i)->author);
    prin«f("Введите цену %d книги«: ", i + 1);
    scanf«s(»%d", &(lib + i)->value);
    getchar();
  }
  for (i = 0; i<3; i++)
  {
    prin«f("\n %d. «s ", i + 1, (lib + i)->author);
    prin«f("%s»%d", (lib + i)->title, (lib + i)->value);
  }
  getchar();
  return 0;
}
```

Результат выполнения аналогичен предыдущему решению.

Задание: Написать конспект. Привести примеры, где можно использовать структуры. Создать кластер по сложным типам данных.

Контрольные вопросы:

1. Что такое структура?
2. Что такое объединение?
3. Как происходит динамическое выделение памяти?

Тема 19. Массив символов

Цель: сформировать представление о массиве символов как о средстве работе со строками.

Существует специальная форма инициализации массива типа char — с помощью символьной строки. Например, объявление

char code[]«= "»bc";

инициализирует массив code четырьмя символами—'a', 'b', 'c' и символом '\0', который завершает символьную строку.

Если в объявлении размер массива указан, а длина инициализирующей строки превышает указанный размер, то лишние символы отбрасываются. Следующее объявление инициализирует трехэлементный массив code типа char:

char code[3]«= "a»cd";

В примере только три первые символа инициализатора заносятся в массив code.

Символ d и символ '\0' отбрасываются.

Если инициализирующая строка короче, чем специфицированный размер массива, то оставшиеся элементы массива инициализируются нулевым значением (символом '\0').

Символьной строкой можно инициализировать не только массив типа char, но и указатель на тип char. Например, в объявлении

char *ptr«= "a»cd";

указатель ptr будет инициализирован адресом массива типа char, содержащего символы 'a', 'b', 'c', 'd', '\0'.

В программе строки могут определяться следующим образом:

- как строковые константы;
- как массивы символов;
- через указатель на символьный тип;
- как массивы строк.
- Кроме того, должно быть предусмотрено выделение памяти для хранения строки.

Любая последовательность символов, заключенная в двойные кавычки «», рассматривается как строковая константа.

Для корректного вывода любая строка должна заканчиваться нуль-символом '\0', целочисленное значение которого равно 0. При объявлении строковой константы нуль-символ добавляется к ней автоматически. Так, последовательность символов, представляющая собой строковую константу, будет размещена в оперативной памяти компьютера, включая нулевой байт.

Под хранение строки выделяются последовательно идущие ячейки оперативной памяти. Таким образом, строка представляет собой массив символов. Для хранения кода каждого символа строки отводится 1 байт.

Для помещения в строковую константу некоторых служебных символов используются символьные комбинации. Так, если необходимо включить в строку символ двойной кавычки, ему должен предшествовать символ «обратный слеш»: ‘\»‘.

Строковые константы размещаются в статической памяти. Начальный адрес последовательности символов в двойных кавычках трактуется как адрес строки. Строковые константы часто используются для осуществления диалога с пользователем в таких функциях, как printf().

При определении массива символов необходимо сообщить компилятору требуемый размер памяти.

char m[82];

Компилятор также может самостоятельно определить размер массива символов, если инициализация массива задана при объявлении строковой константой:

char m2»]="Горные вершины спят во тьме ночн»й."; char m3[]={'Т','и','х','и','е',' ','д','о','л','и','н','ы',' ','п','о','л','н','ы',' ','с','в','е','ж','е','й',' ','м','г','л','о','й','\0'};

В этом случае имена m2 и m3 являются указателями на первые элементы массивов:

m2 эквивалентно &m2[0]

m2[0] эквивалентно ‘Г’

m2[1] эквивалентно ‘о’

m3 эквивалентно &m3[0]

m3[2] эквивалентно ‘x’

При объявлении массива символов и инициализации его строковой константой можно явно указать размер массива, но указанный размер массива должен быть больше, чем размер инициализирующей строковой константы:

char m2[8»]="Горные вершины спят во тьме ночн»й.";

Для задания строки можно использовать указатель на символьный тип.

char *m4;

В этом случае объявление массива переменной m4 может быть присвоен адрес массива:

m4 = m3;

*m4 эквивалентно m3[0]='T'

*(m4+1) эквивалентно m3[1]='и'

Здесь m3 является константой-указателем. Нельзя изменить m3, так как это означало бы изменение положения (адреса) массива в памяти, в отличие от m4.

Для указателя можно использовать операцию увеличения (перемещения на следующий символ):

m4++;

Массивы символьных строк

Иногда в программах возникает необходимость описание массива символьных строк. В этом случае можно использовать индекс строки для доступа к нескольким разным строкам.

char *poet[4] « {"Погиб по»т!«, "- невольник чес–« -«, "П»л,"«, "оклеветанный молв»й…"};

В этом случае poet является массивом, состоящим из четырех указателей на символьные строки. Каждая строка символов представляет собой символьный массив, поэтому имеется четыре указателя на массивы. Указатель poet[0] ссылается на первую строку:

*poet[0] эквивалентно 'П',

*poet[l] эквивалентно '-'

Инициализация выполняется по правилам, определенным для массивов. Тексты в кавычках эквивалентны инициализации каждой строки в массиве. Запятая разделяет соседние последовательности.

Кроме того, можно явно задавать размер строк символов, используя описание, подобное такому:

char poet[4][23];

Разница заключается в том, что такая форма задает «прямоугольный» массив, в котором все строки имеют одинаковую длину.

Свободный массив

Описание

char *poet[4];

определяет свободный массив, где длина каждой строки определяется тем указателем, который эту строку инициализирует. Свободный массив не тратит память напрасно.

Операции со строками

Большинство операций языка Си, имеющих дело со строками, работает с указателями. Для размещения в оперативной памяти строки символов необходимо:

выделить блок оперативной памяти под массив;

проинициализировать строку.

Для выделения памяти под хранение строки могут использоваться функции динамического выделения памяти. При этом необходимо учитывать требуемый размер строки:

```
char *name;
name = (char*)malloc(10);
sca«f("»9s", name);
```

Для ввода строки использована функция scanf(), причем введенная строка не может превышать 9 символов. Последний символ будет содержать '\0'.

Функции ввода строк

Для ввода строки может использоваться функция scanf(). Однако функция scanf() предназначена скорее для получения слова, а не строки. Если применять форм«т »%s" для ввода, строка вводится до (но не включая) следующего пустого символа, которым может быть пробел, табуляция или перевод строки.

Для ввода строки, включая пробелы, используется функция

```
char * gets(char *);
```

или её эквивалент

```
char * gets_s(char *);
```

В качестве аргумента функции передается указатель на строку, в которую осуществляется ввод. Функция просит пользователя ввести строку, которую она помещает в массив, пока пользователь не нажмет Enter.

Функции вывода строк

Для вывода строк можно воспользоваться рассмотренной ранее функцией

```
prin«f(»%s", str); // str — указатель на строку
```

или в сокращенном формате

```
printf(str);
```

Для вывода строк также может использоваться функция

```
int puts (char *s);
```

которая печатает строку s и переводит курсор на новую строку (в отличие от printf()). Функция puts() также может использоваться для вывода строковых констант, заключенных в кавычки.

Функция ввода символов

Для ввода символов может использоваться функция

```
char getchar();
```

которая возвращает значение символа, введенного с клавиатуры. Указанная функция использовалась в рассмотренных ранее примерах для задержки окна консоли после выполнения программы до нажатия клавиши.

Функция вывода символов

Для вывода символов может использоваться функция

```
char putchar(char);
```

которая возвращает значение выводимого символа и выводит на экран символ, переданный в качестве аргумента.

Пример Посчитать количество введенных символов во введенной строке.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
  char s[80], sym;
```

```
  int count, i;
  syst«m("chcp 1»51");
  syst«m("»ls");
  prin«f("Введите строку«: ");
  gets_s(s);
  prin«f("Введите символ«: ");
  sym = getchar();
  count = 0;
  for (i = 0; s[i] != '\0'; i++)
  {
    if (s[i] == sym)
      count++;
  }
  prin«f("В строк»\n");
  puts(s);      // Вывод строки
  prin«f("симв«л ");
  putchar(sym); // Вывод символа
  prin«f(" встречается %d »аз", count);
  getchar(); getchar();
  return 0;
}
```

Контрольные вопросы:

1.Как можно представить строку в языке программирования Си?

2. Какие операторы используются для ввода массива символов?

3. Какие операторы используются для вывода массива символов?

4. С помощью какого оператора происходит инициализация элементов массива строк?

Тема 20. Строки в языке С. Обработка строк

Цель: сформировать знания о работе со строками в языке программирования Си.

Основные функции стандартной библиотеки string.h приведены в таблице.

Библиотека string.h предоставляет функции для работы со строками (zero-terminated strings) в си, а также несколько функций для работы с массивами, которые сильно упрощают жизнь. Рассмотрим функции с примерами.

Функция	Описание
char *strcat(char *s1, char *s2)	присоединяет s2 к s1, возвращает s1
char *strncat(char *s1, char *s2, int n)	присоединяет не более n символов s2 к s1, завершает строку символом '\0', возвращает s1
char *strcpy(char *s1, char *s2)	копирует строку s2 в строку s1, включая '\0', возвращает s1
char *strncpy(char *s1, char *s2, int n)	копирует не более n символов строки s2 в строку s1, возвращает s1;
int strcmp(char *s1, char *s2)	сравнивает s1 и s2, возвращает значение 0, если строки эквивалентны

int strncmp(char *s1, char *s2, int n)	сравнивает не более n символов строк s1 и s2, возвращает значение 0, если начальные n символов строк эквивалентны
int strlen(char *s)	возвращает количество символов в строке s
char *strset(char *s, char c)	заполняет строку s символами, код которых равен значению c, возвращает указатель на строку s
char *strnset(char *s, char c, int n)	заменяет первые n символов строки s символами, код которых равен c, возвращает указатель на строку s

```
  return 0;
}
```

Копирование

void * memcpy (void * destination, const void * source, size_t num);

Копирует участок памяти из source в destination, размером num байт. Функция очень полезная, с помощью неё, например, можно скопировать объект или перенести участок массива, вместо поэлементного копирования. Функция производит бинарное копирование, тип данных не важен. Например, удалим элемент из массива и сдвинем остаток массива влево.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>

#define SIZE 10

int main() {
   int a[SIZE] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   unsigned index;
   int i;

   prin«f("Enter index [0 ..»9]");
   sca«f("»ud", &index);
   index = index < SIZe? index: SIZE-1;
   memcpy(&a[index], &a[index+1], sizeof(int) * (SI–E - ind–x - 1));

   for (i = 0; i < SIZE; i++) {
      prin«f("«d ", a[i]);
   }
   getch();
}
```

Функция меняет местами две переменные

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void swap(void* a, void* b, size_t size) {
```

```
    void *tmp = malloc(size);
    memcpy(tmp, a, size);
    memcpy(a, b, size);
    memcpy(b, tmp, size);
    free(tmp);
}

int main() {
    float a = 300.456;
    float b = 0.645;

    swap(&a, &b, sizeof(float));
    prin«f("a = %.3f\nb = %»3f", a, b);
    getch();
}
```

Здесь хотелось бы отметить, что функция выделяет память под временную переменную. Это дорогостоящая операция. Для улучшения производительности стоит передавать функции временную переменную, которая будет создана один раз.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void swap(void* a, void* b, void* tmp, size_t size) {
    memcpy(tmp, a, size);
    memcpy(a, b, size);
    memcpy(b, tmp, size);
}

int main() {
    float a = 300.456;
    float b = 0.645;
    float tmp;

    swap(&a, &b, &tmp, sizeof(float));
    prin«f("a = %.3f\nb = %»3f", a, b);
    getch();
}
```

void* memmove (void * destination, const void * source, size_t num);

Копирует блок памяти из source в destination размером num байт с той разницей, что области могут пересекаться. Во время копирования используется промежуточный буфер, который предотвращает перекрытие областей.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>

void main () {
    char str[]«= "memmove can be very useful...»..";
    memmove (str + 20, str + 15, 11);
    puts(str);
```

```
    getch();
}
```
Пример взят из cplusplus.com

char* strcpy (char * destination, const char* source);

Копирует одну строку в другую, вместе с нулевым символом. Также возвращает указатель на destination.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>

void main () {
   char buffer[128];
   char *word = NULL;

   sca«f("%1»7s", buffer);
   word = (char*) malloc(strlen(buffer)+1);
   strcpy(word, buffer);

   prin«f(»%s", word);
   free(word);
   getch();
}
```
Можно копировать и по-другому

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>

void main () {
   char buffer[128];
   char *word = NULL;
   char *other = NULL;

   sca«f("%1»7s", buffer);
   word = (char*) malloc(strlen(buffer)+1);
   other = strcpy(word, buffer);

   prin«f(»%s", other);
   free(other);
   getch();
}
```
char* strncpy (char* destination, const char* source, size_t num);

Копирует только num первых букв строки. 0 в конец не добавляется автоматически. При копировании из строки в эту же строку части не должны пересекаться (при пересечении используйте memmove)

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
```

```
void main () {
   char word[]«= "Aloha, Haw»ii";
   char aloha[20];
   char hawaii[20];

   strncpy(aloha, word, 5);
   aloha[5] = 0;
   strncpy(hawaii, &word[7], 7);

   prin«f("%s,»%s", aloha, hawaii);
   getch();
}
```

Конкатенация строк

char* strcat (char * destination, const char * source);

Добавляет в конец destination строку source, при этом затирая первым символом нулевой. Возвращает указатель на destination.

char* strncat (char * destination, const char * source, size_t num);

Добавляет в конец строки destination num символов второй строки. В конец добавляется нулевой символ.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>

void main () {
   char a[255];
   char b[128];
   sca«f("%1»7s", a);
   sca«f("%1»7s", b);
   strncat(a, b, strlen(b)/2);
   prin«f(»%s", a);
   getch();
}
```

Сравнение строк

int strcmp (const char * str1, const char * str2);

Возвращает 0, если строки равны, больше нуля, если первая строка больше, меньше нуля, если первая строка меньше. Сравнение строк происходит посимвольно, сравниваются численные значения. Для сравнения строк на определённом языке используется strcoll

int strcoll (const char * str1, const char * str2);

int strncmp (const char * str1, const char * str2, size_t num);

сравнение строк по первым num символам

Прим–р - сортировка массива строк по первым трём символам

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

int cmp(const void *a, const void *b) {
   return strncmp((char*) a, (char*) b, 3);
}
```

```
void main() {
   char words[][128] = {
    «  "So»ar",
    «  "Obscu»us",
    «  "Tempes»us",
    «  "Ult»ma",
    «  "Pacifi»us"
   };
   int i;

   qsort(words, 5, 128, cmp);
   for (i = 0; i < 5; i++) {
      prin«f("%»\n", words[i]);
   }
   getch();
}
```

size_t strxfrm (char * destination, const char * source, size_t num);

Трансформация строки в соответствии с локалью. В строку destination копируется num трансформированных символов строки source и возвращается её длина. Если num == 0 и destination == NULL, то возвращается просто длина строки.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <locale.h>

void main() {
   char input[128];
   char output[128];

   sca«f("%1»7s", input);
   //Выводи введённую строку
   prin«f("%»\n", input);
   //Проводим преобразование, ничего не меняется
   strxfrm(output, input, 128);
   prin«f("%»\n", output);
   //Изменяем локаль
   setlocale(LC_AL«, ".1»51");
   strxfrm(output, input, 128);
   prin«f("%»\n", output);
   getch();
}
```

Поиск

void* memchr (void * ptr, int value, size_t num);

Проводит поиск среди первых num байтов участка памяти, на который ссылается ptr, первого вхождения значения value, которое трактуется как unsigned char. Возвращает указатель на найденный элемент, либо NULL.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
```

```
void main() {
   char str[]«= "Hello Wor»d!";
   char *ptr = NULL;

   ptr = (char*) memchr(str, '\0', 4000);
   if (ptr != NULL) {
      prin«f("first zero byte address is %p, strlen =»%d", ptr, p–r - str);
   } else {
      prin«f("no null byte in memory bl»ck");
   }
   getch();
}
```

?

char* strchr (char * str, int character);

Возвращает указатель на место первого вхождения character в строку str. Очень похожа на функцию memchr, но работает со строками, а не с произвольным блоком памяти.

size_t strcspn (const char * str1, const char * str2);

Возвращает адрес первого вхождения любой буквы из строки str2 в строке str1. Если ни одно включение не найдено, то возвратит длину строки. Прим–р - найдём положение всех гласных в строке

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main() {
   char str[]«= "So if you want to love m»\n"
         «  "Then darling don't refrai»\n"
         «  "Or I'll just end up walkin»\n"
         «  "In the cold November rai»\n";
   char vowels[]«= "aei»uy";
   int i;

   i = 0;
   while (str[i]) {
      i = i + strcspn(&str[i], vowels);
      prin«f("«d ", i);
      i++;
   }
   getch();
}
```

Здесь обратите внимание на строку i++ после printf. Если бы её не было, то strcspn возвращал бы всегда 0, потому что в начале строки стояла бы гласная, и произошло зацикливание.

Для решения этой задачи гораздо лучше подошла функция, которая возвращает указатель на первую гласную.

char* strpbrk (char * str1, const char * str2)

Функция очень похожа на strcspn, только возвращает указатель на первый символ из строки str1, который есть в строке str2. Выведем все гласные в строке

```
#include <stdio.h>
#include <conio.h>
```

```
#include <string.h>

void main() {
   char str[]«= "Cos' it's a bittersweet symphony this life..»\n"
         «  "Trying to make ends meet, you're a slave to the money then you d»e.";
   char vowels[]«= "aei»uy";
   char *p = NULL;

   p = strpbrk(str, vowels);
   while (p) {
      prin«f("«c ", *p);
      p++;
      p = strpbrk(p, vowels);
   }
   getch();
}
```

char* strrchr (char * str, int character);
Возвращает указатель на последнее вхождение символа в троку.

size_t strspn (const char * str1, const char * str2);
Возвращает длину куска строки str1, начиная от начала, который состоит только из букв строки str2.
Прим–р - вывести число, которое встречается в строке.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main() {
   char str[]«= "on 21st of »ay";
   char nums[]«= "0123456»89";
   char number[10];
   uintptr_t i;

   //Определяем, где начинаются цифры
   size_t start = strcspn(str, nums);
   //Определяем, где они заканчиваются, относительно start
   size_t end = strspn(&str[start], nums);

   for (i = 0; i < end; i++) {
      prin«f(»%c", str[start+i]);
   }

   getch();
}
```

char* strstr (char * str1, const char * str2);
Возвращает указатель на первое вхождение строки str2 в строку str1.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
```

```
void main() {
   char str[]«= "I'll drown my belief»\n"
         «  "To have you be in peac»\n"
         «  "I'll dress like your niec»\n"
         «  "And wash your swollen fee»\n";
   char niece[]«= "ni»ce";

   char* p = strstr(str, niece);
   prin«f(»%s", p);
   getch();
}
```

?

```
char* strtok (char * str, const char * delimiters);
```

Разбивает строку на токены. В данном случае токенами считаются последовательности символов, разделённых символами, входящими в группу разделителей.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main() {
   char str[]«= "After working in India during the late 1970s and 1980«, "
    «  "Shankar's profile in the West began to rise again in the mid-199«s "
    «  "as his music found its way into club DJ sets, particularly in Lond»n.";
   char delim[]«= " \t\n\«.-";
   char *p = strtok(str, delim);
   while (p != NULL) {
      print« ("%»\n",p);
      p = strtok (NULL, delim);
   }
   getch();
}
```

Контрольные вопросы:

1. Что подразумевает понятие «обработка строк»?
2. Что такое конкатенация строк?
3. Какая функция используется для компироания подстроки в строку?
4. Какая функции используется для поиска?

Тема 21. Файловые потоки. Работа с текстовыми файлами

Цель: сформировать знания по работе с файловыми потоками в языке программирования Си, изучить основные функции работы с файлами.

Для удобства обращения информация в запоминающих устройствах хранится в виде файлов.

Файл – именованная область внешней памяти, выделенная для хранения массива данных. Данные, содержащиеся в файлах, имеют самый разнообразный характер: программы на алгоритмическом или машинном языке; исходные данные для работы программ или результаты выполнения программ; произвольные тексты; графические изображения и Т. п.

Каталог (**папка**, **директория**) – именованная совокупность байтов на носителе информации, содержащая название подкаталогов и файлов, используется в файловой системе для упрощения организации файлов.

Файловой системой называется функциональная часть операционной системы, обеспечивающая выполнение операций над файлами. Примерами файловых систем являются FAT (FAT – File Allocation Table, таблица размещения файлов), NTFS, UDF (используется на компакт-дисках).

Существуют три основные версии FAT: FAT12, FAT16 и FAT32. Они отличаются разрядностью записей в дисковой структуре, т.е. количеством бит, отведённых для хранения номера кластера. FAT12 применяется в основном для дискет (до 4 кбайт), FAT16 – для дисков малого объёма, FAT32 – для FLASH-накопителей большой емкости (до 32 Гбайт).

Для программиста открытый файл представляется как последовательность считываемых или записываемых данных. При открытии файла с ним связывается **поток ввода-вывода**. Выводимая информация записывается в поток, вводимая информация считывается из потока.

Когда поток открывается для ввода-вывода, он связывается со стандартной структурой типа FILE, которая определена в stdio.h. Структура FILE содержит необходимую информацию о файле.

Типы файлов (с точки зрения интерпретации информации в программе на Си):

текстовые;

бинарные.

Основные операции производимые над файлами:

Открытие файлов.

Чтение и запись данных.

Закрытие файлов.

Дополнительные операции:

Навигация по файлу.

Обработка ошибок работы с файлами.

Удаление и переименование файлов.

Описание переменной

Для работы с файлами в программах на Си используется заголовочный файл

stdio.h , в котором объявлен специальный тип данных — структура FILE, предназначенная для хранения атрибутов (параметров) файлов (указатель текущей позиции файла, признак конца файла, флаги индикации ошибок, сведения о буферизации и др.). Переменные типа FILE нельзя создавать вручную.

Для работы с файлом в программе нужно созда«ь "указатель на ф»йл".

FILE *имя = NULL;

Нужно осознавать, что каждое обращение к файлу выполняется через системный вызов. При этом этот указатель не нужно разыменовывать при обращении к файлу, библиотечной функции передается сам этот указатель.

Открытие файла

FILE *fopen(const char *filename, const char *mode);

где filename — название файла, mode — режим открытия.

Функция возвращает указатель на файл, если тот был успешно открыт. В противном случае — NULL.

Название файла содержит только имя файла, если файл находится в текущем каталоге. Иначе необходимо указать абсолютный или относительный путь к файлу.

Закрытие файла

int fclose(FILE *stream);

где stream — указатель на открытый файл.

Функция возвращает:

0 – файл успешно закрыт.

1 – произошла ошибка закрытия файла.

Можно закрыть и открыть новый файл в одну команду:

FILE * freopen(const char *filename, const char *mode, FILE *stream);

Открытие файла осуществляется с помощью функции fopen(), которая возвращает указатель на структуру типа FILE, который можно использовать для последующих операций с файлом.

FILE *fopen(name, type);

– name – имя открываемого файла (включая путь), type — указатель на строку символов, определяющих способ доступа к файл«:»"r" — открыть файл для чтения (файл должен существовать«;»"w" — открыть пустой файл для записи; если файл существует, то его содержимое теряетс«;»"a" — открыть файл для записи в конец (для добавления); файл создается, если он не существуе«;

– »r+" — открыть файл для чтения и записи (файл должен существовать«;

– »w+" — открыть пустой файл для чтения и записи; если файл существует, то его содержимое теряетс«;

– »a+" — открыть файл для чтения и дополнения, если файл не существует, то он создаётся.

Возвращаемое значение — указатель на открытый поток. Если обнаружена ошибка, то возвращается значение NULL.

Функция fclose() закрывает поток или потоки, связанные с открытыми при помощи функции fopen() файлами. Закрываемый поток определяется аргументом функции fclose().

Возвращаемое значение: значение 0, если поток успешно закрыт; константа EOF, если произошла ошибка.

Функция возвращает указатель на файл, если все нормально, и NULL, если возникла ошибка открытия нового файла или закрытия старого.

Чтение из текстового файла

Форматированное чтение

int fscanf(FILE *stream, const char * format, [arg] …);

Функция возвращает:

больше 0 — число успешно прочитанных переменных,

0 — ни одна из переменных не была успешно прочитана,

EOF — ошибка или достигнут конец файла.

Чтение строки

char * fgets(char * buffer, int maxlen, FILE *stream);

В maxlen следует указать размер буфера, чтобы при записи в память функция чтения не вышла за его границы.

Функция считывает строку до символа перевода каретки '\n' или

maxlen— что наступит раньше.

Функция возвращает указатель на buffer, если все нормально, и NULL, если возникла ошибка или достигнут конец файла.

Чтение символа

int fgetc(FILE *stream);

Функция возвращает код символа, если все нормально, и EOF, если произошла ошибка или достигнут конец файла.

Помещение символа обратно в поток

int ungetc(int c, FILE *stream);

Функция возвращает код символа, если все успешно, и EOF, если произошла ошибка.

Запись в текстовый файл

Форматированный вывод
int fprintf(FILE *stream, const char *format, [arg] …);
Функция возвращает число записанных символов, если все нормально, и отрицательное значение, если произошла ошибка.

Запись строки
int fputs(const char *string, FILE *stream);
Функция возвращает число записанных символов, если все нормально, и EOF, если произошла ошибка.

Запись символа
int fputc(int c, FILE *stream);
Функция возвращает код записанного символа, если все нормально, и EOF, если произошла ошибка.

Чтение и вывод в двоичные файлы
Чтение из двоичных файлов
size_t fread(void *buffer, size_t size, size_t num,FILE *stream);
Функция возвращает количество прочитанных блоков.
Если оно меньше num, то произошла ошибка или достигнут конец файла.

Запись в двоичный файл
size_t fwrite(const void *buffer, size_t size, size_t num, FILE *stream);
Функция возвращает количество записанных блоков.
Если оно меньше num, то произошла ошибка.

Проверка на достижение конца файла
int feof(FILE *stream);
где stream — указатель на открытый файл
Функция возвращает 0, если файловый поток не кончился, и не ноль, если достигнут конец файла.
Навигация по файлу
Чтение текущего смещения в файле
long int ftell(FILE *stream);

Изменение текущего смещения в файле
int fseek(FILE *stream, long int offset, int origin);

Значение origin может принимать три значения:

SEEK_SET (0) – от начала файла.
SEEK_CUR (1) – от текущей позиции.
SEEK_END (2) – от конца файла.
Функция возвращает 0, если все нормально, и не ноль, если произошла ошибка.

Перемещение к началу файла
void rewind(FILE *stream);

Чтение текущей позиции в файле

```
int fgetpos(FILE *stream, fpos_t *pos);
```

Установка текущей позиции в файле

```
int fsetpos(FILE *stream, const fpos_t *pos);
```

Функции возвращают 0, если все нормально, и не ноль, если произошла ошибка.

Структура fpos_t

```
typedef struct fpos_t {
        long off;
        mbstate_t wstate;
    } fpos_t;
```

Получение признака ошибки

```
int ferror(FILE *stream);
```

Функция возвращает ненулевое значение, если возникла ошибка.

Функция сброса ошибки

```
void clearerr(FILE *stream);
```

Функция вывода сообщения об ошибке

```
void perror(const char *string);
```

Буферизация

Функция очистки буфера:

```
int fflush(FILE *stream);
```

Функция возвращает 0, если все нормально, и EOF, если произошла ошибка.

Функция управления буфером:

```
void setbuf(FILE *stream, char *buffer);
```

Создает буфер размером BUFSIZ. Используется до ввода или вывода в поток.

Временные файлы

Функция создания временного файла:

```
FILE * tmpfile(void);
```

Создает временный файл в режиме wb+. После закрытия файла, последний автоматически удаляется.

Функция генерации имени временного файла:

```
char * tmpnam(char *buffer);
```

Удаление файла

```
int remove(const char *filename);
```

Функция возвращает 0 в случае успеха, не ноль в противном случае.

Переименование файла

```
int rename(const char *fname, const char *nname);
```

Функция возвращает 0 в случае успеха, не ноль в противном случае.

Контрольные вопросы:

1. Что такое файл?
2. Опишите функцию для открытия файла?
3. Каким образом определяется степень доступа к файлу?

Раздел 2 Проектирование технологии программного обеспечения

Тема 22. История развития технологии программирования

Цель: сформировать знания об истории развития технологии программирования, ее основных стадиях и ученных, внесших вклад в становлениие технологии программирования.

В 60-х – 70-х годах XX века данный вопрос активно обсуждался на научных конференциях. Существовало две популярных точки зрения:

1. программирование это искусство,
2. программирование это наука.

Дело в том, что разработка программного обеспечения (ПО) имеет ряд специфических особенностей [1].

– Неформальный характер требований к ПО (постановка задачи), но формализованный объект разработки (программы). Тем самым, разработка ПО содержит определенные этапы формализации, а переход от неформального к формально–у - неформален.

– Разработка ПО носит творческий характер. Тем самым, эта разработка ближе к процессу проектирования сложных устройств, но не к их массовому производству.

Технология программирования

Все эти и другие дополнительные виды деятельности, выполняемые в процессе промышленного программирования и необходимые для успешного выполнения заказов, называют технологией программирования = программной инженерией.

Технология = набор правил + методик + инструментов + процессы планирования, оценки качества + др., позволяющих наладить производственный процесс выпуска продукта, сокращая его стоимость и повышая качество.

Технология программирования (Т–) - технология разработки программного средства (ПС), включающая все процессы, начиная с момента зарождения идеи этого средства. Результатом применения ТП является программа, действующая в заданной

вычислительной среде, хорошо отлаженная и документированная, доступная для понимания и развития в процессе сопровождения [10].

Программная инженерия

Программная инженерия является отраслью информатики, которая изучает вопросы построения компьютерных программ, отражает закономерности развития программирования, обобщает опыт программирования в виде комплекса знаний и правил регламентации инженерной деятельности разработчиков ПС [2, 8].

Международным комитетом при американском объединении компьютерных специалистов ACM (Association for Computing Machinery) и институте инженеров по электронике и электротехнике IEEE было создано ядро знаний SWEBOK (Software Engineering Body of Knowledge) (2001, 2003 гг.).

В этом ядре были систематизированы разнородные знания в области программирования, планирования и управления, сформулировано понятие программной инженерии и областей, которые соответствуют процессам проектирования ПС и методам их поддержки.

Программная инженерия охватывает все аспекты создания ПС, начиная от формирования требований до создания, сопровождения и снятия с эксплуатации, а также включает инженерные методы оценки трудозатрат, стоимости, производительности и качества.

Кроме программистов, занимающихся непосредственно разработкой ПС, в программной инженерии задействованы:

– менеджеры, которые планируют и руководят проектом, отслеживают сроки его исполнения и затраты;

– инженеры службы ведения библиотек и репозитариев компонентов;

– технологи, которые определяют инженерные методы и стандарты, создают для проекта модель жизненного цикла ПС, удовлетворяющую его целям и задачам;

– тестировщики, которые проверяют правильность выполнения процесса разработки ПС путем тестирования, и на основе собранных данных проводят измерения характеристик качества;

– валидаторы, которые проверяют ПС на соответствие заданным требованиям (в технике валидация подтверждает, что требования пользователя удовлетворены);

– верификаторы, которые проверяют правильность реализации алгоритмов и программ в проекте путем их сопоставления с эталонными данными, алгоритмами или программами (верификация — это внутренний процесс управления качеством, обеспечивающий согласование с правилами или спецификацией).

Информатизация общества и технология программирования

Технология программирования играла разные роли на разных этапах развития программирования [1]. По мере повышения мощности компьютеров и развития средств программирования росла и сложность решаемых на компьютерах задач, что привело к повышенному вниманию к ТП.

В пятидесятые годы мощность компьютеров была невелика, а программирование для них велось, в основном, в машинном коде. Решались, главным образом, научно-технические задачи, задание на программирование содержало достаточно точную постановку задачи.

Использовалась интуитивная технология программирования: почти сразу приступали к составлению программы по заданию, при этом часто задание несколько раз изменялось (что сильно увеличивало время процесса составления программы), минимальная документация оформлялась уже после того, как программа начинала работать.

Тем не менее, именно в этот период родилась фундаментальная для технологии программирования концепция модульного программирования, ориентированная на преодоление трудностей программирования в машинном коде. Появились первые языки программирования высокого уровня, из которых только ФОРТРАН пробился для использования в следующие десятилетия.

Джон (Янош) Фон Нейман (28 декабря 1903 — 8 февраля 1957)

Внес большой вклад в создание первых ЭВМ и разработку методов их применения. Широко известны «принципы фон Неймана», определяющие архитектуру современных компьютеров: принцип двоичного кодирования, принцип программного управления, принцип адресации памяти и т.д.

В шестидесятые годы происходило бурное развитие и широкое использование языков программирования высокого уровня (АЛГОЛ 60, ФОРТРАН, КОБОЛ и др.).

В результате повышения мощности компьютеров и накопления опыта программирования на языках высокого уровня быстро росла сложность решаемых на компьютерах задач, но надежда на то, что эти языки решат все проблемы, возникающие в процессе разработки больших программ, не оправдалась. Обнаружилась ограниченность языков, проигнорировавших модульную организацию программ. ФОРТРАН, сохранивший возможность модульного программирования, прошествовал в следующие десятилетия. Пользователи отказаться от его услуг не могли из-за накопления большого фонда программных модулей, которые с успехом использовались в новых программах.

Появилось понимание, что важно не только то, на каком языке осуществляется программирование, но методология и технология программирования.

Появление в компьютерах второго поколения прерываний привело к развитию мультипрограммирования и созданию больших программных систем. Широко стала использоваться коллективная разработка, которая поставила ряд серьёзных технологических проблем.

Джон Бэкус (19–4 - 2007)

Один из авторов языков программирования Фортран и Алгол. В 1950 году Бэкус начал работать программистом в фирме IBM. В 1953 году он предложил создать для компьютера IBM-704 язык, позволяющий записывать команды почти в обычной алгебраической форме, и компилятор для него. Большую популярность получила версия под названием “Фортран IV”, выпущенная в 1962 году. Лауреат премии Тьюринга 1977 года.

Эти события произошли в 1968 году: компьютерный дизайн интерфейса пользователя, «мышь», электронная почта, режим видеоконференции, гиперссылки, сеть ARPAnet, которая впоследствии превратилась в Интернет.

В семидесятые годы получили широкое распространение информационные системы и базы данных. К середине 70-х годов стоимость хранения одного бита информации на компьютерных носителях стала меньше, чем на обычных (ранее использовавшихся) носителях. Это резко повысило интерес к компьютерным системам хранения данных. Началось интенсивное развитие технологии программирования в следующих направлениях:

– обоснование и широкое внедрение нисходящей разработки и структурного программирования;

– развитие абстрактных типов данных и модульного программирования;

– исследование проблем обеспечения надёжности и мобильности ПС;

– создание методики управления коллективной разработкой ПС;

– появление инструментальных программных средств (программных инструментов) поддержки технологии программирования.

Эдсгер Вайб Дейкстра (19–0 - 2002)

Автор работ в области применения математической логики при разработке компьютерных программ. Активно участвовал в разработке языка программирования Алгол. Написал первый компилятор Алгол-60. Один из авторов концепции структурного программирования. Выдвинул идею применения «семафоров» для синхронизации процессов в многозадачных системах, автор алгоритма нахождения кратчайшего пути на ориентированном графе с неотрицательными весами рёбер.

Лауреат премии Тьюринга 1972 года.

Восьмидесятые годы характеризуются широким внедрением персональных компьютеров во все сферы человеческой деятельности и тем самым созданием обширного и разнообразного контингента пользователей ПС. Это привело к бурному развитию пользовательских интерфейсов и созданию концепции качества ПС.

Появляются языки программирования (Ада), учитывающие требования технологии программирования. Развиваются методы и языки спецификации ПС. Начинается бурный процесс стандартизации технологических процессов и, прежде всего, документации, создаваемой в этих процессах. Выходит на передовые позиции объектный подход к разработке ПС. Создаются различные инструментальные среды разработки и сопровождения ПС. Развивается концепция компьютерных сетей.

Деннис Ритчи (родился 9 сентября 1941 года)

Специалист по системному программированию, автор и один из разработчиков языка С (Си), одного из самых универсальных языков программирования. На нем была написана почти вся операционная система UNIX, в разработке которой активно участвовал Ритчи. Лауреат премии Тьюринга 1983 года.

Девяностые годы ознаменовались появлением международной компьютерной сети Интернет, персональные компьютеры стали подключаться к ней как терминалы. Это поставило ряд проблем (как технологического, так и юридического и этического характера) регулирования доступа к информации компьютерных сетей. Остро встала проблема защиты компьютерной информации и передаваемых по сети сообщений. Стали бурно развиваться компьютерная технология (CASE-технология) разработки ПС и связанные с ней формальные методы спецификации программ. Начался решающий этап полной информатизации и компьютеризации общества.

Задание: Конспект. Ответить на вопросы. Описать в 5-6 предложениях, как вы видите технологии разработки ПО в будущем?

Контрольные вопросы:

1. Дайте определение понятия технология.
2. Дайте определение понятия технология программирования.
3. Дайте определение понятия программная инженерия.
4. Какие специалисты заняты в программной инженерии?
5. Охарактеризуйте ТП 50-х годов XX века.
6. Охарактеризуйте ТП 60-х годов XX века.
7. Охарактеризуйте ТП 70-х годов XX века.
8. Охарактеризуйте ТП 80-х годов XX века.
9. Охарактеризуйте ТП 90-х годов XX века.
10. Охарактеризуйте ТП XXI века.

Тема 23. Жизненный цикл и критерии качества программы

Цель: изучить понятие жизненного цикла ПО согласно международным стандартам. Рассмотреть особенности каскадной и спиральной моделей ЖЦ ПО.

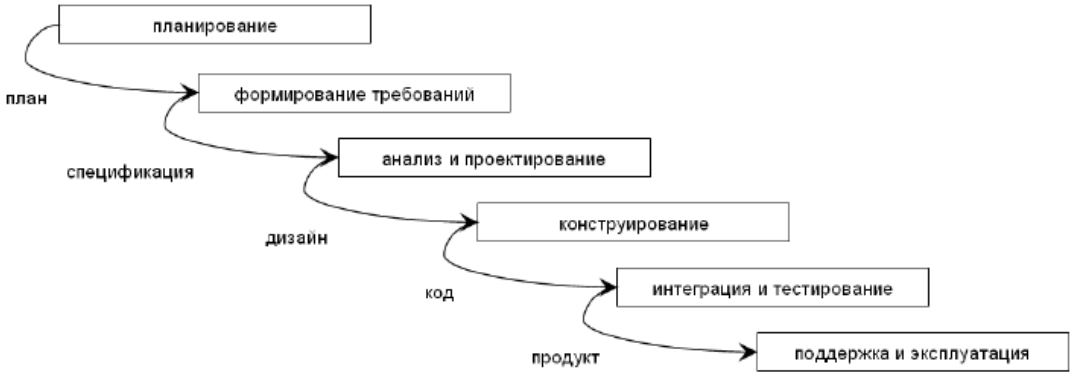
Жизненный цикл программного обеспечения — это период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации. Этот цикл — процесс построения и развития программного обеспечения (ПО). Существует несколько моделей жизненного цикла.

Каскадная модель жизненного цикла (англ. waterfall model) была предложена в 1970 г. Уинстоном Ройсом. Она предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе. Требования, определенные на стадии формирования требований, строго документируются в виде технического задания и фиксируются на все время разработки проекта. Каждая стадия завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Этапы проекта в соответствии с каскадной моделью:

1. Формирование требований;
2. Проектирование;
3. Реализация;
4. Тестирование;
5. Внедрение;
6. Эксплуатация и сопровождение.

В каскадной модели переход от одной фазы проекта к другой предполагает полную корректность результата предыдущей фазы. В больших проектах этого добиться практически невозможно. Поэтому такая модель пригодна только для разработки небольшого проекта. (Сам У. Ройс не придерживался данной модели и использовал модель итерационную).



Область применения каскадной модели

Из-за недостатков каскадной модели ее применение необходимо ограничить ситуациями, в которых требования и их реализация максимально четко определены и понятны.

Каскадная модель хорошо функционирует при ее применении в циклах разработки программного продукта, в которых используется неизменяемое определение продукта и вполне понятные технические методики.

Если компания имеет опыт построения определенного рода системы — автоматизированного бухгалтерского учета, начисления зарплаты, ревизии, компиляции, производства, — тогда в проекте, ориентированном на построение еще одного продукта такого же типа, возможно, даже основанного на существующих разработках, можно эффективно использовать каскадную модель. Другим примером надлежащего применения модели может служить создание и выпуск новой версии уже существующего продукта, если вносимые изменения вполне определены и управляемы. Перенос уже существующего продукта на новую платформу часто приводят в качестве идеального примера использования каскадной модели в проекте.

При всей справедливости критики этой модели все же следует признать, что модифицированная версия каскадной модели является в значительной степени менее жесткой, чем ее первоначальная форма. Здесь включаются итерации между фазами, параллельные фазы и менеджмент изменений. Обратные стрелки предполагают возможность существования итераций между действиями в рамках фаз. Чтобы отобразить согласованность между этапами, их объединяют прямоугольниками или под прямоугольниками перечисляют выполняемые на данных этапах действия, чтобы продемонстрировать согласованность между ними. Несмотря на то, что модифицированная каскадная модель является значительно более гибкой, чем классическая модель, она все же не является наилучшим выбором для выполнения проектов по ускоренной разработке.

Спиральная модель

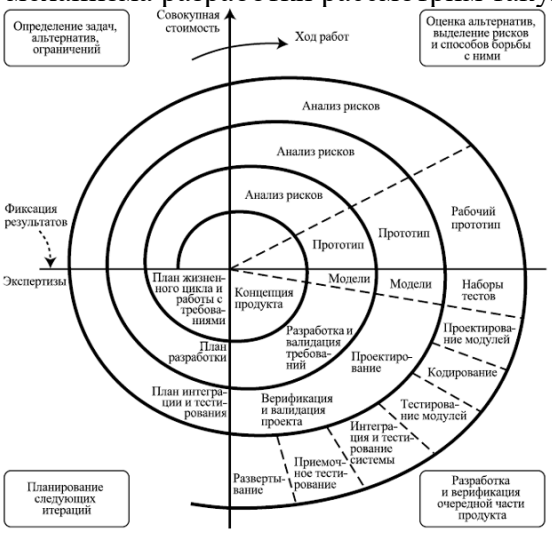
Спиральная модель разработки программного обеспечения не так широко известна, как, например, Scrum или Kanban. Причина в том, что данный подход может оказаться довольно затратным в применении. Именно поэтому он не очень хорошо подходит для небольших проектов. В спиральной модели особое внимание уделяется управлению рисками. На практике это означает, что фаза оценки и разрешения рисков является критичной для успеха проекта. Контроль рисков, в свою очередь, требует проведения специфического анализа на каждой итерации. Для регулярного обзора и анализа текущего состояния проекта необходимы дополнительные навыки и ресурсы.

На первый взгляд может показаться, что данная модель является сложной, неповоротливой и дорогостоящей и нет никаких веских причин для того, чтобы рассматривать ее как один из возможных вариантов. Но, как и любой другой подход к разработке программного обеспечения, спиральная модель имеет, помимо недостатков, также и свои сильные стороны. Например, она позволяет добавлять дополнительный функционал к программному обеспечению на самых поздних стадиях разработки. Поскольку постоянный контроль за рисками и, как следствие, регулярные экспертизы текущего состояния проекта, являются неотъемлемой частью данного подхода, общее видение проекта становится более ясным.

Обзор главных особенностей

Коротко спиральную модель можно описать как повторяющуюся последовательность циклов разработки с непрерывным контролем рисков.

Для лучшего понимая механизма разработки рассмотрим такую схему:



Как вы видите, спиральная модель состоит из четырех главных повторяющихся стадий. В ходе процесса разработки проект несколько раз проходит через все эти фазы. Каждая такая итерация называется спиралью.

Четыре главные фазы это:

1. Определение целей, альтернатив, ограничений, или фаза планирования. С этой стадии начинается работа над проектом. Команда разработчиков формулирует цели проекта, основные требования (такие как, например, Business Requirement Specifications, или BRS, System Requirement Specifications, или SRS), возможный дизайн и т.д. На последующих спиралях требования формируются согласно отзывам, полученным от заказчика. Именно поэтому постоянная коммуникация между заказчиком и командой крайне важна.

2. Анализ, определение и разрешение рисков является одной из самых значимых стадий разработки. В данном контексте, риски — это возможные события и состояния проекта, препятствующие достижению командой разработчиков поставленных целей. Существует довольно обширный диапазон возможных рисков, от тривиальных и легко преодолимых, до крайне серьезных. Главной задачей для команды разработчиков является выявление всех возможных рисков и присвоение им определенного уровня приоритета на основе их значимости. Следующим шагом является разработка возможных стратегий преодоления этих рисков. В итоге этих действий возможны изменения в последующих стадиях разработки. В качестве результата работы на этом этапе создается прототип.

3. Фаза разработки. На этом этапе происходит разработка и последующее тестирование продукта. Во время первой итерации, когда общие требования еще не так четко сформулированы, разрабатывается так называемый концепция будущего продукта (Proof Of Concept), которая необходима для получения отзыва заказчика. На последующих витках спирали рабочие версии продукта, или билды (builds), отправляются заказчику. Это позволяет получить более детальный отзыв и четче сформулировать требования.

4. Планирование следующей фазы. На этом этапе вся полученная информация используется для планирования дальнейших этапов разработки.

Спиральную модель часто называют мета-моделью, поскольку в ней используются два подхода: каскадная модель и модель прототипирования. Но крайне важно понимать, что спиральная модель не является простой последовательностью этапов разработки, следующих каскадной модели. На самом деле, спиральная модель является довольно гибкой. Стоит помнить, что схема, приведенная выше содержит определенные упрощения. Может показаться, что все стадии следуют одной спиральной последовательности. Но реальный жизненный цикл ПО более гибкий, чем это изображено на схеме. Существует даже возможность вернуться к предыдущим фазам в случае необходимости пересмотра принятых решений.

Давайте рассмотрим, как проходила разработка реальных проектов, чтобы понять, как эта модель может быть применена.

Спиральная модель на примере GanttPRO

В качестве примера применения на практике спиральной модели, рассмотрим GanttPRO — приложение для удобного управления проектами и задачами.

Достоинства:

- Мониторинг рисков является одной из главных особенностей, делающих данную модель особенно привлекательной в том случае, если вам предстоит управление большим, сложным и дорогостоящим проектом. Более того, проект будет более прозрачным, поскольку спиральная модель изначально была спроектирована таким образом, чтобы каждая итерация тщательно анализировалась;
- Заказчик может увидеть работающую версию продукта уже на ранних стадиях жизненного цикла ПО;
- Изменения могут быть внесены на поздних стадиях разработки;
- Проект может быть разделен на несколько частей и те из них, которые, согласно анализу, окажутся более рискованными, могут быть реализованы на ранних стадиях. Такой подход может снизить трудности, связанные с управлением проектом; Строгий контроль над документацией, как результат постоянного анализа рисков.

Недостатки:

– Мониторинг рисков требует дополнительных ресурсов, а значит, эта модель может оказаться весьма затратной. Каждая итерация требует отдельной экспертизы, что делает управление проектом сложнее. Именно поэтому спиральная модель плохо подходит для небольших проектов;

– Большое количество промежуточных стадий разработки. Как следствие — большой объем документации;

– На самых ранних стадиях дата завершения работы над проектом может быть неизвестна, что также усложняет контроль над процессом разработки

– Стоит понимать, что спиральную модель стоит применять в проектах такого типа, для которого она изначально была предназначена. Она может оказаться полезной, если вам предстоит работа над проектом со средним или высоким уровнем возможных рисков, заказчик не может предоставить достаточно четкий список требований к конечному продукту или эти требования достаточно сложные, а также в том случае, когда ожидаются значительные изменения в процессе разработки.

Задания:

1. Сделайте сравнительную таблицу преимущества / недостатки каскадной и спиралевидной моделей жизненного цикла.

Преимущества	Недостатки

2. Опишите отличия между каскадной и спиралевидной моделями жизненного цикла

3. Разработать свою идею ПО. Выполнить этапы «Планирование», «Формирование требование».

Контрольные вопросы:

1. Что такое жизненный цикл программного продукта?
2. Какие этапы входят в каскадную модель ЖЦПО?
3. Какие этапы входят в спиральную модель ЖЦПО?

Тема 24. Постановка задачи и разработка внешних спецификации

Цель: сформировать знания в области постановке задач при разработке программного обеспечения и разработке внешних спецификаций (требований).

Постановка задачи — это процесс формулировки назначения программного обеспечения и основных требований к нему. Описываются функциональные требования, определяющие функции, которые должно выполнять программное обеспечение, и эксплуатационные требования, определяющие характеристики его функционирования. Этан постановки задачи заканчивается разработкой технического задания с принятием основных проектных решений.

Техническое задание в соответствии со стандартом ГОСТ 19.201—78 «Техническое задание. Требования к содержанию и оформлению» имеет следующие основные разделы:

– введение: наименование и краткая характеристика программного обеспечения;

– основание для разработки;

– назначение разработки: описание функционального и эксплуатационного назначения, спецификации функций;

– требования к программному изделию: к функциональным характеристикам, к надежности, к техническим средствам;

– требования к программной документации;

– технологические требования.

Технологические требования определяют выбор следующих принципиальных решений, влияющих на процесс проектирования программного обеспечения:

– архитектура программного обеспечения;

– пользовательский интерфейс;

– метод программирования;

– язык программирования;

– среда программирования.

Архитектурой программного обеспечения называют описание создаваемого программного обеспечения на уровне его компонентов и связей между ними.

Архитектура программной системы во многом зависит от предметной области, для которой разрабатывается система. Поэтому часто архитектуры систем, разрабатываемых для одной и той же предметной области, имеют много общего. Следовательно, при проектировании архитектуры новой системы можно воспользоваться решениями, удачно примененными в ранее разработанных системах.

Развитие данного подхода привело к появлению архитектурных шаблонов (паттернов), на основе которых может создаваться архитектура конкретной системы. Архитектурный паттерн представляет собой описание типовой организации программной системы, опробованной в различных условиях и продемонстрировавшей свою эффективность.

Распространенными являются следующие архитектурные паттерны:

– модель-представление-контроллер {Model- View-Controller, MVC) применяется в системах, ориентированных на обслуживание клиентов;

– паттерн хранилища данных применяется в системах, функционирование которых связано с обработкой большого объема данных (информационные системы, системы управления); архитектура таких систем должна содержат компоненты, генерирующие данные, и компоненты, их обрабатывающие;

– паттерн «клиент-сервер» предусматривают распределение заданий между поставщиками и заказчиками услуг. Поставщики (серверы) предоставляют заказчикам (клиентам) определенный набор услуг (сервисов), доступ к которым осуществляется с помощью удаленного вызова соответствующих процедур;

– паттерны потоков данных предполагают построение архитектуры системы из функциональных модулей, которые получают входные данные п преобразуют их в выходные. Преобразования могут осуществляться как последовательно, так и параллельно;

– многоуровневая система представляет собой иерархическую систему, состоящую из нескольких уровней, каждый из которых выполняет определенные функции. Каждый уровень предоставляет услуги вышестоящему уровню и использует услуги нижестоящего уровня.

При процедурном программировании модель построения программы — иерархия множества подпрограмм, при объектно-ориентированном программировании — иерархия множества классов, совокупность обменивающихся сообщениями объектов классов. При этом простом виде архитектуры программа — это совокупность отдельно компилируемых программных единиц, используемая при решении задач.

Пакеты программ — это совокупность программ, решающих задачи некоторой прикладной области. Например, пакет математических программ, пакет графических программ.

Программные комплексы — это совокупность взаимосвязанных программ, совместно решающих небольшой класс задач некоторой прикладной области. Для решения задачи могут использоваться несколько программ комплекса, управляемые интерфейсом с пользователем, причем исходные данные и результаты желательно хранить в пределах одного проекта.

Программные системы — это совокупность подсистем программ, совместно решающих большой класс сложных задач некоторой прикладной области. Отличие от программных комплексов — взаимодействие программ системы через общие данные и наличие развитых пользовательских интерфейсов.

Пользовательский интерфейс представляет собой совокупность программных и аппаратных средств, обеспечивающих диалог пользователя и компьютера. Различают следующие типы пользовательских интерфейсов:

- примитивные интерфейсы;
- интерфейсы-меню;
- интерфейсы со свободной навигацией;
- интерфейсы прямого манипулирования.

К технологическим требованиям к программному обеспечению относятся:

- выбор метода программирования: процедурный или объектно-ориентированный;
- выбор языка программирования: C++, Java, Python и др.;
- выбор среды программирования: Visual Studio фирмы Microsoft, Embarcadero RAD Studio (включающая Delphi и C++ Builder), Eclipse и др.

Спецификациями называют полное н точное описание функций и ограничений разрабатываемого программного обеспечения. Существуют функциональные спецификации, описывающие функции разрабатываемого программного обеспечения, и эксплуатационные спецификации, описывающие требования к техническим средствам. Требование полноты означает строгое соответствие техническому заданию, а требование точности — однозначное толкование спецификаций разработчиком и заказчиком.

Спецификации содержат:

- декомпозицию и содержательную постановку задач;
- эксплуатационные ограничения;
- математические методы решения;
- модели программного обеспечения.

Модели программного обеспечения зависят от технологии программирования (процедурная или объектно-ориентированная).

При процедурном программировании используются следующие модели разрабатываемого программного обеспечения:

- функциональные диаграммы, отражающие взаимосвязи функций разрабатываемого программного обеспечения и ориентированные на иерархию функций, декомпозицию функций;
- диаграммы потоков данных, описывающие взаимодействие источников и потребителей информации и позволяющие специфицировать как функции, так и данные;
- диаграммы структур данных, описывающие базы данных разрабатываемого программного обеспечения;
- диаграммы переходов состояний, описывающие поведение разрабатываемого программного обеспечения при получении управляющих данных извне (например, команды пользователя).

При объектно-ориентированном программировании модели разрабатываемого программного обеспечения основаны на предметах реального мира. На этапе определения спецификаций требуется разработать модель предметной области программного обеспечения на базе иерархии классов (типов, определенных пользователем), объектной декомпозиции. Разрабатываемое программное обеспечение представляется в виде совокупности объектов (экземпляров классов), в результате взаимодействия которых через передачу сообщений происходит выполнение требуемых функций.

В настоящее время стандартным средством описания разрабатываемого программного обеспечения с использованием объектно- ориентированного подхода является фактически графический язык UML {Unified Modeling Language, универсальный язык моделирования), разработанный авторами Г. Буч, Д. Рамбо и И. Якобсоном в 1995 г.

Язык UML описывает модель сложной системы в виде специальных графических конструкций (диаграмм).

Диаграмма представляет собой связный граф, вершины которого представляются в виде геометрических фигур, связанных между собой ребрами (дугами).

Все диаграммы используют единую графическую нотацию.

В диаграммах используется три типа графических элементов, имеющих различное назначение:

– геометрические фигуры обозначают вершины графа. Их форма строго соответствует определенным элементам языка (класс, вариант использования, состояние, деятельность). Каждой вершине присваивается уникальное имя;

– различные линии (ребра), соединяющие геометрические фигуры (вершины графа) обозначают связи и взаимодействия элементов;

– специальные графические символы, располагаются около других элементов диаграммы и содержат дополнительную информацию. Дополнительный текст может размещаться также внутри контуров геометрических фигур.

Нотация языка UML определяет использование следующих основных видов диаграмм:

– вариантов использования (прецедентов) — описывают основные функции системы;

– классов — описывают классы, их характеристики (поля и операции) и связи между ними;

– кооперации — показывают взаимодействие объектов в процессе реализации варианта использования;

– пакетов — показывают связи наборов классов, объединенных в пакеты;

– состояний — демонстрируют состояния объекта и переходы из одного состояния в другое;

– деятельности — представляют собой схему потоков управления для решения некоторой задачи;

– компонентов — описывают компоненты программного обеспечения и их связи между собой;

– развертывания — позволяют связывать программные и аппаратные компоненты системы.

Каждая из этих диаграмм детализирует и конкретизирует различные представления о модели системы в терминах языка. Совокупность этих диаграмм содержит всю необходимую информацию для реализации проекта сложной программной системы. При создании моделей следует придерживаться следующих основных рекомендаций.

Любая модель должна содержать только те элементы, которые определены в нотации языка UML. Каждый элемент может быть использован только в соответствии с назначением и по правилам, определенным в языке.

Каждая диаграмма должна полностью описывать рассматриваемый фрагмент предметной области, на ней должны быть представлены все значимые элементы. Отсутствие каких-либо элементов может привести к серьезным ошибкам в моделировании.

Все элементы диаграммы должны принадлежать к одному уровню представления. При моделировании сложных систем часть используют иерархический подход, при котором диаграммы нижнего уровня уточняют и детализируют элементы диаграмм высших уровней.

Желательно информацию обо всех элементах диаграммы представлять в явном виде, не используя значения, заданные по умолчанию. Это ускорит и упростит процесс реализации модели.

Каждая модель конкретной программной системы может содержать все или только некоторые типы диаграмм.

Контрольные вопросы:

1. Что такое постановка задачи?
2. С помощью какого документа разрабатывается техническое задание?
3. Что содержит спецификация?

Тема 25. Виды контроля и основы доказательства правильности программ

Цель: сформировать знания о видах контроля, методах доказательства правильности программ.

Традиционные методы анализа ПО включают и методы доказательства правильности программ. Концепция, положенная в основу этого направления, была предложена в 1960-х гг. в работах П. Наура и Р. Флойда, в которых сформулирована идея приписывания точке программы так называемого индуктивного, или промежуточного, утверждения и указана возможность доказательства частичной правильности программы (т.е. соответствия друг другу ее предусловия и постусловия), построенного на установлении согласованности индуктивных утверждений.

Фундаментальный вклад в теорию верификации в середине 1970-х гг. внесли Ч. Хоор, который высказал идеи проведения доказательства частичной правильности программы в виде вывода в некоторой логической системе, и Э. Дейкстра, который ввел понятие «слабейшее предусловие», позволяющее одновременно как соответствие друг другу предусловия и постусловия, так и завершимость.

Методы доказательства правильности программ принесли определенную пользу программированию. Было отмечено, что эти методы указывают способ рассуждения о ходе выполнения программ, дают удобную систему комментирования программ и устанавливают взаимосвязи между конструкциями языков программирования и их семантикой. Если принять более широкое толкование термина «анализ программ», подразумевая доказательство разнообразных свойств программ или доказательство теорем о программах, то ценность методов анализа станет более ясной. В частности, можно исследовать характер изменения выходных значений программы, количество операций при выполнении программы, наличие зацикливаний, незадействованных участков программы.

Таким образом, в некоторых частных случаях методы верификации могут применяться и для доказуемого обнаружения программных дефектов.

Основные виды контроля:

- Статический
- Динамический
- Визуальный

Визуальный контроль – это проверка программы без использования компьютера. На первом этапе визуального контроля выполняется чтение программы, при этом особое внимание уделяется следующим элементам:

Комментарии и их соответствие к тексту программ.

Проверка всех условий в условных операторах и циклы.

Проверяется сложность логических выражений.

Возможность не завершения итерационных циклов.

Сквозной контроль программы – насколько корректно написана программа.

Статический контроль – проверка программы по ее тексту без выполнения с помощью инструментальных средств. Синтаксический контроль – выполняется компилятором. Проверка ошибок компилятора, необходимо проверять структурированность программ. Проверка соглашений, возможностей и Т. д. Контроль правдоподобия программы – выявление в тексте программы конструкций, в которых будучи синтаксически корректные могут содержать ошибку.

Основные неправдоподобные ситуации:

Использование в программе неинициализированных переменных.

Наличие в программе описание элементов, переменных, процедур, функций, меток, файлов в дальнейшем неиспользованных.

Наличие в тексте программы переменных, которые ни разу не использовались для чтения, после присвоения их значения.

Наличие в программе заведомо бесконечных циклов.

Верификация программ. Аналитическое доказательство их корректности. Здесь нужно отличать понятие надежность и корректность. Говоря о надежности обычно допускают небольшую вероятность ошибки. И поэтому оценивают вероятность появления таких ошибок. Надежность можно описано следующими характеристиками:

1. Целостность программного средства, т.е. способность его к защите от отказа.

2. «Живучесть» - способность входного контроля данных и их проверки в ходе работы.

3. Завершенность - это бездефектность готового программного средства.

4. Работоспособность – это способность программных средств к восстановлению после сбоев.

С учетом специфики появления ошибок в программе можно выделить 2 стороны понятия корректности.

Корректность, как точное соответствие цели разработки программы. При условии ее завершения.

Завершение программы, т.е. достижение программой конечной точки.

Динамический контроль – проверка правильности программы и ее выполнение на компьютере.

Контрольные вопросы:

1. Назовите формальные методы проверки правильности программ.
2. Назовите основные виды контроля.
3. Назовите цели процессов верификации и валидации программ.

Тема 26. Документирование и стандартизация программ

Цель: сформировать знания о процедурах и регдпменте документирования и стандартизации программ.

Программная и эксплуатационная документация может использоваться для изготовления и сопровождения программного изделия, для его тестирования (испытания), для его эксплуатации.

Документирование должно начинаться одновременно с разработкой продукта или даже раньше. В процессе разработки создаются следующие основные программные документы:

Текст программ — запись программы с необходимыми комментариями.

Описание программы — сведения о логической структуре и функционировании программы.

Программа и методика испытаний — требования, подлежащие проверке при испытании программы, а также порядок и методы их контроля.

Техническое задание — см. лекцию 2.

Пояснительная записка — схема алгоритма, общее описание алгоритма и/или функционирование программы, а также обоснование принятых технических и экономических решений.

Эксплуатационные документы:

Руководство пользователя — сведения о назначении программы, области применения, применяемых методах, ограничениях при применении, конфигурации технических средств; сведения для обеспечения процедуры общения пользователя с вычислительной системой в процессе выполнения программы. Создается на основе документов «Описание применения» и «Руководство оператора», описанных в ЕСПД.

Руководство системного администратора — сведения для обеспечения установки, функционирования и настройки программ на условиях конкретного применения. Создается на основе документа «Руководство системного программиста», описанного в ЕСПД.

Единая система программной документации

Комплекс государственных стандартов, устанавливающих взаимоувязанные правила разработки, оформления и обращения программ и программной документации.

В стандартах ЕСПД устанавливают требования, регламентирующие разработку, сопровождение, изготовление и эксплуатацию программ, что обеспечивает возможность:

– унификации программных изделий для взаимного обмена программами и применения ранее разработанных программ в новых проектах;

– снижения трудоемкости и повышения эффективности разработки, сопровождения, изготовления и эксплуатации программных изделий;

– автоматизации изготовления и хранения программной документации.

Основу отечественной нормативной базы в области документирования ПС составляет комплекс стандартов Единой системы программной документации (ЕСПД). Основная и большая часть комплекса ЕСПД была разработана в 70-е и

80-е гг. Сейчас этот комплекс представляет собой систему межгосударственных стандартов стран СНГ (ГОСТ), действующих на территории Российской

Федерации на основе межгосударственного соглашения по стандартизации.

Стандарты ЕСПД в основном охватывают ту часть документации, которая создается в процессе разработки ПС, и связаны, по большей части, с документированием функциональных характеристик ПС.

Ссылки на стандарты

ГОСТ 19.301-79 Единая система программной документации. Программа и методика испытаний. Требования к содержанию и оформлению

ГОСТ 19.005-85 Единая система программной документации. Р-схемы алгоритмов и программ. Обозначения условные графические и правила выполнения

ГОСТ 19.505-79 Единая система программной документации. Руководство оператора. Требования к содержанию и оформлению

ГОСТ 19.508-79 Единая система программной документации. Руководство по техническому обслуживанию. Требования к содержанию и оформлению

ГОСТ 19.504-79 Единая система программной документации. Руководство программиста. Требования к содержанию и оформлению

ГОСТ 19.503-79 Единая система программной документации. Руководство системного программиста. Требования к содержанию и оформлению

ГОСТ 19.202-78 Единая система программной документации. Спецификация. Требования к содержанию и оформлению

ГОСТ 24.104-85 Единая система стандартов автоматизированных систем управления. Автоматизированные системы управления. Общие требования

ГОСТ 28704-90 Единая система средств коммутационной техники. Термины и определения

ГОСТ 19.501-78 Единая система программной документации. Формуляр. Требования к содержанию и оформлению

ГОСТ 19.106-78 Единая система программной документации. Требования к программным документам, выполненным печатным способом

ГОСТ 19.201-78 Единая система программной документации. Техническое задание. Требования к содержанию и оформлению

ГОСТ 19.401-78 Единая система программной документации. Текст программы. Требования к содержанию и оформлению

ГОСТ 19.701-90 Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения

Процессы жизненного цикла программных средств

Основу стандарта составляют следующие базовые понятия

стадия — период в пределах ЖЦ ПП, который относится к описанию или реализации одного из конкретных состояний;

процесс — совокупность взаимосвязанных или взаимодействующих видов деятельности, преобразующих исходные данные о ПП либо его отдельных компонентах в выходные результаты;

деятельность — совокупность действий, используемых для получения конкретных выходных результатов;

задача — элементарное действие, предназначенное для достижения одного или более выходных результатов процесса, при выполнении которого можно назначить конкретного исполнителя и определить требуемые ресурсы. Задача формулируется в форме требования, рекомендации или допустимого действия, при этом используются глаголы: **«должен» —** для выражения условия, требующего проверки на соответствие чему-либо; **«следует» —** выражения рекомендации среди других возможностей; **«может» —** чтобы отразить направление допустимых действий;

модель жизненного цикла ПП — структура, состоящая из процессов, действий и задач, включающих разработку, эксплуатацию и сопровождение программного продукта, охватывающая период существования ПП — от установления требований к ПП до полного прекращения его использования.

Каждый процесс в стандарте описывается в следующем виде: наименование, цель, перечень действий, задачи. Последовательность действий или задач в каждом процессе нс является жесткой и определяется логическими связями между ними. Результатом выполнения процесса является готовый ПП либо его отдельные компоненты.

Стандарт содержит полный набор описания процессов ЖЦ ПП для некоторого типового проекта с максимально возможным составом процессов, действий и задач, которые используются при приобретении системы, содержащей программные средства, или отдельно поставляемого ПП; при оказании программной услуги; при поставке, разработке, эксплуатации и сопровождении ПП.

Задание:

Изучить стандарты ЕСПД заполнить таблицу

	ЕСПД	Название	Назначение (для чего и что описывает)
1.	ГОСТ 19.301-79		
2.	ГОСТ 19.005-85		
3.	ГОСТ 19.505-79		
4.	ГОСТ 19.508-79		
5.	ГОСТ 19.504-79		
6.	ГОСТ 19.503-79		
7.	ГОСТ 19.202-78		
8.	ГОСТ 24.104-85		
9.	ГОСТ 28704-90		
10.	ГОСТ 19.501-78		
11.	ГОСТ 19.106-78		
12.	ГОСТ 19.401-78		
13.	ГОСТ 19.701-90		

Контрольные вопросы:

1. Изучить предлагаемый стандарт СТ РК 34.019 – 2005
2. Перечислите основные процессы жизненного цикла. Кратко описать содержание каждого процесса.
3. Перечислите организационные процессы жизненного цикла

Тема 27. Тестирование основных процессов и подпрограмм.

Цель: изучить процессы тестирования и отладки программных продуктов.

Тестирование подпрограмм начинается с этапа контроля, основными разновидностями которого являются: визуальный, статический и динамический.

Визуальный контроль — это проверка текст«в "за сто»ом", без использования компьютера.

На первом этапе визуального контроля осуществляется чтение текста подпрограммы, причем особое внимание уделяется: комментариям и их соответствию тексту программы; условиям в операторах условного выбора и цикла; сложным логическим выражениям; возможности незавершения итерационных циклов.

Второй этап визуального контроля — сквозной контроль текста подпрограммы (его ручная прокрутка на нескольких заранее подобранных простых тестах). Распространенное мнение, что более выгодным является перекладывание большей части работы по контролю программных средств на компьютер, ошибочно. Основной довод в пользу этого таков: при работе на компьютере главным образом совершенствуются навыки в использовании клавиатуры, в то время как программистская квалификация приобретается, прежде всего, за столом.

Статический контроль — это проверка текста подпрограммы (без выполнения) с помощью инструментальных средств.

Первой, наиболее известной формой статического контроля является синтаксический контроль программы с помощью компилятора, при котором проверяется соответствие текста программы синтаксическим правилам языка программирования.

Сообщения компилятора обычно делятся на несколько групп в зависимости от уровня тяжести нарушения синтаксиса языка программирования:

1) информационные сообщения и предупреждения, при обнаружении которых компилятор, как правило, строит корректный объектный код и дальнейшая работа с программой (компоновка, выполнение) возможна (тем не менее сообщения этой группы должны тщательно анализироваться, так как их появление также может свидетельствовать об ошибке в программе — например, из-за неверного понимания синтаксиса языка);

2) сообщения об ошибках, при обнаружении которых компилятор пытается их исправить и строит объектный код, но его корректность маловероятна и дальнейшая работа с ним, скорее всего, невозможна;

3) сообщения о серьезных ошибках, при наличии которых построенный компилятором объектный код заведомо некорректен и его дальнейшее использование невозможно;

4) сообщения об ошибках, обнаружение которых привело к прекращению синтаксического контроля и построения объектного кода.

Однако практически любой компилятор пропускает некоторые виды синтаксических ошибок. Место обнаружения ошибки может находиться далеко по тексту подпрограммы от места истинной ошибки, а текст сообщения компилятора может не указывать на истинную причину ошибки. Одна синтаксическая ошибка может повлечь за собой генерацию компилятором нескольких сообщений об ошибках (например, ошибка в описании переменной приводит к появлению сообщения об ошибке в каждом операторе подпрограммы, использующем эту переменную).

Второй формой статического контроля может быть контроль структурированности текста подпрограммы, Т. е. проверка выполнения соглашений и ограничений структурного программирования. Примером подобной проверки может быть выявление в тексте подпрограммы ситуаций, когда цикл образуется с помощью оператора безусловного перехода (использования оператора GOTO для перехода вверх по тексту программы). Для проведения контроля структурированности могут быть созданы специальные инструментальные средства, а при их отсутствии эта форма статического контроля может совмещаться с визуальным контролем.

Третья форма статического контроля — контроль правдоподобия подпрограммы, Т. е. выявление в ее тексте конструкций, которые хотя и синтаксически корректны, но скорее всего содержат ошибку или свидетельствуют о ней. Основные неправдоподобные ситуации:

– использование в программе неинициализированных переменных (Т. е. переменных, не получивших начального значения);

– наличие в программе описаний элементов, переменных, процедур, меток, файлов, в дальнейшем не используемых в ее тексте;

– наличие в тексте подпрограммы фрагментов, никогда не выполняющихся;

– наличие в тексте программы переменных, ни разу не используемых для чтения после присваивания им значений;

– наличие в тексте подпрограммы заведомо бесконечных циклов.

Даже если присутствие в тексте программы неправдоподобных конструкций не приводит к ее неправильной работе, исправление этого фрагмента повысит ясность и эффективность программы, Т. е. благотворно скажется на ее качестве.

Для возможности проведения контроля правдоподобия в полном объеме также должны быть созданы специальные инструментальные средства, хотя ряд возможностей по контролю правдоподобия имеется в существующих отладочных и обычных компиляторах.

Следует отметить, что создание инструментальных средств контроля структурированности и правдоподобия программ может быть упрощено существенно при применении следующих принципов:

1) проведение дополнительных форм статического контроля после завершения компиляции и только для синтаксически корректных программ;

2) максимальное использование результатов компиляции и линковки программы и, в частности, информации, включаемой в листинг компилятора и линкера;

3) вместо полного синтаксического разбора текста проверяемой программы необходимо построение для нее списка идентификаторов и списка операторов с указанием всех их необходимых признаков.

При отсутствии инструментальных средств контроля правдоподобия эта фаза статического контроля также может объединяться с визуальным контролем.

Четвертой формой статического контроля программ является их верификация, Т. е. аналитическое доказательство их корректности.

Несмотря на достаточную сложность процесса верификации программы и на то, что даже успешно завершенная верификация не дает гарантий качества программы (так как ошибка может содержаться и в верификации), применение методов аналитического доказательства правильности очень полезно для уточнения смысла разрабатываемой программы, а знание этих методов благотворно сказывается на квалификации программиста.

Динамический контроль программы — это проверка правильности программы при ее выполнении на компьютере, Т. е. тестирование. Минимальное автономное тестирование подпрограммы должно обеспечивать прохождение всех путей вычислений.

Проектная процедура тестирования подпрограммы заключается в следующем:

— по внешним спецификациям модуля подготовьте тесты для каждой ситуации и каждого недопустимого условия;

— просмотрите текст подпрограммы, чтобы убедиться, что все условные переходы будут выполняться в каждом направлении; при необходимости добавьте тесты;

— убедитесь по тексту подпрограммы, что тесты охватывают достаточно много путей; для циклов должны быть тесты без повторения, с одним повторением и с несколькими повторениями;

— проверьте по тексту подпрограммы ее чувствительность к особым значениям данных (наиболее опасные числа — это ноль и единица), в случае необходимости добавьте тесты.

Отладка – это процесс локализации и исправления ошибок, обнаруженных при тестировании программного обеспечения.

Локализацией называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо определить ее причину, т.е. определить оператор или фрагмент, содержащие ошибку. Причины ошибок могут быть как очевидны, так и очень глубоко скрыты. В целом сложность отладки обусловлена следующими причинами:

– требует от программиста глубоких знаний специфики управления используемыми техническими средствами, операционной системы, среды и языка программирования, реализуемых процессов, природы и специфики различных ошибок, методик отладки и соответствующих программных средств;

– психологически дискомфортна, так как необходимо искать собственные ошибки и, как правило, в условиях ограниченного времени;

– возможно взаимовлияние ошибок в разных частях программы, например, за счет затирания области памяти одного модуля другим из-за ошибок адресации;

– отсутствуют четко сформулированные методики отладки.

Классификация ошибок

В соответствии с этапом обработки, на котором появляются ошибки, различают:

– синтаксические ошибки – ошибки, фиксируемые компилятором (транслятором, интерпретатором) при выполнении синтаксического и частично семантического анализа программы;

– ошибки компоновки – ошибки, обнаруженные компоновщиком (редактором связей) при объединении модулей программы;

– ошибки выполнения – ошибки, обнаруженные операционной системой, аппаратными средствами или пользователем при выполнении программы.

Методы отладки программного обеспечения

Отладка программы в любом случае предполагает обдумывание и логическое осмысление всей имеющейся информации об ошибке. Большинство ошибок можно обнаружить по косвенным признакам посредством тщательного анализа текстов программ и результатов тестирования без получения дополнительной информации. При этом используют различные методы:

– ручного тестирования;

– индукции;

– дедукции;

– обратного прослеживания.

Метод ручного тестирования

Это – самый простой и естественный способ данной группы. При обнаружении ошибки необходимо выполнить тестируемую программу вручную, используя тестовый набор, при работе с которыми была обнаружена ошибка. Метод очень эффективен, но не применим для больших программ, программ со сложными вычислениями и в тех случаях, когда ошибка связана с неверным представлением программиста о выполнении некоторых операций. Данный метод часто используют как составную часть других методов отладки.

Общая методика отладки программных продуктов, написанных для выполнения в операционных системах MS DOS и Win32:

1 этап – изучение проявления ошибки;

2 этап – определение локализации ошибки;

3 этап – определение причины ошибки;

4 этап – исправление ошибки;

5 этап – повторное тестирование.

Процесс отладки можно существенно упростить, если следовать основным рекомендациям структурного подхода к программированию:

–программу наращивать «сверху-вниз», от интерфейса к обрабатывающим подпрограммам, тестируя ее по ходу добавления подпрограмм;

–выводить пользователю вводимые им данные для контроля и проверять их на допустимость сразу после ввода;

–предусматривать вывод основных данных во всех узловых точках алгоритма (ветвлениях, вызовах подпрограмм).

Спецификация программы, программная спецификация (program specificatio–) - точная и полная формулировка определенной задачи или группы задач, содержащая сведения, необходимые для построения алгоритма их решения. Содержит описание результата, который должен быть достигнут с помощью конкретной программы, а также действий, выполняемых программой для достижения конечного результата без упоминания того, как указанный результат достигается

Контрольные вопросы

1. В чем заключается ручная отладка ПО?
2. На каком этапе проводится ручная отладка?
3. Опишите методы отладки.
4. В чем состоит суть тестирования?

5. Методы тестирования «белого ящика».
6. Как определить количество тестов?
7. Какова эффективность тестирования логических выражений разными типами тестов?

Тема 28. Основы UML

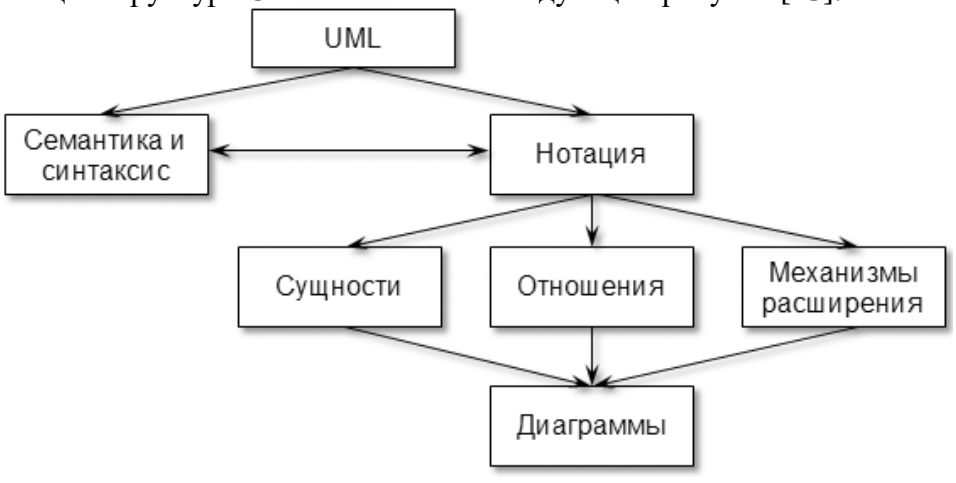
Цель: сформировать зания об основх унифицированного языка моделирования (UML), его семантике и синтаксисе.

Унифицированный язык моделирования (UML) в настоящий момент является стандартом де-факто при описании (документирования) результатов проектирования и разработки объектно-ориентированных систем. Начало разработки UML было положено в 1994 г. Гради Бучем и Джеймсом Рамбо, работавшим в компании Rational Software. Осенью 1995 Г. к ним присоединился Ивар Якобсон и в октябре того же года была выпущена предварительная версия 0.8 унифицированного метода (англ. Unified Method). С этого времени было выпущено несколько версий спецификации UML, две из которых носят статус международного стандарта:

- UML 1.4.2 – ISO/IEC 19501:200«. "Информационные технологии. Открытая распределительная обработка. Унифицированный язык моделирования (UML). Версия 1.».2" (анг«. "Information technology. Open distributed processing. Unified modeling language (UML). Version 1.».2");
- UML 2.4.1 – ISO/IEC 19505-1:201«. "Информационные технологии. Унифицированный язык моделирования группы по управлению объектами (OMG UML). Часть 1. Инфраструкт»ра" (анг«. "Information technolo– -- Object Management Group Unified Modeling Language (OMG UM–) - Part 1: Infrastruct»re") и ISO/IEC 19505-2:201«. "Информационные технологии. Унифицированный язык моделирования группы по управлению объектами (OMG UML). Часть 2. Сверхструкт»ра" (анг«. "Information technolo– -- Object Management Group Unified Modeling Language (OMG UM–) - Part 2: Superstruct»re").

Последнюю официальную спецификацию языка можно найти на сайте www.omg.org.

Общая структура UML показана на следующем рисунке [25].



Семантика и синтаксис UML

Семантика – раздел языкознания, изучающий значение единиц языка, прежде всего его слов и словосочетаний [35].

Синтаксис – способы соединения слов и их форм в словосочетания и предложения, соединения предложений в сложные предложения, способы создания высказываний как части текста [35].

Таким образом, применительно к UML, семантика и синтаксис определяют стиль изложения (построения моделей), который объединяет естественный и формальный языки для представления базовых понятий (элементов модели) и механизмов их расширения.

Нотация UML

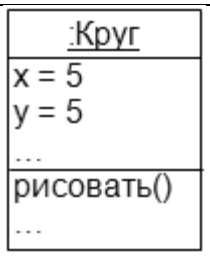
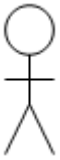
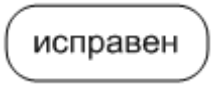
Нотация представляет собой графическую интерпретацию семантики для ее визуального представления.

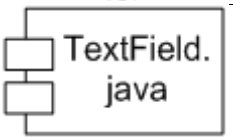
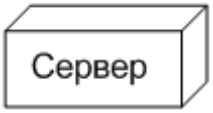
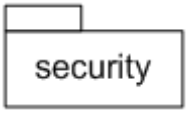
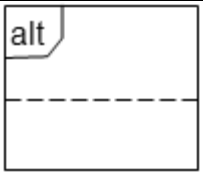
В UML определено три типа сущностей:

- структурная – абстракция, являющаяся отражением концептуального или физического объекта;
- группирующая – элемент, используемый для некоторого смыслового объединения элементов диаграммы;
- поясняющая (аннотационная) – комментарий к элементу диаграммы.

В следующей таблице приведено краткое описание основных сущностей, используемых в графической нотации, и основные способы их отображения.

Таблица 11.1. Сущности

Тип	Наименование	Обозначение	Определение (семантика)
Структурная	Класс (class)		Множество объектов, имеющих общую структуру и поведение
	Объект (object)		Абстракция реальной или воображаемой сущности с четко выраженными концептуальными границами, индивидуальностью (идентичностью), состоянием и поведением. С точки зрения UML объекты являются экземплярами класса (экземплярами сущности)
	Актер (actor)		Внешняя по отношению к системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. Таким образом актер – это внешний источник или приемник информации
	Вариант использования (use case)		Описание выполняемых системой действий, которая приводит к значимому для актера результату
	Состояние (state)		Описание момента в ходе жизни сущности, когда она удовлетворяет некоторому условию, выполняет некоторую деятельность или ждет наступления некоторого события

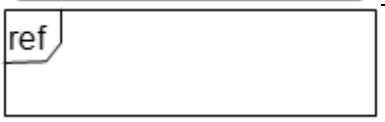
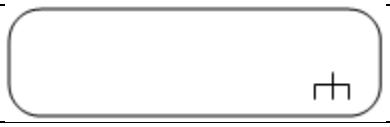
	Кооперация (collaboration)		Описание совокупности экземпляров актеров, объектов и их взаимодействия в процессе решения некоторой задачи
	Компонент (component)		Физическая часть системы (файл), в том числе модули системы, обеспечивающие реализацию согласованного набора интерфейсов
	Интерфейс (interface)		Совокупность операций, определяющая сервис (набор услуг), предоставляемый классом или компонентом
	Узел (node)		Физическая часть системы (компьютер, принтер и Т. д.), предоставляющая ресурсы для решения задачи
Группирующая	Пакет (package)		Общий механизм группировки элементов. В отличие от компонента, пакет – чисто концептуальное (абстрактное) понятие. Частными случаями пакета являются система и модель
	Фрагмент (fragment)		Область специфического взаимодействия экземпляров актеров и объектов. Любая диаграмма в UML также является фрагментом – фрагментом (частью) проекта.
	Раздел деятельности (activity partition)		Группа операций (зона ответственности), выполняемых одной сущностью (актером, объектом, компонентом, узлом и т.д.)
	Прерываемый регион (interruptible activity region)		Группа операций, обычная последовательность выполнения которых может прервана в результате наступления нестандартной ситуации
Поясняющая	Примечание (comment)		Комментарий к элементу. Присоединяется к комментируемому элементу штриховой линией

В некоторых источниках, в частности [24, 29], выделяют также поведенческие сущности взаимодействия и конечные автоматы, но с логической точки зрения их следует отнести к диаграммам.

Некоторые из приведенных выше сущностей в соответствии с прицинами иерархического упорядочивания и декомпозиции подразумевают их подробное описание

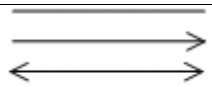
на диаграммах декомпозиции. На диаграмме верхнего уровня они помечаются особым значком или меткой.

Таблица 11.2. Декомпозируемые сущности

Наименование	Обозначение	Диаграмма
Скрытое составное состояние		Диаграмма автоматов
Скрытый фрагмент взаимодействия		Диаграмма последовательности
Деятельность		Диаграмма деятельности

В следующей таблице приведено описание всех видов отношений UML, используемых на диаграммах для указания связей между сущностями.

Таблица. Отношения

Наименование	Обозначение	Определение (семантика)
Ассоциация (association)		Отношение, описывающее значимую связь между двумя и более сущностями. Наиболее общий вид отношения
Агрегация (aggregation)		Подвид ассоциации, описывающей свя«ь "ча»т«"– "це»ое", в котор«м "ча»ть" может существовать отдельно «т "цел»го". Ромб указывается со сторо«ы "цел»го". Отношение указывается только между сущностями одного типа
Композиция (composition)		Подвид агрегации, в котор«й "ча»ти" не могут существовать отдельно «т "цел»го". Как правил«, "ча»ти" создаются и уничтожаются одновременно«с "це»ым"
Зависимость (dependency)		Отношение между двумя сущностями, в котором изменение в одной сущности (независимой) может влиять на состояние или поведение другой сущности (зависимой). Со стороны стрелки указывается независимая сущность
Обобщение (generalization)		Отношение между обобщенной сущностью (предком, родителем) и специализированной сущностью (потомком, дочкой). Треугольник указывается со стороны родителя. Отношение указывается только между сущностями одного типа
Реализация (realization)		Отношение между сущностями, где одна сущность определяет действие, которое другая сущность обязуется выполнить. Отношения используются в двух случаях: между интерфейсами и классами (или компонентами), между вариантами использования и кооперациями. Со стороны стрелки указывается сущность, определяющее действие (интерфейс или вариант использования)

Для ассоциации, агрегации и композиции может указываться кратность (англ. multiplicity), характеризующая общее количество экземпляров сущностей, участвующих в отношении. Она, как правило, указывается с каждой стороны отношения около соответствующей сущности. Кратность может указываться следующими способами:

- * – любое количество экземпляров, в том числе и ни одного;
- целое неотрицательное число – кратность строго фиксирована и равна указанному числу (например: 1, 2 или 5);
- диапазон целых неотрицательных чис«л "первое число .. второе чи»ло" (например: 1..5, 2..10 или 0..5);
- диапазон чисел от конкретного начального значения до произвольного конечно«о "первое число .» *" (например: 1..*, 5..* или 0..*);
- перечисление целых неотрицательных чисел и диапазонов через запятую (например: 1, 3..5, 10, 15..*).

Если кратность не указана, то принимается ее значение, равное 1. Кратность экземпляров сущностей, участвующих в зависимости, обобщении и реализации, всегда принимается равной 1.

В следующей таблице приведено описание механизмов расширения, применяемых для уточнения семантики сущностей и отношений. В общем случае, механизм расширения представляет собой строку текста, заключенную в скобки или кавычки.

Таблица . Механизмы расширения

Наименование	Обозначение	Определение (семантика)
Стереотип (stereotype)	« »	Обозначение, уточняющее семантику элемента нотации (например: зависимость со стереотипом «include» рассматривается, как отношение включения, а класс со стереотипом «boundary» – граничный класс)
Сторожевое условие (guard condition)	[ ]	Логическое условие (например: [A > B] или [идентификация выполнена])
Ограничение (constraint)	{ }	Правило, ограничивающее семантику элемента модели (например, {время выполнения менее 10 мс})
Помеченное значение (tagged value)	{ }	Новое или уточняющее свойство элемента нотации (например: {version = 3.2})

Помимо стереотипов, указываемых в виде строки текста в кавычках, на диаграммах могут использоваться графические стереотипы. На следующем рисунке приведены примеры стандартного и стереотипного отображения класса-сущности.

а)стандартное обозначение

б)стандартное обозначение с текстовым стереотипом

в)графический стереотип

Задание: Конспект. Ответить на вопросы. Найти в интернете и привести примеры приложений для создания UML диаграмм

Контрольные вопросы

1. Назовите типы сущностей и дайте их краткую характеристику.
2. Перечислите структурные сущности.
3. Перечислите виды отношений.
4. Дайте определение понят«ю "сторожевое услов»е".

Тема 29. Диаграмма вариантов использования (UseCase diagram)

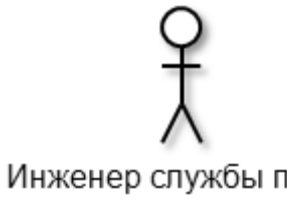
Цель: изучить назначение и состав диаграммы вариантов использования, правила разработки диаграммы.

Диаграмма вариантов использования (сценариев поведения, прецедентов) является исходным концептуальным представлением системы в процессе ее проектирования и разработки. Данная диаграмма состоит из актеров, вариантов использования и отношений между ними. При построении диаграммы могут использоваться также общие элементы нотации: примечания и механизмы расширения.

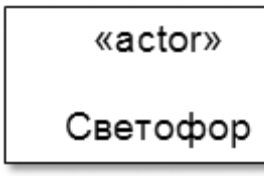
Суть данной диаграммы состоит в следующем [28]: проектируемая система представляется в виде множества актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (действующим лицом, актантом, актором) называется любой объект, субъект или система, взаимодействующая с моделируемой системой извне. В свою очередь вариант использования – это спецификация сервисов (функций), которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемых системой при взаимодействии с актером. При этом в модели никак не отражается то, каким образом будет реализован этот набор действий.

В структурном подходе аналогом диаграммы вариантов использования являются диаграммы IDEF0 и DFD, вариантов использования – работы (IDEF0) и процессы (DFD), актеров – внешние сущности (DFD).

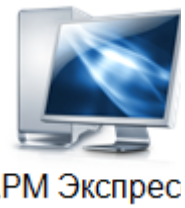
Согласно UML актера графически можно отобразить тремя способами.



а) «проволочный человечек»



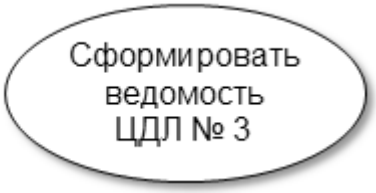
б) класс с текстовым стереотипом «actor»



в) произвольная иконка

Первый способ отображения в виде «проволочного человечка» является самым распространенным.

Вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его описание, обозначающее выполнение какой-либо операции или действия.



Вариант использования, который инициализируется по запросу пользователя, представляет собой законченную последовательность действий. Это означает, что после того, как система закончит обработку запроса актера, она должна возвратиться в состояние, в котором готова к выполнению следующих запросов.

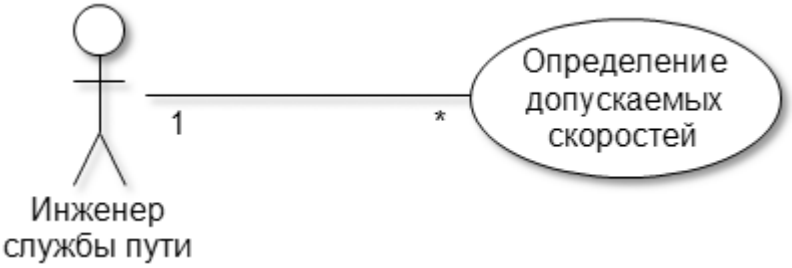
Варианты использования могут включать в себя описание особенностей способов реализации сервиса и различных исключительных ситуаций, таких как корректная обработка ошибок системы.

Примечания предназначены для включения в диаграмму произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемой системы. В качестве такой информации могут быть комментарии разработчика и ограничения. Графически примечания отображаются прямоугольником с загнутым верхним правым уголком, внутри которого содержится текст примечания. Линия, соединяющая примечание и элемент диаграммы, называется якорем (фиксацией).

Связи между актерами и вариантами отображаются с использованием отношений четырех видов:

- ассоциаций;
- обобщения;
- включения (зависимость со стереотипом «include»);
- расширения (зависимость со стереотипом «extend»).

Применительно к рассматриваемой диаграмме отношение ассоциации служит для обозначения взаимодействия актера с вариантом использования.



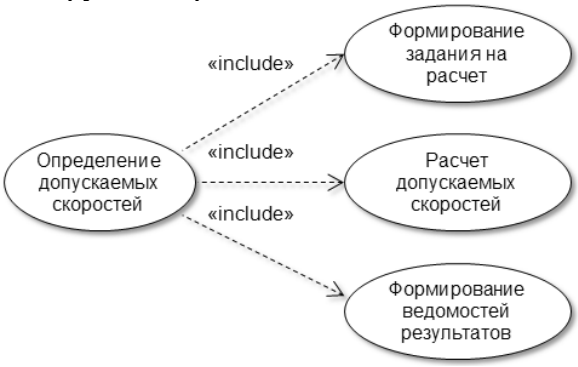
Ассоциация может отображаться в виде однонаправленной или двунаправленной стрелки, показывающей направление потоков информации или управляющих сигналов.

Отношение обобщения служит для указания того факта, что некоторая сущность A может быть обобщена до сущности B. В этом случае сущность A будет являться специализацией сущности B. На диаграмме данный вид отношения можно отображать только между однотипными сущностями (между двумя вариантами использования или двумя актерами).

Графически данное отношение обозначается сплошной линией со стрелкой, в виде незакрашенного треугольника, от потомка к родителю.

Отношения включения и расширения являются частным случаем отношения зависимости и могут иметь место только между двумя вариантами использования. Они отображаются штриховой стрелкой с указанием стереотипа.

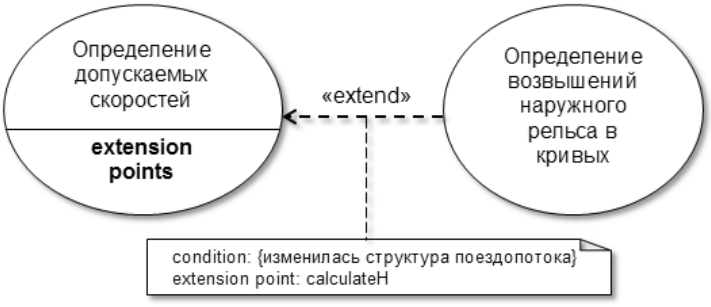
Отношение включения указывает, что некоторое заданное поведение одного варианта использования обязательно включается в качестве составного компонента в последовательность поведения другого варианта использования.



Стрелка включения должна быть направлена от базового (составного) варианта к включаемому и помечена стереотипом «include» (англ. включает) или «uses» (англ. использует).

В отличие от отношения включения, отношение расширения определяет потенциальную возможность включения поведения одного варианта использования в состав другого. Т. е. дочерний вариант использования может как вызываться, так и не вызываться родительским.

Стрелка расширения должна быть направлена от включаемого варианта к базовому и помечена стереотипом «extend» (англ. расширяет).



Ввиду того, что допускаемая скорость в кривых участках пути зависит в том числе и от возвышения наружного рельса, перед определением допускаемых скоростей может потребоваться определение и установление новых возвышений, которые в свою очередь зависят от структуры пропускаемого поездопотока.

Варианты использования, которые расширяют базовый, подключаются к нему (активируются при его выполнении) через так называемые точки расширения (англ.

extension points). Каждая точка расширения маркируется меткой (на рис. 12–8 - calculateH) и условием (англ. condition) активации. Обычно перечень точек расширения указывается в базовом варианте использования ниже горизонтальной линии.

Правила и рекомендации по разработке диаграмм вариантов использования

Вследствие того, что диаграммы вариантов использования являются аналогом диаграмм IDEF0 и DFD, методологии их разработки во многом совпадают.

1. Рекомендуется вначале построить контекстную диаграмму, на которой отображаются основные варианты использования (функции) системы, а затем для каждого из них построить диаграммы декомпозиции (детализации).

2. Контекстная диаграмма может представлять собой несвязный граф (в отличие от IDEF0 и DFD).

3. Чрезмерная детализация вариантов использования не требуется. Следует помнить, что вариант использования – это относительно крупный блок функциональности системы. Для детализации в дальнейшем будут использоваться другие виды диаграмм, более подходящие для этой цели.

4. Для лучшего восприятия отдельная диаграмма (контекстная или декомпозиции) не должна быть перенасыщена элементами. Рекомендуется отображать на диаграмме не более 15 вариантов использования.

5. Располагать элементы следует так, чтобы была видна логическая последовательность выполнения вариантов использования и было минимум пересечений между отношениями.

6. Перед построением диаграммы необходимо задокументировать потоки событий в системе. Поток событий – это процесс обработки данных, реализуемый в рамках одного или нескольких вариантов использования. Описание потока включает информацию о том, какие обязанности возлагаются на актеров, а какие – на систему. Оно содержит:

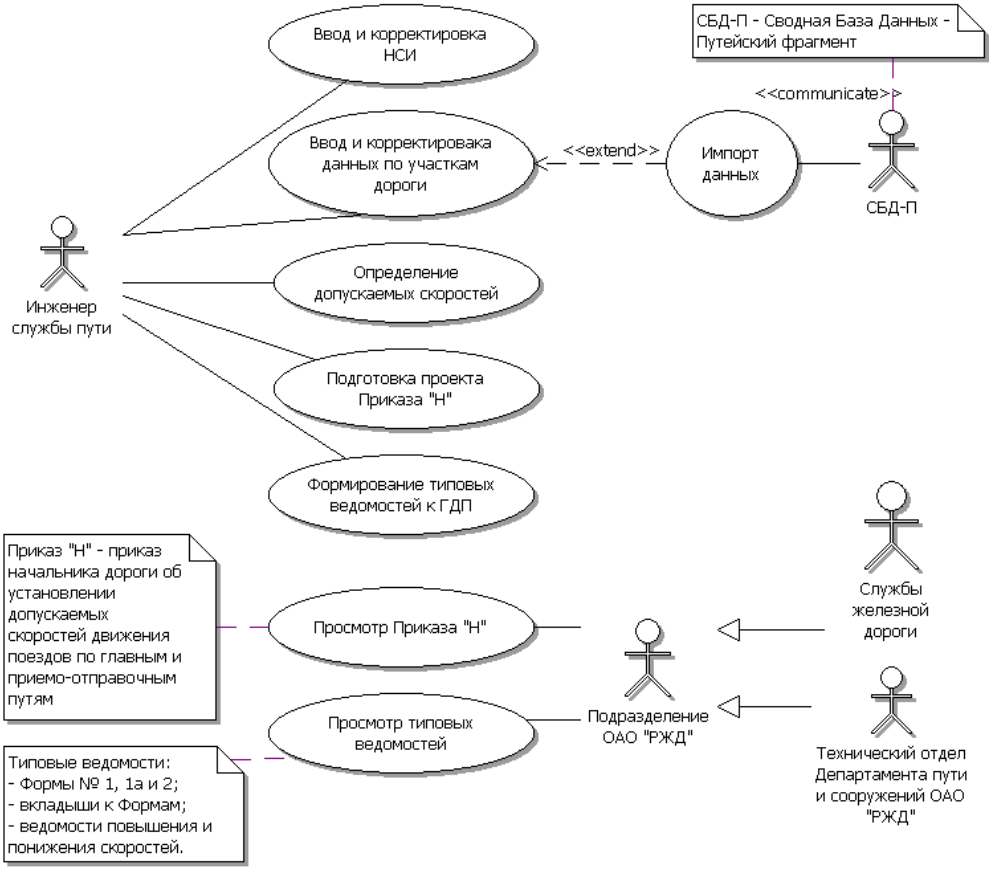
- краткое описание поведения, реализуемого в варианте использования;
- предусловия – условия, которые должны быть соблюдены, прежде чем вариант использования может быть задействован. Например, таким условием может быть завершение выполнения другого варианта использования или наличие у пользователя прав доступа;
- основной поток событий описывает, что должно происходить во время выполнения варианта использования в наиболее распространенном (типовом) случае. В этом случае дочерние варианты использования связаны с базовым отношением включения;
- альтернативные потоки событий описывают исключительные ситуации (например, ввод неправильного пароля или необходимость выполнения дополнительных действий). Дочерние варианты использования при разработке диаграммы связываются с базовым отношением расширения;
- постусловия – условия, которые должны быть выполнены после завершения варианта использования. Например, таким условием может быть обязательное сохранение результатов расчета в базе данных на сервере.

7. На диаграммах не следует отображать особенности реализации вариантов использования и внутренней организации системы, связанные со спецификой используемых программных и аппаратных средств. Данные диаграммы в первую очередь предназначены для совместного с заказчиком определения функциональных требований к системе. Поэтому понимать (интерпретировать) отображенное на диаграммах и заказчик и разработчик должны одинаково.

Примеры построения диаграмм вариантов использования

В качестве примера в этой и последующих разделах будет использоваться проект системы ИСКРА-ПУТЬ, применяемой в службах пути всех железных дорог России. Описание системы приводится в теме № 6. На следующем рисунке показана контекстная

диаграмма вариантов использования, разработанная с помощью Case-средства Borland Together Architect 2006 for Eclipse.



Как видно, на следующем рисунке контекстная диаграмма представляет собой несвязный граф. Ввиду того, что с системой предусматривается работа разных категорий пользователей (актеров), каждая из них задействуют только часть функциональных возможностей.

На рис отображена диаграмма декомпозиции для варианта использования «Определение допускаемых скоростей».

Диаграммы декомпозиции, как правило, представляют собой «ромашку», в центре которой декомпозируемый вариант использования, а вокруг – входящие в него обязательные (include) или расширяющие (extend) составные части.

Задание: Опишите свой пример диаграммы использования вариантов (UseCase diagram).

Контрольные вопросы:

1. Для чего используются диаграммы вариантов использования?
2. Какие существуют отношения между элементами диаграммы?
3. Назовите основные элементы диаграммы.

Тема 30. Диаграммы классов (class diagram)

Цель: сформировать знания о назначении и составе диаграммы классов, правилах ее построения.

Класс — это план, который используется для создания объекта. Класс определяет, что может делать объект.

Что такое диаграмма классов?

UML CLASS DIAGRAM дает обзор программной системы путем отображения классов, атрибутов, операций и их взаимосвязей. Эта диаграмма включает в себя имя класса, атрибуты и операции в отдельных назначенных отсеках.

Диаграмма классов определяет типы объектов в системе и различные типы отношений, которые существуют между ними. Это дает общее представление о приложении. Этот метод моделирования может работать практически со всеми объектно-ориентированными методами. Класс может ссылаться на другой класс. Класс может иметь свои объекты или наследовать от других классов.

Диаграмма классов помогает построить код для разработки программного приложения.

Преимущества диаграммы классов

Диаграмма классов Иллюстрирует модели данных даже для очень сложных информационных систем

Это обеспечивает обзор того, как приложение структурировано перед изучением фактического кода. Это может легко сократить время обслуживания

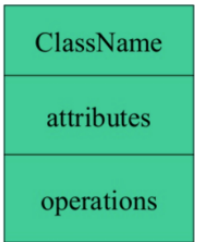
Это помогает лучше понять общие схемы приложения.

Позволяет рисовать подробные диаграммы, которые выделяют код, необходимый для программирования

Полезно для разработчиков и других заинтересованных сторон.
Основные элементы диаграммы классов UML
Основные элементы диаграммы классов UML:

– имя класса
– атрибуты
– операции

Имя класса

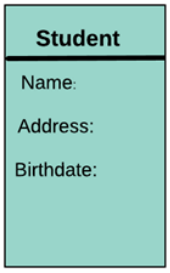


Имя класса требуется только в графическом представлении класса. Появляется в самом верхнем отсеке. Класс — это план объекта, который может иметь одинаковые отношения, атрибуты, операции и семантику. Класс отображается в виде прямоугольника, включая его имя, атрибуты и операции в отдельных отсеках.

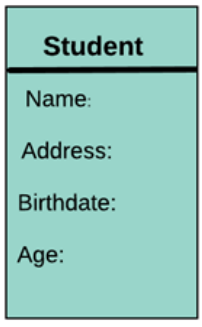
При представлении класса необходимо соблюдать следующие правила:
Имя класса всегда должно начинаться с заглавной буквы.
Название класса всегда должно быть в центре первого отсека.
Имя класса всегда должно быть написано жирным шрифтом .
Имя абстрактного класса должно быть написано курсивом.

Атрибуты:

Атрибут именуется свойством класса, который описывает моделируемый объект. На диаграмме классов этот компонент расположен чуть ниже отсека имени.



Производный атрибут вычисляется из других атрибутов. Например, возраст учащегося можно легко вычислить по дате его рождения.



Атрибуты характеристики
Атрибуты обычно записываются вместе с фактором видимости.

Публичная, частная, защищенная и пакетная — это четыре видимости, которые обозначаются знаками +, -, # или ~ соответственно.

Видимость описывает доступность атрибута класса.

Атрибуты должны иметь осмысленное имя, которое описывает его использование в классе.

Отношения

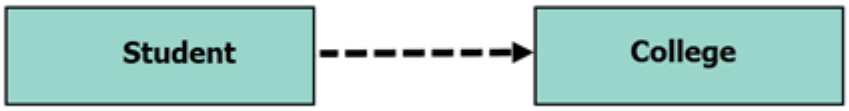
В UML есть в основном три вида отношений:

– зависимости
– обобщения
– ассоциации

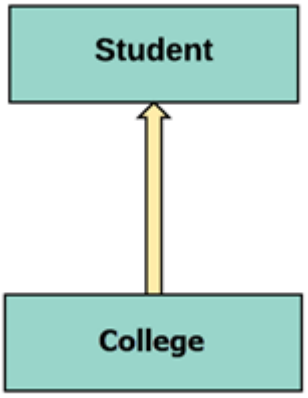
Зависимость

Зависимость означает отношение между двумя или более классами, в котором изменение одного может вызвать изменения другого. Тем не менее, это всегда будет создавать более слабые отношения. Зависимость указывает, что один класс зависит от другого.

В следующем примере студент имеет зависимость от колледжа



Обобщение:



Обобщение помогает связать подкласс с его суперклассом. Подкласс наследуется от своего суперкласса. Отношение обобщения нельзя использовать для моделирования реализации интерфейса. Диаграмма классов позволяет наследовать от нескольких суперклассов.

В этом примере класс Student обобщается из класса Person.

Ассоциация:

Этот тип отношений представляет статические отношения между классами A и B. Например; сотрудник работает на организацию.

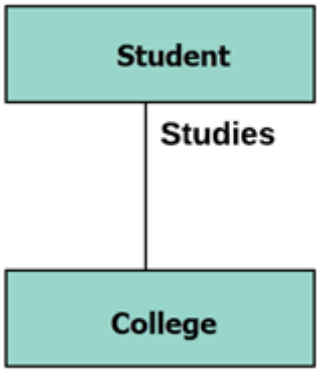
Вот несколько правил для Ассоциации:

Ассоциация — это в основном глагол или глагольная фраза, или существительное, или именная фраза.

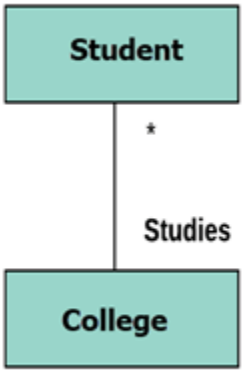
Он должен быть назван так, чтобы указывать роль, которую играет класс, присоединенный в конце пути ассоциации.

Обязательно для рефлексивных ассоциаций

В этом примере показана связь между студентом и колледжем, который является учебой.



множественность

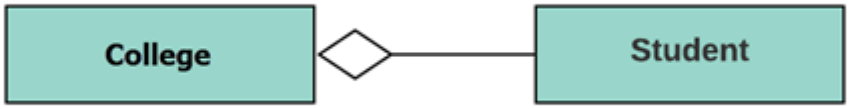


Кратность — это фактор, связанный с атрибутом. Он указывает, сколько экземпляров атрибутов создается при инициализации класса. Если кратность не указана, по умолчанию она считается кратностью по умолчанию.

Допустим, что в одном колледже 100 студентов. Колледж может иметь несколько студентов.

агрегирование

Агрегация — это особый тип ассоциации, который моделирует отношение всей части между агрегатом и его частями.



Например, класс колледжа состоит из одного или нескольких студентов. В совокупности содержащиеся классы никогда полностью не зависят от жизненного цикла контейнера. Здесь класс колледжа останется, даже если ученик недоступен.

Сочинение:

Композиция — это особый тип агрегации, который обозначает сильную собственность между двумя классами, когда один класс является частью другого класса.

Например, если колледж состоит из классов ученика. Колледж может содержать много студентов, в то время как каждый студент принадлежит только одному колледжу. Так что, если колледж не функционирует, все студенты также удаляются.

агрегирование	сочинение
Агрегация указывает на отношение, в котором ребенок может существовать отдельно от своего родительского класса. Пример: автомобиль (родитель) и автомобиль (ребенок). Так что, если вы удалите автомобиль, автомобиль ребенка все еще существует.	Композиция отображает отношения, где ребенок никогда не будет существовать независимо от родителя. Пример: Дом (родитель) и Комната (ребенок). Комнаты никогда не разделятся на дом.

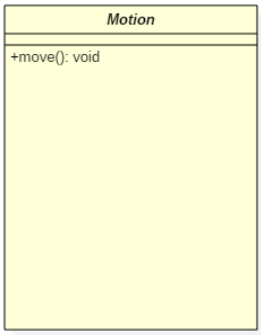
Абстрактные классы

Это класс с прототипом операции, но не с реализацией. Также возможно иметь абстрактный класс без каких-либо операций, объявленных внутри него. Аннотация полезна для определения функциональных возможностей классов. Давайте рассмотрим пример абстрактного класса. Предположим, у нас есть абстрактный класс, называемый движением с методом или операцией, объявленной внутри него. Метод, объявленный внутри абстрактного класса, называется move () .

Этот метод абстрактного класса может использоваться любым объектом, таким как автомобиль, животное, робот и т. Д., Для изменения текущей позиции. Эффективно использовать этот метод абстрактного класса с объектом, потому что для данной функции не предусмотрена реализация. Мы можем использовать его любым способом для нескольких объектов.

В UML абстрактный класс имеет те же обозначения, что и класс. Единственная разница между классом и абстрактным классом состоит в том, что имя класса строго написано курсивом.

Абстрактный класс не может быть инициализирован или создан.



Обозначение абстрактного класса

В приведенной выше записи абстрактного класса есть единственный абстрактный метод, который может использоваться несколькими объектами классов.

Пример диаграммы классов UML

Создание диаграммы классов — простой процесс. Это не связано со многими техническими аспектами. Вот пример:

Система банкоматов очень проста, так как клиенты должны нажать несколько кнопок, чтобы получить наличные. Однако существует несколько уровней безопасности, которые должна пройти любая система ATM. Это помогает предотвратить мошенничество и предоставить наличные или необходимые данные для банковских клиентов.

Ниже приведен пример диаграммы классов UML:

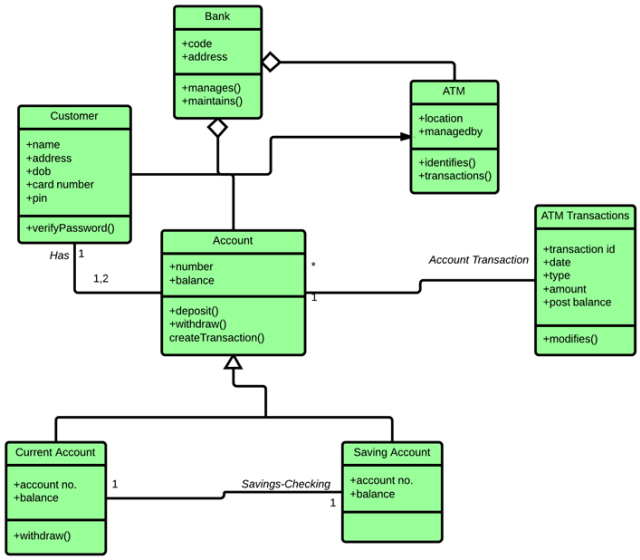


Диаграмма классов в жизненном цикле разработки программного обеспечения

Диаграммы классов могут использоваться на разных этапах разработки программного обеспечения. Это помогает в моделировании диаграмм классов в трех разных ракурсах.

1. Концептуальная перспектива: концептуальные диаграммы описывают вещи в реальном мире. Вы должны нарисовать диаграмму, которая представляет понятия в изучаемой области. Эти понятия относятся к классу и он всегда не зависит от языка.

2. Перспектива спецификации: Перспектива спецификации описывает программные абстракции или компоненты со спецификациями и интерфейсами. Тем не менее, это не дает никаких обязательств для конкретной реализации.

3. Перспектива реализации: этот тип диаграмм классов используется для реализаций на определенном языке или в приложении. Перспектива реализации, использование для реализации программного обеспечения.

Лучшие практики проектирования диаграммы классов

Диаграммы классов являются наиболее важными диаграммами UML, используемыми для разработки программных приложений. Есть много свойств, которые следует учитывать при рисовании диаграммы классов. Они представляют различные аспекты программного приложения.

Вот некоторые моменты, которые следует учитывать при рисовании диаграммы классов:

1. Имя, данное диаграмме классов, должно быть значимым. Более того, следует описать реальный аспект системы.
2. Отношения между каждым элементом должны быть определены заранее.
3. Ответственность за каждый класс должна быть определена.
4. Для каждого класса должно быть указано минимальное количество свойств. Следовательно, нежелательные свойства могут легко усложнить диаграмму.
5. Примечания пользователя должны быть включены всякий раз, когда вам нужно определить некоторые аспекты диаграммы. В конце чертежа это должно быть понятно для команды разработчиков программного обеспечения.
6. Наконец, перед созданием окончательной версии диаграмму необходимо нарисовать на обычной бумаге. Более того, он должен быть переработан до тех пор, пока не будет готов к окончательной сдаче.

Контрольные вопросы:

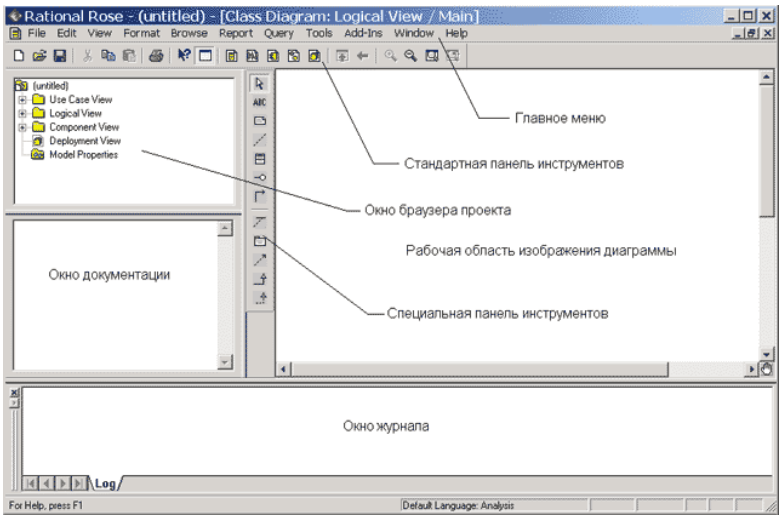
1. Для чего используются диаграммы классов?

2. Какие существуют отношения между элементами диаграммы?
3. Назовите основные элементы диаграммы.

Тема 31. Моделирование предметной области с помощью Rational Rose.

Цель: сформировать знания о предметной области среды разработки UML диаграмм Rational Rose.

В IBM Rational Rose реализованы общепринятые стандарты на рабочий интерфейс программы, аналогично известным средам визуального программирования. После установки IBM Rational Rose на компьютер пользователя, запуск этого средства в среде MS Windows (без выбора готовых шаблонов проектов) приводит к появлению на экране соответствующего рабочего интерфей–а - автоматически создается новый проект и в рабочем окне диаграммы появляется по умолчанию окно диаграммы классов.



Рассмотрим назначение и основные функции каждого из этих элементов.

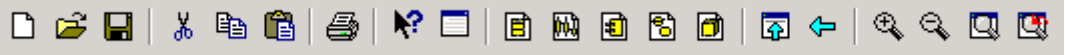
Главное меню и стандартная панель инструментов

Главное меню программы IBM Rational Rose вы полнено в общепринятом стандарте и имеет следующий вид№

Отдельные пункты меню объединяют сходные операции, относящиеся ко всему проекту в целом. Некоторые из пунктов меню содержат хорошо знакомые операции, такие как открытие проекта, вывод на печать диаграмм, копирование в буфер и вставка из буфера различных элементов диаграмм. Другие операции настолько специфичны, что могут потребоваться дополнительные усилия для их изучения (свойства операций генерации программного кода или проверки согласованности моделей).

Стандартная панель инструментов располагается ниже строки главного меню и имеет следующий вид. Некоторые из инструментов недоступны для нового проекта, который не имеет никаких элементов. Стандартная панель инструментов обеспечивает

быстрый доступ к тем командам меню, которые выполняются разработчиками наиболее часто.



(Инструменты→Параметры), открыть вкладку **Toolbars** (Панели инструментов) появившегося диалогового окна и нажать кнопку **Standard** (Стандартная) . В дополнительно открытом окне можно переносить требуемые кнопки из левого списка в правый список, а ненужные кноп–и - из правого списка в левый. Данным способом можно показать или скрыть различные кнопки инструментов, а также изменить их размер. Назначение отдельных кнопок стандартной панели инструментов приводится далее при рассмотрении операций главного меню.

Назначение операций главного меню File и Edit

Операции главного меню **File** (Файл) позволяют создавать новые модели в нотации языка UML, загружать и сохранять разрабатываемую модель во внешнем файле, распечатывать на принтере разработанные диаграммы.

Операции главного меню **Edit** (Редактирование) позволяют выполнять действия по редактированию элементов модели и их свойств, а также выполнять поиск элементов в рамках разрабатываемого проекта.

Назначение операций главного меню View, Format и Browse

Операции главного меню **View** (Вид) позволяют отображать на экране различные элементы рабочего интерфейса и изменять графическое представление диаграмм. Назначение операций этого пункта главного меню представлено в следующей таблице.

Таблица Операции пункта главного меню View (Вид)		
Название операции меню	Наличие кнопки на станартной панели	Назначение операции главного меню
Toolbars		Позволяет настроить внешний вид рабочего интерфейса системы IBM Rational Rose и содержит дополнительные подпункты: Standa–d - делает видимой/невидимой стандартную панель инструментов (рис. 1.3) Toolb–x - делает видимой/невидимой стандартную панель инструментов текущей активной диаграммы Configu–e - вызывает диалоговое окно настройки параметров модели, открытое на вкладке настройки панелей инструментов
Status Bar		Делает видимой/невидимой строку состояния
Documentation		Делает видимым/невидимым окно документации
Browser		Делает видимым/невидимым браузер проекта
Log		Делает видимым/невидимым окно журнала
Editor		Делает видимым/невидимым встроенный текстовый редактор
Time Stamp		Включает/выключает режим отображения времени в записях журнала

Zoom to Selection		Изменяет масштаб изображения выделенных элементов модели , так чтобы они разместились в одном окне
Zoom In		Увеличивает масштаб изображения
Zoom Out		Уменьшает масштаб изображения
Fit in Window		Изменяет (уменьшает) масштаб изображения всех элементов текущей диаграммы, так чтобы все они разместились в одном окне
Undo Fit in Window		Отменяет изменение масштаба изображения размещения элементов в одном окне
Page Breaks		Разбивает текущую диаграмму на страницы для последующей печати
Refresh		Перерисовывает текущую диаграмму
As Booch		Изображает элементы модели в соответствии с нотацией Г. Буча
As OMT		Изображает элементы модели в соответствии с нотацией OMT
As Unified		Изображает элементы модели в соответствии с нотацией языка UML

Операции главного меню **Format** (Формат) позволяют выполнять действия по изменению внешнего вида элементов модели на различных диаграммах. Назначение операций этого пункта главного меню представлено в следующей таблице.

Таблица. Операции пункта главного меню Format (Формат)	
Название операции меню	Назначение операции главного меню
Font Size	Изменяет масштаб используемого шрифта
Font	Вызывает диалоговое окно выбора шрифта
Line Color	Вызывает диалоговое окно выбора цвета линий
Fill Color	Вызывает диалоговое окно выбора цвета для изображения графических элементов диаграмм
Use Fill Color	Включает/выключает режим отображения цвета для изображения графических элементов диаграмм
Automatic Resize	Включает/выключает режим автоматического изменения размеров графических элементов диаграмм для отображения текстовой информации об их свойствах
Stereotype	Позволяет выбрать способ изображения стереотипов выделенных элементов диаграммы и содержит дополнительные подпункты: **No–e** - стереотип не показывается; **Lab–l** - стереотип отображается в форме текста; **Decorati–n** - стереотип отображается в форме небольшой пиктограммы в правом верхнем углу графического элемента; **Ic–n** - элемент диаграммы отображается в форме специального графического стереотипа, если данный стереотип предусмотрен в программе.
Stereotype Label	Включает/выключает режим отображения текстовых стереотипов для взаимосвязей (ассоциаций, зависимостей и пр.) диаграммы

Show Visibility	Включает/выключает режим отображения кванторов видимости атрибутов и операций выделенных классов
Show Compartment Stereotypes	Включает/выключает режим отображения текстовых стереотипов атрибутов и операций выделенных классов
Show Operation Signature	Включает/выключает режим отображения сигнатуры операций выделенных классов
Show All Attributes	Делает видимыми/невидимыми атрибуты выделенных классов
Show All Operations	Делает видимыми/невидимыми операции выделенных классов
Suppress Attributes	Делает видимой/невидимой секцию атрибутов выделенных классов. Скрывает секцию атрибутов даже в том случае, когда выбрана опция Show All Attributes
Suppress Operations	Делает видимой/невидимой секцию операций выделенных классов. Скрывает секцию операций даже в том случае, когда выбрана опция Show All Operations
Line Style	Позволяет выбрать способ графического изображения линий взаимосвязей и содержит дополнительные подпункты: Rectiline–r - линия изображается в форме вертикальных и горизонтальных отрезков; Obliq–e - линия изображается в форме наклонных отрезков; Togg–e - промежуточный вариант изображения линии
Layout Diagram	Позволяет автоматически разместить графические элементы в окне диаграммы с минимальным количеством пересечений и наложений соединительных линий
Autosize All	Позволяет автоматически изменить размеры графических элементов текущей диаграммы таким образом, чтобы текстовая информация помещалась внутри изображений соответствующих элементов
Layout Selected Shapes	Позволяет автоматически разместить выделенные графические элементы в окне диаграммы с минимальным количеством пересечений и наложений соеденительных линий

Операции главного меню **Browse** (Обзор) позволяют отображать рабочие окна с различными каноническими диаграммами разрабатываемой модели и вызывать диалоговые окна редактирования свойств отдельных элементов модели. Назначение операций этого пункта главного меню представлено в следующей таблице.

Таблица. Операции пункта главного меню Browse (Обзор)		
Название операции меню	Наличие кнопки на стандартной панели	Назначение операции главного меню
Use Case Diagram		Вызывает диалоговое окно с предложением выбрать для отображения в рабочем окне одну из существующих диаграмм вариантов использования модели или приступить к разработке новой диаграммы
Class Diagram		Вызывает диалоговое окно с предложением выбрать для отображения в рабочем окне одну из существующих диаграмм классов модели или приступить к разработке новой диаграммы

Component Diagram	Вызывает диалоговое окно с предложением выбрать для отображения в рабочем окне одну из существующих диаграмм компонентов модели или приступить к разработке новой диаграммы	
Deployment Diagram		Позволяет отобразить в рабочем окне диаграмму развертывания разрабатываемой модели
Interaction Diagram		Вызывает диалоговое окно с предложением выбрать для отображения в рабочем окне одну из существующих диаграмм кооперации или последовательности, а также приступить к разработке новой диаграммы взаимодействия
State Machine Diagram		Вызывает диалоговое окно с предложением выбрать для отображения в рабочем окне одну из существующих диаграмм состояний модели или приступить к разработке новой диаграммы
Expand		Отображает в рабочем окне первую из диаграмм выделенного пакета модели
Parent		Отображает в рабочем окне родителя выделенной диаграммы модели
Specification		Вызывает диалоговое окно свойств выделенного элемента модели
Top Level		Отображает в рабочем окне диаграмму самого верхнего уровня для текущей диаграммы модели
Referenced Item		Отображает в рабочем окне диаграмму классов, содержащую класс для выделенного объекта модели
Previous Diagram		Отображает в рабочем окне предыдущую диаграмму модели
Create Message Trace Diagram		Позволяет создать диаграмму трассировки сообщений

Назначение кнопок специальной панели инструментов

Специальная панель инструментов содержит кнопки с изображением графических примитивов, необходимых для разработки различных диаграмм проекта, например, диаграммы классов. Назначение отдельных кнопок панели можно узнать также из всплывающих подсказок.

Таблица. Назначение кнопок специальной панели инструментов (на примере работы с диаграммой классов)		
Графическое изображение	Всплывающая подсказка	Назначение кнопки
	Selection Tool	Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме
	Text Box	Добавляет на диаграмму текстовую область
	Note	Добавляет на диаграмму примечание
	Anchor Note to Item	Добавляет на диаграмму связь примечания с соответствующим графическим элементом диаграммы
	Class	Добавляет на диаграмму класс
	Interface	Добавляет на диаграмму интерфейс
	Unidirectional Association	Добавляет на диаграмму направленную ассоциацию
	Association Class	Добавляет на диаграмму ассоциацию класс
	Package	Добавляет на диаграмму пакет
	Dependency or Instantiates	Добавляет на диаграмму отношение зависимости
	Generalization	Добавляет на диаграмму отношение обобщения
	Realize	Добавляет на диаграмму отношение реализации

На специальной панели инструментов по умолчанию присутствует только часть пиктограмм элементов, которые могут быть использованы для построения диаграммы классов. Добавить кнопки с пиктограммами других графических элементов.

Соответствующее диалоговое окно настройки специальной панели инструментов для диаграммы классов можно вызвать аналогично другим панелям с помощью операции контекстного меню **Customize** (Настройка) при позиционировании курсора на специальной панели инструментов.

Контрольные вопросы:

1. Для чего предназаначена среда Rational Rose?
2. Что входит в интерфейс среды?
3. Опишите назначение операций главного меню View, Format и Browse.
4. Опишите назначение кнопок специальной панели инструментов.

Тема 32. Состав языка программирования С++. Структура программы. Типы данных

Цель: сформировать знания о составе языка программирования С++, структуре программы и типах данных.

Язык С++ – это язык программирования общего назначения, цель которого – сделать работу серьёзных программистов более приятным занятием. За исключением несущественных деталей, язык С++ является надмножеством языка С. Помимо

возможностей, предоставляемых языком С, язык С++ обеспечивает гибкие и эффективные средства определения новых типов.

Язык программирования служит двум взаимосвязанным целям: он предоставляет программисту инструмент для описания подлежащих выполнению действий и набор концепций, которыми оперирует программист, обдумывая, что можно сделать. Первая цель в идеале требует языка, близкого к компьютеру, чтобы все важные элементы компьютера управлялись просто и эффективно способом, достаточно очевидным для программиста. Язык С создавался на основе именно от этой идеи. Вторая цель в идеале требует языка, близкого к решаемой задаче, чтобы концепции решения могли быть выражены понятно и непосредственно. Эта идея привела к пополнению языка С свойствами, превратившими его в язык С++.

Ключевое понятие в языке С++ – **класс**. Классы обеспечивают сокрытие информации, гарантированную инициализацию данных, неявное преобразование определяемых пользователем типов, динамическое определение типа, контроль пользователя над управлением памятью и механизм перегрузки операторов. Язык С++ предоставляет гораздо лучшие, чем язык С, средства для проверки типов и поддержки модульного программирования. Кроме того, язык содержит усовершенствования, непосредственно не связанные с классами, такие как: символические константы, встраивание функций в место вызова, параметры функций по умолчанию, перегруженные имена функций, операторы управления свободной памятью и ссылки. Язык С++ сохраняет способность языка С эффективно работать с аппаратной частью на уровне битов, байтов, слов, адресов и т.д. Это позволяет реализовывать пользовательские типы с достаточной степенью эффективности.

Алфавит

Множество символов языка С включает:

- прописные буквы латинского алфавита;
- строчные буквы латинского алфавита;
- арабские цифры;
- разделители: , . ; : ? !« ' " | / \ ~ _ ^ ( ) { } [ ] < > # % & - = + *

Остальные символы могут быть использованы только в символьных строках, символьных константах и комментариях. Язык С++ различает большие и маленькие буквы, таким образом, name и Name – **разные идентификаторы**.

Литералы

Литералы в языке С++ могут быть целые, вещественные, символьные и строковые.

- Целые (можно испрользовать апостроф как разделитель групп разрядов):
  - десятичные: 10, 132, -32179, 2'147'483'647;
  - двоичные (предваряются символами «0b»): 0b11, 0b1010b, 0b1111'0011;
  - восьмеричные (предваряются символом «0»): 010, 0204, -076663;
  - шестнадцатеричные (предваряются символами «0x»): 0xA, 0x84, 0x7db3.
- Вещественные: 15.75, 1.575e1, .75, -.125
- Символьные: 'a', 'e', '.', '?', '2'.
- Строковы«: "стр»ка".

Комментарии

Комментарий – это последовательность символов, которая игнорируется компилятором языка С++. Комментарий имеет следующий вид: /*<символы>*/. Комментарии могут занимать несколько строк, но не могут быть вложенными. Кроме того, часть строки, следующая за символами //, также рассматривается как комментарий.

Разумное использование комментариев (и согласованное употребление отступов) может сделать чтение и понимание программы более приятным занятием. При неправильном использовании комментариев читабельность программы может, напротив,

серьёзно пострадать. Компилятор не понимает смысл комментариев, поэтому не существует способа проверить, что комментарий:

1. содержателен;
2. имеет какое-то отношение к программе;
3. не устарел.

Удачно подобранный и написанный набор комментариев является существенной частью хорошей программы. Написание «правильных» комментариев может оказаться не менее сложной задачей, чем написание самой программы.

На уровне библиотек/программ/функций комментарии отвечают на вопрос «ЧТО?»: «Что делают эти библиотеки/программы/функции?». Например: // Эта функция использует метод Ньютона для вычисления корня функции. Такие комментарии позволяют понять, что делает программа, без необходимости смотреть на исходный код.

Внутри библиотек/программ/функций комментарии отвечают на вопрос «КАК?»: «Как код выполняет задание?». Например: // Чтобы получить случайный элемент, мы делаем следующее: …

На уровне однострочного кода комментарии отвечают на вопрос «ПОЧЕМУ?»: «Почему код выполняет задание именно так, а не иначе?». Плохой комментарий на уровне операторов объясняет, что делает код. Если вы когда-нибудь писали код, который был настолько сложным, что нужен был комментарий, который бы объяснял, что он делает, то вам нужно было бы не писать комментарий, а переписывать этот код.

Типы данных языка C++

Имя	**Размер**	**Представляемые значения**	**Диапазон**
bool	1 байт	логические	false, true
(signed) char	1 байт	символы целые числа	от –128 до 127
wchar_t	2 байта	символы Unicode	от 0 до 65535
(signed) short int	2 байта	целые числа	от -32768 до 32767
(signed) int	зависит от реализации (в последних компиляторах обычно 4 байта)	целые числа	
(signed) long int	4 байта	целые числа	от -2147483648 до 2147483647
(signed) long long int (signed) __int64 (MS)	8 байт	целые числа	от –9,223,372,036,854,775,808 до 9,223,372,036,854,775,807
unsigned char	1 байт	символы целые числа	от 0 до 255
unsigned short int	2 байта	целые числа	0 до 65535
unsigned int	зависит от реализации (в последних компиляторах обычно 4 байта)	целые числа	
unsigned long int	4 байта	целые числа	от 0 до 4294967295

(unsigned) long long int (unsigned) __int64 (MS)	8 байт	целые числа	от 0 до 18,446,744,073,709,551,615
float	4 байта	вещественные числа	от 1.175494351e–38 до 3.402823466e+38
double	8 байт	вещественные числа	от 2.2250738585072014e–308 до 1.7976931348623158e+308
long double	зависит от реализации	вещественные числа	

В языке C++ также существуют перечислимый тип – enum, который является подмножеством целого типа, и пустой тип – void, который имеет специальное назначение. Он используется для объявления функций, которые не возвращают никакого значения, а также для объявления указателей на значение типа void. Такие указатели могут быть преобразованы к указателям на любой другой тип.

В языке C++ можно объявлять структуры и так называемые объединения.

В языке C++ **нет специальных типов для массивов и строк**, которые представляются массивом символов.

В языке C не существовало логического типа. Логические значения представлялись данными целого типа, при этом значение 0 соответствовало логическому значению ложь, а **все остальные целые значения** соответствовали логическому значению истина. В языке C++ сохранена данная логика. По определению, true имеет значение 1 при преобразовании к целому типу, а false – значение 0. И наоборот, целые можно неявно преобразовать в логические значения: при этом ненулевые целые преобразуются в true, а ноль – в false. В любом месте, где требуется логическое значение, может стоять целочисленное выражение. В арифметических и логических выражениях логические значения преобразуются в целые, операции выполняются над преобразованными величинами.

Указатель можно неявно преобразовать в логическое значение, при этом ненулевой указатель принимает значение true, нулевой – false.

Такой подход позволяет вместо логической и целочисленной переменных объявлять только целочисленную, при этом значение переменной, равное 0, говорит об отсутствии некоторого признака у объекта, а остальные значения говорят о его наличии, и при этом несут какую-либо дополнительную информацию.

Тема 33. Обзор операторов языка C++.

Цель: сформировать знания об операторах языка C++: арифметических, логических, условия и цикла.

Запись действий, которые должен выполнить компьютер, состоит из операторов. Язык C++ включает все операторы C и еще несколько новых операторов. Операторы определяют, какое вычисление следует выполнить с одним или несколькими операндами. При выполнении программы операторы выполняются один за другим, если только оператор

не является оператором управления, который может изменить последовательное выполнение рограммы.

Математические операторы

- Вычитание, а также унарный минус
- + Сложение
- * Умножение
- / Деление
- % Взятие по модулю (остаток)
- --Уменьшение
- ++ Увеличение

Операторы +, -, * и / работают в C++ точно так же, как и в большинстве других языков. Их можно применять практически ко всем встроенным типам данных. Когда применяется / к целому числу или символу, остаток не используется, например: 10/3 равно 3.

Оператор взятия по модулю % работает в C++ так же, как в некоторых других языках. Надо помнить, что оператор взятия по модулю выдает остаток от целочисленногоделения. % не может использоваться с типами float и double.

Пример кода

Следующий фрагмент демонстрирует его использование:

```
int x, y; x = 10; y = 3;
prin«f(»%d", x/y); /* выводит 3 */
print« (»%d", x%y); /*выводит1-остаток целочисленного деления */
x = 1;y = 2;
prin«f("%d»%d", x/y, x%y)/ /* выводит 0 1*/
```

Последняя строка печатает 0 и 1, так как в результате целочисленного деления 1/2 получается 0 с остатком 1. 1 % 2 выдает остаток 1.

Побитовые операторы

Побитовые операции выполняются над отдельными разрядами или битами чисел. Данные операции производятся только над целыми числами.

& битовое И
| битовое ИЛИ
^ битовое ИСКЛЮЧАЮЩЕЕ ИЛИ
~ битовое НЕ
<< двиг влево
>> сдвиг вправо

Операторы присваивания

Операции присваивания позволяют присвоить некоторое значения. Эти операции выполняются над двумя операндами,причем левый операнд может представлять только модифицируемое именованное выражение, например, переменную.

+= действие присваивание после сложения
-= присваивание после вычитания
*= присваивание после умножения
/=присваивание после деления
%=присваивание после деления по модулю
<<=присваивание после сдвига разрядов влево
>>=присваивание после сдвига разрядов вправо

Логические операторы

Логические операции образуют сложное (составное) условие из нескольких простых (два или более) условий. Эти операции упрощают структуру программного кода в

несколько раз. В следующей таблице приведены все логические операции в языке программирования С++, для построения логических условий.

&& логическое И
|| логическое ИЛИ
! логическое НЕ

Операторы сравнения

Сравнивать можно операнды любого типа, но либо они должны быть оба одного и того же встроенного типа (сравнение на равенство и неравенство работает для двух величин любого типа), либо между ними должна быть определена соответствующая операция сравнения. Результат – логическое значение true или false.

== равно
!= не равно
< меньше
> больше
<= меньше или равно
>= больше или равно

Оператор ветвления if

Условный оператор if используется для разветвления процесса вычислений на два направления. Формат оператора: if (выражение) оператор_1; [else оператор_2;]

Сначала вычисляется выражение, которое может иметь арифметический тип или тип указателя. Если оно не равно нулю (имеет значение true), выполняется первый оператор, ина–е - второй. После этого управление передается на оператор, следующий за условным. Одна из ветвей может отсутствовать, логичнее опускать вторую ветвь вместе с ключевым словом else. Если в какой-либо ветви требуется выполнить несколько операторов, их необходимо заключить в блок, иначе компилятор не сможет понять, где заканчивается ветвление. Блок может содержать любые операторы, в том числе описания и другие условные операторы (но не может состоять из одних описаний). Необходимо учитывать, что переменная, описанная в блоке, вне блока не существует.

Структурная схема условного оператора

Далее представлены примеры условного оператора:

```
if (a<0) b = 1;
if (a<b && (a>d || a==0)) b++; else {b* = a; a = 0;}
if (a<b) {if (a<c) m = a; else m = c;} else {if (b<c) m = b; else {m = c;}
if (a++) b++;
if (b>a) max = b; else max = a;
```

В примере отсутствует ветвь else. Подобная конструкция называется «пропуск оператора», поскольку присваивание либо выполняется, либо пропускается в зависимости от выполнения условия.

Если требуется проверить несколько условий, их объединяют знаками логических операций. Например, выражение в примере 2 будет истинно в том случае, если выполнится одновременно условие a<b и одно из условий в скобках. Если опустить внутренние скобки, будет выполнено сначала логическое И, а пот–м - ИЛИ. Оператор в примере 3 вычисляет наибольшее значение из трех переменных. Фигурные скобки в данном случае не обязательны, так как компилятор относит часть else к ближайшему if.

Пример 4 напоминает о том, что хотя в качестве выражений в операторе if чаще всего используются операции отношения, это не обязательно. Конструкции, подобные оператору в примере 5, проще и нагляднее записывать в виде условной операции (в данном случае: max = (b > a) ? b : a;).

Оператор switch Оператор switch (переключатель) предназначен для разветвления процесса вычислений на несколько направлений. Формат оператора:

```
switch ( выражение ){
case константное_выражение_1: [список_операторов_1]
case константное_выражение_2:1[список_операторов_2]
…
case константное_выражение _n:[список_операторов _n]
[default: операторы ] \
}
```

Выполнение оператора начинается с вычисления выражения (оно должно быть целочисленным), а затем управление передается первому оператору из списка, помеченного константным выражением, значение которого совпало с вычисленным.

После этого, если выход из переключателя явно не указан, последовательно выполняются все остальные ветви. Выход из переключателя обычно выполняется с помощью операторов break или return. Оператор break выполняет выход из самого внутреннего из объемлющих его операторов switch, for, while и do. Оператор return выполняет выход из функции, в теле которой он записан.

Цикл с параметром (for)

Цикл с параметром имеет следующий формат:

for (инициализация; выражение : модификации) оператор;

Инициализация используется для объявления и присвоения начальных значений величинам, используемым в цикле. В этой части можно записать несколько операторов, разделенных запятой (операцией «последовательное выполнение»)

Чтобы избежать ошибок, рекомендуется:

проверить, всем ли переменным, встречающимся в правой части операторов присваивания в теле цикла, присвоены до этого начальные значения (а также возможно ли выполнение других операторов); проверить, изменяется ли в цикле хотя бы одна переменная, входящая в условие выхода из цикла; предусмотреть аварийный выход из цикла по достижению некоторого количества итераций (см. пример в следующем разделе); если в теле цикла требуется выполнить более одного оператора, нужно заключать их в фигурные скобки.

Оператор goto

Оператор безусловного перехода goto имеет формат: goto метка;

В теле той же функции должна присутствовать ровно одна конструкция вида: метка: оператор; Оператор goto передает управление на помеченный оператор. Мет–а - это обычный идентификатор, областью видимости которого является функция, в теле которой он задан.

Использование оператора безусловного перехода оправдано в двух случаях:

1. принудительный выход вниз по тексту программы из нескольких вложенных циклов или переключателей;

2. переход из нескольких мест функции в одно (например, если перед выходом из функции необходимо всегда выполнять какие-либо действия).

В любом случае не следует передавать управление внутрь операторов if, switch и циклов. Нельзяпереходить внутрь блоков, содержащих инициализацию переменных, на операторы, расположенные после нее, поскольку в этом случае инициализация не будет выполнена:

```
int k; …
goto metka; …
{ int a = 3, b = 4;
k = a + b;
```

metka: int m = k + 1; …

}

После выполнения этого фрагмента программы значение переменной m не определено.

Оператор break

Оператор break используется внутри операторов цикла, if или switch для обеспечения перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится break.

Операторы continue и return

Оператор перехода к следующей итерации цикла continue пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации. Оператор возврата из функции return завершает выполнение функции и передает управление в точку ее вызова. Вид оператора: return [выражение]; Выражение должно иметь скалярный тип. Если тип возвращаемого функцией значения описан как void, выражение должно отсутствовать.

Функции ввода/вывода

Ввод/вывод в C++ реализуется либо с помощью функций, унаследованных от библиотеки C, либо с помощью потоков C++. Смешивать эти два способа в одной программе можно только синхронизировав ввод с помощью функции sync_with_iostream(). Каждый способ имеет свои преимущества.

Преимущество спользования потоков в том, что они легче в использовании в простых случаях ввода/вывода, не требующих форматирования, а, главное, потоковые операции можно переопределить для собственных классов.

Ввод/вывод в стиле C удобнее использовать при форматированном выводе в программах, не использующих объектно-ориентированную технику. Кроме того, существуют миллионы строк, написанных на C и перенесенных на C++, с которыми программисту приходится сталкиваться. Для использования функций ввода/вывода в стиле C необходимо подключить к программе заголовочный файл <iostream> или <ciostream>. При вводе/выводе данные рассматриваются как поток байтов. Физически поток представляет собой файл или устройство, например, клавиатуру или дисплей, рассматривающийся как частный случай файла.

Математические функции

Тригонометрические функции

tan() вычисление тангенса

cos() вычисление косинуса

sin() вычисление синуса

acos() вычисление арккосинуса

asin() вычисление арксинуса

atan() вычисление арктангенса

atan2() вычисление арктангенса с двумя параметрами

ceil() Функции округления

floor() округление до большего округление к меньшему

fmod() вычисление остатка от деления

Другие математические функции библиотеки <cmath>

fabs() вычисление абсолютного значени

abs() вычисление абсолютного значения

fma() умножение и добавлен–е - C++

Контрольные вопросы

1. В чем заключается задача операторов?
2. Какие операторы упрощают структуру программного кода в несколько раз?
3. Какой оператор является заменой множественного использования операторов if ?

4. В чем заключается задача оператора switch?
5. Когда вызываются функции?
6. Какая функция используется для вычисления квадратного корня?

Тема 34. Введение в объект–о - ориентированное программирование.

Цель: сформировать знания об основных понятиях объектно-ориентированного программирования.

Объектно-ориентированное программирование (ООП) — это методика разработки программ, в основе которой лежит понятие объект.

Объект — это некоторая структура, соответствующая объекту реального мира, его поведению. Задача, решаемая с использованием методики ООП, описывается в терминах объектов и операций над ними, а программа при таком подходе представляет собой набор объектов и связей между ними. Системы объектно-ориентированного программирования (ООП) дают возможность визуализировать процесс создания графического интерфейса разрабатываемого приложения.

Взаимодействие программных объектов между собой и их изменения описываются с помощью программного кода. Создание программного кода в ООП базируется на использовании алгоритмических структур различных типов (линейной, ветвления, цикла), исполнителями которых выступают программные объекты

Класс — разновидность абстрактного типа данных в объектно-ориентированном программировании (ООП), характеризуемый способом своего построения. Другие абстрактные типы данных — метаклассы, интерфейсы, структуры, перечисления, — характеризуются какими-то своими, другими особенностями. Наряду с понятием «объекта» класс является ключевым понятием в ООП.

Программные объекты обладают свойствами, могут использовать методы и реагируют на события Классы объектов являются «шаблонами», определяющими наборы свойств, методов и событий, по которым создаются объекты Основными классами объектов являются классы, реализующие графический интерфейс проектов Объект в программировании — некоторая сущность в виртуальном пространстве, обладающая определѐнным состоянием и поведением, имеющая заданные значения свойств (атрибутов) и операций над ними. Как правило, при рассмотрении объектов выделяется то, что объекты принадлежат одному или нескольким классам, которые определяют поведение (являются моделью) объекта.

Термины «экземпляр класса» и «объект» взаимозаменяемы. Объект, созданный по шаблону класса объектов, является экземпляром класса и наследует весь набор свойств, методов и событий данного класса Каждый экземпляр класса имеет уникальное для данного класса имя Различные экземпляры класса обладают одинаковым набором свойств, однако значения свойств у них могут отличаться Каждый объект обладает определенным набором свойств, первоначальные значения которых можно установить

Значения свойств объектов можно изменять и в программном коде:

Объект.Свойство := ЗначениеСвойства

Метод в объектно-ориентированном программировании — это функция или процедура, принадлежащая какому-то классу или объекту. Как и процедура в процедурном программировании, метод состоит из некоторого количества операторов для выполнения какого-то действия и имеет набор входных аргументов.

Различают простые методы и статические методы (методы класса): простые методы имеют доступ к данным объекта (конкретного экземпляра данного класса), статические методы не имеют доступа к данным объекта и для их использования не нужно создавать

экземпляры (данного класса). Методы предоставляют интерфейс, при помощи которого осуществляется доступ к данным объекта некоторого класса, тем самым, обеспечивая инкапсуляцию данных.

В зависимости от того, какой уровень доступа предоставляет тот или иной метод, выделяют:

- ✓ открытый (public) интерфейс — общий интерфейс для всех пользователей данного класса;
- ✓ защищѐнный (protected) интерфейс — внутренний интерфейс для всех наследников данного класса;
- ✓ закрытый (private) интерфейс — интерфейс, доступный только изнутри данного класса.

Такое разделение интерфейсов позволяет сохранять неизменным открытый интерфейс, но изменять внутреннюю реализацию.

Обратиться к методу объекта можно также с использованием точечной нотации: Объект.Метод

Событие в объектно-ориентированном программировании — это сообщение, которое возникает в различных точках исполняемого кода при выполнении определѐнных условий. События предназначены для того, чтобы иметь возможность предусмотреть реакцию программного обеспечения. Для решения поставленной задачи создаются обработчики событий: как только программа попадает в заданное состояние, происходит событие, посылается сообщение, а обработчик перехватывает это сообщение. В общем случае в обработчик не передаѐтся ничего, либо передаѐтся ссылка на объект, инициировавший (породивший) обрабатываемое событие. В особых случаях в обработчик передаются значения некоторых переменных или ссылки на какие-то другие объекты, чтобы обработка данного события могла учесть контекст возникновения события.

Самое простое событие — это событие, сообщающее о начале или о завершении некоторой процедуры. Событие, по сути, сообщает об изменении состояния некоторого объекта. Наиболее наглядно события представлены в пользовательском интерфейсе, когда каждое действие пользователя порождает цепочку событий, которые, затем обрабатываются в приложении. Событие может создаваться пользователем (щелчок мышью или нажатие клавиши) или быть результатом воздействия других объектов В качестве реакции на событие вызывается определенная процедура, которая может изменять свойства объекта или вызывать его методы Графический интерфейс. Визуальное программирование позволяет делать графический интерфейс разрабатываемых приложений на основе форм и управляющих элементов.

В роли основных объектов при визуальном программировании выступают формы (Forms). Форма представляет собой окно, на котором размещаются управляющие элементы. Управляющие элементы — это командные кнопки (CommandButton), переключатели, или «флажки» (Checkbox), поля выбора, или «радиокнопки» (OptionsButton), списки (ListBox), текстовые поля (TextBox) и др.

Все основанные на объектах языки (C#, Java, C++, Smalltalk, Visual Basic и т.п.) должны отвечать трем основным принципам объектно-ориентированного программирования (ООП):

- ✓ Инкапсуляция
- ✓ Наследование
- ✓ Полиморфизм

Инкапсуляция — это механизм программирования, объединяющий вместе код и данные, которыми он манипулирует, исключая как вмешательство извне, так и неправильное использование данных. В объектно-ориентированном языке данные и код могут быть объединены в совершенно автономный черный ящик. Внутри такого ящика находятся все необходимые данные и код. Когда код и данные связываются вместе

подобным образом, создается объект. Иными словами, объект — это элемент, поддерживающий инкапсуляцию.

Следующий принцип ООП — **наследование —** касается способности языка позволять строить новые определения классов на основе определений существующих классов. По сути, наследование позволяет расширять поведение базового (или родительского) класса, наследуя основную функциональность в производном подклассе (также именуемом дочерним классом) Т.е. наследование представляет собой процесс, в ходе которого один объект приобретает свойства другого объекта. Это очень важный процесс, поскольку он обеспечивает принцип иерархической классификации.

Полиморфи–м - это свойство, которое позволяет одно и тоже имя использовать для решения нескольких технически разных задач. В общем смысле, концепцией полиморфизма является ид«я "один интерфейс, множество мето»ов". Это означает, что можно создать общий интерфейс для группы близких по смыслу действий. Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование одного интерфейса для единого класса действий. Выбор конкретного действия, в зависимости от ситуации, возлагается на компилятор.

Контрольные вопросы:

1.Что такое ООП?

2.Опишите понят«я "объ»кт«, "кл»сс«, "ме»од«, "собы»ие".

3 Опишите принципы ООП.

Тема 35. Природа классов. Природа объекта.

Цель: сформировать знания о природе класса, природе объекта и их зависимостях.

По определению будем называть объектом понятие, абстракцию или любую вещь с четко очерченными границами, имеющую смысл в контексте рассматриваемой прикладной проблемы. Введение объектов преследует две цели:

- понимание прикладной задачи (проблемы);
- введение основы для реализации на компьютере.

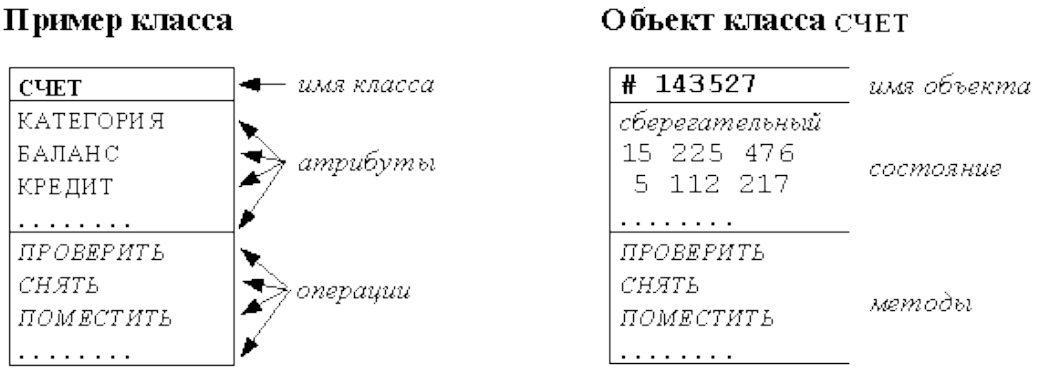
Примеры объектов: форточка, Ба«к "Импер»ал", Петр Сидоров, дело № 7461, сберкнижка и т.д.

Все объекты могут быть отличены один от другого: пусть у нас есть два яблока, имеющие одинаковый цвет, форму, вес и вкус; все равно это два яблока (а не одно), в чем легко убедиться, съев одно из них (другое останется). Между объектами можно установить отношение тождества: объекты, удовлетворяющие этому отношению, одинаковы (тождественны), как вышеупомянутые яблоки. В случае с яблоками иногда говорят о двух экземплярах объекта яблоко. Мы будем считать здесь, что объект и экземпляр объек–а - это одно и то же.

Классы

Два яблока из предыдущего примера принадлежат одному и тому же классу объектов (именно с этим связана их одинаковость). Цвет, форма, вес и вкус ябло–а - это его атрибуты: совокупность атрибутов и их значений (например, красное, овальное, стограммовое, кисло-сладкое) характеризует объект.

Все объекты одного и того же класса характеризуются одинаковыми наборами атрибутов. Однако объединение объектов в классы определяется не наборами атрибутов, а семантикой. Так, например, объекты конюшня и лошадь могут иметь одинаковые атрибуты: цена и возраст. При этом они могут относиться к одному классу, если рассматриваются в задаче просто как товар, либо к разным классам, что более естественно.



Объединение объектов в классы позволяет ввести в задачу абстракцию и рассмотреть ее в более общей постановке. Класс имеет имя (например лошадь), которое относится ко всем объектам этого класса. Кроме того, в классе вводятся имена атрибутов, которые определены для объектов. В этом смысле описание класса аналогично описанию типа структуры (записи); при этом каждый объект имеет тот же смысл, что и экземпляр структуры (переменная или константа соответствующего типа). Пример класса и объекта этого класса приведен на рисунке 2.1.

Атрибуты объектов

Атриб–т - это значение, характеризующее объект в его классе. Примеры атрибутов: категория, баланс, кредит (атрибуты объектов класса счет); имя, возраст, вес (атрибуты объектов класса человек) и т.д.

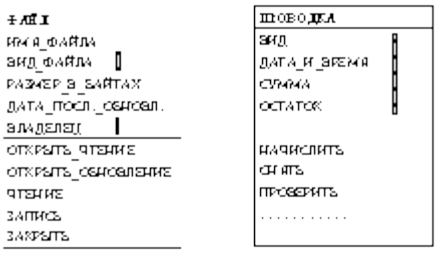
Среди атрибутов различаются постоянные атрибуты (константы) и переменные атрибуты. Постоянные атрибуты характеризуют объект в его классе (например, номер счета, категория, имя человека и т.п.). Текущие значения переменных атрибутов характеризуют текущее состояние объекта (например, баланс счета, возраст человека и т.п.); изменяя значения этих атрибутов, мы изменяем состояние объекта.

Атрибуты перечисляются во второй части прямоугольника, изображающего класс (см. рисунок 2.1). Иногда указывается тип атрибутов (ведь каждый атриб–т - это некоторое значение) и начальное значение переменных атрибутов (совокупность начальных значений этих атрибутов задает начальное состояние объекта).

Следует отметить, что, говоря об объектах и их классах, мы не подразумеваем никакого объектно-ориентированного языка программирования. Это, в частности, выражается в том, что на данном этапе разработки программной системы следует рассматривать только такие атрибуты, которые имеют смысл в реальности (все атрибуты объектов класса сч–т - рисунок 2.– - обладают этим свойством). Атрибуты связаны с особенностями общей реализации. Например, если известно, что будет использоваться база данных, в которой каждый объект имеет уникальный идентификатор, то включать этот идентификатор в число атрибутов объекта на данном этапе не следует. Дело в том, что, вводя такие атрибуты, мы ограничиваем возможности реализации системы. Так, вводя в качестве атрибута уникальный идентификатор объекта в базе данных, мы уже в самом начале проектирования отказываемся от использования СУБД, которые такой идентификатор не поддерживают.

Операции и методы

Операц–я - это функция (или преобразование), которую можно применять к объектам данного класса. Примеры операций: проверить, снять, поместить (для объектов класса сч–т - рисунок 2.1), открыть_на_чтение, читать, закрыть (для объектов класса фа–л - рисунок 2.2) и т.п.



Все объекты данного класса используют один и тот же экземпляр каждой операции (т.е. увеличение количества объектов некоторого класса не приводит к увеличению количества загруженного программного кода). Объект, из которого вызвана операция, передается ей в качестве ее неявного аргумента (параметра).

Одна и та же операция может, вообще говоря, применяться к объектам разных классов: такая операция называется полиморфной, так как она может иметь разные формы для разных классов. Например, для объектов классов вектор и комплексное_число можно определить операцию +; эта операция будет полиморфной, так как сложение векторов и сложение комплексных чисел, вообще говоря, разные операции.

Каждой операции соответствует мет–д - реализация этой операции для объектов данного класса. Таким образом, операц–я - это спецификация метода, мет–д - реализация операции. Например, в классе файл может быть определена операция печать (print). Эта операция может быть реализована разными методами: (а) печать двоичного файла; (б) печать текстового файла и др. Логически эти методы выполняют одну и ту же операцию, хотя реализуются они разными фрагментами кода.

Каждая операция имеет один неявный аргуме–т - объект к которому она применяется. Кроме того, операция может иметь и другие аргументы (параметры). Эти дополнительные аргументы параметризуют операцию, но не связаны с выбором метода. Метод связан только с классом и объектом (некоторые объектно-ориентированные языки, например C++, допускают одну и ту же операцию с разным числом аргументов, причем используя то или иное число аргументов, мы практически выбираем один из методов, связанных с такой операцией; на этапе предварительного проектирования системы удобнее считать эти операции различными, давая им разные имена, чтобы не усложнять проектирование).

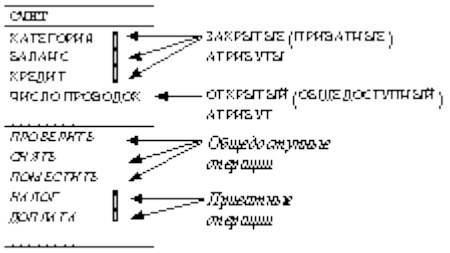
Операция (и реализующие ее методы) определяется своей сигнатурой, которая включает, помимо имени операции, типы (классы) всех ее аргументов и тип (класс) результата (возвращаемого значения). Все методы, реализующие операцию должны иметь такую же сигнатуру, что и реализуемая ими операция.

Операции перечисляются в третьей части прямоугольника (рисунок 2.1), описывающего класс. Каждая операция должна быть представлена своей сигнатурой, однако на ранних стадиях проектирования можно ограничиваться указанием имени операции, отложив полное определение сигнатуры на конец рассматриваемой фазы жизненного цикла (либо даже на последующие фазы). В графическом языке технологии OMT тип любого объекта данных указывается вслед за именем этого объекта после двоеточия.

В настоящее время существует несколько технологий объектно-ориентированной разработки прикладных программных систем, в основе которых лежит построение и интерпретация на компьютере моделей этих систем OMT (Object Modeling Techniques).

При моделировании системы полезно различать операции, имеющие побочные эффекты (эти эффекты выражаются в изменении значений атрибутов объекта, т.е. в изменении его состояния), и операции, которые выдают требуемое значение, не меняя состояния объекта. Эти последние операции называются запросами.

Значения некоторых атрибутов объекта могут быть доступны только операциям этого объекта. Такие атрибуты называются закрытыми. На рисунке 2.3 показаны закрытые атрибуты для объектов класса счет. Значения закрытых атрибутов объекта можно узнать вне объекта только в том случае, если среди операций этого объекта определены соответствующие запросы. Аналогично, в объекте можно определить и закрытые (вспомогательные) операции, однако на ранних стадиях проектирования этого, как правило, не делают, так как выделение закрытых операций связано, в основном, с реализацией системы.



Запросы без аргументов (за исключением неявного аргумен–а - объекта, к которому применяется операция) могут рассматриваться как производные атрибуты. Значения производных атрибутов зависят от значений основных атрибутов. В этом их отличие от основных атрибутов, значения которых независимы. Следовательно, значения основных атрибутов объекта определяют как его состояние, так и значения его производных атрибутов. Так, например, длина, ширина и высота комна–ы - ее основные атрибуты, а площадь и кубату–а - производные атрибуты; такой атрибут как кубатура нужен для того, чтобы не вычислять кубатуру комнаты всякий раз, когда понадобится ее значение.

Выбор основных атрибутов объектов произволен, но в число основных атрибутов не следует включать такие атрибуты, значения которых определяются значениями других атрибутов, так что на самом деле они являются производными.

Таким образом, для задания класса необходимо указать имя этого класса, а затем перечислить его атрибуты и операции (или методы). Полное описание объекта на графическом языке ОМТ имеет вид, изображенный на рисунке 2.4. Однако иногда удобно бывает пользоваться сокращенным описанием класса, когда в прямоугольнике, изображающем этот класс, указывается только имя класса. Так, на рисунке 2.5 приведены сокращения обозначения классов для нашего основного приме–а - системы обслуживания клиентов банковского консорциума.

Зависимости между классами (объектами)

С каждым объектом связана структура данных, полями которой являются атрибуты этого объекта и указатели функций (фрагментов кода), реализующих операции этого объекта (отметим, что указатели функций в результате оптимизации кода обычно заменяются на обращения к этим функциям). Таким образом, объе–т - это некоторая структура данных, тип которой соответствует классу этого объекта.

Между объектами можно устанавливать зависимости по данным. Эти зависимости выражают связи или отношения между классами указанных объектов. Примеры таких зависимостей изображены на рисунке 2.6 (первые две зависимос–и - бинарные, третья зависимос–ь - тренарная). Зависимость изображается линией, соединяющей классы над которой надписано имя этой зависимости, или указаны роли объектов (классов) в этой зависимости (указание рол–й - наиболее удобный способ идентификации зависимости).

Зависимости между классами являются двусторонними: все классы в зависимости равноправны. Это так даже в тех случаях, когда имя зависимости как бы вносит направление в эту зависимость. Так, в первом примере на рисунке 2.6 имя зависимости имеет_столицу предполагает, что зависимость направлена от класса страна к классу город (двусторонность зависимости вроде бы пропала); но следует иметь в виду, что эта

зависимость двусторонняя в том смысле, что одновременно с ней существует и обратная зависимость является_столицей. Точно таким же образом, во втором примере на рисунке 2.6 можно рассматривать пару зависимостей владеет-принадлежит. Подобных недоразумений можно избежать, если идентифицировать зависимости не по именам, а по наименованиям ролей классов, составляющих зависимость.

В языках программирования зависимости между классами (объектами) обычно реализуются с помощью ссылок (указателей) из одного класса (объекта) на другой. Представление зависимостей с помощью ссылок обнаруживает тот факт, что зависимость является свойством пары классов, а не какого-либо одного из них, т.е. зависимос–ь - это отношение. Отметим, что хотя зависимости между объектами двунаправлены, их не обязательно реализовать в программах как двунаправленные, оставляя ссылки лишь в тех классах, где это необходимо для программы.

Дальнейшие примеры зависимостей между классами приведены на рисунке 2.7. Первый пример показывает зависимость между клиентом банка и его счетами. Клиент банка может иметь одновременно несколько счетов в этом банке, либо вовсе не иметь счета (когда он впервые становится клиентом банка). Таким образом, нужно изобразить зависимость между клиентом и несколькими счетами, что и сделано на рисунке 2.7. Второй пример показывает зависимость между пересекающимися кривыми (в частности, прямыми) линиями. Можно рассматривать 2, 3, и более таких линий, причем они могут иметь несколько точек пересечения. Наконец, третий пример показывает необязательную (optional) зависимость: компьютер может иметь, а может и не иметь мышь.

Контрольные вопросы:

1. Что такое класс?
2. Что такое объект?
3. Каким образом происходит инициализация объекта?
4. Что такое атрибуты класса?
5. Что такое методы класса?

Тема 36. Наследование и полиморфизм

Наследование является одним из основополагающих принципов ООП. В соответствии с ним, класс может использовать переменные и методы другого класса как свои собственные.

Класс, который наследует данные, называется подклассом (subclass), производным классом (derived class) или дочерним классом (child). Класс, от которого наследуются данные или методы, называется суперклассом (super class), базовым классом (base class) или родительским классом (parent). Термины “родительский” и “дочерний” чрезвычайно полезны для понимания наследования. Как ребенок получает характеристики своих родителей, производный класс получает методы и переменные базового класса.

Наследование полезно, поскольку оно позволяет структурировать и повторно использовать код, что, в свою очередь, может значительно ускорить процесс разработки. Несмотря на это, наследование следует использовать с осторожностью, поскольку большинство изменений в суперклассе затронут все подклассы, что может привести к непредвиденным последствиям.

В этом примере, метод turn_on() и переменная serial_number не были объявлены или определены в подклассе Computer. Однако их можно использовать, поскольку они унаследованы от базового класса.

Важное примечание: приватные переменные и методы не могут быть унаследованы.

```
#include <iostream>
using namespace std;
```

```
class Device {
   public:
      int serial_number = 12345678;

      void turn_on() {
         cout «< "Device is»on" << endl;
      }
   private:
      int pincode = 87654321;
};

class Computer: public Device {};

int main() {
   Computer Computer_instance;

   Computer_instance.turn_on();
   cout «< "Serial number i«: " << Computer_instance.serial_number << endl;
   // cout «< "Pin code i«: " << Computer_instance.pincode << endl;
   // will cause compile time error
   return 0;
}
```

Типы наследования

В C ++ есть несколько типов наследования:

публичный (public)- публичные (public) и защищенные (protected) данные наследуются без изменения уровня доступа к ним;

защищенный (protected) — все унаследованные данные становятся защищенными;

приватный (private) — все унаследованные данные становятся приватными.

Для базового класса Device, уровень доступа к данным не изменяется, но поскольку производный класс Computer наследует данные как приватные, данные становятся приватными для класса Computer.

```
#include <iostream>
using namespace std;
class Device {
   public:
      int serial_number = 12345678;
      void turn_on() {
         cout «< "Device is»on" << endl;
      }
};

class Computer: private Device {
   public:
      void say_hello() {
         turn_on();
         cout «< "Welcome to Windows »5!" << endl;
      }
};
```

```
int main() {
   Device Device_instance;
   Computer Computer_instance;

   cout «< "\t Dev»ce" << endl;
   cout «< "Serial number i«: "<< Device_instance.serial_number << endl;
   Device_instance.turn_on();

   // cout «< "Serial number i«: " << Computer_instance.serial_number << endl;
   // Computer_instance.turn_on();
   // will cause compile time error

   cout «< "\t Compu»er" << endl;
   Computer_instance.say_hello();
   return 0;
}
```

Класс Computer теперь использует метод turn_on() как и любой приватный метод: turn_on() может быть вызван изнутри класса, но попытка вызвать его напрямую из main приведет к ошибке во время компиляции. Для базового класса Device, метод turn_on() остался публичным, и может быть вызван из main.

Конструкторы и деструкторы

В C ++ конструкторы и деструкторы не наследуются. Однако они вызываются, когда дочерний класс инициализирует свой объект. Конструкторы вызываются один за другим иерархически, начиная с базового класса и заканчивая последним производным классом. Деструкторы вызываются в обратном порядке.

```
#include <iostream>
   using namespace std;

   class Device {
      public:
         // constructor
         Device() {
            cout «< "Device constructor cal»ed" << endl;
         }
         // destructor
         ~Device() {
            cout «< "Device destructor cal»ed" << endl;
         }
   };

   class Computer: public Device {
      public:
         Computer() {
            cout «< "Computer constructor cal»ed" << endl;
         }
         ~Computer() {
            cout «< "Computer destructor cal»ed" << endl;
         }
   };
```

```
class Laptop: public Computer {
   public:
      Laptop() {
         cout «< "Laptop constructor cal»ed" << endl;
      }
      ~Laptop() {
         cout «< "Laptop destructor cal»ed" << endl;
      }
};

int main() {
   cout «< "\tConstruct»rs" << endl;
   Laptop Laptop_instance;
   cout «< "\tDestruct»rs" << endl;
   return 0;
}
```

Конструкторы: Device -> Computer -> Laptop. Деструкторы: Laptop -> Computer -> Device.

Множественное наследование

Множественное наследование происходит, когда подкласс имеет два или более суперкласса. В этом примере, класс Laptop наследует и Monitor и Computer одновременно.

```
#include <iostream>
using namespace std;

class Computer {
   public:
      void turn_on() {
         cout «< "Welcome to Windows»95" << endl;
      }
};

class Monitor {
   public:
      void show_image() {
         cout «< "Imagine image h»re" << endl;
      }
};

class Laptop: public Computer, public Monitor {};

int main() {
   Laptop Laptop_instance;
   Laptop_instance.turn_on();
   Laptop_instance.show_image();
   return 0;
```

```
}
```

Абстрактный класс

В C++, класс в котором существует хотя бы один чистый виртуальный метод (pure virtual) принято считать абстрактным. Если виртуальный метод не переопределен в дочернем классе, код не скомпилируется. Также, в C++ создать объект абстрактного класса невозможно — попытка тоже вызовет ошибку при компиляции.

```
#include <iostream>
using namespace std;

class Device {
   public:
     void turn_on() {
         cout «< "Device is »n." << endl;
      }
      virtual void say_hello() = 0;
};

class Laptop: public Device {
   public:
     void say_hello() {
         cout «< "Hello wor»d!" << endl;
      }
};

int main() {
   Laptop Laptop_instance;
   Laptop_instance.turn_on();
   Laptop_instance.say_hello();

   // Device Device_instance;
   // will cause compile time error
   return 0;
}
```

Полиморфизм

Когда программисты говорят о C++ и объектно-ориентированном программировании, то очень часто употребляют термин полиморфизм. В общем случае полиморфизм представляет собой способность объекта изменять форму. Если вы разделите этот термин на части, то обнаружите, что поли означает много, а морфизм относится к изменению формы. Полиморфный объект, следовательно, представляет собой объект, который может принимать разные формы. Этот урок вводит понятие полиморфизма и рассматривает, как использовать полиморфные объекты внутри ваших программ для упрощения и уменьшения кода. К концу данного урока вы освоите следующие основные концепции:

- Полиморфизм представляет собой способность объекта изменять форму во время выполнения программы.
- C++ упрощает создание полиморфных объектов.
- Для создания полиморфных объектов ваши программы должны использовать виртуальные (virtual) функции.

- Виртуальная (virtual) функция — это функция базового класса, перед именем которой стоит ключевое слово virtual.
- Любой производный от базового класс может использовать или перегружать виртуальные функции.
- Для создания полиморфного объекта вам следует использовать указатель на объект базового класса.

Котрольные вопросы и задания:

1. Приведите свои примеры класса и объекта (пример, класс деревья, объе–т - береза, сосна). Определите их постоянные и переменные атрибуты. Определите зависимости между объектами
2. Найдите и запишите ответы на вопросы в учебни«е "С/С++ Программирование на языке высокого уро»ня" Т.А. Павловская стр. 178-199

– Что такое элемент класса?
– Чем отличаются спецификаторы private и public
– Что такое конструктор?
– Для чего применяются статистические поля?
– Что такое деструктор?

Тема 37. Исключения в С++ (exception)

Цель: изучить основные понятия обработки исключительных ситуаций, конструкцию try… catch.

Исключение — это событие при выполнении программы, которое приводит к её ненормальному или неправильному поведению.

Существует два вида исключений:

Аппаратные (структурные, SE-Structured Exception), которые генерируются процессором. К ним относятся, например,

– деление на 0;
– выход за границы массива;
– обращение к невыделенной памяти;
– переполнение разрядной сетки.

Программные, генерируемые операционной системой и прикладными программами – возникают тогда, когда программа их явно инициирует. Когда встречается аномальная ситуация, та часть программы, которая ее обнаружила, может сгенерировать, или возбудить, исключение.

Механизм структурной обработки исключений позволяет однотипно обрабатывать как программные, так и аппаратные исключения.

Обработка программных исключений

Фундаментальная идея обработки исключительных ситуаций состоит в том, что функция, обнаружившая проблему, но не знающая как её решить, генерирует исключение в надежде, что вызвавшая её (непосредственно или косвенно) функция сможет решить возникшую проблему. Функция, которая может решать проблемы данного типа, указывает, что она перехватывает такие исключения.

Для реализации обработки исключений в С++ используйте выражения try, throw и catch.

Блок try {…} позволяет включить один или несколько операторов, которые могут создавать исключение.

Выражение throw используется только в программных исключениях и означает, что исключительное условие произошло в блоке try. В качестве операнда выражения throw можно использовать объект любого типа. Обычно этот объект используется для передачи информации об ошибке.

Для обработки исключений, которые могут быть созданы, необходимо реализовать один или несколько блоков catch сразу после блока try. Каждый блок catch указывает тип исключения, которое он может обрабатывать.

Сразу за блоком try находится защищенный раздел кода. Выражение throw вызывает исключение, т.е. создает его.

Блок кода после catch является обработчиком исключения. Он перехватывает исключение, вызываемое, если типы в выражениях throw и catch совместимы. Если оператор catch задает многоточие (…) вместо типа, блок catch обрабатывает все типы исключений.

Поскольку блоки catch обрабатываются в порядке программы для поиска подходящего типа, обработчик с многоточием должен быть последним обработчиком для соответствующего блока try. Как правило, блок catch(…) используется для ведения журнала ошибок и выполнения специальной очистки перед остановкой выполнения программы.

```
try { … // защищенный раздел кода
  throw параметр;
}
catch (параметр) { // обработка исключения }
catch (…) { // обработка остальных исключений }
```

Ниже приведен пример обработки программного исключения. В реальных программах посылка исключения командой throw, как правило, является следствием проверки какого-либо условия.

```
#include <iostream>
using namespace std;
int main() {
  try {
    cout << «Exception: «;
    throw 1;
    cout << «No exception!»;
  } catch (int a) {
    cout << a;
  }
cin.get(); return 0;
}
#include <iostream>
using namespace std;
int main() {
  try {
    cout << «Exception: «;
    //throw 1;
    cout << «No exception!»;
  } catch (int a) {
    cout << a;
```

```
  }
cin.get(); return 0;
}
```

Обработка структурных исключений

Рассмотрим пример программы, генерирующей исключительную ситуацию «деление на 0».

```
#include <iostream>
using namespace std;
int main() {
  int a = 0, b =10;
  cout << b/a << endl;cin.get();
  return 0;
}
```

При попытке запустить программу на выполнение видим следующее:

Необработанное исключение

Для обработки исключительной ситуации необходимо операцию деления поместить в блок защищенного кода:

```
#include <iostream>
using namespace std;
int main() {
  int a = 0, b = 10;
  try {
    cout << b / a << endl;
  }
  catch (…)
  {
    cout << «error»;
  }
  cin.get();
  return 0;
}
```

Для корректного запуска программы необходимо также произвести настройки среды разработки и разрешить обработку структурных исключений. Для этого переходим к меню Имя проекта->Свойства.

Контрольные вопросы:

1. Что такое исключение?
2. Что подразумевает обработка исключительной ситуации?
3. Какая конструкция используется для обработки исключительной ситуации?

Тема 38. Понятие визуального программирования

Цель: сформировать знания о методе визуального программирования, среде быстрой разработки приложений Rad Studio C++Builder.

Библиоте́ка визуа́льных компоне́нтов (англ. Visual Component Library, **VCL**) — объектно-ориентированная библиотека для разработки программного обеспечения, разработанная компанией Borland (на данный момент поддерживается Embarcadero) для поддержки принципов визуального программирования. VCL входит в комплект поставки

Delphi, C++ Builder и Embarcadero RAD Studio и является, по сути, частью среды разработки, хотя разработка приложений в этих средах возможна и без использования VCL. VCL предоставляет огромное количество готовых к использованию компонентов для работы в самых разных областях программирования, таких, например, как интерфейс пользователя (экранные формы и элементы управления — т. н. «контролы», «контроли»), работа с базами данных, взаимодействие с операционной системой, программирование сетевых приложений и прочее.

C++Builder представляет собой систему программирования. Как любая подобная система, C++Builder предназначена для разработки программ и имеет две характерные особенности: создаваемые с ее помощью программы могут работать не только под управлением Windows, а сама она относится к классу инструментальных средств ускоренной разработки программ (Rapid Application Development, RAD).

Первый инструмент RAD создан корпорацией Microsoft и называется Visual Basic. Среда C++Builder, созданная тремя годами позже, вместе с такими продуктами как Visual C++, C++ Builder, J Builder, PowerBuilder также относится к классу инструментов ускоренной разработки программ.

Это ускорение достигается за счет двух характерных свойств C++Builder: визуального конструирования форм и широкого использования библиотеки визуальных компонентов (Visual Component Library, VCL).

Визуальное конструирование форм избавляет программиста от многих аспектов разработки интерфейса программы, так как C++Builder автоматически готовит необходимые программные заготовки и соответствующий файл ресурсов. Программист использует специальное окно, которое называется окном формы, как прототип будущего окна программы и наполняет его компонентами, реализующими нужные интерфейсные свойства (разного рода списки, кнопки, полосы прокрутки и т. п.). После размещения на форме очередного компонента C++Builder автоматически вставляет в связанный с формой модуль ссылку на компонент и корректирует специальный файл описания формы с расширением DFM, который после компиляции преобразуется в ресурсный файл Windows.

Библиотека визуальных компонентов предоставляет программисту огромное разнообразие созданных разработчиками C++Builder программных заготовок, которые немедленно или после несложной настройки готовы к работе в рамках вашей программы. Компоненты характеризуются важным свойством: они включают в себя программный код и все необходимые для его работы данные, что избавляет программиста от рутинной работы по «изобретению велосипедов» — нет нужды писать то, что уже написано (и в подавляющем большинстве — очень грамотно!) другими. Как уже упоминалось, с C++Builder поставляется великое множество компонентов, рассчитанных на самые разные аспекты применения — от простеньких компонентов, создающих поясняющие надписи, до сложных текстовых процессоров или инструментов принятия решений. Если по каким-либо причинам в C++Builder нет компонента С нужной функциональностью, его можно создать средствами самой среды C++Builder и включить затем в VCL (другой вариант — обратиться к ресурсам Интернета, где на тысячах сайтов предлагаются бесплатные, условно-бесплатные и платные компоненты, созданные специально для C++Builder как профессиональными программистами, так и любителями).

Общая характеристика среды визуального программирования

Состав и структура среды.

Интегрированная среда разработки C++Builder 6 представляет собой многооконную систему, определяемую настройками пользовательского интерфейса.

✓ Главное окно (C++Builder6 – Project1)

- главное меню
- набор инструментальных кнопок

- палитра компонентов

- ✓ Окно Редактора формы (Form1).
- ✓ Окно дерева объектов (Object Tree View)
- ✓ Окно Редактора Инспектора объектов (Object Inspector)
- ✓ Окно Редактора кода программы (Unit1.pas)

Главное окно

Осуществляет основные функции управления проектом создаваемой програм. Оно всегда присутствует на экране. Все элементы главного окна располагаются на специальных панелях, в левой или верхней части которых имеются вешки перемещения, позволяющие с помощью мыши перетаскивать панели вместе с находящимися на них элементами. Связано это с функциональностью главного окна: с одной стороны, оно несет в себе элементы, которые всегда должны быть «под рукой» у программиста, с другой, — окно не должно отнимать у остальных окон C++Builder значительного пространства экрана. Свертывание главного окна приводит к исчезновению с экрана других окон C++Builder, а его закрытие означает окончание работы программиста с системой программирования.

В главном окне располагается главное меню C++Builder, набор инструментальных кнопок и палитра компонентов.

Все элементы главного окна располагаются на специальных панелях, в левой М или верхней части которых имеются вешки перемещения, позволяющие с помощью мыши перетаскивать панели вместе с находящимися на них элементами. Любую панель (кроме главного меню) можно убрать из окна (сделать ее невидимой) или «пустить плавать» по экрану в отдельном окне. Для этого нужно лишь «стащить» панель с помощью мыши за вешку из пределов главного окна. Для изменения состава представленных на панели кнопок нужно предварительно щелкнуть на ней правой кнопкой мыши. В появившемся после этого контекстном меню (рис. 1, слева) перечислены названия всех панелей и указан их статус (отмеченные флажками панели видны в главном окне; если флажок снять, панель исчезнет). После выбора команды Customize (Настройка) появляется окно настройки (рис. 1, справа). Теперь можно «стаскивать» с панелей ненужные кнопки или, перейдя на вкладку Commands, выбирать в списке названия нужных кнопок и перетаскивать их на экран.

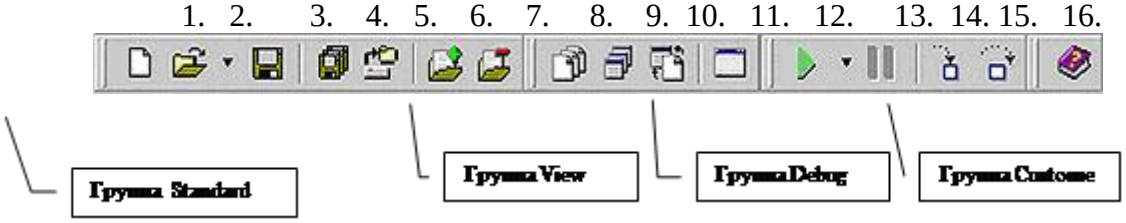
В главном окне находятся:

Главное меню содержит все необходимые средства для управления проектом;

Все пункты главного меню открывают доступ к меню второго уровня (подменю).

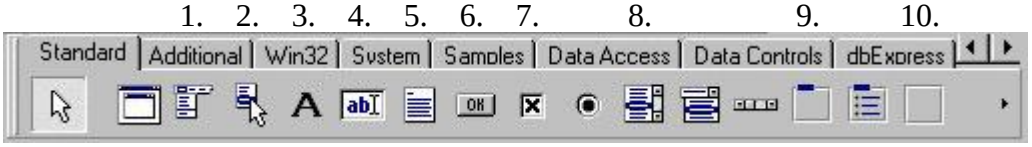
Инструментальные кнопки - открывают быстрый доступ к наиболее важным командам главного меню. По функциональному признаку они разделены на 7 групп. Каждая группа имеет свою отдельную панель.

Инструментальные кнопки представляют собой пиктографическое меню, которое содержит кнопки быстрого доступа к наиболее часто вызываемым опциям главного меню.



кнопка	Главное меню	Быстрый доступ	Команда
1.	File/ New/Other		Открывает доступ в Репозиторию Объектов.
2.	File/ Open File		Открывает существующий файл.
3.	File/ Save File	**Ctrl-S**	Сохраняет файл на диске.
4.	File/ Save All		Сохраняет все файлы проекта.
5.	File/Open Project	**Ctrl-F11**	Открывает созданный ране проект программы.
6.	Project/Add to Project	**Shift-F11**	Добавляет новый файл к проекту.
7.	Project/Remove from Project		Удаляет файл из проекта.
8.	View/Units	**Shift-F12**	Выбор модуля текущего проекта.
9.	View/Forms	**Ctrl-F12**	Выбор формы текущего проекта.
10.	View/Toggle Form/Units	**F12**	Переключатель между Конструктором формы и кодом.
11.	File/New/Form		Создает новую форму и добавляет ее к проекту.
12.	Run/Run	**F9**	Компилирует и выполняет программу.
13.	Run/Program Pouse		Реализует паузу в работе отлаживаемой программы.
14.	Run/Trace Into	**F7**	Пошаговая трассировка программы и подпрограмм.
15.	Run/Step Over	**F8**	Пошаговая трассировка программы, но без прослеживания работы вызываемых подпрограмм.
16.	Help/C++Builder Help		Встроенная справочная служба C++Builder.

Палитра компонентов – занимает правую часть главного окна и имеет вкладки, обеспечивающие быстрый поиск нужного компонента. Под компонентом понимается некий функциональный элемент, содержащий определенные свойства и размещаемый программистом в окне формы. С помощью компонентов создается каркас программы - основные видимые на экране элементы: окна, кнопки, списки и т.п. Палитра компонентов может настраиваться в специальном редакторе (на любом компоненте нажать правую кнопку мыши и выбрать свойства Properties).



Окно редактора формы

Представляет собой проект окна будущей программы. Данное окно формируется автоматически при формировании нового приложения и вначале содержит стандартные для Windows элементы: кнопки вызова системного меню, развертывания, свертывания и закрытия окна, строку заголовка и габаритную рамку.

Окно дерева объектов

Предназначено для наглядного отображения взаимосвязей между отдельными компонентами, размещенными на активной форме или в активном модуле данных.

Окно редактора инспектора объектов

Каждый размещаемый на форме объект характеризуется набором параметров (положением, размером, цветом и т.д.), которые могут задаваться с помощью редактора инспектора объектов.

Объект – замкнутый, обособленный фрагмент программы, сочетающий в себе данные (поля) и действия над ними (методы). Объекты в виду своей самодостаточности легко могут переноситься из одной программы в другую и использоваться программистом как готовый материал, облегчающий разработку приложений одновременно повышая его качество и профессионализм.

Класс – специальный тип данных, используемый для описания объекта. Класс, как особый тип записи, обязательно содержит три элемента: поля (данные), методы (процедуры) и свойства (реализуют механизм доступа к полям). Как и любой другой тип, класс служит лишь типовым описанием для создания конкретных экземпляров реализации – объектов.

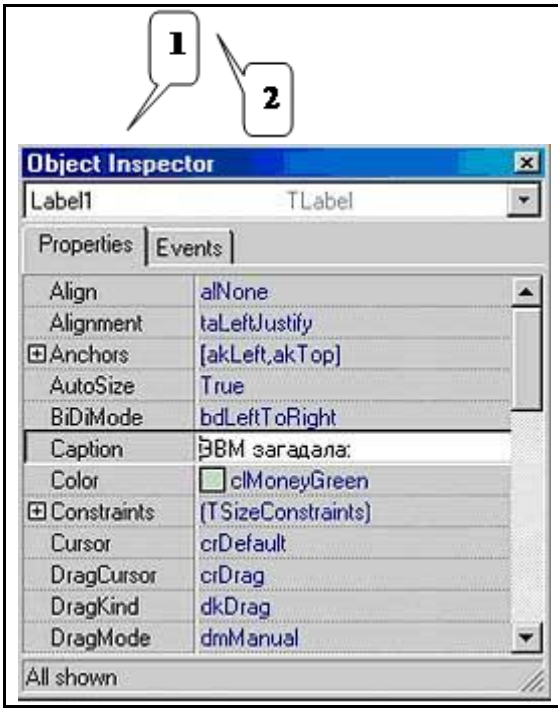
C++Builder содержит ряд стандартных компонентов – объектов того или иного класса, подключаемых по желанию программиста путем вызова их с **Палитры компонентов**. Библиотека визуальных компонентов (Visual Component Library, **VCL**) – это большое количество классов, предназначенных для быстрой разработки приложений.

Библиотеки компонентов для разных версий C++Builder строятся по принципу расширения. Так в первой версии было около 70 компонентов, в то время как в составе C++Builder 6 – 373 компонента, распределенные по функциональным группам (страницы палитры).

Для создания интерфейса приложений система C++Builder предлагает обширный набор визуальных компонентов, основные из которых располагаются на страницах **Standard**, **Additional** и **Win32** Палитры компонентов.

Свойства позволяют управлять внешним представлением компонента в создаваемом приложении.

Поля свойств содержат конкретные значения, по которым объекты принимают тот или иной вид. Установка большинства свойств производится на этапе проектирования при помощи Инспектора объектов.



Правая колонка Инспектора объектов содержит перечень свойств, левая – значения.

Для объекта «надпись» (Label1) заданы значения полям:

(**1**) «ЭВМ загадала:» – свойство Caption (текст надписи);

(**2**) св. зеленый – цвет фона (свойство Color).

Значение свойствам можно присвоить и непосредственно в коде программы, используя **оператор присваивания** «**:=**»:

Label.Caption:='Введите 2-й операнд: Y=';

Слева от оператора присваивания стоит свойство, справа – его значение. Если присваивается текстовая информация, то в C++Builder, как и в C(C++), она записывается между одиночными апострофами. В конце оператора – «**;**».

Методы задают объектам определенное поведение. В общем случае это ответ на конкретное событие компьютерной жизни: клик мыши, нажатие клавиши клавиатуры, отсчет интервала времени.

Перечень событий, для которых формируются методы данных объектов приведены во вкладке **Events** инспектора объектов.

Как правило, описание метода оформляется в коде программы в виде процедуры, причем имя метода отражает суть события вызова:

```
void TForm1.Button1Click(Sender: TObject);
  {
    Close;
  };
```

В приведенном примере метод обработки события «клик мыши по кнопке» закроет окно формы. Среда IDE C++Builder в значительной степени автоматизирует процесс написания кода.

Свойства объектов, предъявляемые C++Builder по умолчанию в окне Инспектора Объектов, программисту часто приходится изменять. Предлагаю некоторые рекомендации.

Объект/Свойство	**Значение**	
Форма · **Position**	poScreenCenter	Расположение формы при открытии/запуске приложения симметрично относительно экрана.
· **Align**		Способ выравнивания компонента относительно своего контейнера.
	alBottom	Внизу. Объект прижимается к нижней кромке контейнера и растягивается по всей длине. При распахивании окна формы, компонент всегда будет

		позиционироваться внизу, а не «зависать» на определенном месте.
Панель **BevelOther**	bvNone	Нет внешней кромки. Панель сливается с формой.
Name		Имя, под которым компонент вызывается в коде программы.
Однострочная надпись (метка - Label)		
· **Caption**		Текст, отображаемый в надписи.
Однострочный редактор (Edit)		
· **Text**		Текст, отображаемый однострочным редактором.
Многострочный редактор (Memo)		
· **Lines**		Текст, отображаемый многострочным редактором.
· **ScrollBars**	ssBoth	Оба. Две полосы прокрутки на компоненте.
· **Align**	alClient	Компонент занимает всю оставшуюся часть формы, независимо от размеров окна.
· **WordWrap**	False	Отказ от переноса слов на другую строку редактора.
Стандартная кнопка BitBtn **Kind**		Сорт. Определяет типовые кнопки с заданной функциональностью.
	bkClose	Закрыть приложение.
	bkOk	Дублирует Enter (ввод информации)

Окно редактора кода программы

Окно кода предназначено для создания и редактирования текста программы. Этот текст составляется по специальным правилам и описывает алгоритм работы программы. Совокупность правил записи текста называется языком программирования. В системе C++Builder используется язык программирования C++Builder, который представляет собой значительно расширенную версию широко распространенного языка Паскаль, впервые предложенного швейцарским профессором Н. Виртом еще в 1970 г. и усовершенствованного сотрудниками корпорации Borland (созданные ими языки назывались Turbo C(C++), Borland C(C++) и Object C(C++); последний в версии 7 переименован в C++Builder). Несмотря на то, что визуальная среда C++Builder берет на себя многие рутинные аспекты программирования, знание языка C++Builder является непременным условием для любого программиста, работающего с этой системой.

Первоначально окно кода содержит минимальный исходный текст, обеспечивающий нормальное функционирование пустой формы в качестве полноценного окна Windows-программы. В ходе работы над проектом программист вносит в него необходимые дополнения, чтобы придать программе нужную функциональность. Поскольку для создания даже простых программ вам понадобится создавать и изменять (редактировать) код программы, ниже описываются основные приемы работы с окном кода:

Чтобы вставить в окно новую строку (строки), нужно сначала с помощью клавиш управления курсором или щелчком мыши установить текстовый курсор (мигающую вертикальную черту) в нужное место, а затем ввести текст с клавиатуры. Обычно текст кода программы располагается в нескольких строках. Для перехода на новую строку используйте клавишу Enter.

Контрольные вопросы:

1. Назначение среды визуального программирования
2. Структура среды визуального программирования.

3. Для чего необходим Инспектор объектов (Object Inspector)?
4. В каком файле хранятся данные Окна формы?
5. Для чего предназначен Редактор кода программы (Code Editor)?

Тема 39. Компоненты ввода и отображения текстовой информации

В библиотеке визуальных компонентов **C++Builder** существует множество компонентов, позволяющих отображать, вводить и редактировать текстовую информацию. В таблице ниже приведен их перечень для C++Builder 6 с краткими характеристиками и указанием основных параметров, содержащих отображаемый или вводимый текст. В этой таблице не указаны аналогичные элементы отображения и редактирования текстов, содержащихся в базах данных.

Таблица. Компоненты ввода и отображения текстовой информации

Компонент	**Страница**	**Описание**
Label (метка)	Standard	Отображение текста, который не изменяется пользователем. Никакого оформления текста не предусмотрено, кроме цвета метки и текста. Основное свойство Caption.
StaticText (метка с бордюром)	Additional	Подобен компоненту Label, но обеспечивает возможность задания стиля бордюра. Основное свойство Caption.
Panel (панель)	Standard	Компонент является контейнером для группирования органов управления, но может использоваться и для отображения текста с возможностями объемного оформления. Основное свойство Caption.
Edit (окно редактирования)	Standard	Отображение, ввод и редактирование однострочных текстов. Имеется возможность оформления объемного бордюра. Основное свойство Text.
MaskEdit (окно маскированного редактирования)	Additional	Используется для форматирования данных или для ввода символов в соответствии с шаблоном. Основные свойства Text и EditText.
LabeledEdit (окно редактирования с привязанной к нему меткой)	Additional	Комбинация Editи Label. Основные свойства Text и EditLabel.Caption. Только в C++Builder 6.
Memo (многострочное окно редактирования)	Standard	Отображение, ввод и редактирование многострочных текстов. Имеется возможность оформления объемного бордюра. Основное свойство Lines.
RichEdit (многострочное окно редактирования в формате RTF)	Win32	Компонент представляет собой окно редактирования в стиле Windowsв обогащенном формате RTF, позволяющее производить выбор атрибутов шрифта, поиск текста и многое другое. Основное свойство Lines.

ListBox (окно списка)	Standard	Отображение стандартного окна списка Windows, позволяющего пользователю выбирать из него пункты. Основное свойство Items.
CheckListBox (список с индикаторами)	Additional	Компонент является комбинацией списка ListBoxи индикаторов CheckBox.
ValueListEditor (список специального вида)	Additional	Окно редактирования списков строк вида «имя = значение». Основные свойства: Keys - имена, Values - значения. Только в C++Builder 6.
ComboBox (редактируемый список)	Standard	Объединяет функции ListBoxи Edit. Пользователь может либо ввести текст, либо выбрать его из списка. Основное свойство Items.
StringGrid (таблица строк)	Additional	Отображения текстовой информации в таблице из строк и столбцов с возможностью перемещаться по строкам и столбцам и осуществлять выбор. Основное свойство — Cells.

Помимо перечисленных компонентов отображать текстовые надписи можно непосредственно на свойстве **Canvas** (холст) любого компонента, имеющего это свойство, в частности, непосредственно на форме. Например, оператор вида:

Canvas->TextOut(60,16,"Canvas");

обеспечивает печать, начиная с точки с координатами (60, 16), текста «Canvas». Но это неудобно, так как при этом теряются преимущества визуального проектирования и приходится рассчитывать координаты размещения надписи.

Во всех компонентах шрифт текста, его размер, стиль (жирный, курсив и т.п.), цвет определяются свойством Font, которое имеет множество подсвойств, устанавливаемых в процессе проектирования или программно во время выполнения приложения.

Компонент Edit

Компонент Edit – это простейшее окна редактирования, позволяющее вводить и отображать текстовую информацию в свойстве Text типа AnsiString, но обладает множеством полезных функций. Компонент Edit поддерживает популярные комбинации горячих клавиш: Ctrl+C (копирование текста), Ctrl+X (вырезание текста), Ctrl+V (вставка текста), Ctrl+Z (отмена последнего редактирования текста).

Выравнивание текста в поле Edit невозможно, и он выравнивается по левому краю. Перенос строк так же невозможен. Текст, который не помещается в длину окна Edit, сдвигается за рамки и, чтобы его отобразить, достаточно перемещать курсор мыши в сторону.

Свойства Edit

BorderStyle – позволяет редактировать внешний вид окна редактирования Edit. Имеет несколько значений:

bsSingle – отображает рамку вокруг окна (по умолчанию);

bsNone – скрывает рамку, делая окно редактирования плоским.

AutoSize – автоматическое выставление высоты окна редактирования.

AutoSelect – при фокусе окна Edit автоматически выделяет весь текст. Имеет несколько значений:

SelLength – определяет длину выделенного текста;

SelStart – возвращает индекс первого выделенного символа текста;

SelText – возвращает выделенный текст.

Readonly – запрещает редактировать текст в окне Edit.

Text – строка типа AnsiString. Автоматически преобразовывает числовые данные в строку.

MaxLength – задаёт максимальное количество вводимых символов в окно Edit. Если установить MaxLength в 0, то длина ввода текста не будет ограничиваться.

Modified – определяет было ли редактирование текста в окне Edit или нет.

PasswordChar – маскирует все вводимые символы, превращая окно Edit в окно ввода пароля. Принимает любое значение, которое будет заменять каждый символ в строке. По умолчанию указано «#0», т.е. нулевой символ.

Свойства LabeledEdit

Компонент LabeledEdit имеет все свойства Edit, но есть и некоторые дополнения.

LabelPosition – устанавливает расположение своей метки и имеет несколько значений:

lpAbove – расположение по верхнему левому краю;

lpBelow – расположение по нижнему левому краю;

lpLeft – расположение по левому краю;

lpRight – расположение по правому краю.

Компонент LabeledEdit появился в C++Builder 6 и значительно упростил разработку приложений, т.к. в большинстве случаев Edit всегда использовалось с Label для пояснения назначения окна редактирования.

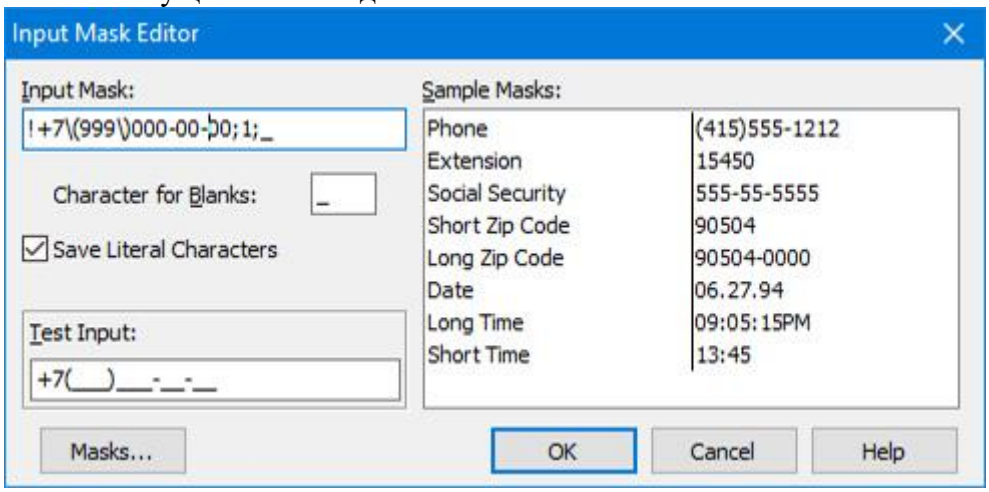
Компонент MaskEdit

Компонент MaskEdit по функционалу и назначению схож с Edit и LabeledEdit, но имеет одно преимущество – позволяет задавать маску в свойстве EditMask. Это позволяет ограничить ввод текста, обеспечив получение синтаксически безошибочных данных, например, ввод номера телефонов, контактных данных и т.п. Маска состоит из трёх частей, разделяющихся точкой с запятой «;»:

В первую часть записываются специальные символы, которые разрешается вводить в каждую позицию, и символы маски.

Вторая часть содержит 1 или 0, т.е. надо или нет, чтобы символы маски добавлялись в свойство Text компонента MaskEdit.

В третьей части нужно указывать символ, отвечающий за обозначение позиции, в которой не был осуществлён ввод.



Символы шаблона маски

Символ	Описание
!	Если указан символ, то EditText добавляет пробелы в недостающие символы. В противном случае пробелы будут размещаться в конце.
>	Все последующие за «>» символы должны вводиться в верхнем регистре до окончания маски или символа «<».

<	Все последующие за «<» символы должны вводиться в нижнем регистре до окончания маски или символа «>».
<>	Не учитывает регистр вводимых символов.
\	Указывает, что следующим за ним символ должен быть буквенным.
L	Указывает, что следующим за ним символ должна быть буква.
l	Указывает, что следующим за ним символ должна быть только буква или ничего.
A	Указывает, что следующим за ним символ должна быть буква или цифра.
a	Указывает, что следующим за ним символ должна быть буква, или цифра, или ничего.
C	Указывает, что следующим за ним символ может быть любым символом.
c	Указывает, что следующим за ним символ может быть любым символом или ничего.
0	Указывает, что следующим за ним символ должна быть цифра.
9	Указывает, что следующим за ним символ должна быть цифра или ничего.
#	Указывает, что следующим за ним символ должна быть цифра, знак «+», знак «-» или ничего.
:	Используется для разделения времени на часы, минуты и секунды.
/	Используется для разделения даты на дни, месяцы и годы.
_	Автоматически вставляет пробел в текст.

Текстовая метка Label

Компонент Label – это простая текстовая метка, предназначенная для вывода текста. Свойство Caption типа AnsiString позволяет принимать текст и выводить его в приложении. Задавать значение свойству можно как через «Инспектор объектов», так и программно. Свойство Caption автоматически преобразовывает любой тип в строку.

```
Label1->Caption = "Здесь какой-то текст";
Label2->Caption = 3.6; //Преобразовывает число в строку
```

Компонент Label позволяет вносить в метку так же и смешанную информацию, т.е. принимать строки символов и числа. Для этого необходимо применить функции преобразования FloatToStr (из вещественного в строки) и IntToStr (из целочисленного в строки). При формировании текста из нескольких частей их следует объединять операцией «+» (склеивание или конкатенация). Например, в приложении рассчитывается математическая операция и её нужно вывести в текстовую метку Label:

```
int project = 6;
float sum = 2450850.65;
Label1->Caption = "За текущий месяц выполненно проектов: " + IntToStr(project) + ", на сумму: " + FloatToStr(sum);
// => За текущий месяц выполненно проектов: 6, на сумму: 2450850,65
```

Свойства Label

Color – задаёт цвет фона. Если не указывать, то по умолчанию выставляется с цветом формы, имитируя прозрачность фона компонента Label.

Font – задаёт тексту цвет, шрифт, размер и прочие параметры.

Top, Left, Height, Width, Aline – задают координаты (размещение) компонента, размеры на форме и изменение при изменении родительского компонента.

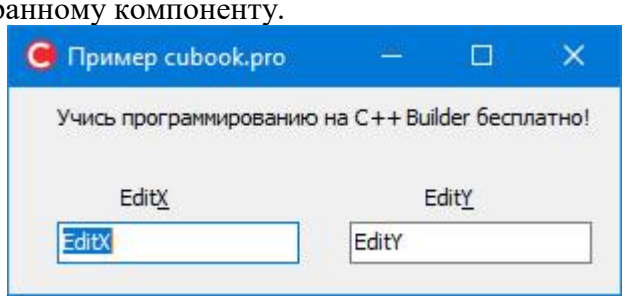
AutoSize – устанавливает автоматическое изменение размера. Если значение установлено в true, то горизонтальный и вертикальный размеры компонента Label будет изменяться по соотношению его текста. Если установлено false, то текст будет выравниваться внутри компонента Label значением свойства Alignment.

Alignment – задаёт расположение текста по левому или правому краю, или же по центру области метки.

WordWrap – разрешает перенос по словам на новую строчку при длинных надписях, если значение установлено в true. Если AutoSize установить в false и выставить небольшую длину компонента Label при AutoSize установленном false, то текст, не влезающий в область метки, будет обрезаться.

FocusControl – превращает компонент Label в некое подобие управляющего элемента в фокусируемый компонент. Если в свойстве Caption поставить символ амперсанд «&» в тексте перед каким-нибудь символом, то он будет подчёркнутым. Теперь этот символ становится управляющим элементом.

В свойстве FocusControl компонента текстовой метки Label из выпадающего списка нужно выбрать нужный компонент (список соответствует всем компонентам размещённых на форме приложения). Свойство ShowAccelChar должно быть установлено в true. Комбинация кнопок Alt+«подчеркнутый символ» позволят выполнить ускоренный доступ к выбранному компоненту.



Текстовая метка StaticText

Компонент StaticText – это текстовая метка, предназначенная для вывода текста. Свойство Caption, как и у компонент Label, имеет тип AnsiString и позволяет принимать не только текст, но и смешанную информацию. Также автоматически преобразовывает любой тип в строку.

```
StaticText1->Caption = "Здесь какой-то текст";
StaticText2->Caption = 3.6; //Преобразовывает число в строку
```

При использовании смешанной информации в свойстве Caption компонента StaticText необходимо применять функции преобразования FloatToStr (из вещественного в строки) и IntToStr (из целочисленного в строки). Компонент StaticText поддерживает операцию объединения «+» (склеивание или конкатенация).

```
int project = 12;
float sum = 1500750.11;
StaticText1->Caption = "За текущий месяц выполненно проектов: " + IntToStr(project) + ", на сумму: " + FloatToStr(sum);
// => За текущий месяц выполненно проектов: 12, на сумму: 1500750,11
```

Свойства StaticText

StaticText поддерживает все свойства, что и метка Label, но имеет одно дополнение.

Свойство Border – устанавливает рамку вокруг текста и имеет несколько значений:

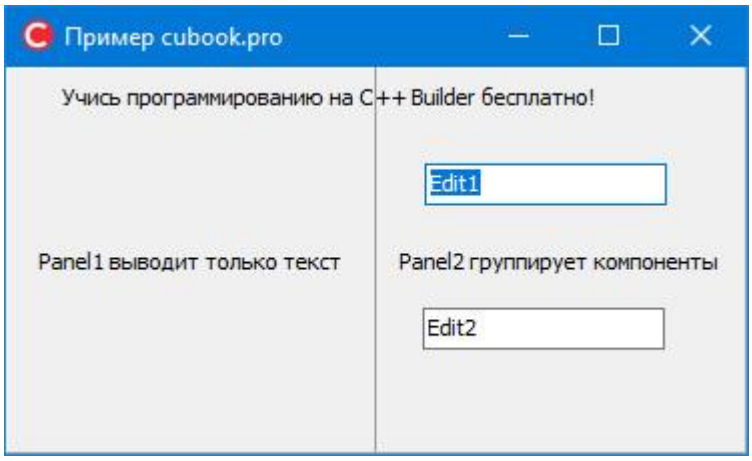
sbsNone – скрывает рамку и метка StaticText становится как Label;

sbsSingle – устанавливает чёрную рамку;

sbsSunken – устанавливает тип утопленного текста в область StaticText.

Панель Panel

Компонент Panel – это панель, позволяющая компоновать компоненты в своей области на форме приложения, но и прекрасно подходит для отображения текста.



Компонент Panel обладает всем функционалом, что Label и StaticText. Свойство Caption имеет тип AnsiString и автоматически преобразует вводимое значение в строку.

```
Panel1->Caption = "Здесь какой-то текст";
Panel2->Caption = 7.2; //Преобразовывает число в строку
```

Поддерживает операцию объединения «+» (склеивание или конкатенация) при вводе смешанной информации:

```
int project = 7;
float sum = 810330.41;
Panel1->Caption = "За текущий месяц выполненно проектов: " + IntToStr(project) + ", на сумму: " + FloatToStr(sum);
// => За текущий месяц выполненно проектов: 7, на сумму: 810330,41
```

Свойства Panel

Все свойства панели Panel идентичны свойствам Label и StaticText.

Memo

Компонент Memo – это многострочное окно редактирования текста. Компонент Memo обладает теми же функциями, что и Edit. Для быстрого редактирования текста имеются комбинации «горячих» клавиш:

Ctrl+C – копирование выделенного текста.

Ctrl+V – вставка текста.

Ctrl+X – вырезание выделенного текста.

Ctrl+Z – отмена последней команды редактирования.

Свойство Font компонента Memo одинаковый формат (шрифт и его атрибуты) для всего текста. Отредактированный текст при копировании из окна Memo не будет содержать элементы форматирования. Или отредактированный текст скопированный, например, из Word с элементами форматирования, то при вставке в окно Memo, будет вставлен как текст и к нему применяться те стили, которые указаны в свойстве Font.

Свойства Memo

Lines типа TStrings – содержит текст окна и представляет из себя список строк. Через «Инспектор Объектов» можно задать начальный текст в Lines. Lines имеет ряд параметров:

Add или Append – добавляет текст в конец строки;

Clear – очистка Memo от всего текста.

Alignment – устанавливает выравнивание текста.

WordWrap – разрешает перенос длинных строк.

Readonly – запрещает редактировать текст в Memo.

MaxLength – устанавливает максимально допустимое количество символов вводимого текста.

WantTabs – устанавливает действие над кнопкой Tab. Если установлено true, при нажатии кнопки Tab в Memo, будет добавлена табуляция. Если установлено false, то будет переключаться фокус между компонентами на форме.

ScrollBars – устанавливает полосу прокрутки в окне. В компоненте Memo полосы всегда отображаются, независимо от размера окна и текста. ScrollBars имеет ряд параметров:

ssNone – скрывает полосу прокрутки;

ssHorizontal – устанавливает горизонтальную прокрутку;

ssVertical – устанавливает вертикальную прокрутку;

ssBoth – устанавливает и горизонтальную и вертикальную.

Count – отображает количество строк текста в Memo.

Strings[int Index] типа AnsiString – предоставляет доступ к любой строке текста. В C++ Builder, индекс начинаются с 0.

```
Memo1->Lines->Strings[0]; //Текст первой строки
```

SelStart – указывает индекс текста, на котором установлен указатель мыши.

CaretPos – отображает структуру:

X – содержит индекс символа перед курсором мыши;

Y – отображает индекс строки, где расположен курсор.

Пример добавления текста в Memo

```
Edit1->Text = "C++ Builder";
Memo1->Lines->Add("Учимся программировать");
Memo1->Lines->Add("Язык " + Edit1->Text);
```

RichEdit

Компонент RichEdit – это удобное многострочное окно редактирования текста. Компонент RichEdit имеет большинство функций компонентов Memo и Edit. Так же есть наличие комбинации «горячих» клавиш. Окно редактирования RichEdit обладает множеством сообщений Windows, которые позволяют управлять текстом.

Компонент RichEdit позволяет редактировать текст формате RTF. Свойство SelAttributes типа TTextAttributes позволяет менять атрибуты добавляемого текста и имеет ряд подсвойств:

Color – устанавливает цвет текста.

Name – устанавливает имя шрифта.

Size – устанавливает размер.

Style – определяет стиль текста.

С помощью диалога выбора шрифта FontDialog можно легко менять атрибуты текста в RichEdit.

Свойства RichEdit

Lines типа TStrings – содержит текст окна и представляет из себя список строк. Через «Инспектор Объектов» можно задать начальный текст в Lines. Lines имеет ряд параметров:

Add или Append – добавляет текст в конец строки;

Clear – очистка RichEdit от всего текста.

DefAttributes – содержит начальные атрибуты текста. Они действуют на всём промежутке времени, даже после изменения атрибутов в SelAttributes, которые могут быть методом Assign. Этот метод можно присвоить к атрибутам SelAttributes, чтобы можно было быстро вернуть начальный стиль. Свойство DefAttributes доступно во время выполнения приложения. Это очень полезно, т.к. его атрибуты можно назначить в обработчике события на создание формы OnCreate.

Paragraph типа TParaAttributes – служит для выравнивания, отступов и т.д. текста в пределах текущего абзаца и имеет ряд свойств:

Значение	Описание
Alignment	Устанавливает выравнивание текста:

	taLeftJustify – по левому краю; taRightJustify – по правому краю; taCenter (по центру).
Firstlndent	Устанавливает размер красной строки в пикселях.
Numbering	Устанавливает маркерный список: nsNone –маркеры отключены; nsBullet – добавляет маркеры к списку.
Leftlndent	Устанавливает отступ от левого поля в пикселях.
Rightlndent	Устанавливает отступ от правого поля в пикселях.

```
RichEdit1->Paragraph->Alignment = taLeftJustify; //Выравнивание по левому краю
RichEdit1->Paragraph->Alignment = taCenter; //Выравнивание по центру
RichEdit1->Paragraph->Alignment = taRightJustify; //Выравнивание по правому краю
RichEdit1->Paragraph->Numbering = nsBullet; //Делает список маркированным
RichEdit1->Paragraph->Numbering = nsNone; //Убирает маркер
```

WantTabs – устанавливает действие над кнопкой Tab. Если установлено true, при нажатии кнопки Tab в RichEdit, будет добавлена табуляция. Если установлено false, то будет переключаться фокус между компонентами на форме.

Alignment – устанавливает выравнивание текста.

WordWrap – разрешает перенос длинных строк.

MaxLength – устанавливает максимально допустимое количество символов вводимого текста.

Readonly – запрещает редактировать текст в RichEdit.

ScrollBars – устанавливает полосу прокрутки в окне. В компоненте RichEdit полосы отображаются автоматически, когда текст не помещается в окно. ScrollBars имеет ряд параметров:

ssNone – скрывает полосу прокрутки;

ssHorizontal – устанавливает горизонтальную прокрутку;

ssVertical – устанавливает вертикальную прокрутку;

ssBoth – устанавливает и горизонтальную и вертикальную.

Count – отображает количество строк текста в RichEdit.

Strings[int Index] типа AnsiString – предоставляет доступ к любой строке текста.

SelStart – указывает индекс текста, на котором установлен указатель мыши.

CaretPos – отображает структуру:

X – содержит индекс символа перед курсором мыши;

Y – отображает индекс строки, где расположен курсор.

StringGrid – таблица строк и столбцов

Компонент StringGrid – это таблица, которая содержит строки и столбцы, напоминающая таблицы Excel. Данные в ячейках таблицы могут быть использованы как только чтение, так и с возможностью ввода/редактирования. Если таблица компонента StringGrid слишком большая и не помещается в размеры формы, то есть возможность подключить полосы прокрутки по горизонтали, вертикали и совместный режим. При необходимости можно задать фиксированные строки и столбы, причём любое количество, которые не будут прокручиваться. Достаточно полезный функционал, если таблица содержит большое количество данных.

Компонент StringGrid – это отличное решение для отображения структурированных текстовых данных. Но StringGrid позволяет отображать и графические данные.

Основные свойства

Значение	Описание

Cells [int ACol][int ARow]	Строка таблицы, которая содержится в ячейке с индексами столбца ACol и строки ARow.
Cols[int Index]	Список строк, связанные с объектом, которые содержатся в столбце с индексом Index.
Rows [int Index]	Список строк, связанные с объектом, которые содержатся в строке с индексом Index.

Все основные свойства компонента StringGrid доступны во время программирования приложения. Задавать текст в ячейки можно как программно на все сразу, так и по отдельности сразу по строкам и столбцам методов класса TStrings.

Дополнительные свойства

ColCount – задаёт количество столбцов.

RowCount – задаёт количество строк.

FixedCols – задаёт количество фиксированных столбцов.

FixedRows – задаёт количество фиксированных строк.

FixedColor – задаёт цвет фона фиксированных строк и столбцов.

LeftCol – первый видимый столбец при прокручивании таблицы.

TopRow – первая видимая строка при прокручивании таблицы.

ScrollBars – включает отображение полос прокрутки содержимого таблицы. Если содержимое таблицы умещается в форму приложения, то полосы прокрутки автоматически скрываются.

Options – дополнительные настройки взаимодействия с таблицей:

goFixedVertLine и goFixedHorzLine – отображает разделительные горизонтальные и вертикальные линий в фиксированных ячейках;

goVertLine и goHorzLine отображает разделительные горизонтальные и вертикальные линий в нефиксированных ячейках;

goCoISizing и goRowSizing – позволяют изменять размеры строк и столбцов с помощью указателя мыши;

goColMoving и goRowMoving – позволяет перемещать строки и столбцы;

goEditing – позволяет редактировать содержимое таблицы.

Свойства Row и Col показывают индексы строки и столбца ячейки. Так же есть возможность множественного выделения ячеек, строк и столбцов.

В компоненте StringGrid есть полезное событие OnSelectCell, которое срабатывает в момент активации ячейки (нажатие на неё указателем мыши). В обработчик события передаются несколько параметров:

ACol – номер столбца, типа int;

ARow – номер строки, типа int;

CanSelect – допустимость выбора, типа bool. При указании значения false, запрещает выделять указанную ячейку.

Тема 40. Проектирование пользовательского интерфейса.

MainMenu – главное меню. MainMenu, расположенный на странице Standard, это не визуальный компонент, т.е. место его размещения на форме в процессе проектирования не имеет никакого значения для пользователя - он все равно увидит не сам компонент, а только меню, сгенерированное им.

Обычно на форму помещается один компонент MainMenu. В этом случае его имя автоматически заносится в свойство формы Menu. Но можно поместить на форму и несколько компонентов MainMenu с разными наборами разделов, соответствующими различным режимам работы приложения. В этом случае во время проектирования свойству Menu формы присваивается ссылка на один из этих компонентов. А в процессе выполнения

в нужные моменты это свойство можно изменять, меняя соответственно состав главного меню приложения.

Основное свойство компонента Items. Его заполнение производится с помощью Конструктора Меню, вызываемого двойным щелчком на компоненте MainMenu или нажатием кнопки с многоточием рядом со свойством Items в окне Инспектора Объектов. В результате откроется окно. В этом окне можно спроектировать меню.

Свойство Caption обозначает надпись раздела. Заполнение этого свойства подчиняется тем же правилам, что и заполнение аналогичного свойства в кнопках, включая использование символа амперсанда для обозначения клавиш быстрого доступа. Если в качестве значения Caption очередного раздела вести символ минус "-", то вместо раздела в меню появится разделитель.

Свойство Name задает имя объекта, соответствующего разделу меню. Очень полезно давать этим объектам осмысленные имена, так как иначе вскоре можно запутаться в ничего не говорящих именах типа N21. Куда понятнее имена типа MFile, MOpen, MSave и т.п.

Свойство Shortcut определяет клавиши быстрого доступа к разделу меню - «горячие» клавиши, с помощью которых пользователь, даже не заходя в меню, может в любой момент вызвать выполнение процедуры, связанной с данным разделом. Чтобы определить клавиши быстрого доступа, надо открыть выпадающий список свойства Shortcut в окне Инспектора Объектов и выбрать из него нужную комбинацию клавиш. Эта комбинация появится в строке раздела меню.

Свойство Default определяет, является ли данный раздел разделом по умолчанию своего подменю, т.е. разделом, выполняемым при двойном щелчке пользователя на родительском разделе. Подменю может содержать только один раздел по умолчанию, выделяемый жирным шрифтом.
Свойство Break используется в длинных меню, чтобы разбить список разделов на несколько столбцов. Возможные значение Break: mbNone - отсутствие разбиения меню (это значение принято по умолчанию), mbBarBreak и mbBreak - в меню вводится новый столбец разделов, отделенный от предыдущего полосой (mbBarBreak) или пробелами (mbBreak).

Свойство Checked, установленное в true, указывает, что в разделе меню будет отображаться маркер флажка, показывающий, что данный раздел выбран. В C++Builder 6 для разделов меню введено новое свойство - AutoCheck. Если его установить в true, то при каждом выборе пользователем данного раздела маркер будет автоматически переключаться, указывая то на выбранное состояние, то на отсутствие выбора. В предшествующих версиях C++Builder маркер сам по себе не переключается и в обработчик события OnClick раздела надо вставлять оператор типа (в приведенном операторе подразумевается, что раздел меню назван MAutoSave):

MAutoSave->Checked =! MAutoSave->Checked;

Еще одним свойством, позволяющим вводить маркеры в разделы меню, является Radioltem. Это свойство, установленное в true, определяет, что данный раздел должен работать в режиме радиокнопки совместно с другими разделами, имеющими то же значение свойства Grouplndex. По умолчанию значение GroupIndex равно 0. Но можно задать его больше нуля и тогда, если имеется несколько разделов с одинаковым значением Grouplndex и с Radioltem = true, то в них могут появляться маркеры флажков, причем только в одном из них. Если задать программно в одном из этих разделов Checked = true, то в остальных разделах Checked автоматически сбросится в false.

Для подобных групп разделов, работающих как радиокнопки, можно установить в true свойство AutoCheck. Тогда при щелчке пользователя на невыбранном разделе все будет работать нормально: этот раздел включится, а ранее включенный выключится. Но если пользователь щелкнет на выбранном разделе, то он выключится и окажется, что все разделы данной группы выключены. Если по смыслу это допустимое состояние, то все хорошо. Но если в любом случае один из разделов должен быть включен, то надо отказаться от

использования свойства AutoCheck (установить его в false) и переключать свойство Checked разделов программно. Например, в обработчики щелчков этих разделов можно ввести оператор:

```
if(!((TMenuItem *)Sender)->Checked)
((TMenuItem *)Sender)->Checked = true;
```

Он переключает маркер только в случае, если сделан щелчок на не выбранном разделе меню. Оператор записан в общем виде и в качестве источника события может фигурировать любой раздел меню.

Описанные маркеры флажков в режиме радиокнопок и в обычном режиме используются для разделов меню, представляющих собой различные опции, взаимоисключающие или совместимые. Для каждого раздела могут быть установлены во время проектирования или программно во время выполнения свойства Enabled (доступен) и Visible (видимый). Если установить Enabled = false, то раздел будет изображаться серой надписью и не будет реагировать на щелчок пользователя. Если же задать Visible = false, то раздел вообще не будет виден, а остальные разделы сомкнутся, заняв место невидимого. Свойства Enabled и Visible используются для того, чтобы изменять состав доступных пользователю разделов в зависимости от режима работы приложения.

В C++ Builder предусмотрена возможность ввода в разделы меню изображений. За это ответственны свойства разделов Bitmap и ImageIndex. Bitmap позволяет непосредственно ввести изображение в раздел, выбрав его из указанного вами файла. ImageIndex позволяет указать индекс изображения, хранящегося во внешнем компоненте ImageList. Указание на этот компонент можно задать в свойстве Images компонента MainMenu. Индексы начинаются с 0. Если указать индекс -1 (значение по умолчанию), изображения не будет.

Основное событие раздела - OnClick, возникающее при щелчке пользователя на разделе или при нажатии «горячих» клавиш быстрого доступа. Еще одно очень важное свойство - Action. Сославшись на ранее описанное действие, можно избавиться от необходимости задавать большинство из указанных выше свойств, так как они наследуются от объекта действия. Вам не требуется также писать обработчик события OnClick, так как оно наследуется от действия.

Рассмотрим теперь вопросы объединения главных меню вторичных форм с меню главной формы. Речь идет о приложениях с несколькими формами, в которых и главная, и вспомогательные формы имеют свои главные меню компоненты MainMenu. Конечно, пользователю неудобно работать одновременно с несколькими окнами, каждое из которых имеет свое меню. Обычно надо, чтобы эти меню сливались в одно меню главной формы.

Приложения с несколькими формами могут быть двух видов: приложения с интерфейсом множества документов так называемые MDI приложения, и обычные приложения с главной и вспомогательными формами. Типичными примерами приложений MDI являются программы Word и Excel. В MDI приложениях меню дочерних форм всегда объединяются с меню родительской формы. А в приложениях с несколькими формами наличие или отсутствие объединение определяется свойством AutoMerge компонентов TMainMenu. Если требуется, чтобы меню вторичных форм объединялись с меню главной формы, то в каждой такой вторичной форме надо установить AutoMerge в true. При этом свойство AutoMerge главной формы должно оставаться в false.

Способ объединения меню определяется свойством разделов GroupIndex. По умолчанию все разделы меню имеют одинаковое значение GroupIndex, равное нулю. Если требуется объединение меню, то разделам надо задать неубывающие номера свойств GroupIndex. Тогда, если разделы встраиваемого меню имеют те же значения GroupIndex, что и какие-то разделы меню основной формы, то эти разделы заменяют соответствующие разделы основного меню. В противном случае разделы вспомогательного меню встраиваются между элементами основного меню в соответствии с номерами GroupIndex.

Если встраиваемый раздел имеет GroupIndex меньший, чем любой из разделов основного меню, то разделы встраиваются в начало.

Если в меню имеются разделы, работающие как радиокнопки, то нельзя забывать, что их взаимодействие также определяется свойствами Grouplndex. Связанные с меню в приложениях MDI, пользователь может открывать столько окон документов, сколько ему требуется. Обычно в подобных приложениях имеется меню Окно, которое содержит такие разделы, как Упорядочить. В конце меню идет обычно список открытых окон документов, в который заносятся названия открытых пользователем окон. Выбирая в этом списке, пользователь может переключаться между окнами документов.

Для включения в меню раздела списка открытых окон, надо в свойстве WindowMenu главной формы приложения MDI указать имя меню, в конец которого должен помещаться список. Указывается именно имя меню, а не разделов выпадающего списка.

Один из требований является стандартизация меню и их разделов. Этому помогает команда Save As Template в контекстном меню, всплывающем при щелчке правой кнопкой мыши в окне Конструктора Меню. Эта команда вызывает диалог. В этом диалоге можно в верхнем окне указать описание (заголовок), под которым нужно сохранить меню. Впоследствии в любом новом приложении можно загрузить этот шаблон в меню, выбирая из всплывающего меню в окне Конструктора Меню команду Insert From Template.

PopupMenu – контекстное всплывающее меню

Контекстное меню привязано к конкретным компонентам. Оно всплывает, если во время, когда данный компонент в фокусе, пользователь щелкнет правой кнопкой мыши. Обычно в контекстное меню включают те команды главного меню, которые в первую очередь могут потребоваться при работе с данным компонентом.

Контекстному меню соответствует компонент PopupMenu. Поскольку в приложении может быть несколько контекстных меню, то и компонентов PopupMenu может быть несколько. Оконные компоненты: панели, окна редактирования, метки и другие имеют свойство PopupMenu, которое по умолчанию пусто, но куда можно поместить имя того компонента PopupMenu, с которым будет связан данный компонент.

Формирование контекстного всплывающего меню производится с помощью Конструктора Меню, вызываемого двойным щелчком на PopupMenu, точно так же, как это делалось для главного меню. Обратите внимание на возможность упрощения этой работы. Поскольку разделы контекстного меню обычно повторяют некоторые разделы уже сформированного главного меню, то можно обойтись копированием соответствующих разделов. Для этого, войдя в Конструктор Меню из компонента PopupMenu, щелкните правой кнопкой мыши и из всплывшего меню выберите команду Select Menu (выбрать меню). Вам будет предложено диалоговое окно, в котором вы можете перейти в главное меню. В нем вы можете выделить нужный вам раздел или разделы (при нажатой клавише Shift выделяются разделы в заданном диапазоне, при нажатой клавише Ctrl можно выделить совокупность разделов, не являющихся соседними). Затем выполните копирование их в буфер обмена, нажав клавиши Ctrl+C. После этого опять щелкните правой кнопкой мыши, выберите команду Select Menu и вернитесь в контекстное меню. Укажите курсором место, в которое хотите вставить скопированные разделы, и нажмите клавиши чтения из буфера обмена Ctrl+V. Разделы меню вместе со всеми их свойствами будут скопированы в создаваемое вами контекстное меню.

В остальном работа с PopupMenu не отличается от работы с MainMenu. Только не возникает вопросов объединения меню разных форм, т.к. контекстные меню не объединяются.

Контрольные вопросы:

1. Назовите компонент главного меню?
2. Назовите свойства компонента Main Menu?
3. Назовите компонент контекстного меню?

4. Назовите свойства компонента PopupMenu?

Тема 41. Основные понятия баз данных

Цель: сформировать знания об основных понятиях базы данных, а также изучить компоненты среды C++Builder, использующие технологию ADO

Базы данных и системы управления базами данных СУБД. Пользователи базы данных. Архитектура базы данных. Модели представления данных (иерархическая, сетевая, реляционная). Классификация БД по способу хранения БД. Элементы реляционных БД. Языковые средства БД.

Одним из важнейших условий обеспечения эффективного функционирования любого предприятия или организации является наличие развитой информационной системы.

Информационная система представляет собой систему, реализующую автоматизированный сбор, обработку и манипулирование данными и включающую технические средства обработки данных, программное обеспечение и обслуживающий персонал. Со-временной формой информационных систем являются банки данных1.

Банк данных – это система специальным образом организованных данных – баз данных, а также технических, программных, языковых и организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

Основными компонентами банка данных являются:

- ✓ вычислительная система (технические средства и операционная система);
- ✓ база данных (непосредственно вся информация);
- ✓ система управления базой данных, СУБД (программное обеспечение для организации хранения и использования информации);
- ✓ набор прикладных программ.

К основным функциям банка данных относятся:

- ✓ хранение данных и их защита;
- ✓ изменение (обновление, добавление и удаление) хранимых данных;
- ✓ поиск и отбор данных по запросам пользователей;
- ✓ обработка данных и вывод результатов.

База данных (БД) является ядром банка данных и представляет совокупность взаимосвязанных и вместе хранящихся данных из определенной предметной области, организованных специальным образом и хранимых во внешней памяти (файлах базы данных).

В компьютерных базах данных может содержаться любая информация: от простого текста (например, фамилия, имя и адрес) до сложной структуры, включая рисунки, звуки и изображения. Хранение данных в заранее известном формате позволяет извлекать данные в желаемом формате благодаря использованию разных методов обработки. Функционирование базы данных обеспечивает администратор базы данных.

Администратор базы данных — лицо, отвечающее за выработку требований к базе данных, её проектирование, реализацию, эффективное использование и сопровождение, включая управление учётными записями пользователей БД и защиту от несанкционированного доступа. Не менее важной функцией администратора БД является поддержка целостности базы данных. Целостность БД – свойство БД, означающее, что база данных содержит полную и непротиворечивую информацию, необходимую и достаточную для корректного функционирования приложений.

Система управления базой данных (СУБД) – это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

К функциям СУБД относится:

- ✓ перевод схемы, определяющей структуру данных и записанной на языке определения данных в некоторое внутреннее представление, используемой системой при дальнейшей работе с данными;
- ✓ создание БД (загрузка данных в БД);
- ✓ реализация запросов пользователей (формулируемых на специальном языке, принятом в данной СУБД) на сортировку и отбор по заданным критериям, а также извлечение некоторой части БД, что может сопровождаться редактированием и обработкой информации;
- ✓ обновление некоторой части БД без изменения структуры данных; обеспечение защиты данных и приоритетов в их использовании.

Можно сказать, что основная функция СУБД – это предоставление пользователю БД возможности работы с ней, не вникая в детали на уровне аппаратного обеспечения. То есть все запросы пользователя к БД, добавление и удаление данных, выборки, обновление данных – все это обеспечивает СУБД.

Программы, с помощью которых пользователи работают с базой данных, называются приложениями. В общем случае с одной базой данных могут работать множество раз-личных приложений. Например, если база данных моделирует некоторое предприятие, то для работы с ней может быть создано приложение, которое обслуживает подсистему учета кадров, другое приложение может использоваться для расчета заработной платы сотрудников, третье предназначено для планирования производственного процесса и т. д. При рассмотрении приложений, работающих с одной базой данных, предполагается, что они могут работать параллельно и независимо друг от друга, и именно СУБД призвана обеспечить работу множества приложений с единой базой данных таким образом, чтобы каждое из них выполнялось корректно, но учитывало все изменения в базе данных, вносимые другими приложениями. Приложения могут создаваться как в среде СУБД, так и вне СУБД – с помощью системы программирования, использующей средства доступа к БД (например, Delphi или C++ Builder).

Для работы с базой данных во многих случаях можно обойтись только средствами СУБД, скажем, создавая запросы и отчеты. Приложения разрабатывают главным образом в случаях, когда требуется обеспечить удобство работы с БД неквалифицированным пользователям или интерфейс СУБД не устраивает пользователя.

Пользователи банков данных

Как любой программно-организационный и технический комплекс, банк данных существует во времени и в пространстве. Он имеет определенные стадии своего развития:

1. Проектирование.
2. Реализация.
3. Эксплуатация.
4. Модернизация и развитие.
5. Полная реорганизация.

На каждом этапе своего существования с банком данных связаны разные категории пользователей. Определим основные категории пользователей и их роль в функционировании банка данных.

Системы управления базами данных (СУБД) — это программные средства, предназначенные для создания, наполнения, обновления и удаления баз данных. По реализации СУБД можно выделить три основных вида. Заказные СУБД требуют существенных затрат при разработке, однако заказные СУБД в максимальной степени

учитывают специфику работы заказчика (того или иного предприятия), их интерфейс обычно интуитивно понятен пользователям и не требует от них специальных знаний.

Среда программирования C++Builder обладает мощными средствами для разработки СУБД .

Мы будем рассматривать работу с БД, основанную на технологии ADO. И хотя скорость доступа к данным, обеспечиваемая технологией ADO, ниже, чем при использовании других технологий, она не требует установки дополнительных программных средств.

Технология ADO (ActiveX Data Object) обеспечивает механизм доступа к данным, разработанный корпорацией Microsoft, посредством которого можно связываться с различными данными приложений Microsoft. Позволяет представлять данные из разнообразных источников (реляционных баз данных, текстовых файлов и т. д.) в объектно-ориентированном виде.

В соответствии с терминологией ADO любой источник данных является хранилищем данных. Приложение взаимодействует с хранилищем данных с помощью интерфейса OLE. Для каждого типа хранилища данных используется свой интефейс ADO. Интерфейс обеспечивает обращение к данным хранилища с запросами, интерпретацию возвращаемой служебной информации и результатов выполнения запросов для передачи их приложению .

Применяя техногию ADO, можно в качестве хранилища данных использовать базы данных, созданные в MS Access. Файл БД должен иметь формат MBD.

1) Все таблицы БД хранятся в одном файле, что облегчает создание резервных копий, переносимость на другие компьютеры.

2) Поля в таблице можно называть по-русски.

Компоненты для работы с базой данных

Компоненты для работы находятся на панели dbGo.

ADOConnection — ADO-соединение, используется для установки соединения с ADO-источником данных;

ADODataSet — набор данных ADO;

ADOTable — таблица ADO, обеспечивает доступ к одной таблице ADO-источника данных и позволяет другим компонентам управлять этими данными, связываясь с компонентом ADOTable через компонент DataSource;

ADOQuery — запрос ADO, позволяет выполнять SQL-команды.

Для визуализации данных используются компоненты панели Data Controls. Для обеспечения связи между визуальными компонентами и компонентами, обеспечивающими технологию ADO, используют компонент DataSource, расположенный на вкладке Data Access.

Таким образом, для работы с базами данных по технологии ADO на форме необходимо размещать компоненты:

- обеспечивающие реализацию технологии ADO;
- обеспечивающие визуализацию табличных данных;
- обеспечивающие связь между технологическими и визуальными компонентами.

Контрольные вопросы:

1. Какую систему называют информационной?
2. Что называется банком данных, базой данных?
3. Приведите примеры баз данных.
4. Перечислите основные модели данных.
5. Приведите пример иерархической модели представления данных.
6. Что называется СУБД?

ЛАБОРАТОРНО -ПРАКТИЧЕСКИЕ РАБОТЫ

Лабораторная работа № 1.
Способы реализации алгоритмов

Цель работы

Усвоить понятия: алгоритм как фундаментальное понятие информатики, способы описания, основные типы алгоритмов, освоить принципы решения задач с использованием основных алгоритмических конструкций.

Общие теоретические сведения

Решение любой задачи на ЭВМ можно разбить на следующие этапы: разработка алгоритма решения задачи, составление программы решения задачи на алгоритмическом языке, ввод программы в ЭВМ, отладка программы (исправление ошибок), выполнение программы на ПК, анализ полученных результатов.

Первый этап решения задачи состоит в разработке алгоритма.

Алгоритм – это точная конечная система правил, определяющая содержание и порядок действий исполнителя над некоторыми объектами (исходными и промежуточными данными) для получения после конечного числа шагов искомого результата.

Алгоритм может быть описан одним из трех способов:

– словесным (пример в начале раздела);
– графическим (виде специальной блок-схемы);
– с помощью специальных языков программирования.

Блок-схема – распространенный тип схем, описывающий алгоритмы или процессы, изображая шаги в виде блоков различной формы, соединенных между собой стрелками.

1. Линейный алгоритм – это такой алгоритм, в котором все операции выполняются последовательно одна за другой.

2. Алгоритмы разветвленной структуры применяются, когда в зависимости от некоторого условия необходимо выполнить либо одно, либо другое действие.

3. Алгоритмы циклической структуры.

Циклом называют повторение одних и тех же действий (шагов). Последовательность действий, которые повторяются в цикле, называют телом цикла.

Циклические алгоритмы подразделяют на алгоритмы с предусловием, постусловием и алгоритмы с конечным числом повторов. В алгоритмах с предусловием сначала выполняется проверка условия окончания цикла и затем, в зависимости от результата проверки, выполняется (или не выполняется) так называемое тело цикла.

Задание 1. Определить площадь трапеции по введенным значениям оснований (a и b) и высоты (h).

Запись решения задачи на алгоритмическом языке:

```
алг трапеция
вещ a,b,h,s
нач
   ввод f,b,h
            s:=((a+b)/2)*h
      вывод s
кон
```

Запись алгоритма в виде блок-схемы (рис. 1):

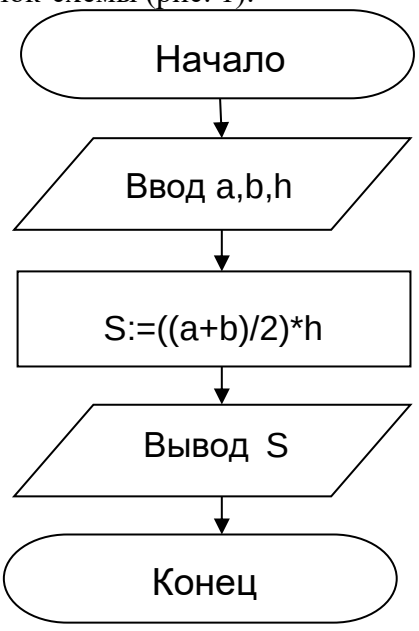


Рисунок 1 – Блок-схема линейного алгоритма

Задание 2. Определить среднее арифметическое двух чисел, если а положительное и частное (a/b) в противном случае.

Запись решения задачи на алгоритмическом языке:

алг числа
вещ a,b,c
нач
ввод a,b
если a>0
то c:=(a+b)/2
иначе c:=a/b
все
вывод c
кон

Запись алгоритма в виде блок-схемы (рис. 2):

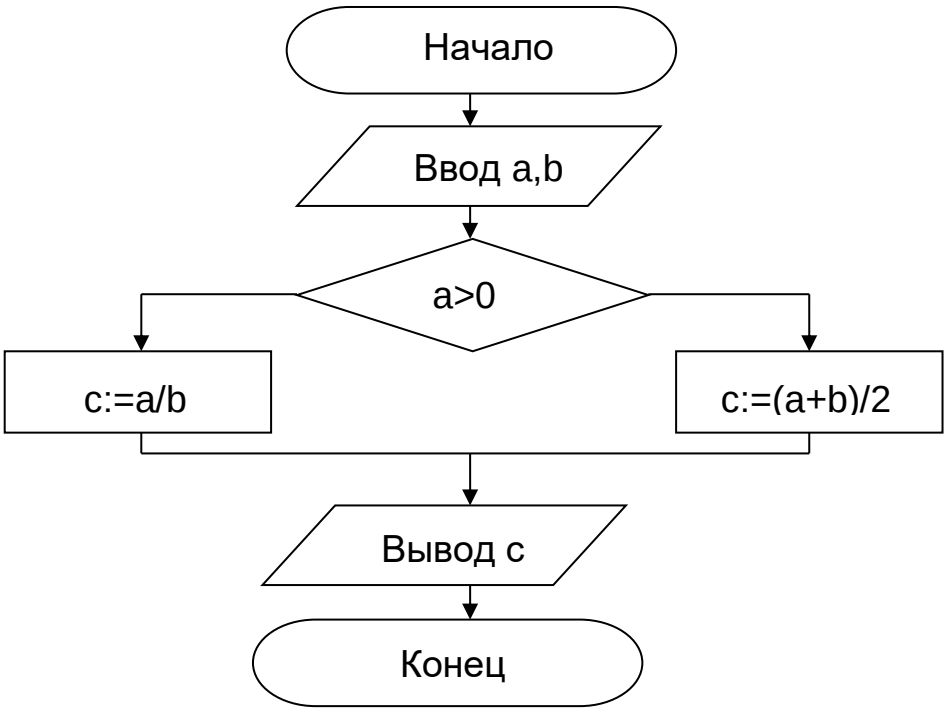


Рисунок 2 – Блок-схема алгоритма с ветвлением

Задание 3. Составить алгоритм нахождения суммы целых чисел в диапазоне от 1 до 10.

Запись решения задачи на алгоритмическом языке:

алг сумма
вещ a,s
нач
S:=0;
A:=1;
нц
пока a<=10
S:=S+a;

A:=a+1;
кц
вывод S
кон

Запись алгоритма в виде блок-схемы (рис. 3):

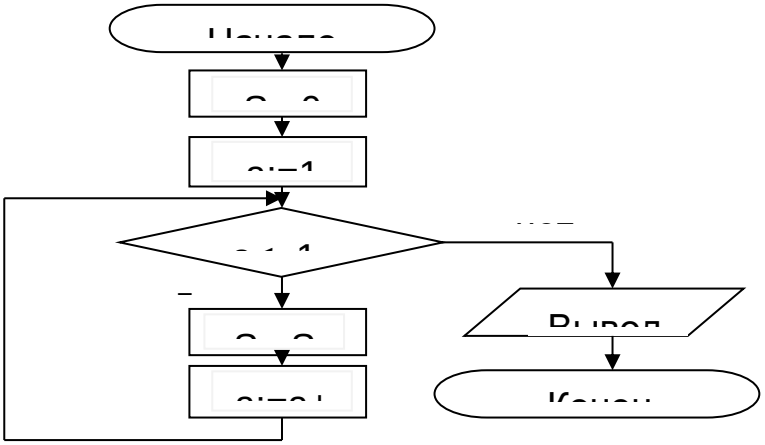


Рисунок 3 – Циклический алгоритм с предусловием

В алгоритме с постусловием сначала выполняется тело цикла, а затем проверяется условие окончания цикла. Решение задачи нахождения суммы первых десяти целых чисел в данном случае будет выглядеть следующим образом:

алг сумма
вещ a,s
нач
S:=0;
A:=1;
нц
S:=S+a;
A:=a+1;
пока a<=10
кц
вывод S
кон

Запись алгоритма в виде блок-схемы (рис. 4):

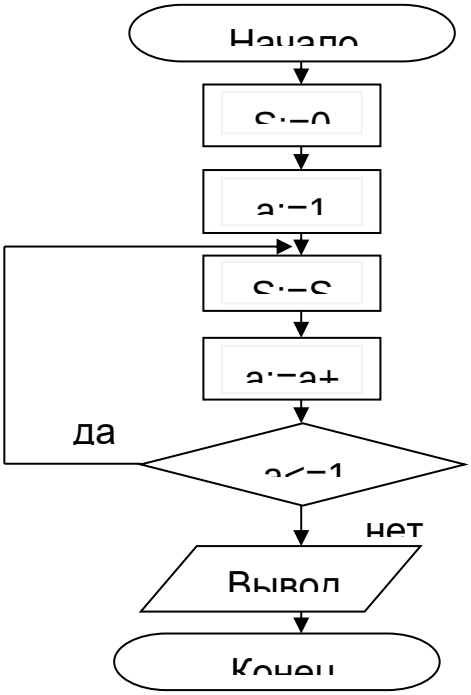
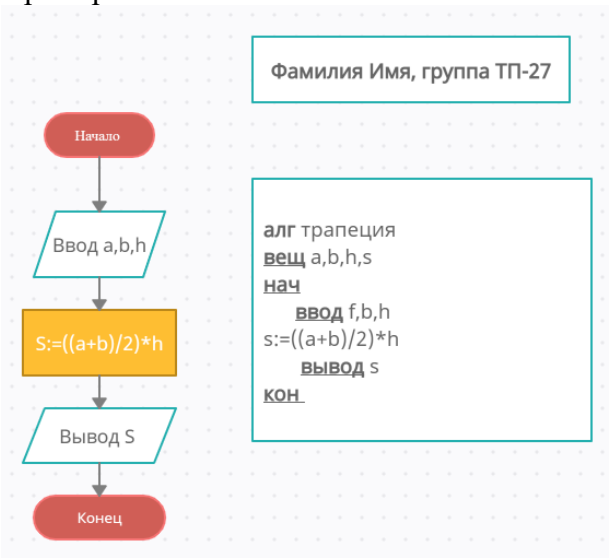


Рисунок 4 – Циклический алгоритм с постусловием

Варианты задания

Задание 1. Составить алгоритм решения задачи с помощью алгоритмического языка псевдокод и с помощью блок-схем, используя конструкцию линейного алгоритма. Выполнить блок схемы с помощью интернет сервисов **creately.com или programforyou.ru.** Сделать скриншот и прикрепить в Платонус.

Пример.



1. Вычислить площадь поверхности и объем усеченного конуса по следующим формулам

$S = \pi (R + r) l + \pi R^2 + \pi r^2$;

$V = (1/3) \pi (R^2 + r^2 + Rr) h$.

2. Вычислить координаты центра тяжести трех материальных точек с массами m_1, m_2, m_3 и координатами (x_1,y_1), (x_2,y_2), (x_3,y_3) по формулам:

$x_c = (m_1x_1 + m_2x_2 + m_3x_3) / (m_1 + m_2 + m_3)$;

$y_c = (m_1y_1 + m_2y_2 + m_3y_3) / (m_1 + m_2 + m_3)$.

3. Вычислить площадь треугольника со сторонами a, b, c по формуле Герона:

$S = \sqrt{p\,(p-a)(p-b)(p-c)}$,

где p – полупериметр, вычисляемый по формуле

$\frac{a+b+c}{2}$,

4. Вычислить координаты точки, делящей отрезок a_1a_2 в отношении $n_1:n_2$ по формулам:

$x = (x_1 + \gamma x_2)/(1+\gamma)$;

$y = (y_1 + \gamma y_2)/(1+\gamma)$,

где $\gamma = n_1/n_2$.

5. Вычислить медианы треугольника со сторонами a, b, c по формулам:

$m_a = 0.5\sqrt{2b^2 + 2c^2 - a^2}$;

$m_b = 0.5\sqrt{2a^2 + 2c^2 - b^2}$;

$m_c = 0.5\sqrt{2b^2 + 2b^2 - c^2}$;

6. Вычислить площадь круга и длину окружности по введенному значению радиуса.

7. Вычислить площадь S и периметр L эллипса по введенным значениям полуосей a и b:

$S := \pi \cdot a \cdot b$;

$L = 2 \cdot \pi \cdot \sqrt{\frac{1}{2}(a^2 + b^2)}$.

8. Вычислить объем V и площадь боковой поверхности цилиндра S по введенным значениям радиуса основания R и высоты цилиндра H.

$V = \pi \cdot R^2 \cdot H$;

$S = 2 \cdot \pi \cdot R \cdot H$.

9. Вычислить объем V и площадь боковой поверхности конуса S по введенным значениям радиуса основания r, высоты h и образующей l:

$V = \frac{1}{3}\pi \cdot r^2 \cdot h$;

$S = \pi \cdot r \cdot l$.

10. Вычислить объем V и площадь поверхности S сферы по введенному значению радиуса r:

$V = \frac{4}{3} \cdot \pi \cdot r^3$;

$S = 4 \cdot \pi \cdot r^2$.

11. Дано целое четырехзначное число. Используя операции div и mod, найти сумму его цифр.

12. Дана сторона равностороннего треугольника. Найти площадь этого треугольника и радиусы вписанной и описанной окружностей.

13. Даны координаты трех вершин треугольника (x1, y1), (x2, y2), (x3, y3). Найти его периметр и площадь.

14. Дана длина окружности. Найти площадь круга, ограниченного этой окружностью.

15. Дана площадь круга. Найти длину окружности, ограничивающей этот круг.

Задание 2. Составить алгоритм решения задачи с помощью алгоритмического языка псевдокод и с помощью блок-схем, используя конструкцию алгоритма с ветвлением. Выполнить блок схемы с помощью интернет сервисов **creately.com или programforyou.ru.** Сделать скриншот и прикрепить в Платонус.

1. Составить программу для решения квадратного уравнения $ax^2 + bx + c = 0$.

2. Определить максимальное четное число из двух введенных.

3. Определить, можно ли из отрезков с длинами x, y и z построить треугольник.

4. Ввести два числа a и b. Большее число заменить утроенным произведением, меньшее – полусуммой.

5. Если среди трех чисел a, b, c имеется хотя бы одно четное, то найти максимальное число, иначе – минимальное.

6. Определить, в каком квадранте находится точка с координатами x и y и вывести номер квадранта на экран.

7. Найти квадрат наибольшего из двух чисел a и b. Вывести на экран число 1, если наибольшим является число a, число 2 – если наибольшим числом является b.

8. Определить, попадает ли точка с координатами x и y в круг радиусом R. Если точка попадает в круг, вывести на экран единицу, в противном случае – ноль.

9. Написать алгоритм решения задачи, которая решает уравнение $ax + b = 0$ относительно x для любых чисел a и b, введенных с клавиатуры. Все числа считаются действительными.

10. Написать алгоритм решения задачи, которая определяет, лежит ли точка A(x,y) внутри некоторого кольца («внутри» понимается в строгом смысле, т.е. случай, когда точка A лежит на границе кольца, недопустим). Центр кольца находится в начале координат. Для кольца заданы внутренний и внешний радиусы r1, r2. Координаты x и y вводятся с клавиатуры.

11. Даны две переменные целого типа: A и B. Если их значения не равны, то присвоить каждой переменной произведение этих значений, а если равны, то присвоить переменным нулевые значения.

12. Даны две переменные целого типа: A и B. Если их значения не равны, то присвоить каждой переменной минимальное из этих значений, а если равны, то присвоить переменным нулевые значения.

13. Даны целочисленные координаты точки на плоскости. Если точка не лежит на координатных осях, то вывести 0. Если точка совпадает с началом координат, то вывести 1. Если точка не совпадает с началом координат, но лежит на оси OX или OY, то вывести соответственно 2 или 3.

14. Даны вещественные координаты точки, не лежащей на координатных осях OX и OY. Вывести номер координатной четверти, в которой находится данная точка.

15. Дано целое число, лежащее в диапазоне от –999 до 999. Вывести строку – словесное описание данного числа вида "отрицательное двузначное число", "нулевое число", "положительное однозначное число" и т.д.

Задание 3. Составить алгоритм решения задачи с помощью алгоритмического языка псевдокод и с помощью блок-схем, используя конструкцию циклического алгоритма. Выполнить блок схемы с помощью интернет сервисов **creately.com или programforyou.ru.** Сделать скриншот и прикрепить в Платонус.

1. Найти сумму чисел, кратных трем, в диапазоне от 0 до 50.

2. Найти сумму первых десяти чисел, кратных пяти.

3. Найти произведение четных чисел в диапазоне от 2 до 30.

4. Вводятся положительные числа. Прекратить ввод, когда сумма введенных чисел превысит 100.

5. Требуется найти сумму чисел, кратных 7, в диапазоне от 0 до 100. Вывести на экран сумму чисел и их количество.

6. Определить количество целых чисел, кратных 3 (от 3 и далее), дающих в сумме число, превышающее 200.

7. Вводятся 10 чисел. Вывести на экран суммы положительных и отрицательных чисел и их количество.

8. Вывести на экран значения функции y=sin(x) для $0 \le x \le 180$ с шагом в 1^0.

9. Подсчитать площади десяти кругов с радиусами от 1 см с шагом 2 см и вывести значения площадей на экран.

10. Вводятся положительные числа. Прекратить ввод чисел, когда их сумма превысит 100. Результат вывести на экран.

11. Вводятся числа. Прекратить ввод чисел, когда сумма положительных чисел превысит 100. Результат вывести на экран.

12. Вывести на экран значения произведений чисел a и b. Числа a изменяются от 1 до 11 с шагом 1, b – от 1 до 3 с шагом 0,2.

13. Вывести на экран таблицу перевода километров в мили в диапазоне от 2 до 20 километров с шагом 2 км.

14. Вы положили в банк 1500 рублей. Определить, сколько денег будет на Вашем вкладе через 1 год, если каждый месяц вклад увеличивается на 0.76 % от суммы предыдущего месяца.

15. Решив заняться легкой атлетикой, Вы пробежали в первый день 2 км. Сколько километров Вы пробежите за 2 недели, если каждый день Вы увеличиваете дистанцию на 10 % от предыдущего дня?

Технология выполнения работы

В рамках выполнения работы необходимо составить алгоритм решения задачи в виде блок-схемы и с помощью языка псевдокода.

Содержание отчета

1. Цель работы и задание.
2. Условие задачи.
3. Алгоритм, написанный с помощью псевдокода и блок- схемы.

Вопросы для защиты работы

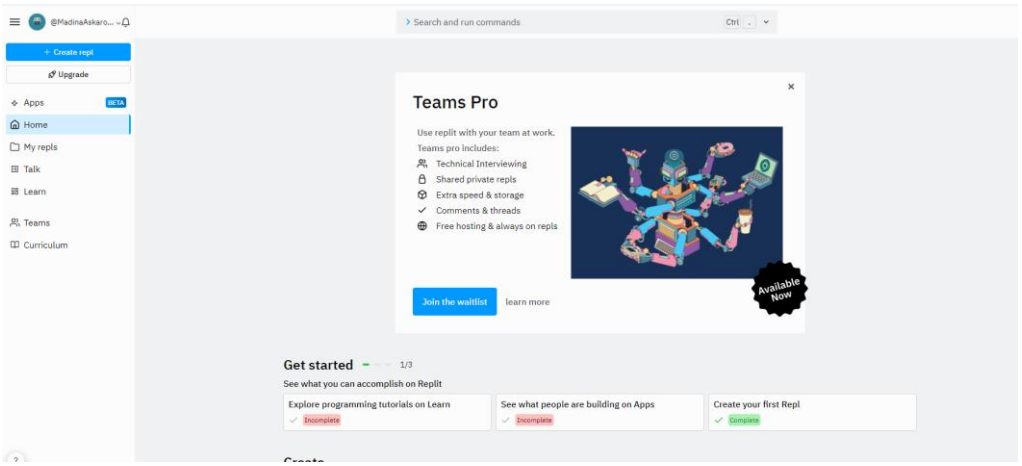
1. Что такое алгоритм?
2. Свойства алгоритма.
3. Способы записи алгоритма.
4. Основные элементы блок-схемы.
5. Виды алгоритмов.
6. Отличительные особенности алгоритмов с предусловием и постусловием.

Лабораторная работа №2.
Знакомство со средой программирования

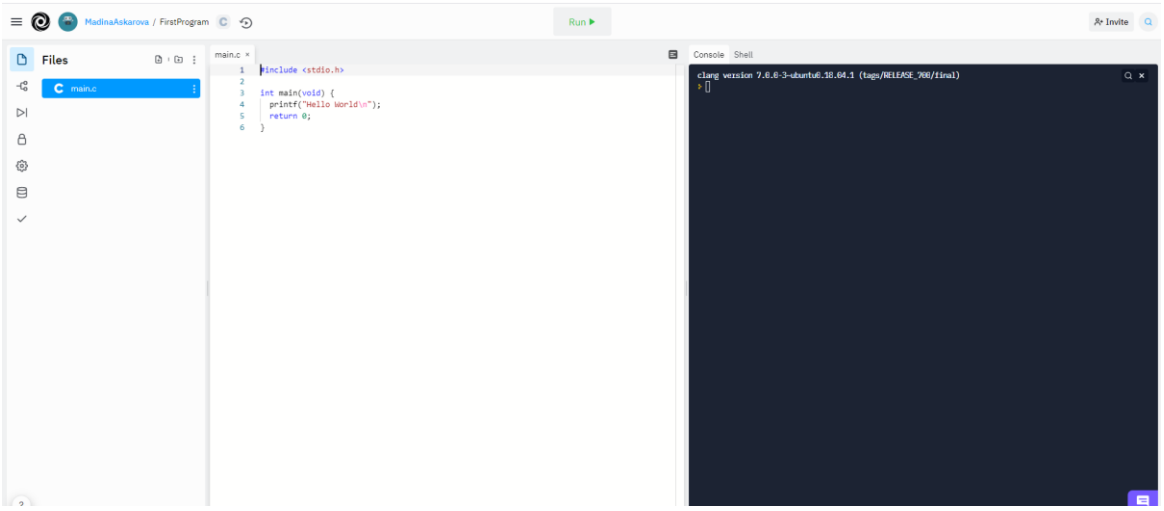
Цели:

1. Ознакомиться с со средой программирования онлайн компилятора Repl.it.
2. Изучить возможности ввода кода и компиляции.
3. Выработать навыки реализации программного кода, минимальной отладки программ.

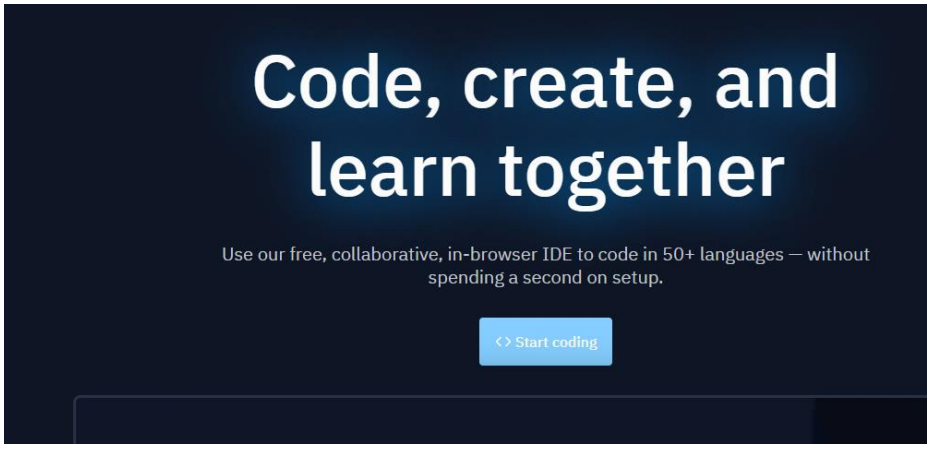
Repl.it — среда для совместной работы с кодом в браузере. Поддерживает более 50 языков, среди которых C, C++, C#, Java, Python, R, JavaScript.



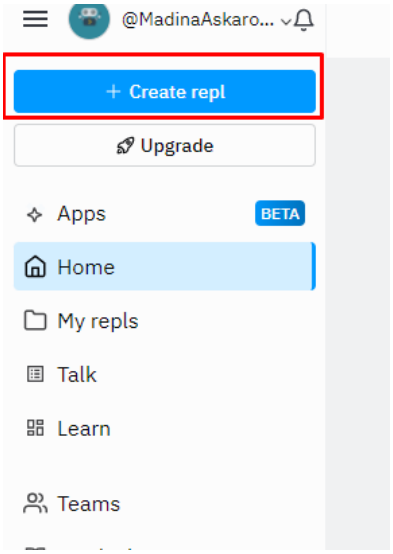
В бесплатной версии доступно многопользовательское сотрудничество, 500 МБ хранилища и 500 МБ памяти, 0.2 — 0.5 vCPUs. Есть также платная версия с приватными проектами, хостингом до 5 реплов, 5 ГБ хранилища, 2 ГБ памяти и 2 vCPUs.



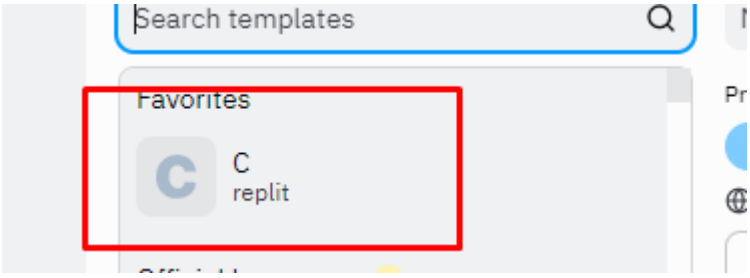
Для знакомства с онлайн компилятором необходимо пройти по ссылке https://replit.com/, нажать на кнопку Start coding и пройти процедуру регистрации /авторизации.



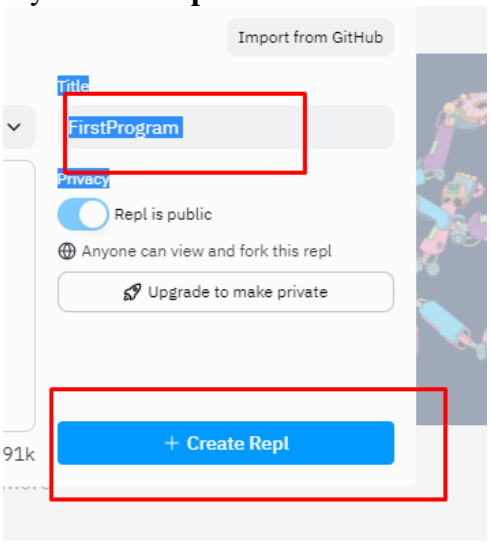
Далее для создания кода программы необходимо нажать на кнопку **Create repl.**



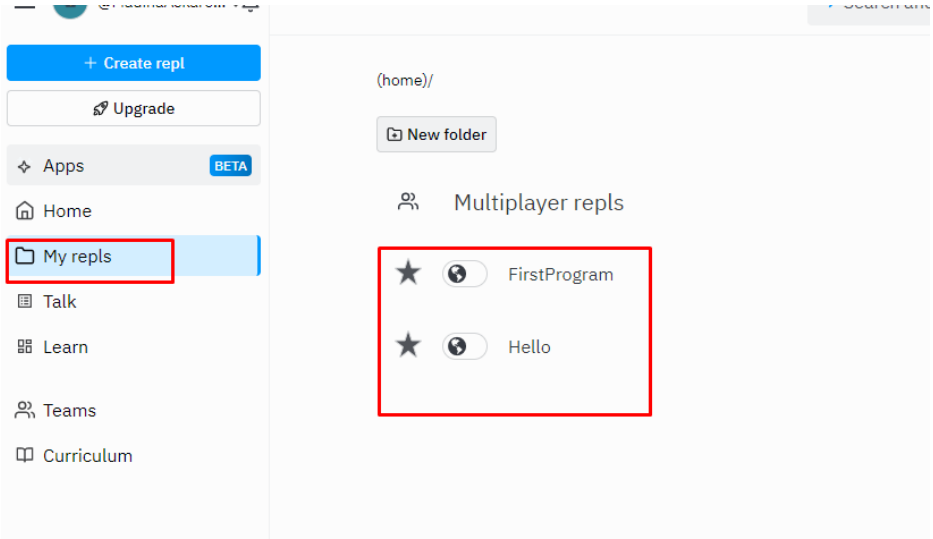
Выбрать необходимый компилятор из списка предложенных, в нашем случае Си.



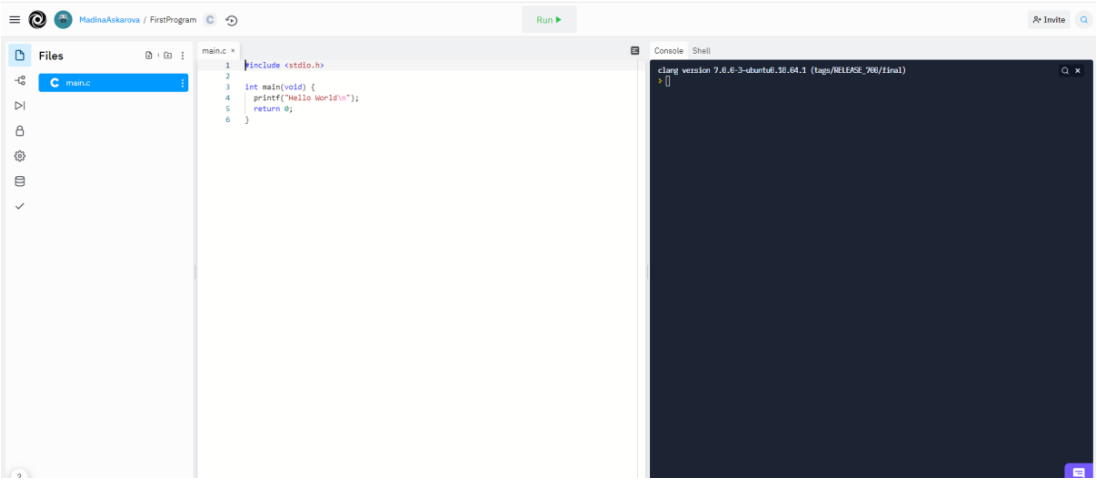
В появившемся окне необходимо ввести название программы или проекта и нажать на кнопку **Create Repl.**

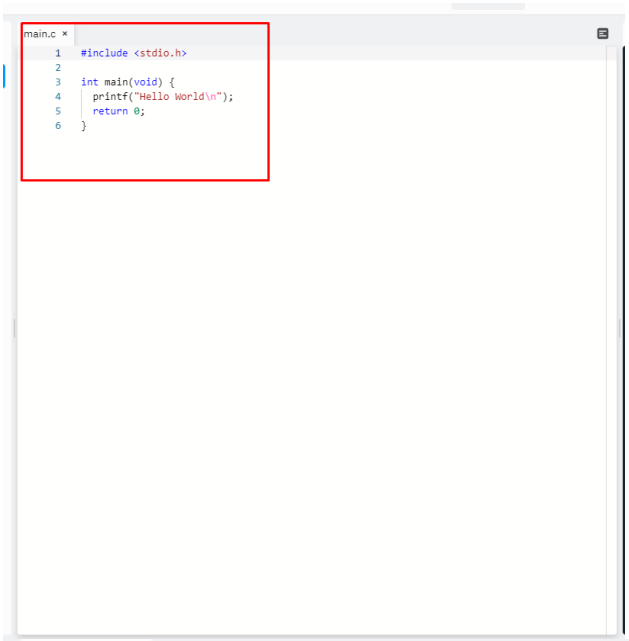


Все ваши созданные файлы программ будут хранится во вкладке My repls.

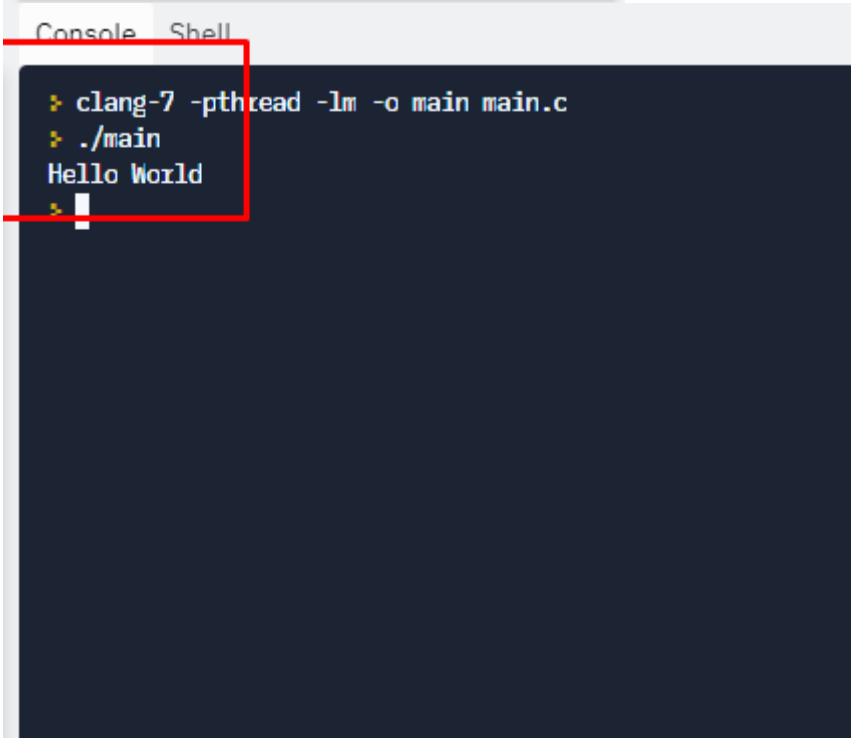


Интерфейс редактора состоит из двух столбцов, первый - сам редактор, второй – консоль. В редакторе есть подсветка синтаксиса и автодополенение, которое поможет вам программировать быстрее.



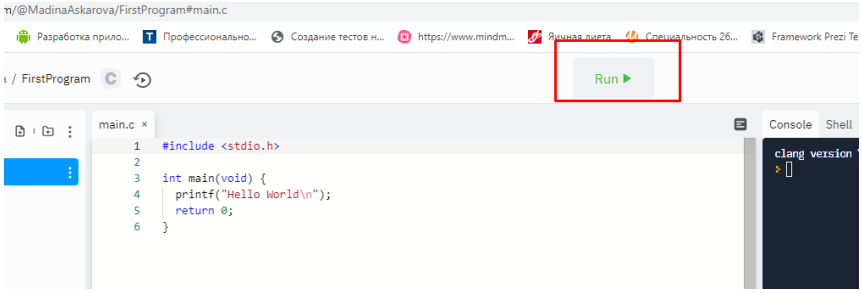


Окно редактора кода

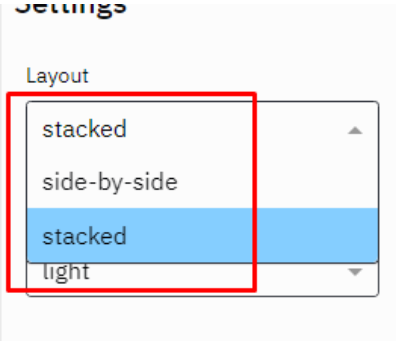


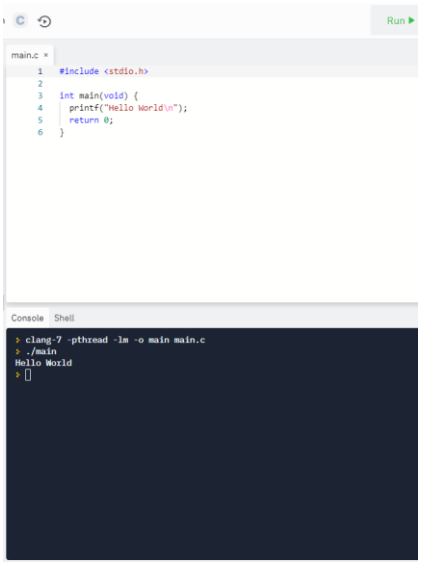
Окно консоли

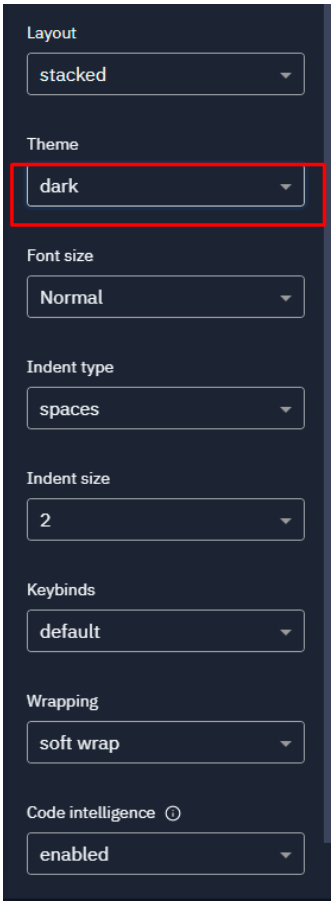
Для компиляции кода программы необходим нажать кнопку Run.

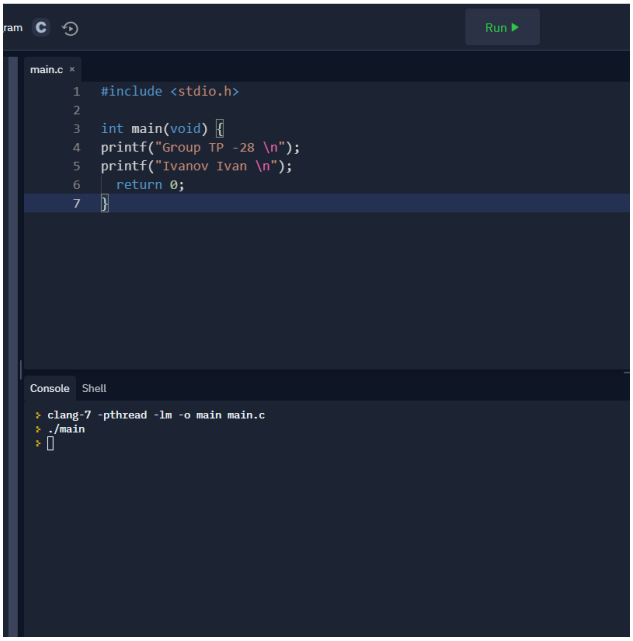


С помощью вкладки настроек можно изменить расположение окон, цветовую гамму и т.д.









Задание лабораторной работы.

1. Изучить интерфейс онлайн компилятора Repl.it. Запишите следующий код в окно редактора кода:

```
#include <stdio.h>
int main(void) {
printf("Group TP -28 \n");
printf("Ivanov Ivan \n");
```

return 0;
}

Измените код так, чтобы после компиляции консоль выдавала название вашей группы и вашу Фамилию Имя.

Пример

2 .Выполните следующие пункты работы:

Откройте программу и создайте проект. Имя проекта должно иметь следующую структуру: LW1_Фамилия, при этом Фамилия должна быть написана латинскими буквами.

После создания проекта и добавления файла исходного кода напишите следующую программу (включая комментарии) и откомпилируйте ее, убедившись в отсутствии ошибок. Исправьте ошибки

#include <stdio.h> //подключение заголовочного файла
int a; //объявление переменной
void main()//основной цикл
{//начало основного цикла
}//конец основного цикла

Добавьте в данную программу глобальное объявление переменных в табл. 1 и выполните компиляцию на наличие ошибок.

Таблица 1.

Типы переменных

Имя переменных	Тип переменных
a,b	integer
c	double
d,e	float
letter, digits	char

a1,b1,c1,d1	signed integer
a2, b2,c2,d2	long integer

В случае наличия ошибок попробуйте исправить их и вновь выполнить компиляцию. При этом необходимо внести комментарии в строки объявления переменных с указанием их типов.

Добавить в программу глобальное определение констант из табл. 2, посредствам, соответствующим им идентификаторов.

Таблица 2.

Константы

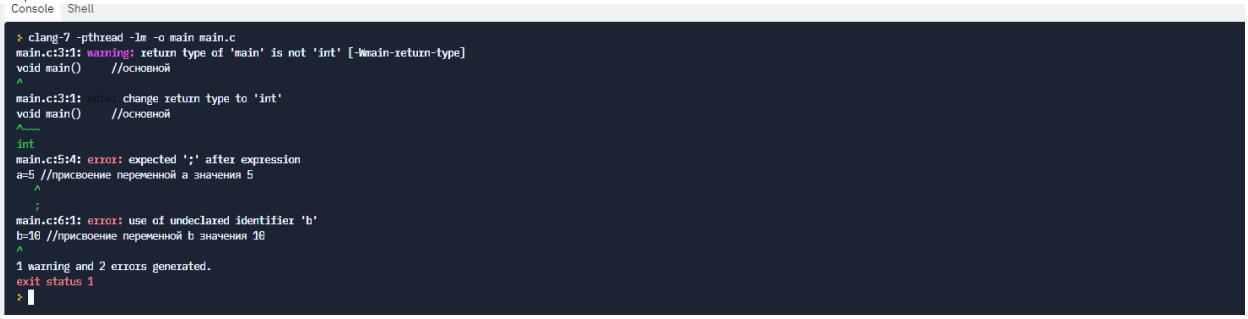
Имя констант	Значения	Тип идентификатора
min	60	define
max	100	define
avr	80	define
con1	10	const
con2	1.53	const
con3	-20	const

В случае наличия ошибок попробуйте исправить их и вновь выполнить компиляцию. При этом необходимо внести комментарии в строки объявления констант.

Запишите следующую программу в том виде в котором Вы ее видите ниже (не внося исправлений):

```
#include <stdio.h>//подключение заголовочного файла
int a; //объявление переменной
void main()//основной
{ //начало
a=5 //присвоение переменной a значения 5
b=10 //присвоение переменной b значения 10
c=a+b//присвоение переменной c значения a+b
prentf("c=%d \n",c)
} //конец
```

Откомпилируйте данную программу. После компиляции будут обнаружены следующие ошибки:



Исправьте ошибки и добейтесь корректной компиляции проекта!!!

Покажите результат исправленной программы преподавателю.

Самостоятельные задания:

1. Вводится вещественное число a. Не пользуясь никакими арифметическими

операциями, кроме сложения, получить 7а за четыре операции.

2. Найти площадь круга и длину окружности. Используемые переменные: r - радиус, d – длина окружности, s – площадь круга

3. Составить программу расчета значения функции. Z = |cos x 4 – 3 tg x 2 |+0.8 sin yx 2 + 10 при любых значениях x и y.

Лабораторная работа №3
Решение задач с использованием операторов ввода/вывода данных

Цель работы: приобретение практических навыков работы с операторами ввода/вывода

Операторы ввода/вывода

1) Оператор вывода:

Printf (<форматная строка>, переменные);

Форматная строка может включать текст для вывода на экран, спецификации преобразования данных, управляющие символы.

В строке символов могут присутствовать спецификации преобразования данных, каждой из которой должен соответствовать аргумент. Если имеются спецификации преобразования, то вместо них выводятся значения соответствующих аргументов.

Для аргументов различных типов необходимо использовать соответствующие спецификации преобразования:

%d - для вывода целых чисел;

%c - для вывода образа символа, соответствующий аргумент должен содержать код символа;

%f - для вывода вещественного числа в виде целой и дробной части;

%e - для вывода вещественного числа в виде мантиссы и порядка;

%g - для вывода вещественного числа в виде %f или %e в зависимости от значения числа;

%u - для вывода беззнакового целого числа в десятичной системе счисления;

%o - для вывода беззнакового целого числа в восмеричной системе счисления;

%x - для вывода беззнакового целого числа в шестнадцатеричной системе счисления;

%s - для вывода на экран символьной строки, соответствующий аргумент должен быть адресом строки (т. е. именем символьного массива или строковой константой).

Для дополнительного управления преобразованием данных используются модификаторы преобразования

Они записываются между символом % и спецификацией преобразования.

%[флаг] [ширина] [. точность] [l] [спецификатор типа преобразования]

[флаг] : “-“ или “+” ; ”-“ выравнивает текст вывода по левому краю; “+” выводит знак для положительных и отрицательных значений;

[ширина] : минимальный размер поля вывода;

[.точность] : для вещественных чисел количество знаков после десятичной точки;

[l] : для вывода целых чисел типа long;

Примеры спецификаций преобразований с модификаторами:

%-20.6lf , %6d, %8.4f

Пример:

double a=-78.98;

```
int c=24;
printf("a=%8.4lf, c=%6d", a, c);
```
На экране:
```
a=-78.9800, c=    24
```

Управляющие символы:
\n – перевод строки;
\t – табуляция;
\v – вертикальная табуляция;
\b – возврат на 1 символ;
\r – возврат на начало строки;
\a – звуковой сигнал.

2) Оператор ввода:
scanf(<форматная строка>, адреса переменных);

Функцию scanf рекомендуется использовать без лишних символов в формате,только спецификации преобразования, иначе может возникнуть непредсказуемая ситуация. Количество форматов определяется количеством переменных. Форматы при вводе переменных различных типов такие же, как и при выводе.

Обратите внимание, что в качестве аргументов передаются не значения переменных, а их адреса. Нужно сообщить функции адрес ячейки, в которую необходимо занести данные. Адреса переменных большинства типов обозначаются знаком &.

Пример:
```
int n;
scanf( "%d", &n );
```
Перед именами строк и массивов операция взятия адреса & не ставится, т. к. имя строки или массива само определяет их адрес.

Пример:
```
char name[41];
scanf( "%s", name );
```

ЗАДАНИЯ

На оценку «удовлетворительно»

Составить программу, которая запрашивает у пользователя целое число, действительное число, произвольный символ, а затем все выводит в одной строчке.

1. Выведите на экран свою фамилию, имя и отчество, а через две строки - дату своего рождения.

На оценку «хорошо»

2. Написать программу для печати звездочками одной из фигур:
а) елочки (нескольких елочек);
б) снежинки (нескольких снежинок);
в) домика.
Например,

```
    *
  *   *
*       *
```

```
**********
*        *
*        *
*        *
*        *
**********
```

3. Составить свою визитную карточку.

```
******************************
*          Иванов Сергей          *
*     Пролетарская 74 кв. 55      *
*        Телефон 45-72-88         *
******************************
```

На оценку «отлично»

4. Составьте диалог пользователя с компьютером на произвольную тему.

Например, машина задает два вопроса “Как тебя зовут?” и “Сколько тебе лет?”; после введения имени (Антон) и числа (15) выводит на экран “Да... Через 50 лет тебе уже будет 65 лет, а звать тебя будут не Антон, а дед Антон”

5. Запросить у пользователя два числа и вывести на экран результат суммы, разности, произведения и частного этих чисел полным ответом.

6. Написать программу, которая запрашивает название животного и число, а затем выводит на экран фразу типа "Белка съест 10 грибов" (при вводе слова "белка" и числа 10).

7. Организуйте диалог продавца (компьютер) и покупателя (пользователь) при покупке какого-либо товара по следующей схеме: предложение товара по определенной цене, запрашивание количества покупаемого товара, определение и вывод на экран денежной суммы, которую должен заплатить покупатель за покупку.

Лабораторная работа №4
Решение задач с использованием линейных алгоритмов

1. Цель работы: приобретение практических навыков работы в интегрированной среде, изучение структуры программы на языке С.

2. ОСНОВНЫЕ СВЕДЕНИЯ

2.1. Алфавит и идентификаторы языка Си

Алфавит.

В алфавит языка Си входят:

- прописные и строчные буквы латинского алфавита;
- цифры;
- специальные знаки “ , {} | [ ] ( ) + - / % \ ; ‘ . : ? < = > _ ! & * # ~ ^
- неизображаемые символы, например, пробел, табуляция.

Идентификатор.

Последовательность букв, цифр, символов подчеркивания, начинающаяся с буквы или символа подчёркивания, считается идентификатором языка Си. Например, Lab_1, size22, _min, TIME, time.

Прописные и строчные буквы различаются, т.е. два последних идентификатора различны. Идентификаторы могут иметь различную длину, но компилятор учитывает не более 31-го символа.

2.2. Переменные

Переменная – это именованная область памяти. Отличительная особенность переменной – возможность связывать с её именем различные значения, совокупность которых определяется типом переменной. Каждая переменная перед её использованием в программе должна быть определена, т.е. для неё должна быть выделена память. Размер участка памяти зависит от типа переменной.

Основные типы переменных и их предельно допустимые значения:

char	-128...127	1 байт
unsigned char	0...255	1 байт
int	-32768...32767	2 байта
unsigned int	0...65535	2 байта
long	-2147483648...2147483647	4 байта
unsigned long	0...4294967295	4 байта
float	3.4E-38...3.4E+38	4 байта
double	1.7E-308...1.7E+308	8 байт
long double	3.4E-4932...1.1E+4932	10 байт

2.3. Операции

1) Операция присваивания : “=”.

2) Аддитивные операции : “ +”, ”-“;

3) Мультипликативные : “*”, ”/”, ”%”; % - получение остатка от деления, “/” — целочисленное деление (остаток отбрасывается);

4) Логические операции : && (and), || (or), ! (not);

5) Операции отношения : == (равно), != (не равно), < (меньше), > (больше),<= (меньше или равно), >= (больше или равно);

6) Операции модификации :

= присваивание после умножения (p= 2; эквивалентно p=p*2;)

/= присваивание после деления (p /= 2;эквивалентно p=p/2;)

+= присваивание после суммирования (a += b; эквивалентно a=a+b;)

-= присваивание после вычитания (a -= b; эквивалентно a=a-b;)

%= присваивание после деления по модулю (n %= 3; эквивалентно n=n % 3;)

a++ (--) постфиксное изменение (a++; эквивалентно a=a+1; значение a сначала используется, затем изменяется)

++(--) a префиксное изменение (++a; эквивалентно a=a+1; значение a сначала изменяется, затем используется).

2.4. Программирование алгоритма линейной структуры на языке Си

Чаще всего линейные алгоритмы используются для программирования вычислений по формулам. В этом случае удобно использовать набор стандартных функций Си, содержащихся в библиотеке и доступных при подключении заголовочного файла math.h:

sin (x) — sin x;

cos (x) — cos x;

tan(x) — tg x

log (x) — ln x;

log10(x) – lg x;

exp(x) — показательная функция e^x;

sqrt (x) — корень квадратный от x;

pow(x, y) — x в степени y;
abs(x) — модуль x;
acos(x) — arccos x;
asin(x) – arcsin x;
atan(x) — arctg x;
sinh(x) — sh x;
cosh (x) — ch x;
tanh(x) — th x.
Для тригонометрических функций аргумент x измеряется в радианах и имеет тип double, как и значения функций.

Пример программы линейной структуры:

С клавиатуры вводятся значения длины гипотенузы c и угла x. Вычислить длины катетов и площадь прямоугольного треугольника. Для вычислений воспользуемся формулами: a = c·sin x; b = c cos x; S = ab/2;

Программа вычислений имеет вид:

```
// вычисление длин сторон и площади треугольника:
#include <stdio.h>
#include <math.h>
int main(void)
{ float a, b, c, x, S;
printf ("\nEnter c:");
scanf ("%f", &c);
printf ("\nEnter x:");
scanf ("%f", &x);
a=c*sin(x);
b=c*cos(x);
S=a*b/2;
printf("\na = %.2f     b = %.2f      S = %.2f ", a,
b, S);
getchar();
return 0;
}
```

Объяснение примера:

Комментарий в программе не влияет на компиляцию программы, а служит для разъяснения действий основных блоков текста и программы в целом. Однострочный комментарий действует от двух символов // до конца строки. Многострочный комментарий заключается в пары символов /* и */.

Первая строка программы #include <stdio.h> является директивой препроцессора для доступа к средствам ввода-вывода (связи с внешними устройствами), отсутствующими в самом языке Си. Вторая строка — #include<math .h> является директивой компилятора для включения заголовочного файла math.h, обеспечивающего выполнение математических функций.

Программа на языке Си состоит из ряда функций, из которых функция main (главная) является обязательной и служит точкой входа в программу. В круглые скобки заключаются параметры функции, причем наличие круглых скобок обязательно, если даже список параметров пуст. В фигурные скобки заключаются составной оператор (несколько операторов).

Для придания тексту программы наглядности открывающая и соответствующая ей закрывающая фигурные скобки печатаются на одном уровне, а заключенный между ними текст сдвигается на 1-2 символа вправо,вложенный блок

также сдвигается вправо и т.д. Образуется иерархия вложенных блоков, придающая программе на Си характерный вид.

В программе описаны переменные a, b, c, x, S вещественного типа (float). В отличие от других языков в Си учитывается регистр при определении имени переменной, т. е. s и S – разные переменные.

Функции printf и scanf доступны при подключении заголовочного файла stdio.h и служат для вывода на экран и ввода с клавиатуры соответственно.

Управляющие символы \n в функции printf служат для перевода на новую строку. Символ & в функции scanf указывает на адрес вводимой переменной.

Ввод и вывод переменных вещественного типа производится в формате f. Признаком форматного вывода в функции printf является %. При выводе между знаком процента и форматной переменной f можно включить общую ширину поля вывода и число позиций после десятичной точки.

Оператор return служит для выхода из функции main. Завершает текст программы закрывающая фигурная скобка, означающая конец функции main.

3 Задание

На оценку «удовлетворительно»

Наберите и выполните приведенную выше программу вычисления длин катетов и площади треугольника по значениям длины гипотенузы и острого угла, вводимым с клавиатуры.

Контрольные задания:

1. Добавить вывод значения cos(x).

2. Изменить строку в программе с выводом a, b, S так, чтобы значения a, b и S выводились каждое с новой строки.

3. Изменить строку программу так, чтобы числовые значения a, b, S были бы выровнены относительно точки.

Например:

a = 1.13
b = 21.53
S = 10.79

4. Изменить строку с выводом так, чтобы значения a, b, S выводились до третьего знака после точки.

5. Изменить программу так, чтобы в начале она спрашивала ваше имя и выводила затем строку "Hello, <ваше имя>!".

На оценку «хорошо»

Вычислить значение функции

$y=x^3+\sin(x)$, при x=1.2**На оценку «отлично»**

Составьте и выполните программы линейной структуры согласно следующим заданиям.

Варианты согласно номеру в журнале:

№	Задание	Формула
1	Дана длина ребра куба H. Найти объем куба V и площадь его боковой поверхности S.	$V = H^3$ $S = H^2$
2	Определить время t падения камня на поверхность земли с высоты h.	$t = \sqrt{\frac{2H}{g}}$, g=9.81523 м/с^2

3	Известна длина окружности. Найти площадь круга S, ограниченного этой окружностью.	$S = \frac{\pi}{4} D^2$
4	Треугольник задан координатами своих вершин. Найти: периметр треугольника P; площадь треугольника S.	$a = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$ $b = \sqrt{(x3 - x2)^2 + (y3 - y2)^2}$ $c = \sqrt{(x1 - x3)^2 + (y1 - y3)^2}$ $P = (a + b + c)/2$ $s = \frac{1}{2}[(x_1 - x_3)(y_2 - y_3) - (x_2 - x_3)(y_1 - y_3)]$
5	Три сопротивления R1, R2, R3 соединены параллельно. Найти сопротивление соединения.	$\frac{1}{R} = \frac{1}{R1} + \frac{1}{R2} + \frac{1}{R3}$
6	По данным сторонам прямоугольника вычислить его периметр P, площадь S и длину диагонали.	$d = \sqrt{a^2 + b^2}$ $S = ab$ $P = 2(a + b)$
7	Определить координаты вершины параболы. Коэффициенты a, b, c заданы.	$x = -\frac{b}{2a}$, $y = ax^2 + bx + c$
8	Вычислить площадь треугольника по формуле Герона, если заданы его стороны.	$P = \frac{A + B + C}{2}$ $S = \sqrt{P(P - A)(P - B)(P - C)}$
9	Определить расстояние S и скорость v, пройденное физическим телом за время t, если тело движется с постоянным ускорением a и имеет в начальный момент времени скорость v0.	$v = v0 + at$ $S = v0 * t + \frac{at^2}{2}$
10	Вычислить объем и площадь цилиндра с радиусом основания r и высотой h.	$V = \pi r^2 h$ $S = 2\pi rh$
11	Вычислить высоту треугольника, опущенную на сторону a, по известным значениям длин его сторон a, b, c.	$H = \frac{2}{a}\sqrt{p(p - a)(p - b)(p - c)}$

12	В квадратной комнате шириной A и высотой B есть окно и дверь с размерами $c * d$ и $n * m$ соответственно. Вычислите площадь стен для оклеивания их обоями.	$S1 = c * d$ $S2 = n * m$ $S = 4ab - s1 - s2$
13	Найти: площадь трапеции S, если заданы стороны a, b, c, d.	$P = \frac{A + B + C + D}{2}$ S $= \frac{A + B}{4\lvert A - B\rvert}\sqrt{(P - A)(P - B)(P - A - C)(P - A}$
14	Вычислить путь, пройденный лодкой S, если ее скорость в стоячей воде v км/ч, скорость течения реки v1 км/ч, время движения по озеру t1 ч, а против течения реки – t2 ч.	$s1 = v * t1$ – путь по озеру $s2 = t2 * (v - v1)$– путь по реке $S = S1 + S2$– весь путь
15	Определить объем V и температуру T смеси двух жидкостей при заданных значениях $v1, t1, v2, t2$.	$V = V1 + V2$ $T = (v1 * t1 + v2 * t2)/(v1 + v2)$

Контрольные вопросы:

1.Понятие идентификатора.
2.Определение переменной. Основные типы переменных.
3.Типы операций.
4.Структура программы на языке Си.
5.Для чего используются заголовочные файлы?
6.Как откомпилировать программу?
7.Как выполнить программу?
8.Как просмотреть результат выполнения?

Лабораторная работа № 5
Решение задач с использованием разветвляющихся алгоритмов

Цель работы: научиться правильно использовать условный оператор if; научиться составлять программы решения задач на разветвляющиеся алгоритмы.

Общие сведения

Алгоритм называется разветвляющимся, если он содержит несколько ветвей, отличающихся друг от друга содержанием вычислений. Выход вычислительного процесса на ту или иную ветвь алгоритма определяется исходными данными задачи.

Ветвление осуществляется с помощью оператора if, который имеет следующий формат:

if (выражение) оператор-1; [else оператор-2;]

Выполнение оператора if начинается с вычисления выражения. Далее выполнение осуществляется по следующей схеме:

- если выражение истинно (т.е. отлично от 0), то выполняется оператор-1.

- если выражение ложно (т.е. равно 0),то выполняется оператор-2.
- если выражение ложно и отсутствует оператор-2 (в квадратные скобки заключена необязательная конструкция), то выполняется следующий за if оператор.

Пример:

```
if (i < j)i++;
else {j = i-3; i++; }
```

В данном примере показано, что на месте оператор-1, так же как и на месте оператор-2могут находиться сложные конструкции.

Допускается использование вложенных операторов if. Чтобы сделать программу более читабельной, рекомендуется группировать операторы и конструкции во вложенных операторах if, используя фигурные скобки даже если они не требуются. Если же фигурные скобки опущены, то компилятор связывает каждое ключевое слово else с наиболее близким if, для которого нет else.

Примеры:

```
if (t>b)
{
if (b < r)r=b;
}
else r=t;
```

Если же в программе опустить фигурные скобки, стоящие после оператора if, то программа будет иметь следующий вид:

```
if ( a>b )
if ( b < c ) t=b; else r=t;
```

В этом случае else воспринимается как часть второго оператора if.

В операторе ветвления, используемом в языке Си, для написания выражений используются следующие операции отношения:

```
>больше
<меньше
>=больше или равно
<=меньше или равно
==равно
!=не равно
```

Операнды операций отношения могут быть целого типа, плавающего типа или типа указателя, при этом в каждой операции могут участвовать операнды различных типов. Результат целый: 0 (ложь) или 1 (истина).

Кроме арифметических операций языка С можно применять логические операции, к которым относятся:

```
&& операция логического "И",
|| операция логического "ИЛИ",
! операция логического "НЕ".
```

Операнды логических операций могут быть целого типа, плавающего типа или типа указателя, при этом в каждой операции могут участвовать операнды различных типов.

Операнды логических выражений вычисляются слева направо. Если значения первого операнда достаточно, чтобы определить результат операции, то второй операнд не вычисляется.

В языке Си нет логического типа как в языке Паскаль. Вместо значений true и false в языке Си принято нулевое значение за истину, а не нулевое (чаще единица) – за ложь.

Результатом логической операции является 0 или 1, тип результата int. Например:

```
3<5результат 1
3>5результат 0
6!=1 || 2>5результат 1
```

8>2 && 4>5результат 0

В языке СИ имеется одна тернарная операция – операция условие (не путать с условным оператором), которая имеет следующий формат:

операнд-1 ? операнд-2 : операнд-3;

Операнд-1 должен быть целого или плавающего типа или быть указателем. Он оценивается с точки зрения его эквивалентности 0. Если операнд-1 не равен 0 (истина), то вычисляется операнд-2 и его значение является результатом операции. Если операнд-1 равен 0 (ложь), то вычисляется операнд-3 и его значение является результатом операции. Следует отметить, что вычисляется либо операнд-2, либо операнд-3, но не оба. Тип результата зависит от типов операнда-2и операнда-3.

Пример:

max = (d<=b) ? b : d;

Здесь переменной max присваивается максимальное значение переменных d и b.

Обращаем внимание на тот факт, что каждый оператор, каждое определение и каждое описание в программе на языке Си завершает точка с запятой ";". Любое допустимое выражение, за которым следует ";", воспринимается как оператор.

При выполнении арифметических и логических операций следует следить за приоритетами этих операций и порядком их вычисления (таблица 2.1).

Табл. 2.1 Приоритеты операций

Ранг	Операция	Ассоциативность
1	() [] -> .	справа налево
2	! ~ -(смена знака) ++ -- &(адрес) * (тип) sizeof	слева направо
3	* / %	справа налево
4	+ -	справа налево
5	<< >>	справа налево
6	< <= >= >	справа налево
7	== !=	справа налево
8	&	справа налево
9	^	справа налево
10	|	справа налево
11	&&	справа налево
12	||	справа налево
13	?	слева направо
14	= *= /= %= += -= &= ^= |= <<= >>=	слева направо
15	,	справа налево

В языке СИ операции с высшими приоритетами вычисляются первыми. Наивысшим приоритетом является приоритет равный 1. Операции одного ранга имеют одинаковый приоритет, и если их в выражении несколько, то они выполняются в соответствии с правилом ассоциативности, либо слева направо, либо справа налево.

Использование оператора if рассмотрим на следующем примере.

Пример1: Дано действительное a. Для функций f(a),график которой представлен на рисунке 2.1, вычислить f(a).

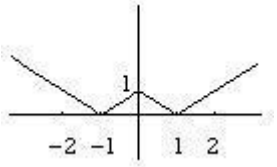


Рис. 2.1. График функции.

Этапы решения задачи:

1. Математическая модель: функция вычисляется по следующей формуле

$$f(x)=\begin{cases} -x-1, & x<-1 \\ x-1, & -1\le x<0 \\ -x+1, & 0\le x<1 \\ x+1, & x\ge 1 \end{cases}$$

2. Составим схему алгоритма (рисунок 2.2).

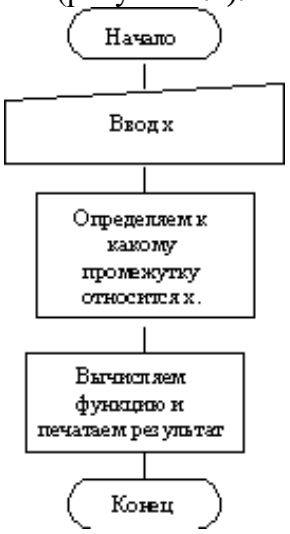


Рис. 2.2. Алгоритм решения задачи.

Детализируем блок «Определяем к какому промежутку относится x» (рисунок 2.3).

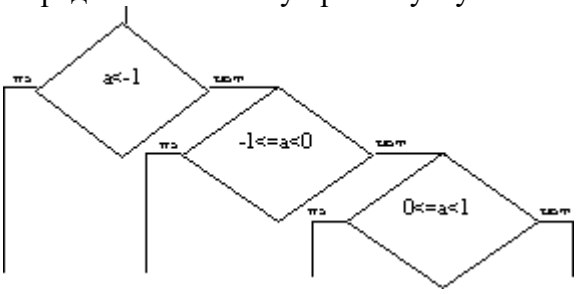


Рис. 2.3. Ветвление блока.

Добавим блоки вычисления функции на каждом из промежутков (рисунок 2.4).

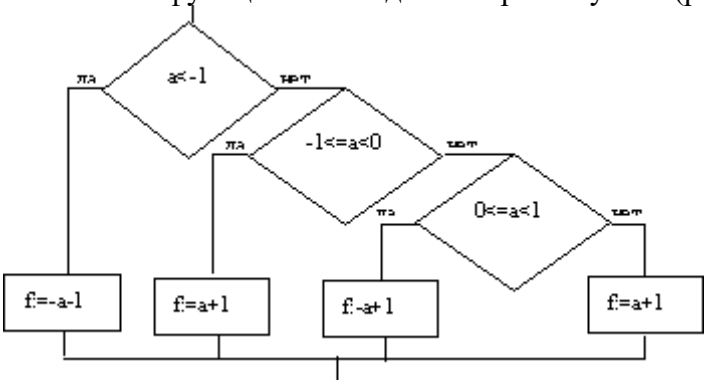


Рис. 2.4. Блок определения промежутка для переменной x.

Окончательный алгоритм представлен на рисунке 2.5.

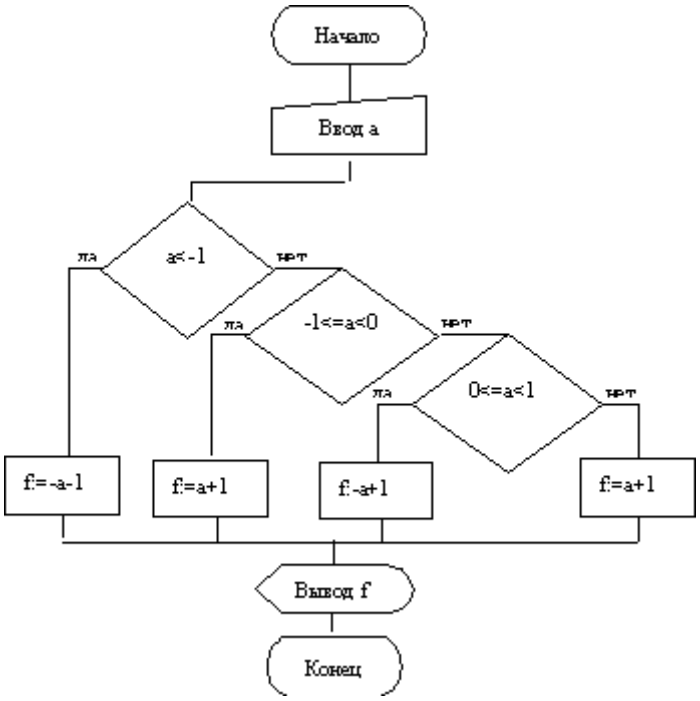
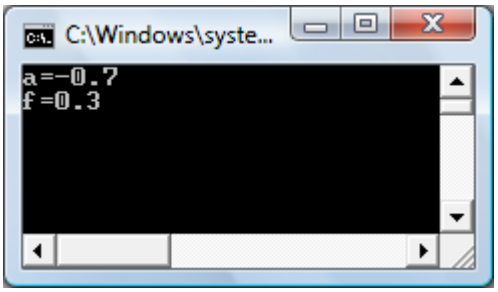


Рис. 2.5. Общая схема вычисления функции по заданному значению x. Дальнейшая детализация не требуется. Переводим блок-схему на язык С:

```
#include <stdio.h>
int main()
{
float a, f;
printf("a=");
scanf("%f",&a);
if (a<-1)  f= -a-1;
else if ((a>=-1) && (a<0))
f = a+1;
else  if ((a>=0) && (a<1))
f= -a+1;
else f = a+1;
printf("%5.2f f=",f);
return 0;
}
```

Реализовав данную программу в интегрированной среде программирования, получим результирующее окно, представленное на рисунке 2.6.

Рис. 2.6. Окно выполнения задачи.



В завершение работы хотелось бы привести перечень служебных символов (табл. 2.2), которые используются с одной стороны для организации процесса вычислений, а с

другой – для передачи компилятору определенного набора инструкций.

Таблица 2.2 Специальные символьные комбинации

Комбинация	Наименование
\a	Звонок
\t	Горизонтальная табуляция
\n	Переход на новую строку
\v	Вертикальная табуляция
\r	Возврат каретки
\"	Кавычки
\'	Апостроф
\0	Символ с кодом ноль
\\	Обратная дробная черта

Примеры использования управляющего символа:

cout << "Это строка один\nЭто строка два"; \\вывод двух строк cout << 1 << '\n' << 0 << '\n' << 0 << '\n' << 1 \\выводкаждого

\\ из чисел 1, 0, 0 и 1, каждое на отдельной строке

Задание 1. Составить программу в соответствии с вариантом. Задачи для индивидуального решения:

! Внимание! Необходимо выбрать одну задачу под тем же номером, под которым стоит ваша фамилия в списке подгруппы

1) $y = \begin{cases} m^2 + 4n, m > n \\ mn, m \le n \end{cases}$

2) $y = \begin{cases} m + 2n, m > n \\ m^2 + 4, m \le n \end{cases}$

3) $y = \begin{cases} m^3 + 2, m > 0 \\ m^2, m \le 0 \end{cases}$

4) $y = \begin{cases} m - n, m > n \\ n - m, m \le n \end{cases}$

5) $y = \begin{cases} n^5, m > n \\ mn, m \le n \end{cases}$

6) $y = \begin{cases} a^2 - b, a > b \\ b^4 - a, a \le b \end{cases}$

7) $y = \begin{cases} 4n^4, m > n \\ m + n, m \le n \end{cases}$

8) $y = \begin{cases} a + 3b, a > b \\ a^2 b, a \le b \end{cases}$

9) $y = \begin{cases} a + b, a > b \\ a + b^2, a \le b \end{cases}$

10) $y = \begin{cases} a^2 + b, a > b \\ a^3 + b, a \le b \end{cases}$

11) $y = \begin{cases} mn^2, m > 0 \\ n^4 - m, m \le 0 \end{cases}$

12) $y=\begin{cases}a^3-4ab, a>0\\ b^2+4a, a\le 0\end{cases}$

13) $y=\begin{cases}a-b, a>b\\ b-a^3, a\le b\end{cases}$

14) $y=\begin{cases}m^3, m>n\\ (m+n)^3, m\le n\end{cases}$

15) $y=\begin{cases}m-\dfrac{n}{2}, m>n\\ mn, m\le n\end{cases}$

Задание 2. Составить программу в соответствии с вариантом. Задачи для индивидуального решения:

1. $f(x)=\begin{cases}5x^2+6, \ если \quad -2<x<5;\\ x^3+7, \ если \quad x\ge 5;\end{cases}$ 2. $f(x)=\begin{cases}\sqrt{x^3+5}, \ если \quad x\ge 0;\\ 3x^4+9, \ если \quad -3<x<0;\end{cases}$

3. $f(x)=\begin{cases}\sqrt[3]{x^2+3}+6x^2, \ если \quad -4<x\le 5;\\ x^5+3.5, \qquad если \quad x>0;\end{cases}$ 4. $f(x)=\begin{cases}x\cos x, \qquad если \quad x<1.22;\\ 5x^3+1.7, \ если \quad x\ge 1.22;\end{cases}$

5. $f(x)=\begin{cases}x^3\cos x, \qquad если \quad 0<x<2;\\ 3x^4+7, \qquad если \quad x\ge 2;\\ \sqrt{5x^2+1.6}, \ если \quad 5<x\le 9;\end{cases}$ 6. $f(x)=\begin{cases}xtgx-\sin x, \ если \quad x<1.5;\\ x^3+\sin x, \quad если \quad 1.5\le x<2.5;\\ 3x^3+5, \qquad если \quad x\ge 2.5;\end{cases}$

7. $f(x)=\begin{cases}\sqrt{3x^3+4}+\cos, \ если \quad 0<x<1;\\ 5-\sin^2 x9, \qquad если \quad x\ge 1;\end{cases}$ 8. $f(x)=\begin{cases}\sqrt{x^2+|x|}, \quad если \quad -5<x<0;\\ 5x^3+\cos x, \ если \quad 0\le x<2;\end{cases}$

9. $f(x)=\begin{cases}x\cos^2 x+\sin x, \ если \quad 0\le x<1;\\ \sqrt[3]{x^2+6\sin x}, \ если \quad 1\le x\le 2;\\ 1.7x^3+7, \qquad если \quad x>2;\end{cases}$ 10.

$f(x)=\begin{cases}x^2\sin x+\sqrt{x^2+1.2}, \ если \quad x>0;\\ tg^2 x+\cos x+3.5, \quad если \quad 2<x\le 6;\end{cases}$

11. $f(x)=\begin{cases}\sin \pi x+\sqrt[5]{x^2+6}, \ если \quad 0<x\le 4;\\ \ln x+\sqrt{3x+7}, \quad если \quad x>4;\end{cases}$ 12.

$f(x)=\begin{cases}\sqrt[3]{1.7x+\sin x}, \ если \quad 0<x\le 2;\\ \cos \pi x+tgx, \quad если \quad 2<x\le 6;\end{cases}$

13. $f(x)=\begin{cases}\sin 5x*(\sqrt{1+x^2}+\ln^2 x), & если \quad 1<x\le 3;\\ e^x*\sqrt[3]{x+e^{2x}}, & если \quad 3<x\le 5;\end{cases}$

14. $f(x)=\begin{cases}\sqrt[3]{|x|+x^4}, & если \quad -1<x\le 1;\\ tg^2 3x+\ln 5x, & если \quad 1<x\le 4;\end{cases}$

15. $f(x)=\begin{cases}\sqrt[4]{x^3+\sin \pi x}+\ln x, & если \quad 1<x\le 3;\\ \sqrt[3]{1.5x^2-e^{2x}}+\ln^3 x, & если \quad 3<x\le 7;\end{cases}$

Лабораторная работа № 6
Решение задач с использованием оператора выбора

В программах с множественным выбором используется переключатель switch , который сравнивает значение выражения, указанного за ним, и выполняет один из операторов, метка которого совпадает с этим значением. Общий вид:

```
Switch (выражение)
{ case метка_1: список_операторов_1;
…….
case метка_n: список_операторов_n;
default : операторы; }
```

Значения выражения и меток должны быть целочисленными константами.

Например, определим количество дней по введенному номеру месяца.

```
//lab2_2   множественный выбор
 #include<stdio.h>
int m;
 int main()
{
 printf("\n Введите номер месяца :");  scanf("%d",&m);
 printf("\n В  %4d месяце дней: ",m);
 switch(m){
 case 1:case 3:case 5: case 7:case 8:case 10:case 12:printf("тридцать один\n");break;
 case 2: printf ("двадцать восемь \n");break;
 case 4: case 6: case 9: case 11: printf("тридцать\n");break;
 default: printf("\n Номер  месяца   неверен \n");}
}
```

В примере программы, если номер месяца превышает 12, выводится сообщение о неверном вводе месяца, для чего используется default. Оператор break служит для прерывания цикла проверки и перехода в конец переключателя. В случае отсутствия break, происходит переход на следующую ветвь.

3. Выполнение работы

Задание 1 Составить программы разветвляющейся структуры, используя SWITCH. Дано целое число в диапазоне 1 – 5. Вывести строку — словесное описание

соответствующей оценки (1 — "плохо", 2 — "неудовлетворительно", 3 — "удовлетворительно", 4 — "хорошо", 5 — "отлично").

Задание 2. В зависимости от введённого символа L, S, V программа должна вычислять длину окружности; площадь круга; объём цилиндра.

Задание 3. Составить программы разветвляющейся структуры согласно вариантам задания, используя SWITCH .

Варианты задания

! Внимание! Необходимо выбрать одну задачу под тем же номером, под которым стоит ваша фамилия в списке подгруппы

1) Дан порядковый номер месяца, вывести на экран количество месяцев, оставшихся до конца года.
2) Дан порядковый номер дня месяца, вывести на экран количество дней, оставшихся до конца месяца.
3) Дан номер масти m (1 <= m <= 4), определить название масти. Масти нумеруются: "пики" - 1, "трефы" - 2, "бубны" - 3, "червы" - 4.
4) Дан номер карты k (6 <= k <= 14), определить достоинство карты. Достоинства определяются по следующему правилу: "туз" - 14, "король" - 13, "дама" - 12, "валет" - 11, "десятка" – 10,…., "шестерка" - 6.
5) Дано число M - номер месяца. Определить номер квартала по введенному номеру месяца и номер полугодия.
6) Дано расписание приемных часов врача. Вывести на экран приемные часы врача в заданный день недели (расписание придумать самостоятельно).
7) Проведен тест, оцениваемый в целочисленный баллах от нуля до ста. Вывести на экран оценку тестируемого в зависимости от набранного количества баллов: от 90 до 100 - "отлично", от 70 до 89 - "хорошо", от 50 до 69 - "удовлетворительно", менее 50 - "неудовлетворительно".
8) Дан год. Вывести на экран название животного, символизирующего заданный год по восточному календарю.
9) Дан возраст человека мужского пола в годах. Вывести на экран возрастную категорию: до года - "младенец", от года до 11 лет - "ребенок", от 12 до 15 лет - "подросток", от 16 до 25 лет - "юноша", от 26 до 70 лет - "мужчина", более 70 лет - "старик".
10) Дан признак транспортного средства: а - автомобиль, в - велосипед, м - мотоцикл, с - самолет, п - поезд. Вывести на экран максимальную скорость транспортного средства в зависимости от введенного признака.
11) Дан признак геометрической фигуры на плоскости: к - круг, п - прямоугольник, т - треугольник. Вывести на экран периметр и площадь заданной фигуры (данные, необходимые для расчетов, запросить у пользователя).
12) Написать программу, которая выводит название дня недели по его порядковому номеру
13) Задан номер текущего дня недели. Вывести названия дней, оставшихся до конца недели, включая текущий день
14) Написать алгоритм нахождения числа дней в месяце, если даны: Номер месяца n - целое число a, равное 1 для високосного года и равное 0 в противном случае.
15) Написать алгоритм нахождения числа дней в месяце, если даны: Номер месяца n - целое число a, равное 1 для високосного года и равное 0 в противном случае.

Контрольные вопросы

1) Какой оператор используется для выхода из оператора switch?
2) Для чего используется метка default в операторе switch?
3) Можно ли в телеоператора switch использовать вложенные операторы switch?

Лабораторная работа 7.
Решение задач с использованием операторов итерационного типа

Цель работы: ознакомиться с циклическими алгоритмами и операторами, реализующими эти алгоритмы. Освоить особенности применения каждого оператора. Составить программы с использованием всех операторов цикла.

Оператор цикла for

Описание: for (выражение 1; выражение 2; выражение 3) оператор;

Действие:

В круглых скобках содержится три выражения. Первое из них служит для инициализации счетчика. Она осуществляется только один раз – когда цикл for начинает выполняться. Второе выражение необходимо для проверки условия, которая осуществляется перед каждым возможным выполнением тела цикла. Когда выражение становится ложным, цикл завершается. Третье выражение вычисляется в конце каждого выполнения тела цикла, происходит приращение числа на шаг.

Комментарий: в операторе цикла for точка с запятой после закрывающейся круглой скобки не ставится. Любое из трех или все три выражения в операторе могут отсутствовать, однако разделяющие их точки с запятыми опускать нельзя. Если отсутствует выражение 2, имеем бесконечный цикл. Например:for (scanf("%d",&p);;p++) оператор;

В языке СИ предусмотрены две нетрадиционные операции: (++) – для увеличения на единицу и (--) – для уменьшения на единицу значения операнда. Операции ++ и -- можно записывать как перед операндом, так и после него. В первом случае (++n или --n) значение операнда (n) изменяется перед его использованием в соответствующем выражении, а во втором (n++ или n--) – после его использования.

Если отсутствуют выражения 1 и 3, цикл становится эквивалентным while. Например:for (;a<20;) оператор;

Каждое из выражений может состоять из нескольких выражений, объединенных операцией "запятая". Например: for(i=0, j=1; i<100; i++, j++) a[i]=b[j];

Тело цикла заключается в фигурные скобки, если в нем более одного оператора.

Пример:

```
/*демонстрация цикла for*/
#include <stdio.h>
main()
{int i,j=1,k;
for (i=1;i<=3;i++)
printf("Костанай\t");
/*В цикле for три раза выполняется функция вывода*/
/*Здесь i-управляющая переменная цикла*/
printf("\nУкажите число повторений цикла\n");
scanf("%d",&k);
for (i=1;i<=k;i++)
{j*=i;
printf("%d",j);}
/*Здесь две инструкции (более одной), поэтому они заключаются в фигурные скобки*/
j=i;
printf("\n");
/*Переменной j присваивается значение 1 и осуществляется перевод курсора*/
/*В следующем цикле for выполняются те же действия, что и в предыдущем*/
for (i=1;i<=k;i++) printf("%d ", j*=i);
```

}
Результаты выполнения программы следующие:
Костанай Костанай Костанай
Укажите число повторений цикла; 5
1 2 6 24 120
1 2 6 24 120

Задание
Задание взять согласно варианту. Написать и отладить программы.

1 $$k=\sum_{i=1}^{5}\frac{(x+y)i}{(xi+yi)^2}$$

2 $$k=\prod_{i=1}^{25}\sqrt{|xi+y|}$$

3 $$k=\sum_{i=1}^{5}(a+i)\cos(xi)$$

4 $$k=\sum_{i=1}^{5}(2x^2i+xi+17i)$$

5 $$k=\prod_{i=1}^{7}(2x+17i)$$

6 $$k=\prod_{i=1}^{7}\frac{\sqrt{xi+y+z}}{i}$$

7 $$k=\sum_{i=1}^{7}(x+yi)$$

8 $$k=\prod_{i=1}^{10}(x+i)$$

9 $$k=\prod_{i=1}^{5}(2x^3+25i)$$

10 $$k=(x^2+y^2)!$$

11 $$k=\sum_{i=5}^{10}(x+yi)$$

12 $$k=\sum_{i=1}^{10}(xi+25)$$

13 $$k = \sum_{i=5}^{10} (x + 25yi + 13z)$$

Лабораторная работа № 8.
Решение задач с использованием операторов пост условия и предусловия

1. Теоретические сведения

Оператор цикла while

Описание: while (выражение) оператор;

Действие: Выполняется оператор до тех пор, пока значение выражения в скобках истинно. Проверка значения выражения происходит перед каждым выполнением оператора. Когда значение выражения ложно, цикл while заканчивается. Если выражение ложно с самого начала, оператор не выполняется ни разу.

Комментарий: Следует заметить, что после ключевого слова while и выражения, заключенного в круглые скобки, точка с запятой не ставится.

Оператор иногда называется телом цикла. В теле цикла должны выполняться действия, в результате которых меняется значение управляющего выражения. В противном случае можем получить бесконечный цикл.

Пример:

```
/*Демонстрация цикла while*/
#include <stdio.h>
main( )
{
int i=1
while (getchar()!=’r’) i++;
/*оператор getchar() вводит любой символ с клавиатуры*/
printf("Символ R %d-й",i);
}
```

Приведенная программа позволяет определить порядковый номер первой введенной буквы R в последовательности символов. Она показывает использование цикла while, в теле которого всего одна инструкция (i++ - увеличение значения целого числа i на единицу). Если запустить эту программу на выполнение и ввести последовательность символов, например: abFk!Rgm, то на экране появится строка: Символ R 6-й.

Оператор цикла do- while

Описание: do оператор while (выражение);

Действие: В операторе do-while тело цикла выполняется по крайней мере один раз. Тело цикла будет выполняться до тех пор, пока выражение в скобках не примет ложное значение. Если оно ложно при входе в цикл, то его тело выполняется ровно один раз.

Комментарий: после слова while и выражения, заключенного в скобки, ставится точка с запятой. Если в теле цикла содержится более одной инструкции, то операторы цикла заключаются в фигурные скобки.

Пример:

```
/*Демонстрация цикла do-while */
#include <stdio.h>
main()
{
int i=0; /*i=0, а не единице*/
```

```
do i++; while (getchar()!='r');
printf("Символ R %d-й",i);
}
```

Программа, представленная выше, теперь написана с циклом do-while. Результат программы будет таким же.

Задание

Написать программу двумя способами и отладить программы.

1. Составьте программу, выводящую на экран квадраты чисел от 10 до 20 включительно.
2. Составить программу вычисления суммы квадратов чисел от 1 до n.
3. Составить программу вычисления суммы первых 10 членов последовательности, с использованием циклов пост и предусловия. Вариант последовательности выбрать согласно номеру в журнале

1. $\frac{1}{n+12}$	2. $\frac{n+1}{3n+3}$	3. $\frac{n^2+1}{n^3}$
4. $\frac{2n+3}{4n+5}$	5. $\frac{\sin(n)}{2n+12}$	6. $\frac{n+1}{2n}$
7. $\frac{\cos(n)}{4n+1}$	8. $\frac{arctg(n)}{5n+1}$	9. $\frac{2n^2+1}{\ln(n)}$
10. $\frac{\log(n)}{n+1}$	11. $\frac{\sin(n)}{2n+1}$	12. $\frac{2n+3}{n^3}$
13. $\frac{2n+2+\ln(n)}{n^3}$	14. $\frac{\cos(2n+1)}{n^2+1}$	15. $\frac{\sin(3n+1)}{n+1}$

Лабораторная работа № 9.
Оператор безусловного перехода goto, операторы continue, break

Оператор break применим в случаях, когда требуется преждевременно прервать цикл. Например:

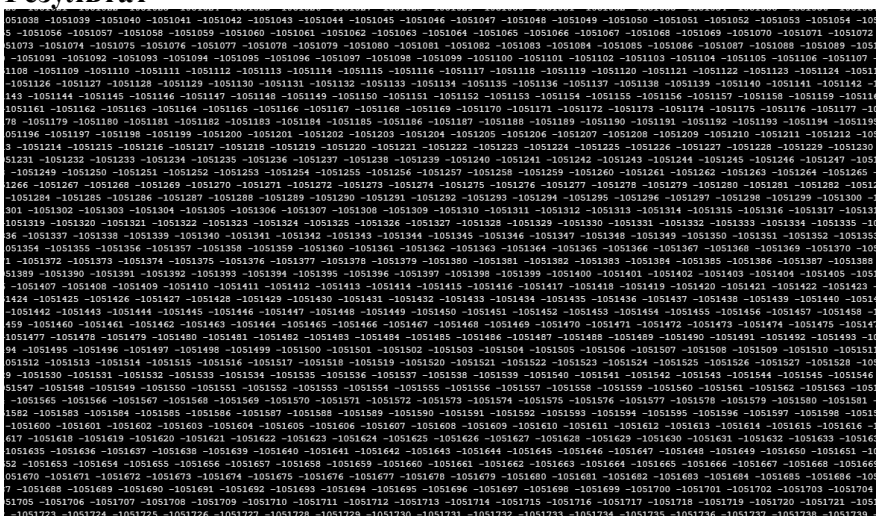
```c
#include <stdio.h>

int main()
{
  int a;
  scanf ("%d", &a);
  do
  {
      a--;
      printf ("%d",a);
      printf("\n");
       }
  while (a<30);

  return 0;
}
```

В данном случае, есть цикл с постусловием. Переменная целочисленного типа задается с клавиатуры и далее в цикле осуществляется уменьшение значения переменной на единицу с последующим выводом результата. Цикл будет продолжаться если значение переменной меньше 30-ти

Результат



Допустим прерывание требуется на моменте, когда значение переменной будет равно -1, тогда код будет выглядеть следующим образом:

```c
#include <stdio.h>

int main()
{
  int a;
  scanf ("%d", &a);
  do
  {
      if (a==-1)
      break;
      a--;
      printf ("%d",a);
      printf("\n");
       }
  while (a<30);

  return 0;
}
```

Результат

Существует возможность перенаправлять последовательность выполнения кода. Это реализуется при помощи оператора continue. Предположим, что на вход поступает натуральное число, в цикле к нему прибавляется единица, до тех пор пока значение не достигнет 100. Из всей последовательности, на экран требуется вывести только четные числа. Подобная задача легко решается следующим алгоритмом:

```
8
9  #include <iostream>
10 using namespace std;
11
12 int main()
13 {
14    int a;
15    scanf ("%d", &a);
16    do
17    { a++;
18    if (a%2==1) continue;
19    printf ("%d\n",a);
20    }
21
22    while (a<100);
23    return 0;
24 }
25
```

Сначала к введенному значению прибавляется 1, далее полученный результат проверяется на четность и если он четный, то цикл продолжается и на экран выводится значение. Но, если результат нечетный, то исполнение перенаправляется в начало цикла

Задание:

1. На вход поступает натуральное число, создается последовательность путем прибавления к начальному значению единицы. Последовательность оканчивается, когда значение достигает 200. Требуется вывести на экран только нечетные числа.
2. На вход поступает натуральное число, создается последовательность путем прибавления к начальному значению единицы. Последовательность оканчивается, когда значение достигает 100. Требуется вывести на экран каждое третье число.
3. На вход поступает натуральное число, создается последовательность путем прибавления к начальному значению единицы. Последовательность оканчивается, когда значение достигает 500. Требуется вывести на экран каждое десятое число.
4. На вход поступает натуральное число, создается последовательность путем прибавления к начальному значению единицы. Последовательность оканчивается, когда значение достигает 300. Требуется вывести на экран каждое пятое число.

5. По следующему коду программы опишите поставленную задачу

C goto Statement

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   LOOP:do {

      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }

      printf("value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```

When the above code is compiled and executed, it produces the following result

```
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Лабораторная работа 10. Использование функции при решении задач.

Функция — это самостоятельная единица программы, которая спроектирована для реализации конкретной подзадачи.

Функция является подпрограммой, которая может содержаться в основной программе, а может быть создана отдельно (в библиотеке). Каждая функция выполняет в программе определенные действия.

Сигнатура функции определяет правила использования функции. Обычно сигнатура представляет собой описание функции, включающее имя функции, перечень формальных параметров с их типами и тип возвращаемого значения.

Семантика функции определяет способ реализации функции. Обычно представляет собой тело функции.

Определение функции

Каждая функция в языке Си должна быть определена, то есть должны быть указаны:

тип возвращаемого значения;

имя функции;

информация о формальных аргументах;

тело функции.

Определение функции имеет следующий синтаксис:

```
 ТипВозвращаемогоЗначения ИмяФункции(СписокФормальныхАргументов)
{
  ТелоФункции;
 ...
 return(ВозвращаемоеЗначение);
}
```

Пример: Функция сложения двух вещественных чисел

```
float function(float x, float z)
{
 float y;
 y=x+z;
 return(y);
}
```

В указанном примере возвращаемое значение имеет тип float. В качестве возвращаемого значения в вызывающую функцию передается значение переменной y. Формальными аргументами являются значения переменных x и z.

Если функция не возвращает значения, то тип возвращаемого значения для нее указывается как void. При этом операция return может быть опущена. Если функция не принимает аргументов, в круглых скобках также указывается void.

Различают системные (в составе систем программирования) и собственные функции.

Системные функции хранятся в стандартных библиотеках, и пользователю не нужно вдаваться в подробности их реализации. Достаточно знать лишь их сигнатуру. Примером системных функций, используемых ранее, являются функции printf() и scanf().

Собственные функции — это функции, написанные пользователем для решения конкретной подзадачи.

Разбиение программ на функции дает следующие преимущества:

Функцию можно вызвать из различных мест программы, что позволяет избежать повторения программного кода.

Одну и ту же функцию можно использовать в разных программах.

Функции повышают уровень модульности программы и облегчают ее проектирование.

Использование функций облегчает чтение и понимание программы и ускоряет поиск и исправление ошибок.

С точки зрения вызывающей программы функцию можно представить как некий «черный ящик», у которого есть несколько входов и один выход. С точки зрения вызывающей программы неважно, каким образом производится обработка информации внутри функции. Для корректного использования функции достаточно знать лишь ее сигнатуру.

Вызов функции

Общий вид вызова функции

Переменная = ИмяФункции(СписокФактическихАргументов);

Фактический аргумент — это величина, которая присваивается формальному аргументу при вызове функции. Таким образом, формальный аргумент — это переменная

в вызываемой функции, а фактический аргумент — это конкретное значение, присвоенное этой переменной вызывающей функцией. Фактический аргумент может быть константой, переменной или выражением. Если фактический аргумент представлен в виде выражения, то его значение сначала вычисляется, а затем передается в вызываемую функцию. Если в функцию требуется передать несколько значений, то они записываются через запятую. При этом формальные параметры заменяются значениями фактических параметров в порядке их следования в сигнатуре функции.

Возврат в вызывающую функцию

По окончании выполнения вызываемой функции осуществляется возврат значения в точку ее вызова. Это значение присваивается переменной, тип которой должен соответствовать типу возвращаемого значения функции. Функция может передать в вызывающую программу только одно значение. Для передачи возвращаемого значения в вызывающую функцию используется оператор return в одной из форм:

return(ВозвращаемоеЗначение);

return ВозвращаемоеЗначение;

Действие оператора следующее: значение выражения, заключенного в скобки, вычисляется и передается в вызывающую функцию. Возвращаемое значение может использоваться в вызывающей программе как часть некоторого выражения.

Оператор return также завершает выполнение функции и передает управление следующему оператору в вызывающей функции. Оператор return не обязательно должен находиться в конце тела функции.

Функции могут и не возвращать значения, а просто выполнять некоторые вычисления. В этом случае указывается пустой тип возвращаемого значения void, а оператор return может либо отсутствовать, либо не возвращать никакого значения:

return;

Пример: Посчитать сумму двух чисел.

```
#include <stdio.h>
// Функция вычисления суммы двух чисел
int sum(int x, int y) // в функцию передаются два целых числа
{
  int k = x + y;  // вычисляем сумму чисел и сохраняем в k
  return k;       // возвращаем значение k
}
int main()
{
  int a, r;      // описание двух целых переменных
  printf("a= ");
  scanf("%d", &a); // вводим a
  r = sum(a, 5);    // вызов функции: x=a, y=5
  printf("%d + 5 = %d", a, r); // вывод: a + 5 = r
  getchar(); getchar(); // мы использовали scanf(),
  return 0;  // поэтому getchar() вызываем дважжы
}
```

В языке Си нельзя определять одну функцию внутри другой.

В языке Си нет требования, чтобы семантика функции обязательно предшествовало её вызову. Функции могут определяться как до вызывающей функции, так и после нее. Однако если семантика вызываемой функции описывается ниже ее вызова, необходимо до вызова функции определить прототип этой функции, содержащий:

тип возвращаемого значения;

имя функции;

типы формальных аргументов в порядке их следования.

Прототип необходим для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических аргументов типам формальных аргументов. Имена формальных аргументов в прототипе функции могут отсутствовать.

Если в примере выше тело функции сложения чисел разместить после тела функции main, то код будет выглядеть следующим образом:

```
#include <stdio.h>
int sum(int, int);   // сигнатура
int main()
{
  int a, r;
  printf("a= ");
  scanf("%d", &a);
  r = sum(a, 5);    // вызов функции: x=a, y=5
  printf("%d + 5 = %d", a, r);
  getchar(); getchar();
  return 0;
}
int sum(int x, int y) // семантика
{
  int k;
  k = x + y;
  return(k);
}
```

Задание 1. Решить задачи согласно вашего варианта:

1. Описать функцию Sign(X) целого типа, возвращающую для вещественного числа X следующие значения: –1, если X < 0; 0, если X = 0; 1, если X > 0. С помощью этой функции найти значение выражения Sign(A) + Sign(B) для данных вещественных чисел A и B.

2. Описать функцию RootsCount(A, B, C) целого типа, определяющую количество корней квадратного уравнения A·x 2 + B·x + C = 0 (A, B, C — вещественные параметры, A ≠ 0). С ее помощью найти количество корней для каждого из трех квадратных уравнений с данными коэффициентами. Количество корней определять по значению дискриминанта: D = B2 – 4·A·C.

3. Описать функцию CircleS(R) вещественного типа, находящую площадь круга радиуса R (R — вещественное). С помощью этой функции найти площади трех кругов с данными радиусами. Площадь круга радиуса R вычисляется по формуле S = π·R2 . В качестве значения π использовать 3.14.

4. Описать функцию RingS(R1, R2) вещественного типа, находящую площадь кольца, заключенного между двумя окружностями с общим центром и радиусами R1 и R2 (R1 и R2 — вещественные, R1 > R2). С ее помощью найти площади трех колец, для которых даны внешние и внутренние радиусы. Воспользоваться формулой площади круга радиуса R: S = π·R2 . В качестве значения π использовать 3.14.

5. Описать функцию TriangleP(a, h), находящую периметр равнобедренного треугольника по его основанию a и высоте h, проведенной к основанию (a и h — вещественные). С помощью этой функции найти периметры трех треугольников, для которых даны основания и высоты. Для нахождения боковой стороны b треугольника использовать теорему Пифагора: b2 = (a/2)2 + h2 . 37

6. Описать функцию SumRange(A, B) целого типа, находящую сумму всех целых чисел от A до B включительно (A и B — целые). Если A > B, то функция возвращает 0. С помощью этой функции найти суммы чисел от A до B и от B до C, если даны числа A, B, C.

7. Описать функцию Calc(A, B, Op) вещественного типа, выполняющую над ненулевыми вещественными числами A и B одну из арифметических операций и возвращающую ее результат. Вид операции определяется целым параметром Op: 1 — вычитание, 2 — умножение, 3 — деление, остальные значения — сложение. С помощью Calc выполнить для данных A и B операции, определяемые данными целыми N1, N2, N3.

8. Описать функцию Quarter(x, y) целого типа, определяющую номер координатной четверти, в которой находится точка с ненулевыми вещественными координатами (x, y). С помощью этой функции найти номера координатных четвертей для трех точек с данными ненулевыми координатами.

9. Описать функцию Even(K) логического типа, возвращающую True, если целый параметр K является четным, и False в противном случае. С ее помощью найти количество четных чисел в наборе из 10 целых чисел.

10. Описать функцию IsSquare(K) логического типа, возвращающую True, если целый параметр K (> 0) является квадратом некоторого целого числа, и False в противном случае. С ее помощью найти количество квадратов в наборе из 10 целых положительных чисел.

11. Описать функцию IsPower5(K) логического типа, возвращающую True, если целый параметр K (> 0) является степенью числа 5, и False в противном случае. С ее помощью найти количество степеней числа 5 в наборе из 10 целых положительных чисел.

12. Описать функцию IsPowerN(K, N) логического типа, возвращающую True, если целый параметр K (> 0) является степенью числа N (> 1), и False в противном случае. Дано число N (> 1) и набор из 10 целых положительных чисел. С помощью функции IsPowerN найти количество степеней числа N в данном наборе.

13. Описать функцию IsPrime(N) логического типа, возвращающую True, если целый параметр N (> 1) является простым числом, и False в противном случае (число, большее 1, называется простым, если оно не имеет положительных делителей, кроме 1 и самого себя). Дан набор из 10 целых чисел, больших 1. С помощью функции IsPrime найти количество простых чисел в данном наборе.

14. Описать функцию DigitCount(K) целого типа, находящую количество цифр целого положительного числа K. Используя эту функцию, найти количество цифр для каждого из пяти данных целых положительных чисел.

15. Описать функцию DigitN(K, N) целого типа, возвращающую N-ю цифру целого положительного числа K (цифры в числе нумеруются справа налево). Если количество цифр в числе K меньше N, то функция возвращает –1. Для каждого из пяти данных целых положительных чисел K1, K2,

Задание 2. Решить задачи согласно вашего варианта:

1. Описать функцию PowerA3(A, B), вычисляющую третью степень числа A и возвращающую ее в переменной B (A — входной, B — выходной параметр; оба параметра являются вещественными). С помощью этой функции найти третьи степени пяти данных чисел.

2. Описать функцию PowerA234(A, B, C, D), вычисляющую вторую, третью и четвертую степень числа A и возвращающую эти степени соответственно в переменных B, C и D (A — входной, B, C, D — выходные параметры; все параметры являются вещественными). С помощью этой функции найти вторую, третью и четвертую степень пяти данных чисел.

3. Описать функцию Mean(X, Y, AMean, GMean), вычисляющую среднее арифметическое AMean = (X + Y)/2 и среднее геометрическое GMean = = X ·Y двух

положительных чисел X и Y (X и Y — входные, AMean и GMean — выходные параметры вещественного типа). С помощью этой функции найти среднее арифметическое и среднее геометрическое для пар (A, B), (A, C), (A, D), если даны A, B, C, D.

4. Описать функцию TrianglePS(a, P, S), вычисляющую по стороне a равностороннего треугольника его периметр P = 3·a и площадь S = 2 a 3 / 4 (a — входной, P и S — выходные параметры; все параметры являются вещественными). С помощью этой функции найти периметры и площади трех равносторонних треугольников с данными сторонами.

5. Описать функцию RectPS(x1, y1, x2, y2, P, S), вычисляющую периметр P и площадь S прямоугольника со сторонами, параллельными осям координат, по координатам (x1, y1), (x2, y2) его противоположных вершин (x1, y1, x2, y2 — входные, P и S — выходные параметры вещественного типа). С помощью этой функции найти периметры и площади трех прямоугольников с данными противоположными вершинами.

6. Описать функцию DigitCountSum(K, C, S), находящую количество C цифр целого положительного числа K, а также их сумму S (K — входной, C и S — выходные параметры целого типа). С помощью этой функции найти количество и сумму цифр для каждого из пяти данных целых чисел. 35

7. Описать функцию InvertDigits(K), меняющую порядок следования цифр целого положительного числа K на обратный (K — параметр целого типа, являющийся одновременно входным и выходным). С помощью этой функции поменять порядок следования цифр на обратный для каждого из пяти данных целых чисел.

8. Описать функцию AddRightDigit(D, K), добавляющую к целому положительному числу K справа цифру D (D — входной параметр целого типа, лежащий в диапазоне 0–9, K — параметр целого типа, являющийся одновременно входным и выходным). С помощью этой функции последовательно добавить к данному числу K справа данные цифры D1 и D2, выводя результат каждого добавления.

9. Описать функцию AddLeftDigit(D, K), добавляющую к целому положительному числу K слева цифру D (D — входной параметр целого типа, лежащий в диапазоне 1–9, K — параметр целого типа, являющийся одновременно входным и выходным). С помощью этой функции последовательно добавить к данному числу K слева данные цифры D1 и D2, выводя результат каждого добавления.

10. Описать функцию Swap(X, Y), меняющую содержимое переменных X и Y (X и Y — вещественные параметры, являющиеся одновременно входными и выходными). С ее помощью для данных переменных A, B, C, D последовательно поменять содержимое следующих пар: A и B, C и D, B и C и вывести новые значения A, B, C, D.

11. Описать функцию Minmax(X, Y), записывающую в переменную X минимальное из значений X и Y, а в переменную Y — максимальное из этих значений (X и Y — вещественные параметры, являющиеся одновременно входными и выходными). Используя четыре вызова этой функции, найти минимальное и максимальное из данных чисел A, B, C, D.

12. Описать функцию SortInc3(A, B, C), меняющую содержимое переменных A, B, C таким образом, чтобы их значения оказались упорядоченными по возрастанию (A, B, C — вещественные параметры, являющиеся одновременно входными и выходными). С помощью этой функции упорядочить по возрастанию два данных набора из трех чисел: (A1, B1, C1) и (A2, B2, C2).

13. Описать функцию SortDec3(A, B, C), меняющую содержимое переменных A, B, C таким образом, чтобы их значения оказались упорядоченными по убыванию (A, B, C — вещественные параметры, являющиеся одновременно входными и выходными). С помощью этой функции упорядочить по убыванию два данных набора из трех чисел: (A1, B1, C1) и (A2, B2, C2).

14. Описать функцию ShiftRight3(A, B, C), выполняющую правый циклический сдвиг: значение A переходит в B, значение B — в C, значение C — в A (A, B, C — вещественные параметры, являющиеся одновременно входными и выходными). С помощью этой функции выполнить правый циклический сдвиг для двух данных наборов из трех чисел: (A1, B1, C1) и (A2, B2, C2).

15. Описать функцию ShiftLeft3(A, B, C), выполняющую левый циклический сдвиг: значение A переходит в C, значение C — в B, значение B — в A (A, B, C — вещественные параметры, являющиеся одновременно входными и выходными). С помощью этой функции выполнить левый циклический сдвиг для двух данных наборов из трех чисел: (A1, B1, C1) и (A2, B2, C2).

Лабораторная работа №11
Создание библиотеки подпрограмм.

Цель работы: приобретение навыков создания библиотеки подпрограмм.

Методические указания

Часто в программе требуется повторить определенную последовательность операторов в разных частях программы. Для того, чтобы описывать эту последовательность один раз, а применять многократно, в языках программирования применяются подпрограммы. Подпрограмма - автономная часть программы, выполняющая определенный алгоритм и допускающая обращение к ней из различных частей общей программы.

В языке Си существует один вид подпрограмм - функции. Каждая программа в своем составе должна иметь главную функцию main(), служащую точкой входа в программу. Кроме функции main(), в программу может входить произвольное число функций, выполнение которых инициализируется либо прямо, либо вызовами из функции main(). Каждая функция по отношению к другой является внешней. Для того, чтобы функция была доступной, необходимо, чтобы до ее вызова о ней было известно компилятору. Форма записи функции:

<тип результата > <имя функции>(<формальные параметры>){<тело функции >}

Если тип возвращаемого функцией значения не указан, то подразумевается int. Если с именем функции не связан результат, то нужно указать тип функции void. Параметры, записываемые в обращении к функции, называются фактическими; параметры, указанные в описании функции - формальными. Фактические параметры должны соответствовать формальным по количеству, порядку следования и типу. Объекты, объявленные вне функции, действуют в любой функции и называются глобальными. Объекты, объявленные в функции, действуют только в ней и называются локальными. В теле функции обычно присутствует оператор return <выражение>, определяющий возвращаемое функцией значение.

Все параметры функции, кроме массивов, передаются по значению, т.е. внутри функции создаются локальные копии параметров. Если необходимо передать саму переменную, а не её копию, то в функцию передаётся адрес этой переменной. Таким образом, через параметры можно передавать результат выполнения функции. То есть, параметры, с помощью которых результаты должны передаваться из функции в точку вызова, описываются как указатели. Вызов функции может быть оформлен в виде оператора, если с именем функции не связано возвращаемое значение, или в виде выражения, если возвращаемое значение связано с именем функции.

Прототип функции может указываться до вызова функции вместо описания функции для того, чтобы компилятор мог выполнить проверку соответствия типов аргументов и параметров. Прототип функции по форме такой же, как и заголовок функции. В конце него ставится «;».

Функции можно подключать с помощью директивы #include <имя файла>. Такие файлы с функциями удобно использовать в диалоговых программах с пользовательским меню, позволяющих выбрать один из режимов.

Пример 1: Функция с параметрами-значениями. Результат связан с именем функции. В программе объявляется прототип функции, а сама функция описывается ниже.

```
#include <stdio.h>
int max(int,int); //Прототип функции
void main()
{ int x,y,z;
printf(" input x,y ");
scanf("%d %d",&x,&y);
z=max(x,y); //Вызов функции с фактическими параметрами
printf("x=%d y=%d max=%d",x,y,z);
getch();
}
int max(int a ,int b) //Заголовок функции с формальными параметрами
{ int c;
if (a>b) c=a;
else c=b;
return c;
```

Задание. По предложенному номеру составить программу. В программе должны быть использованы две функции P(x) и S(t).

1. $y = \begin{cases} 2P(a) + S(b), a < 0 \\ S(a) - P(b), a \ge 0 \end{cases}$, где $P(x) = \begin{cases} 2x, x < 3 \\ x, x \ge 3 \end{cases}$, a S(t) = sin(2t+2)

2. $y = \sum_{n=1}^{10} (P(n) + S(n))$, где $P(x) = \begin{cases} x^2, < 5 \\ 2x, x \ge 5 \end{cases}$, a S(t) = cos(t3)

3. $y = \begin{cases} P(a) + S(b), b < 0 \\ S(a) + P(b), b \ge 0 \end{cases}$, где $P(x) = \begin{cases} x^3, x > 5 \\ \sin(x), x \le 5 \end{cases}$, a S(t) = 2t + 12

4. $y = \sum_{n=1}^{25} (P(n) - S(n))$, где $P(x) = \begin{cases} \ln(x^2), x < 10 \\ \log(x), x \ge 10 \end{cases}$, a S(t) = arctg(t-2)

5. $y = \begin{cases} 2P(a) + S(b), a < 10 \\ S(a) - 3P(b), a \ge 10 \end{cases}$, где $P(x) = \sum_{n=1}^{x} \frac{1}{n}$, a S(t) = sin(t)-cos(t)

6. $y = \sum_{n=1}^{5} \frac{R(n)}{S(n)}$, где $P(x) = \begin{cases} 2x+3, x<2 \\ x^3, x \geq 2 \end{cases}$, а $S(t) = \frac{2t^2+12}{t^2+1}$

7. $y = \begin{cases} P(a)-S(b), a<b \\ S(a)-S(b), b \geq a \end{cases}$, где $P(x) = \sum_{n=1}^{x} \frac{1}{n}$, а $S(t) = \frac{12\sin(t-2)}{t^3}$

8. $y = \sum_{n=1}^{12} (P(n)+S(n))$, где $P(x) = \begin{cases} 2x, x<6 \\ x^2, x \geq 6 \end{cases}$, а $S(t) = 12t + \sin(t)$

9. $y = \begin{cases} 2P(a)+S(b), a<0 \\ S(a)-P(b), a \geq 0 \end{cases}$, где $P(x) = \begin{cases} 2x, x<3 \\ x, x \geq 3 \end{cases}$, а $S(t) = \cos(3t+2)$

10. $y = \sum_{n=1}^{10} (P(n)+S(n))$, где $P(x) = \begin{cases} x^2, <5 \\ 2x, x \geq 5 \end{cases}$, а $S(t) = \sin(t)+10$

11. $y = \begin{cases} P(a)+S(b), b<0 \\ S(a)+P(b), b \geq 0 \end{cases}$, где $P(x) = \begin{cases} x^3, x>5 \\ \sin(x), x \leq 5 \end{cases}$, а $S(t) = 10t + 2$

12. $y = \sum_{n=1}^{25} (P(n)-S(n))$, где $P(x) = \begin{cases} \ln(x^2), x<10 \\ \log(x), x \geq 10 \end{cases}$, а $S(t) = \sin(t-2)$

13. $y = \begin{cases} 2P(a)+S(b), a<10 \\ S(a)-3P(b), a \geq 10 \end{cases}$, где $P(x) = \sum_{n=1}^{x} \frac{1}{n}$, а $S(t) = \cos(t)-\sin(t)$

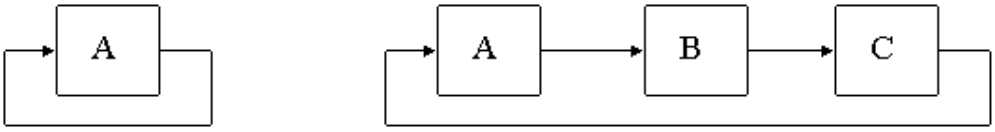
14. $y = \sum_{n=1}^{5} \frac{R(n)}{S(n)}$, где $P(x) = \begin{cases} 2x+3, x<2 \\ x^3, x \geq 2 \end{cases}$, а $S(t) = \frac{2 \quad +12t}{t^2+2}$

15. $y = \begin{cases} P(b)-S(a), a<b \\ S(a)-S(b), b \geq a \end{cases}$, где $P(x) = \sum_{n=1}^{x} \frac{1}{n}$, а $S(t) = \frac{12\cos(t-2)}{t^2}$

Лабораторная работа №12
Использование рекурсивных функции

Цели: формирования представления о рекурсивном объекте и рекурсивном алгоритме, развития логического мышления, самостоятельно составлять рекурсивные алгоритмы

Рекурсия — вызов функции из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция A вызывает функцию B, а функция B — функцию A.



Программа разрабатывается сведением исходной задачи к более простым. Среди этих задач может оказаться и первоначальная, но в упрощенной форме.

Например, для вычисления F(N) может понадобиться вычислить F(N−1). Иными словами, частью алгоритма вычисления функции будет вычисление этой же функции.

Итак, функция является **рекурсивной**, если она обращается сама к себе прямо или косвенно (через другие функции). Заметим, что при косвенном обращении все функции в цепочке – рекурсивные.

Рассмотрим это на примере функции вычисления факториала. Хорошо известно, что , . А как вычислить величину для бОльших ? Если бы мы могли вычислить величину , то тогда мы легко вычислили бы , поскольку . Но как вычислить ? Если бы мы вычислили , то мы сможем вычисли и . А как вычислить ? Если бы… В конце концов, мы дойдем до величины , которая равна . Таким образом, для вычисления факториала мы можем использовать значение факториала для меньшего числа. Это можно сделать и в программе на Си:

```
int factorial(int n)
{
   if (n == 0)
      return 1;
   else
      return factorial(n - 1) * n;
}
```

Краткая формулировка обоснования рекурсивного алгоритма вычисления факториала:

N!=(N−1)! ∗ N,

если N = 0, то N! = 1

Как это работает?

Допустим, мы вызвали функцию factorial(4). Будет вызвана функция, у которой значение параметра n равно 4. Она проверит условие n == 0, поскольку условие ложно, то будет выполнена инструкция return n * factorial(n - 1). Но чтобы вычислить это значение, будет вызвана функция factorial(3), так как параметр n имеет значение, равное 4. Теперь в памяти будет находиться две функции factorial — одна со значением параметра n равным 4, а другая — со значением 3. При этом активна будет последняя функция.

Эта функция в свою очередь вызовет функцию factorial(2), та вызовет функцию factorial(1), затем factorial(0). В случае этой функции ничего более вызвано не будет, функция просто вернет значение 1, и управление вернется в функцию factorial(1). Та умножит значение n = 1 на значение 1, которое вернула функция factorial(0), и вернет полученное произведение, равное 1. Управление вернется в функцию factorial(2), которая

умножит n = 2 на значение 1, которое вернула функция factorial(1) и вернет полученное произведение, равное 2. Функция factorial(3) вернет , а функция factorial(4) вернет .

Таблица последовательности вызовов функции приведена ниже. Значения функции возвращают в порядке, обратном порядку их вызова, то есть сначала заканчивает работу функция factorial(0), затем factorial(1) и т. д.

С какими параметрами вызвана функция	**Какое значение вернула**
factorial(4)	4 * 6 = 24
factorial(3)	3 * 2 = 6
factorial(2)	2 * 1 = 2
factorial(1)	1 * 1 = 1
factorial(0)	1

При отладке программы всю последовательность вложенных вызовов рекуррентных функций можно изучить в окне «Call Stack» («стек вызовов») в режиме отладки.

Для каждой вызываемой функции значения локальных переменных будут своими.

Для того, чтобы реализовать рекурсию нужно ответить на следующие вопросы:

1. Какой случай (для какого набора параметров) будет крайним (простым) и что функция возвращает в этом случае?
2. Как свести задачу для какого-то набора параметров (за исключением крайнего случая) к задаче, для другого набора параметров (как правило, с меньшими значениями)?

При этом программирование рекурсии выглядит так. Функция должна сначала проверить, не является ли переданный набор параметров простым (крайним) случаем. В этом случае функция должна вернуть значение (или выполнить действия), соответствующие простому случаю. Иначе функция должна вызвать себя рекурсивно для другого набора параметров, и на основе полученных значений вычислить значение, которое она должна вернуть.

Количество вложенных вызовов функции или процедуры называется глубиной рекурсии.

Рекурсия изнутри

Это может показаться удивительным, но самовызов функции/процедуры ничем не отличается от вызова другой функции/процедуры. Что происходит, если одна функция вызывает другую? В общих чертах следующее:

- в памяти размещаются параметры, передаваемые функции (но не параметры-переменные, т. к. они передаются по ссылке!);
- для внутренних переменных вызываемой функции также отводится новая область памяти (несмотря на совпадение их имен и типов с переменными вызывающей функции);
- запоминается адрес возврата в вызывающую функцию;
- управление передается вызванной функции.

Если функцию или процедуру вызвать повторно из другой функции/процедуры или из нее самой, будет выполняться тот же код, но работать он будет с другими значениями параметров и внутренних переменных. Это и дает возможность рекурсии.

Пример

Числа Фибоначчи – это ряд чисел, в котором каждое последующее число равно сумме двух предыдущих:

1, 1, 2, 3, 5, 8, 13 и т. д.

То есть последовательность всегда начинается с двух единиц. А каждое следующее число является определяется по формуле:

$$F_n = F_{n-1} + F_{n-2}$$

Для определения чисел Фибоначчи часто используется рекурсивный алгоритм:

- Если n = 1 или n = 2, вернуть 1 (поскольку первый и второй элементы ряда Фибоначчи равны 1).
- Вызвать рекурсивно функцию с аргументами n-1 и n-2.
- Результат двух вызовов сложить и вернуть полученное значение.

Реализация с использованием рекурсии

```
#include <stdio.h>
int fibonacci(int N)  // рекурсивная функция
{
 if (N == 1 || N == 2)
   return 1; // первые 2 числа равны 1
 return fibonacci(N - 1) + fibonacci(N - 2); // складываем предыдущие 2 числа
}
int main()
{
 int N;
 printf("N=");
 scanf("%d", &N); // вводим число N
 for (int i = 1; i <= N; i++) // В цикле выводим N чисел Фибоначчи
   printf("%d ", fibonacci(i));
 getchar(); getchar();
 return 0;
}
```

Индивидуальные задания

1. Дано натуральное число. Составить рекурсивную процедуру вывода на экран цифр заданного натурального числа, определить количество и сумму его цифр.
2. Определить цифровой корень заданного натурального числа. Например, для числа 123456 цифровой корень равен 3, т.к. 1+2+3+4+5+6=21, 2+1=3.
3. Описать рекурсивную процедуру печати Вашего имени. Предусмотреть условие выхода из рекурсии.
4. Программа принимает на вход число и рекурсивно определяет, четное это число или нечетное.
5. Дано натуральное число n. Выведите все числа от 1 до n.
6. Дано натуральное число N. Выведите слово YES, если число N является точной степенью двойки, или слово NO в противном случае. Операцией возведения в степень пользоваться нельзя!
7. Дано натуральное число N. Вычислите сумму его цифр. При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется).
8. Дано натуральное число n>1. Проверьте, является ли оно простым. Программа должна вывести слово YES, если число простое и NO, если число составное. Алгоритм должен иметь сложность O(logn).

Указание. Понятно, что задача сама по себе нерекурсивна, т.к. проверка числа n на простоту никак не сводится к проверке на простоту меньших чисел. Поэтому нужно сделать еще один параметр рекурсии: делитель числа, и именно по этому параметру и делать рекурсию.

9. Дано натуральное число n>1. Выведите все простые множители этого числа в порядке неубывания с учетом кратности.

10. Программа принимает на вход два числа и находит наименьшее общее кратное (НОК) при помощи рекурсии.

11. Программа принимает на вход два числа и находит наибольший общий делитель (НОД) с использованием рекурсии.

12. Программа принимает на вход два числа и при помощи рекурсии находит их произведение.

13. Программа принимает на вход число и его степень, и при помощи рекурсии вычисляет результат.

14. Написать функцию, вычисляющую НОД(a, b) для неотрицательных целых a и b (без циклов).

15. Дано натуральное число N и последовательность из N элементов. Требуется вывести эту последовательность в обратном порядке.

Лабораторная работа №13
Решение задач с использованием одномерных массивов

В отличие от обычных переменных, которые хранят одно значение, массивы используются для хранения целого набора однотипных значений. Массив – набор однотипных элементов, последовательно располагающихся в памяти компьютера. Для определения массива необходимо указать тип элементов, которые в него входят, и максимальное количество элементов, которое может быть помещено в массив. Например:

```
#include <stdio.h> #include <stdlib.h> #include <time.h>
#define N 10/* количество элементов массива */ int main( )
{
int a[N],/* массив a размера N */ i;/* счетчик */
srand(time(NULL)); for (i = 0; i < N; i++)
{
a[i] = rand()%100; printf("%d\n", a[i]);
}
return 0;
}
```

В нашем случае определяется массив a из 10 целых элементов (то есть блок из 10 последовательно расположенных элементов типа int), который заполняется случайными числами в диапазоне от 0 до 99. Заметим, что нумерация элементов массива начинается с 0. Поэтому к элементам массива a необходимо обращаться таким образом: a[0], a[1], …, a[N-1].

В нашем примере фигурирует функция srand() из библиотеки

<stdlib.h>, которая устанавливает исходное число для последовательности, которая генерируется функцией rand():

void srand(unsigned long x)

Часто данная функция применяется для того, чтобы при различных запусках программы функция rand() генерировала различные последовательности псевдослучайных чисел. Например, в приведенной программе в качестве параметра функция srand() получает

системное время.

Номер позиции элемента, содержащийся в квадратных скобках, называют индексом элемента. Индекс должен быть целым числом или целочисленным выражением.

После того как массив определен, его можно инициализировать. Инициализация массива означает, что при его объявлении элементам данного массива будут присвоены начальные значения. Например,

int a[5] = {10, 20, 30, 40, 50};

После такой инициализации мы будем иметь целочисленный массив из пяти элементов со следующими значениями:

a[0]=10, a[1]=20, a[2]=30, a[3]=40, a[4]=50.

Если при инициализации размер массива не указан, то размер определяется по числу начальных значений. Например, массив из 5 целых чисел, приведенный выше, можно сконструировать следующим образом:

int a[] = {10, 20, 30, 40, 50};

Если инициализируемых значений массива меньше, чем всех элементов массива, тогда остальные элементы автоматически инициализируются нулями. Например, если инициализировать массив следующим образом:

int a[10] = {0};

то все элементы данного массива будут иметь нулевое значение.

Заметим, что массивы не инициализируются нулями автоматически. Для этой цели следует инициализировать хотя бы один элемент.

Чтобы узнать объем памяти, занимаемый тем или иным массивом, можно к имени данного массива применить оператор sizeof(), например:

```
#include <stdio.h> int main( )
{
int a[10];
printf("%d\n", sizeof(a)); return 0;
}
```

В этом случае также можно узнать и размерность (количество элементов) массива с помощью выражения: sizeof(a)/sizeof(*a).

Примеры работы с массивами

Пример 1. Проверить, образуют ли элементы целочисленного массива арифметическую прогрессию.

```
#include <stdio.h>
#include <stdlib.h>
#define N 10
/* модификатор const указывает на то, что значения элементов */
/* массива a и значение n менять нельзя*/
int Progression(const int *a, const int n)
{
int d, i;
d = a[1] - a[0];
i = 1;
while ((++i < n) && (a[i] - a[i - 1] == d))
return (i >= n);
}
int main( )
{
```

```
int i, a[N];
for (i = 0; i < N; i++)
{a[i] = rand()%10;
printf(" %d",a[i]); }
printf("%s\n", Progression(a, N) ? "progression" :"no progression");
return 0;
}
```

Пример 2. Определить, все ли элементы массива a попарно различны.

Произвольный случай. В нижеприведенной программе переменная flag играет роль логической переменной. Она инициализируется с начальным значением 1 (истина), то есть будем считать, что по умолчанию все элементы массива попарно различны. Сначала будем сравнивать все элементы с индексами 1, 2, …, n-1 с элементом a[0], затем все элементы с индексами 2, 3, …, n-1 с элементом a[1] и т.д. Как только находится пара одинаковых элементов, процесс останав- ливается и функция Check() возвращает нулевое (ложное) значение. В противном случае функция возвращает значение 1 (истинна).

```
int Check(const double *a, const int n)
{
int i, j, flag = 1;
for(i = 1; i < n && flag; i++)
{
for(j = 0; j < i && a[i] != a[j]; j++)
;
flag = (i == j);
}
return flag;
}
```

Массив с достаточно узким диапазоном значений элементов. Как и в предыдущем примере, рассмотрим случай, когда множество значений элементов массива составляет довольно узкий диапазон. Пусть, например, значения элементов массива принадлежат множеству {0,1,…,255}. Опять же в этом случае можно очень быстро под-считать количество различных элементов, не требуя при этом условия упорядоченности. Для этого определим массив count, содержащий 256 элементов, индексы которого будут соответствовать значениям элементов массива a. На начальном этапе инициализируем элементы массива count нулями. Затем при считывании i-го элемента массива a будем увеличивать значение cout[a[i]] на единицу. После прохождения по всему массиву a количество различных элементов массива a будет равно количеству ненулевых элементов массива count.

```
int Count(const int *a, const int n)
{
int k, i, count[256] = {0}; for (i = 0; i < n; i++)
count[a[i]]++;
for (i = 0; i < 256; i++) if (count[i] != 0)
k++;
return k;
}
```

Пример 4. Пусть имеется некоторый массив действительных чисел a и некоторое

число x. Требуется переставить элементы массива так, чтобы сначала следовали (в произвольном порядке) элементы, меньшие числа x, а затем элементы, не меньшие числа x. Использовать не более одного прохода по массиву.

Будем просматривать элементы массива слева направо, пока не встретим элемент a[i]>=x. Затем будем просматривать элементы массива справа налево, пока не встретим элемент a[j]<x. После этого меняем элементы a[i] и a[j] местами и продолжаем данныйпроцесс.

```
void Exchange(double *a, const int n, const double x)
{
int i = 0, j = n – 1; double buf;
while (i < j)
{
while (i < j && a[i] < x) i++;
while (i < j && a[j] >= x) j--;
if (i < j)
{
/* меняем местами a[i] и a[j] */ buf = a[i]; a[i] = a[j]; a[j] = buf; i++; j--;
}
}
}
```

Например, если требуется переставить элементы массива так, чтобы сначала следовали отрицательные элементы, а затем неотрицательные,то задачаоченьпросторешаетсяспомощьюфункции Exchange(), где в качестве второго параметра нужно передать число 0. Немного усложним задачу. Пусть требуется переставить эле- менты массива так, чтобы сначала следовали (в произвольном порядке) элементы, меньшие числа x, затем элементы, равные x, а затем элементы, большие числа x. При этом по массиву придется пройтись

не более двух раз и алгоритм будет выглядеть следующим образом:

```
void Exchange(double *a, const int n, const double x)
{
int i = 0, j = n – 1; double buf;
while (i < j)
{
while (i < j && a[i] < x)
i++;
while (i < j && a[j] >= x) j--;
if (i < j)
{
buf = a[i]; a[i] = a[j]; a[j] = buf; i++; j--;
}
}
j = n – 1; while (i < j)
{
while (i < j && a[i] == x) i++;
while (i < j && a[j] != x) j--;
if (i < j)
{
a[j] = a[i]; a[i] = x; i++; j--;
}
```

```
}
}
```

Пример 5. Пусть имеется n-элементный целочисленный массив a, который является перестановкой чисел 0, 1,…, n-1. Требуется в n- элементный целочисленный массив b записать обратную перестановку к перестановке a.

```
/* генерирование обратной перестановки b */
void ObratnayaPerest(unsigned int *a, unsigned int *b, int n)
{
int i;
for (i = 0; i < n; i++) b[a[i]] = i;
}
```

Пример 6. Требуется переставить цифры в натуральном числе x так, чтобы полученное число имело максимальное значение.

```
unsigned long Max(unsigned long x)
{
int count[10] = {0}, i, j; do {
count[x%10]++; x /= 10;
}while(x);
for (i = 9; i >= 0; i--)
for (j = 0; j < count[i]; j++) x = x*10 + i;
return x;
}
```

Пример 7. Пусть B – некоторое фиксированное целое число, причем $0 \le B \le 1000$, и M – некоторое подмножество в {0,1,2,…,B-1}. Пусть также имеется некоторый целочисленный массив a размера n. Требуется подсчитать количество элементов массива a, принадлежащих множеству M.

Элементы множества M запишем в массив set размера m, где m – мощность множества M. Пусть, для примера, B=1000, M={1,10,100,500}. Поэтому в данном случае int set[4] =

{1,10,100,500}.

Для эффективной проверки на принадлежность элементов массива a множеству M рассмотрим следующий алгоритм. Пусть flag[B]

– массив, состоящий из нулей и единиц, каждый индекс которого будет соответствовать элементам множества M: если элемент i, $0 \le i < B$, принадлежит множеству M, то определим flag[i]=1, иначе, flag[i]=0. Теперь проверить на принадлежность произвольного элемента a[i] множеству M можно очень просто: if (a[i] >= 0 && a[i] < B && flag[a[i]]).

```
#define B 1000
#define N 100/* размерность массива a */

/* вычисление количества элементов массива a, принадлежащих M: */
int Count (int *a, int n, int *set, int m)
{
int flag[B] = {0}; int i, count;
for (i = 0; i < m; i++) flag[set[i]] = 1;
```

```
count = 0;
for (i = 0; i < n; i++)
if (a[i] >= 0 && a[i] < B && flag[a[i]]) count++;
return count;
}

int main()
{
int set[4] = {1, 10, 100, 500};
int a[N];

…/* заполняем массив a */

printf("%d\n", Count(a, N, set, 4)); return 0;
}
```

Пример 8. Обобщим предыдущий пример. Пусть A и B – некоторые целые числа, причем -1000≤A≤B≤1000, и M – некоторое под- множество в {A,A+1, A+2,…,B-1}. Пусть также имеется некоторый целочисленный массив a размера n. Требуется подсчитать количество элементов массива a, принадлежащих множеству M. После модернизации алгоритма из предыдущего примера эффективный алгоритм решения данной задачи будет выглядеть следующим образом:

```
#define A -1000
#define B  1000
#define N  10

int Count (int *a, int n, int *set, int m)
{
int flag[B – A] = {0}; int i, count;
for (i = 0; i < m; i++)
flag[set[i] – A] = 1;

count = 0;
for (i = 0; i < n; i++)
if (a[i] >= A && a[i] < B && flag[a[i] – A]) count++;
return count;
}
```

Задачи

1. Первый положительный элемент массива
2. Заменить элементы массива на противоположные
3. Поменять местами минимальный и максимальный элементы массива
4. Найти разность между максимальным и минимальным элементами массива
5. Удалить из массива четные элементы
6. Найти сумму четных отрицательных элементов массива
7. Минимальный из элементов массива с нечетными индексами
8. Вывести элементы массива, которые больше среднего арифметического

9. Сумма положительных элементов массива

10. Найти количество положительных элементов массива

11. Расстояние между точками в n-мерном пространстве

12. Сформировать массив B из положительных элементов массива A, имеющих четный индекс

13. Найти среднее арифметическое отрицательных элементов массива. Заменить на него минимальный элемент.

14. Определить индексы элементов массива, значение которых лежит в указанном пределе

15. Среднее арифметическое всех чётных элементов массива, стоящих на нечётных местах

Лабораторная работа 14.
Решение задач с использованием двумерных массивов
Определение и инициализация двумерных массивов

В языке Си имеется возможность создавать n-мерные массивы. Стандарт ANSI предусматривает, что значение n может быть по крайней мере любым числом в пределах от 1 до 12.

В данном разделе будут рассматриваться двумерные массивы.

Такие массивы описываются следующим образом: int a[10][20];

То есть тем самым определен двумерный массив целых чисел a, состоящий из 10 строк и 20 столбцов. Каждый элемент двумерного массива однозначно определяется парой индексов i и j: a[i][j], где i – номер строки, a j – номер столбца. Как и в случае одномерных массивов, нумерация по переменным i и j начинается с нуля: a[0][0] – эле- мент, находящийся на пересечении нулевой строки и нулевого столбца… a[9][19] – элемент, находящийся на пересечении девятой строки и девятнадцатого столбца.

Двумерный массив можно инициализировать при его объявлении:

int a[3][2] = {{1, 2}, {3, 4}, {5, 6}};

При этом значения в фигурных скобках группируются по строкам. Если дополнительные фигурные скобки не присутствуют, то числа распределяются по строкам: сначала заполняется нулевая строка, затем первая и т.д. Например, массив a можно инициализировать итак:

int a[3][2] = {1, 2, 3, 4, 5, 6};

Если для какой-либо строки указано недостаточно значений, то оставшиеся значения данной строки инициализируются нулями. Например, если матрицу a инициализировать таким образом:

int a[3][2] = {{1}, {3}, {5}};

тогда второй столбец данной матрицы будет содержать нулевые значения:

Если необходимо все элементы матрицы инициализировать нулями, то для этого инициализацию достаточно просто записать следующим образом:

int a[10][20] = {0};

Примеры с двумерными массивами

Пример 1. Инициализация и вывод двумерного массива на экран.

```
#include <stdio.h>
```

```
#include<stdlib.h>
#include<time.h>
#define M 5 /* Число строк матрицы */
#define N 10 /* Число столбцов матрицы */
/* заполнение матрицы случайными числами от 0 до 99 */
void Init(int a[][N], int m, int n)
{
int i, j;
for (i = 0; i < m; i++) for (j = 0; j < n; j++)
a[i][j] = rand()%100;
}

/* вывод матрицы на экран */
void Print(int a[][N], int m, int n)
{
int i, j;
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++) printf("%5d", a[i][j]);
printf("\n");
}
}

int main()
{
int a[M][N];/* матрица размера M на N */ srand(time(NULL));
Init(a, M, N);/* инициализация элементов матрицы */
Print(a, M, N); /* вывод матрицы на экран */
return 0;
}
```

При этом заметим очень важный момент. Массив int a[M][N] является одномерным массивом размера M, каждый элемент которого является целочисленным массивом размерности N, то есть каждый из M элементов массива a имеет тип int a[N]. В соответствии с данной логикой элемент a[0] является одномерным массивом, представляющим первую строку матрицы a. Поэтому a[i] – одномерный массив, представляющий i-ую строку матрицы a.

Двумерное представление — это всего лишь удобный способ работы с массивом, используя два индекса – номер строки и номер столбца. В памяти компьютера такой двумерный массив хранится последовательно в виде одномерного массива: сначала следует первая строка матрицы, затем вторая и т.д.

Таким образом, с каждой строкой матрицы a можно работать как с обычным одномерным массивом, что иллюстрируется в следующей программе.

```
#include<stdio.h> #include<stdlib.h>
#define M 5/* Число строк матрицы */ #define N 10/* Число столбцов матрицы */

/* заполнение одномерного массива a случайными числами */ void Init(int *a, int n)
{
int i;
for (i = 0; i < n; i++) a[i] = rand()%100;
}
```

```
/* вывод одномерного массива a на экран */ void Print(int *a, int n)
int i;
for(i = 0; i < n; i++) printf("%5d", a[i]);
putchar('\n');
}

int main()
{
int i, a[M][N]; /* матрица размера M на N */
/* инициализация каждой строки матрицы: */ for (i = 0; i < M; i++)
Init(a[i], N);
/* вывод элементов матрицы построчно: */ for (i = 0; i < M; i++)
Print(a[i], N); return 0;
}
```

Пример 2. Пусть имеется двумерный массив int a[M][N] и одномерный массив int b[MN]. Требуется элементы массива b построчно записать в матрицу a.

Для решения данной задачи учтем важное замечание из предыдущего примера. Если представить двумерный массив a[M][N] в виде последовательно расположенных одномерных массивом размера N, то k-ый элемент ($0 \le k \le MN-1$) полученного ‘большого’ одномерного массива соответствует a[k/N][k%N] элементу матрицы, что следует из однозначного разложения числа k по модулю числа N:

$k = i*N + j$, $0 \le j < N$.

Поэтому решить задачу можно с помощью такого алгоритма: int k;

for(k = 0; k < M*N; k++) a[k/N][k%N] = b[k];

Заметим, что данный алгоритм является действенным, но не практичным, так как используется операция деления в каждом витке итерации. Намного практичнее использовать адресную арифметику в этом случае, о которой речь пойдет в следующей главе. С использо- ванием адресной арифметики алгоритм будет иметь такой вид:

```
for(k = 0; k < M*N; k++)
*(*a + k) = b[k];
```

где *a – адрес первого элемента матрицы a, *a + k – адрес k-го эле- мента матрицы, *(*a + k) – значение k-го элемента.

Если же требуется, наоборот, в массив b записать последова- тельно строки матрицы a, то это делается таким образом:

```
for (i = 0; i < M; i++) for(j = 0; j < N; j++)
b[i*N + j] = a[i][j];
```

Если учесть замечание, связанное с эффективностью использо- вания адресной арифметики, то данный алгоритм можно записать в такой форме:

```
for(k = 0; k < M*N; k++) b[k] = *(*a + k);
```

Пример 3. Подсчитать количество строк целочисленной матри- цы, упорядоченных по возрастанию.

```
#include<stdio.h> #include<stdlib.h> #define M 5
#define N 10

/* проверка одномерного массива на упорядоченность */ int IsSort(int *a, int n)
{
int i;
```

```
for(i = 0; i < n && a[i-1] <= a[i]; i++)
;
return (i >= n);
}

int main()
{
int a[M][N], i, count;
Init(a, M, N); /* функция из первого примера */ for(i = count = 0; i < M; i++)
if (IsSort(a[i], N)) count++;
printf("count = %d\n", count); return 0;
}
```

Пример 4. Пусть имеется некоторая квадратная матрица a раз- мера n. Требуется вывести на экран все подматрицы данной матрицы порядка 1, 2,…,n.

```
#include<stdio.h> #define N 5

/* вывод подматрицы порядка size, левая верхняя вершина которой имеет координаты (i0, j0) в матрице a:
*/
void Print(int a[][N], int i0, int j0, int size)
{
int i, j;
for (i = i0; i < i0 + size; i++)
{
for (j = j0; j < j0 + size; j++) printf("%5d", a[i][j]);
printf("\n");
}
printf("\n");
}

/* функция рассматривает все подматрицы в матрице a */ void SubMatrix(int a[][N], int n)
{
int i0, j0, size;
for(size = 1; size <= n; size++) for(i0 = 0; i0 < n - size + 1; i0++)
for(j0 = 0; j0 < n - size + 1; j0++) Print(a, i0, j0, size);
}

int main()
{
int a[N][N];
…/* инициализация матрицы a */ SubMatrix(a, N);
return 0;
}
```

Задачи

1. Дана матрица размером m × n. Найти суммы элементов всех ее четных строк.
2. Дана матрица размером m × n. Найти суммы элементов всех ее нечетных строк.
3. Дана матрица размером m × n. Найти минимальное значение сумм элементов.

4. Дана матрица размером m × n. Найти произведение элементов всех ее нечетных строк.

5. Дана матрица размером m × n. Найти произведение элементов всех ее четных строк.

6. Вывести номер первой строки матрицы, содержащей равное количество положительных и отрицательных элементов, причем нулевые элементы не учитываются.

7. Найти количество строк матрицы, все элементы которых различны.

8. Дана квадратная матрица порядка n. Найти сумму элементов ее главной и обратной диагоналей.

9. Дана квадратная матрица порядка n. Требуется транспонировать данную матрицу.

10. Дана квадратная матрица порядка n. Найти суммы элементов ее диагоналей, параллельных главной, начиная с одноэлементной диагонали.

11. Дана квадратная матрица порядка n. Заменить нулями элементы, лежащие одновременно выше главной и обратной диагоналей.

12. Удалить все столбцы матрицы, содержащие только положительные элементы.

13. Дана матрица размером m × n, где m и n – четные числа. Поменять местами левую верхнюю и правую нижнюю четверти матрицы.

14. Две строки матрицы назовем эквивалентными, если совпадают множества элементов, встречающихся в этих строках. Найти количество строк, эквивалентных первой строке.

15. Дана квадратная матрица порядка n. Найти сумму элементов ее главной и обратной диагоналей

Лабораторная работа 15.
Использование методов сортировки при программировании массивов.

Пример 1. Задан массив из 30 вещественных чисел. Найти элемент (индекс), имеющий максимальное значение в массиве.

```
// поиск позиции (индекса), содержащего максимальное значение
float B[30];
float max; // переменная, содержащая максимум
int index; // позиция элемента, содержащего максимальное значение
int i; // дополнительная переменная

// ввод массива
// ...

// поиск максимума
// установить максимум как 1-й элемент массива
index = 0;
max = B[0];
for (i=1; i<30; i++)
   if (max<B[i])
   {
      max = B[i]; // запомнить максимум
      index = i;  // запомнить позицию максимального элемента
   }
```

В вышеприведенном примере переменная max содержит максимальное значение. Переменная index содержит позицию элемента, который имеет максимальное значение. В начале переменной max присваивается значение первого элемента массива. Затем, начиная со второго элемента, происходит прохождение всего массива в цикле for. Одновременно проверяется условие

if (max<B[i])

Если условие выполняется (найден другой максимум), тогда новое значение максимума фиксируется в переменных max и index.

Вышеприведенный пример находит только один максимум. Однако, в массивах может быть несколько максимальных значений. В этом случае для сохранения позиций (индексов) максимальных значений нужно использовать дополнительный массив как показано в следующем примере.

Пример 2. Задан массив содержащий 50 целых чисел. Найти позицию (позиции) элемента, который имеет минимальное значение. Если таких элементов несколько, сформировать дополнительный массив индексов.

```
// поиск позиций (индексов), содержащих минимальное значение
int A[50];
int min; // переменная, содержащая минимальное значение
int INDEXES[50]; // позиции элементов, содержащих минимальное значение
int n; // число одинаковых минимальных значений
int i; // дополнительная переменная


// ввод массива
// ...

// 1. Поиск минимального значения
// установить минимальное значение в первом элементе массива
min = A[0];
for (i=1; i<50; i++)
   if (min>A[i])
      min = A[i]; // запомнить минимальное значение

// 2. Формирование массива
n = 0; // обнулить счетчик в массиве INDEXES
for (i=0; i<50; i++)
   if (min == A[i])
      {
         n++; // увеличить число элементов в INDEXES
         INDEXES[n-1] = i; // запомнить позицию
      }

listBox1->Items->Clear();

// 3. Вывод массива INDEXES в listBox1
for (i=0; i<n; i++)
   listBox1->Items->Add(INDEXES[i].ToString());
```

В вышеприведенном листинге сначала ищется минимальное значение min.

На втором шаге формируется массив INDEXES, в котором число элементов записывается в переменную n. Происходит поиск минимального значения в массиве A с одновременным формированием массива INDEXES.

На третьем шаге приведен пример, как вывести массив INDEXES в элементе управления listBox1(ListBox).

3. Сортировка массива методом вставки

Пример. Пусть дан массив A, содержащий 10 целых чисел. Отсортировать элементы массива в нисходящем порядке с помощью метода вставки.

```
// сортировка массива методом вставки
int A[10];
int i, j; // дополнительные переменные - счетчики
int t; // дополнительная переменная

// ввод массива A
// ...

// сортировка
for (i=0; i<9; i++)
   for (j=i; j>=0; j--)
      if (A[j]<A[j+1])
      {
         // поменять местами A[j] и A[j+1]
         t = A[j];
         A[j] = A[j+1];
         A[j+1] = t;
      }
```

4. Поиск элемента в массиве. Примеры

Пример 1. Определить, находится ли число k в массиве M состоящем из 50 целых чисел.

```
// определение наличия заданного числа в массиве чисел
int M[50];
int i;
int k; // искомое значение
bool f_is; // результат поиска, true - число k есть в массиве, иначе false

// ввод массива M
// ...
// ввод числа k
// ...

// поиск числа в массиве
f_is = false;
for (i=0; i<50; i++)
   if (k==M[i])
   {
      f_is = true; // число найдено
      break; // выход из цикла, дальнейший поиск не имеет смысла
   }

// вывод результата
if (f_is)
   label1->Text = "Число " + k.ToString() + " есть в массиве M.";
else
   label1->Text = "Числа " + k.ToString() + " нет в массиве M.";
```

Пример 2. Найти все позиции вхождения числа k в массиве M состоящим из 50 целых чисел.

```
// определение всех позиций заданного числа в массиве чисел
int M[50]; // массив чисел
int i; // вспомогательная переменная
int k; // искомое значение
int INDEXES[50]; // искомый массив позиций вхождения числа k
int n; // количество найденных позиций или количество элементов в массиве INDEXES

// ввод массива M
// ...
// ввод числа k
// ...

// поиск числа k в массиве M и одновременное формирование массива INDEXES
n = 0;
for (i=0; i<50; i++)
   if (k==M[i])
   {
      // число найдено
      n++;
      INDEXES[n-1] = i;
   }

// вывод результата в listBox1
listBox1->Items->Clear();

for (i=0; i<n; i++)
   listBox1->Items->Add(INDEXES[i].ToString());
```

Индивидуальные задания

1. Составить программу сортировки массива A по убыванию величин его элементов.
2. Дан упорядоченный массив A - числа, расположенные в порядке возрастания, и число a, которое необходимо вставить в массив A, так, чтобы упорядоченность массива сохранилась.
3. Составить программу для быстрой перестройки данного массива A, в котором элементы расположены в порядке возрастания, так, чтобы после перестройки эти же элементы оказались расположенными в порядке убывания.
4. Дан массив A, содержащий как отрицательные, так и положительные числа. Составить программу исключения из него всех отрицательных чисел, а оставшиеся положительные расположить в порядке их возрастания.
5. Составить программу, которая будет из массива A брать одно число за другим и формировать из них массив B, располагая числа в порядке возрастания.
6. Дан список авторов в форме массива A. Составить программу формирования указателя авторов в алфавитном порядке и вывести его на экран.
7. Имеется n абонентов телефонной станции. Составить программу, в которой формируется список по форме: номер телефона, фамилия (номера идут в порядке возрастания).

8. Имеется n слов различной длины, все они занесены в массив A. Составить программу упорядочения слов по возрастанию их длин.

9. Составить программу для сортировки массива A по возрастанию и убыванию модулей его элементов.

10. В отсортированный массив A между каждой соседней парой элементов вставить число больше левого и меньше правого элемента в массиве и вывести полученный массив на экран.

11. Дан массив из 10 элементов. Первые 4 упорядочить по возрастанию, последние 4 по убыванию.

12. Измените алгоритм предыдущей задачи для n элементов.

13. Дан массив 20 целых чисел на отрезке [-2;5]. Упорядочить массив, удалив нули со сдвигом влево, ненулевыми элементами.

14. Дан массив 15 целых чисел на отрезке [-5;5]. Упорядочить массив, удалив повторяющиеся элементы.

Лабораторная работа №16
Программирование массивов с использованием методов сортировки

Цель работы: Выполнить анализ эмпирической эффективности (практической сложности) следующих методов сортировки: подсчетом, включением, извлечением, обменом.

Планируемые результаты обучения

В результате выполнения лабораторной работы у студентов должны быть сформированы и развиты следующие профессиональные компетенции:

– **на уровне применения:** выполнять программную реализацию методов сортировки и подготовку входных данных для анализа эмпирической эффективности методов сортировки;

– **на уровне анализа:** выполнять анализ эмпирический эффективности методов сортировки;

– **на уровне оценки:** аргументировать выбор метода сортировки для решения задачи обработки данных.

План выполнения лабораторной работы

1) изучение методов сортировки: подсчетом, включением, извлечением, обменом;
2) разработка алгоритмов указанных методов сортировки и их представление в виде блок-схем;
3) программная реализация указанных методов сортировки;
4) генерация исходных данных;
5) проведение вычислительных экспериментов с помощью разработанной программы с целью измерения времени работы каждого метода.
6) построение графика зависимости времени работы методов сортировки от размера исходной последовательности.
7) сравнительный анализ полученных результатов.

Теоретический материал

Многие задачи, связанные с обработкой и поиском информации, решаются быстрее, проще и эффективнее, если данные хранятся в памяти ЭВМ в определенном порядке.

Процесс упорядочивания записей по возрастанию или убыванию значений ключа называется сортировкой.

В зависимости от состава технических средств, используемых в процессе

сортировки, различают внутреннюю и внешнюю сортировки. Сортировка внутренняя, если весь упорядочиваемый массив целиком помещается в ОП и находится там в течение всего процесса сортировки. Внешняя сортировка производится в массивах данных, объем которых превышает свободный объем ОП.

Существует множество методов внутренней сортировки, каждый из которых имеет свои преимущества и недостатки, оказываясь эффективнее других при определенных конфигурациях данных и аппаратуры. Оценка характеристик методов сортировки позволяет в каждом конкретном случае сделать правильный выбор.

Исходная последовательность записей может иметь различную степень упорядоченности. Возможно, что ее элементы (записи) уже расположены в установленном порядке. В другом крайнем случае элементы следуют в порядке, обратном установленному, т.е. исходная последовательность имеет обратную упорядоченность. В общем случае элементы последовательности размещены в любом произвольном порядке.

Критериями оценки различных методов обычно являются среднее число операций сравнения, выполняемых в процессе сортировки С, и среднее число перестановок или обменов элементов. Эффективность сортировки определяется как частное от деления среднего числа сравнений на число элементов массива.

Сортировка подсчетом

Упорядоченная последовательность В создается в свободной области памяти в результате сортировки исходной последовательности А. Метод основан на том, что (k+1)-й элемент упорядоченной последовательности превышает ровно k элементов и, следовательно, занимает (k+1)-ю позицию. В процессе сортировки на каждом i-м проходе i-й элемент исходной последовательности попарно сравнивается со всеми остальными элементами. Если в результате сравнения установлено, что $A_i > A_j$, то значение числа k увеличивается на единицу. По окончании прохода значение k оказывается равным числу элементов, меньших чем A_i Номер позиции i-го элемента в последовательности В равен k + 1.

Пример сортировки методом подсчета приведен таблице 1.

Таблица 1 – Пример сортировки подсчетом

i	1	2	3	4	5	6
A_i	10	4	11	9	7	2
№ прохода	1	2	3	4	5	6
k	4	1	5	3	2	0
B(k+1)	2	4	7	9	10	11

В результате первого прохода устанавливается, что первый элемент исходной последовательности А (1) = 10 оказался больше четырех элементов и для него устанавливается К = 4. Этот элемент займет пятую позицию в упорядоченной последовательности В. Аналогично определяются позиции всех остальных элементов последовательности.

Для сортировки последовательности из N элементов требуется N проходов, на каждом проходе выполняется N сравнений. Число проходов и сравнений не зависит от степени упорядоченности исходной последовательности. Поэтому для данного метода максимальное, минимальное и среднее число сравнений равно N^2.

Рассмотренный алгоритм сортировки методом подсчета можно использовать лишь в тех случаях, когда в исходной последовательности отсутствуют одинаковые элементы, иными словами, когда в упорядочиваемом массиве нет записей с одинаковыми значениями ключей. Для сортировки массивов, имеющих записи с одинаковыми значениями ключей,

алгоритм нужно модифицировать.

Сортировка включением

При использовании этого метода сортировки упорядоченная последовательность создается на свободном участке памяти. Под сортировку выделяется объем памяти, равный длине отсортированного массива записей. Так как исходная и упорядоченная последовательности располагаются в разных участках памяти, используем для их обозначения различные символы. Элементы исходной последовательности обозначим A_i, а элементы упорядоченной последовательности – Bj .

Первый элемент A_1 исходной последовательности A занимает первую позицию в свободном участке памяти, т.е. становится первым элементом B_1 последовательности B. Элемент A_2 сравнивается с B_1. Если в результате сравнения оказалось, что $A_2 < B_1$, то элемент B_1 передвигается на одну позицию, а элемент A_2 занимает его прежнее место. Теперь в свободном участке памяти размещены два элемента B_1 и B_2, образующих последовательность, упорядоченную по возрастанию значений ключа.

В течение каждого i-го прохода процесса сортировки элемент A_t сравнивается поочередно со всеми элементами последовательности B, начиная с B_1. При обнаруженииB_j, большего A_i элементы Bj, B_{j+1}, B_{j+2}, ..., B_{i-1}, передвигаются на одну позицию, освобождая место для элемента A_i, который занимает j-ю позицию.

В таблице 2 приведен пример сортировки включением.

Последовательность из N элементов будет отсортирована за N проходов. В первом проходе сравнений не требуется, так как первый элемент просто размещается в первой ячейке памяти. В дальнейшем в течение каждого /-го прохода будет выполнено в худшем случае i - 1 сравнение. Худшим окажется тот случай, когда исходная последовательность уже отсортирована в нужном порядке.

Таблица 2 – Пример сортировки включением

A		10	4	11	9	7	2
B	*Проход 1*	10					
	Проход 2	4	10				
	Проход 3	4	10	11			
	Проход 4	4	9	10	11		
	Проход 5	4	7	9	10	11	
	Проход 6	2	4	7	9	10	11

Максимальное число сравнений равно сумме членов арифметической прогрессии 1 + 2 + 3 + ... + (N– 1) и определится формулой

$$C_{max} = \sum_{i-1}^{N-1} (N - i) = 0{,}5N(N-1)$$

Если исходная последовательность имеет обратную упорядоченность, то для сортировки потребуется минимальное число сравнений $C_{min} = N – 1$.

Среднее число сравнений пропорционально $0{,}25N^2$.

Минимальное число перестановок равно нулю, и будет это в том случае, когда исходная последовательность уже упорядочена. Максимальное число перестановок C_{max} потребуется для сортировки исходной последовательности, имеющей обратную упорядоченность. Среднее число перестановок пропорционально $0{,}25N^2$.

Сортировка извлечением.

При сортировке этим методом упорядоченная последовательность записей создается на том же участке памяти, что и исходная последовательность. В течение первого прохода осуществляется поиск наименьшего элемента. После того как этот элемент найден, его меняют местами с первым элементом исходной последовательности, в результате чего наименьший элемент занимает первую позицию в формируемой упорядоченной последовательности. Затем осуществляется поиск следующего наименьшего элемента среди оставшихся. Найденный элемент меняется местами со вторым элементом исходной последовательности. После второго прохода окажется сформированной последовательность из двух элементов, первый из которых меньше второго. Поиск элементов со следующими наименьшими значениями ключа и помещение их в соответствующие позиции исходной последовательности продолжается до тех пор, пока все элементы не будут отсортированы в восходящем порядке.

Пример сортировки посредством выбора приведен в таблице 3. При иллюстрации методов сортировки предполагается, что записи состоят только из поля ключа, т.е. элементами упорядочиваемых последовательностей являются значения ключей записей. Цифры, отмеченные кружочками, означают записи с наименьшим ключом, выбранные на данном проходе. Элементы, расположенные слева от двойной черты для данного прохода, уже расставлены по порядку. Исходная последовательность A, состоящая из шести элементов, отсортирована за пять проходов.

Таблица 3 – Пример сортировки методом извлечения

i	1	2	3	4	5	6
A_i	10	4	11	9	7	(2)
Проход 1	2 ‖	(4)	11	9	7	10
Проход 2	2	4 ‖	11	9	(7)	10
Проход 3	2	4	7	(9)	11	10
Проход 4	2	4	7	9	11	(10)
Проход 5	2	4	7	9	10	11

Произведем оценку характеристик данного метода. Для сортировки последовательности из N элементов требуется N – 1 проходов, так как на каждом проходе в соответствующую позицию упорядоченной последовательности заносится только один элемент. Для i-го прохода требуется N – i сравнений. Следовательно, общее число сравнений

$$C = \sum_{i-1}^{N-1} (N - i) = 0{,}5N(N-1)$$

При сортировке рассмотренным методом число сравнений не зависит от степени упорядоченности исходной последовательности. Поэтому полученное выражение определяет минимальное, максимальное и среднее число сравнений. Для оценки среднего числа сравнений можно использовать следующую аппроксимацию выражения (1): $0{,}5N^2$. Такая аппроксимация дает ошибку 1 % при N = 100 и 0,1 % при N = 1000. Можно считать, что среднее число сравнений при сортировке методом выбора пропорционально 0,5 N^2.

Количество перестановок элементов зависит от того, как расположены элементы исходной последовательности. Однако в любом случае в течение одного прохода потребуется не более одной перестановки; следовательно, максимальное количество

перестановок равно N – 1. В лучшем случае, когда исходная последовательность уже упорядочена, не потребуется ни одной перестановки. Следовательно, среднее число перестановок пропорционально N/2.

Сортировка обменом

При сортировке этим методом упорядоченная последовательность создается на том же месте в памяти, где располагалась исходная последовательность. В процессе сортировки производится попарное сравнение соседних элементов. Если порядок между сравниваемыми элементами нарушен, они меняются местами. Этот метод обмена называется часто методом пузырька, так как наименьшие элементы с каждым проходом, подобно пузырькам, "всплывают" по направлению к первой позиции последовательности. В таблице 4 приведен пример сортировки методом пузырька. В течение первого прохода сравниваются элементы A_2 и A_2 . Если $A_2 < Ai$, то элементы меняются местами, при этом элемент A_2 занимает первую позицию, а элемент A_1 – вторую.

Таблица 4 – Пример сортировки методом пузырька

i	A_i	*Проход 1*	*Проход 2*	*Проход 3*	*Проход 4*	*Проход 5*
1	10	4	4	4	4	2
2	4	10	9	7	2	4
3	11	9	7	2	7	7
4	9	7	2	9	9	9
5	7	2	10	10	10	10
6	2	11	11	11	11	11

Этот процесс повторяется для пар элементов A_2 и A_3, A_3 и $Л_4$ и т.д. После первого прохода наибольший элемент займет N-ю позицию, а минимальный – переместится на одну позицию вверх ("всплывет").

На каждом следующем проходе очередные наибольшие элементы занимают, соответственно, позиции N – 1, N – 2 и т.д., в результате чего будет сформирован упорядоченный массив.

После каждого прохода может быть сделана проверка, были ли совершены перестановки в течение данного прохода. Если перестановок не было, то это означает, что последовательность упорядочена и дальнейших проходов не требуется. В течение прохода фиксируется последний элемент, участвующий в обмене (в таблице 2 эти элементы подчеркнуты двойной чертой). В очередном проходе подчеркнутый элемент и все последующие в сравнении не участвуют, так как начиная с этой позиции последовательность уже упорядочена.

Число сравнений для этого метода зависит от числа проходов, необходимых для сортировки. В худшем случае, когда последовательность имеет обратную упорядоченность, в течение каждого i-го прохода выполняются перестановки, а число проходов равно N – 1. При этом для сортировки потребуется максимальное число сравнений C_{max}, равное сумме членов арифметической прогрессии (N-1) + (N-2)+ (N- 3)+... + 1 , т.е.

$$C_{max} = \sum_{i-1}^{N-1} (N - i) = 0{,}5N(N - 1)$$

В лучшем случае, когда исходная последовательность уже упорядочена, потребуется всего один проход и N – 1 сравнение.

Минимальное число сравнений C_{min} = N – 1. Среднее число сравнений

пропорционально $0,25N^2$.

Число обменов при сортировке методом пузырька зависит от степени упорядоченности исходной последовательности. Если исходная последовательность полностью упорядочена, то обменов нет. Максимальное число обменов будет в последовательности, имеющей исходный обратный порядок, т.е. в том случае, когда надо отсортировать по возрастанию ключа записи, расположенные по убыванию ключа. В этом случае обмен происходит при каждом сравнении и общее число обменов равно 0,5 N (N – 1). Среднее число обменов пропорционально $0,25N^2$.

2. Методические указания по выполнению работы

Для выполнения работы необходимо:

1. Построить блок-схемы алгоритмов сортировки (включением, обменом).
2. Выполнить программную реализацию алгоритмов сортировки. Правильность работы программ подтвердить скриншотами.
3. Сгенерировать исходные последовательности в зависимости от заданного размера выборки.

− Для генерации исходной числовой последовательности можно использовать датчик случайных чисел.

– Для чистоты эксперимента данные должны быть из одной сгенерированной выборки.

4. Выполнить эксперименты, в ходе которых определить время работы каждого алгоритма в зависимости от размера выборки, используя функцию clock(), которая должна быть описана в заголовочном файле <time.h> в следующем виде:

clock_t clock (void);

Тип данных clock_t определяется в файле time.h. Функция clock_t возвращает процессорное время в единицах, которые зависят от реализации языка. (Если процессорное время недоступно или не может быть представлено, функция возвращает значение -1.) Однако в файле time.h также определена константа CLOCKS_PER_SEC, которая представляет количество единиц процессорного времени в секунде. Следовательно, в результате деления разницы между двумя возвращаемыми значениями функции clock () на константу CLOCKS_PER_SEC получается количество секунд, прошедшее между двумя вызовами функции.

Приведение значений к типу double до операции деления позволит получать результат в долях секунды. Например, если вам необходимо получить время выполнения каких-либо действий, то вам необходимо вставить в программу следующий код:

```
#include <time.h >
.....

clock_t time;
time = clock();
/* какие-то действия */
time = clock() - time;
printf("%f", (double)time/CLOCKS_PER_SEC); //время выполнения "каких-то действий"
```

Результаты проведенных экспериментов по сортировке данных представить таблице 1.

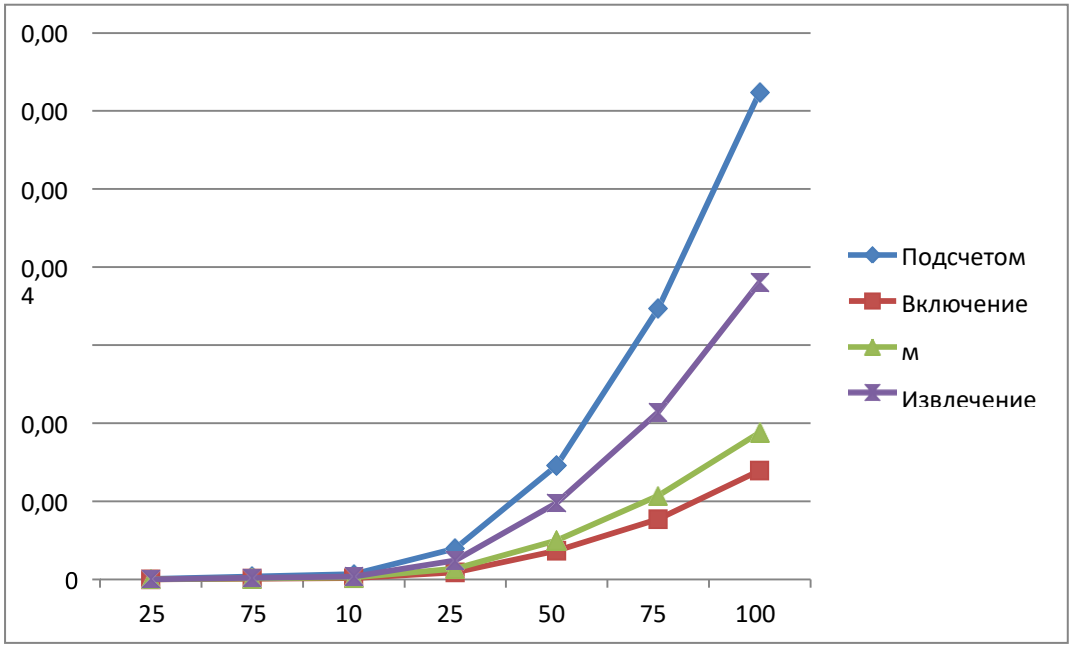
Таблица 1 - Результаты сортировки

Объ	Время, ед.вр. (сек./млсек/мсек/тик)

ём выборки, N	Включением	Обмеом
100		
500		
1000		
2500		
5000		
7500		
10000		
20000		
30000		

4. Построить графики зависимостей времени работы алгоритмов сортировки от размера выборки. Пример графика приведен на рисунке 1.

Рисунок 1 – График зависимости времени работы алгоритмов сортировки от объема выборки



Для выполнения лабораторной работы сгенерировать следующие выборку из 30000 целочисленных элементов, в дальнейшем из нее формировать выборки: 100, 500, 750, 1000, 2500, 5000, 10000, 20000, 30000.

На следующей странице дан пример оформления титульного листа отчета и оглавление.

Лабораторная работа 17.
Создание структурных переменных.

Структура – объединение одного или нескольких объектов, в качестве которых могут выступать переменные, массивы, указатели и т.д. Некоторые объекты, которые используются в конкретной программе, так или иначе могут быть логически связаны между собой, например: абсцисса (double x) и ордината (double y) могут являться координатами некоторой точки плоскости, или фамилия (char name[20]), год рождения (int year) и группа (char group[30]) могут со- держать сведения о том или ином студенте и т.д. Поэтому для удобства работы с такими объектами их можно группировать под одним именем. Если массив представляет собой набор однотипных данных, то структура является более универсальным типом данных, поскольку она может содержать объекты различных типов.

Объявление структуры осуществляется с помощью ключевого слова struct, за которым следует имя структуры, и далее идет перечисление списка объектов, заключенных в фигурные скобки. Например, описание структуры, представляющей собой точку плоскости, выглядит следующим образом:

```
struct DOT
{
double x;/* абсцисса */ double y;/* ордината */
};
```

Описание же структуры, которая представляет собой сведения о студенте, может выглядеть таким образом:

```
struct STUDENT
{
        char name[20];/* фамилия студента*/
        int year;/* год рождения*/
        char group[30];/* группа*/
};
```

Переменные, объявленные внутри структуры, называются ее элементами. Элементы одной структуры должны иметь уникальные имена, хотя имена элементов различных структур могут содержать одинаковые имена. Каждое объявление структуры должно заканчиваться точкой с запятой.

При объявлении структуры внутри программы резервирования памяти под нее не происходит, она просто описывает новый тип данных. Объявив структуру, можно создавать переменные этого типа:

struct DOT A, B;/* переменные A и B типа DOT */ struct STUDENT stud; /* переменная stud типа STUDENT */

Этим самым мы объявили две переменные-структуры A и B, которые могут содержать координаты точек плоскости, и одну переменную- структуру stud. Обращение к отдельным элементам структуры происходит через операцию "точка" (.): A.x, A.y, stud.name, stud.year, stud.group. В следующей программе вводятся координаты точки плоскости A, а затем выводится на экран расстояние от этой точки до начала координат.

```
#include <stdio.h> #include <math.h> struct DOT
{
double x; double y;
};
int main( )
```

```
{
struct DOT A; scanf("%lf", &A.x);
scanf("%lf", &A.y);
printf("d = %.2f\n", sqrt(A.x*A.x + A.y*A.y)); return 0;
}
```

Важным моментом при описании структуры является место, где она объявляется. Если объявление происходит вне всех функций (такое объявление называется внешним), тогда это объявление может использоваться всеми функциями, следующими за ним. Если же структура объявляется внутри какой-то функции, тогда это объявление может использоваться только этой функцией.

Как и массивы, структурные переменные при их определении можно инициализировать, например:

```
struct DOT A = {1.2, -2.5};
struct STUDENT stud = {"Иванов", 1990, "КБ-11"};
```

Если структура передается в качестве аргумента функции, то, в отличие от массивов, передача происходит по значению. Чтобы осуществить передачу структуры по ссылке, необходимо передать ее адрес.

Помимо того, что функциям можно передавать структуры в качестве аргументов, функции могут также возвращать структуры, например:

```
#include <stdio.h> struct DOT
{
double x; double y;
};
struct DOT Init(double x, double y)
{
struct DOT A;
A.x = x;
A.y = y; return A;
}
int main( )
{
struct DOT B = Init(2.4, 1.5); return 0;
}
```

Если две структурные переменные имеют один тип, то одну пе- ременную можно присвоить другой, например:

```
struct DOT A = {1.2, -2.5}, B; B = A;
```

При этом значениями элементов одной структуры будут значения соответствующих элементов другой структуры, если даже среди элементов структуры присутствуют массивы.

Также можно создавать массивы структур. Например, можно задать последовательность из 10 элементов точек плоскости:

```
struct DOT  a[10];
```

Тогда обращение к координатам точек будет происходит следующим образом:

```
a[0].x, a[0].y,…, a[9].x, a[9].y
```

Объединения

Объединение – такая переменная, которая в разные моменты времени может хранить объекты различного типа и размера. В каждый определенный момент времени может

использоваться только один из элементов объединения. Главной особенностью объединения является то обстоятельство, что для каждого из объявленных элементов выделяется одна и та же область памяти, то есть они перекрываются. Поэтому если в данный момент времени используется некоторый элемент объединения, то значения остальных элементов становятся неопределенными. Объединения экономят пространство вместо того, чтобы просто так тратить память на неиспользуемые в данный момент переменные.

Для описания объединения используется ключевое слово union, например:

```
union NUMBER
{
long l; double d;
};
```

Здесь переменная l имеет размер 4 байта, а переменная d – 8 байт. Поэтому на переменную типа NUMBER будет отведено 8 байт, то есть достаточно, чтобы хранить самую большую из данных двух переменных.

При объявлении объединение можно инициализировать только значением, имеющим тип его первого элемента. Таким образом, в нашем случае объединение можно инициализировать только значением типа long, например можно записать

union NUMBER value = {10};/* то есть value.l == 10 */ и была бы ошибочной запись вида

union NUMBER value = {10.0};

Ниже приведен небольшой пример с использованием объединения:

```
#include <stdio.h> union NUMBER
{
long l; double d;
};
int main( )
{
union NUMBER value = {10}; printf("l = %d\n", value.l); value.d = 5.0;
printf("d = %f\n", value.d); return 0;
}
```

Объединения могут входить в структуры и массивы. Обратное также возможно. Например, ниже приводятся два эквивалентных объявления структуры STR:

```
union NUMBER
{
long l; double d;
};
struct STR
{
char s[10];
union NUMBER value;
};
```

```
struct STR
{
char s[10]; union
NUMBER
{
long l; double d;
} value;
};
```

После одного из данных объявлений обращение к элементом структуры STR происходит следующим образом:

```
int main( )
{
struct STR st ={"abc", 10};/* st.s == “abc”, st.value.l == 10 */ printf("s = %s\n", st.s);
printf("l = %d\n", st.value.l); st.value.d = 1.0;
printf("d = %f\n", st.value.d); return 0;
}
```

Индивидуальные задания:

Ввести массив структур в соответствии с вариантом. Рассортировать массив в алфавитном порядке по первому полю, входящему в структуру. В программе реализовать меню:

1)Ввод структуры;

2)Вывод структуры;

1.Структура «Автосервис»: регистрационный номер автомобиля, марка, пробег, мастер, выполнивший ремонт, сумма ремонта.

2.Структура «Сотрудник»: фамилия, имя, отчество; должность; год рождения; заработная плата.

3.Структура «Государство»: название; столица; численность населения; занимаемая площадь.

4.Структура «Человек»: фамилия, имя, отчество; домашний адрес; номер телефона; возраст.

5.Структура «Читатель»: Фамилия И.О., номер читательского билета, название книги, срок возврата.

6.Структура «Школьник»: фамилия, имя, отчество; класс; номер телефона; оценки по предметам (математика, физика, русский язык, литература).

7.Структура «Студент»: фамилия, имя, отчество; домашний адрес; группа; рейтинг.

8.Структура «Покупатель»: фамилия, имя, отчество; домашний адрес; номер телефона; номер кредитной карточки.

9.Структура «Пациент»: фамилия, имя, отчество; домашний адрес; номер медицинской карты; номер страхового полиса.

10.Структура «Информация»: носитель; объем; название;

автор.

11.Структура «Клиент банка»: Фамилия И.О., номер счета, сумма на счете, дата последнего изменения.

12. Структура «Склад»: наименование товара, цена, количество, процент торговой надбавки.

13.Структура «Авиарейсы»: номер рейса, пункт назначения, время вылета, дата вылета, стоимость билета.

14.Структура «Вокзал»: номер поезда, пункт назначения, дни следования, время прибытия, время стоянки.

15.Структура «Кинотеатр»: название кинофильма, сеанс, стоимость билета, количество зрителей.

Лабораторная работа 18.
Решение задач с использованием массива символов

В языке Си нет такого отдельного типа данных, как строка. В Си под строкой понимается последовательность символов, которая заканчивается символом ‘\0’. Строку

можно определить несколькими способами.

Строка как массив символов. Во-первых, строку можно определить как массив символов, который заканчивается нулевым символом ‘\0’. Нуль-символ (‘\0’) используется для того, чтобы отмечать конец строки. В таблице символов ASCII данный символ имеет номер

0. При определении строки как массива символов ей присваивается имя и указывается максимальное количество символов, которое может содержаться в ней с учетом нулевого символа. Например, char s[10]. Такое объявление строки в виде массива символов требует определенного места в памяти для десяти элементов. Значением строки s является адрес ее первого символа, через который осуществляется доступ ко всем ее элементам.

Например, в следующей программе объявляется массив символов s, который заполняется пользователем с помощью функции fgets(), а затем выводится на экран при помощи функции puts().

```
#include <stdio.h> int main( )
{
char s[10];/* массив символов */ printf("введите не более 9 символов: "); fgets(s, 10, stdin);
puts(s);/* вывод строки */ return 0;
}
```

При объявлении строка может быть сразу же инициализирована, то есть она может быть присвоена массиву символов:

char s[] = “moon”;

При таком объявлении создается массив s, состоящий из 5 элементов: ‘m’, ‘o’, ‘o’, ‘n’, ‘\0’. Отдельные символы внутри массива могут изменяться, но сама переменная s будет указывать на одно и то же место памяти.

Объявление строки в виде массива символов с последующей инициализацией можно записать также следующим образом:

char s1[5] = {‘m’, ‘o’, ‘o’, ‘n’, ‘\0’};

char s2[7] = {‘m’, ‘o’, ‘o’, ‘n’, ‘\0’}.

То же самое можно сделать и в более удобной форме, а именно: char s1[5] = “moon”;

char s1[7] = “moon”;

Подчеркнем, что наличие нулевого символа означает, что количество элементов массива символов должно быть по крайней мере на один больше, чем максимальное количество символов, планируемых для размещения в памяти.

Заметим, что после объявления строки, например char s[10];

абсолютно недопустима запись вида s = “Hello”;

то есть имя массива нельзя использовать в левой части операции присваивания.

Строка как указатель на первый символ. Строку можно также определить и другим способом – с использованием указателя на символ. Если объявить char *ps, то этим самым задается переменная ps, которая может содержать адрес первого символа строки. В этом случае не происходит резервирования памяти для хранения символов, как в случае с массивом символов, и сама переменная ps не инициализируется конкретным значением.

Если где-то в программе после объявления char *ps встречается инструкция вида ps = “sun”, тогда компилятор производит следующие действия: в некоторую область памяти (скорее всего в область памяти read only) последовательно друг за другом записываются символы ‘s’, ‘u’, ‘n’ и ‘\0’; переменной ps присваивается адрес первого символа, то есть, в нашем случае, адрес символа ‘s’. После инициализации строки ps к каждому ее символу можно обращаться через индекс: ps[0], ps[1], ps[2].

Ниже приводится пример использования указателя на строку:

```
#include <stdio.h> int main( )
{
char s[17] = "this is a string";
/* переменной ps присваивается адрес первого символа строки s */ char *ps = s;
int i;
for (i = 0; i < 16; i++)
/* обращение к элементам строки ps по их индексам: */ printf("%c", ps[i]);
printf("\n");
ps = ps + 8;/* указателю ps присваивается адрес буквы ‘a’ */ printf("%s\n", ps);/*
вывод на экран новой строки */
return 0;
}
```

В результате на экране появятся две строчки: this is a string
a string

Как и в случае с массивом символов, строку, объявленную через указатель на первый символ, можно сразу же инициализировать, например

```
char *ps = “sun”;
```

В этом случае создается переменная-указатель ps, которая указывает на строку “sun”.

Заметим, что строка, определенная как указатель на первый символ строки, может быть помещена в неизменяемую область памяти, то есть значения элементов в этом случае изменить будет нельзя, хотя можно будет изменить значение самого указателя ps.

Еще раз подчеркнем, что при объявлении строки как массива символов (например, char s[10]) происходит резервирование места в памяти, в то время как объявление указателя (char *s) требует место только для самого указателя, который может указывать на любой символ или массив символов.

Ниже приводятся несколько примеров работы с символами и строками.

Пример 1. Преобразование строки s в целое число. int Atoi(const char *s)

```
{
int i, n;
i = n = 0;
while (s[i] >= '0' && s[i] <= '9')
{
n = n*10 + (s[i] - '0'); i++;
}
return n;
}
```

В данном случае выражение s[i] – ‘0’ дает численное значение символа, который хранится в s[i], поскольку в таблице символов цифры ‘0’, ‘1’, …, ’9’ идут подряд. То есть ‘0’ преобразуется в 0, ‘1’ преобразуется в 1 и так до символа ‘9’, который преобразуется в число 9.

Пример 2. Преобразование прописного латинского символа в строчный. Если же символ не является прописной латинской буквой, то он остается без изменения.

```
int ToLower(const int c)
{
if (c >= 'A' && c <= 'Z')
return 'a' + (c - 'A'); else return c;
```

```
}
```

Пример 3 (эффективное удаление символов из строки). Функция удаления из строки s всех символов, совпадающих с символом c.

В функции DeleteSymbol() будем перезаписывать все символы, входящие в строку s, таким образом, чтобы среди них не оказался символ c.

```
void DeleteSymbol(char *s, const int c)
{
int i, j;
for(j = 0; s[j] && s[j] != c; j++)
;
for(i = j; s[i]; i++)
if (s[i] != c)
s[j++] = s[i];
s[j] = '\0';
}
```

В данной функции не совершается ни одной лишней перестановки, поэтому она является очень эффективной и оптимальной.

Пример 4. Определение длины строки без учета символа конца строки ‘\0’.

```
int Strlen(const char *s)
{
int i = 0;
while (s[i] != ‘\0’) i++;
return i;
}
```

Пример 5. Копирование строки t в строку s.

```
void Strcpy(char *s, const char *t)
{
int i = 0;
while ((s[i] = t[i]) != '\0') i++;
}
```

Пример 6. Добавление строки t к строке s.

```
void Strcat(char *s, const char *t)
{
int i, j;
i = j = 0;
while (s[i] != '\0') i++;
while ((s[i++] = t[j++]) != '\0')
;
}
```

Индивидуальные задания

1. Составить из букв введенной строки слова. Вводится строка, состоящая из букв и пробелов. Составить из входящих в нее букв несколько любых их сочетаний

(слов) любой длины. Каждую букву строки можно использовать неограниченное количество раз.

2. Является ли строка идентификатором. Определить, является ли введенное слово идентификатором, т.е. начинается ли оно с английской буквы в любом регистре или знака подчеркивания и не содержит других символов, кроме букв английского алфавита (в любом регистре), цифр и знака подчеркивания.

3. Удалить из строки пробелы и определить, является ли она перевертышем. Вводится строка. Удалить из нее все пробелы. После этого определить, является ли она палиндромом (перевертышем), т.е. одинаково пишется как с начала, так и с конца.

4. Замена подстроки. Найти в строке указанную подстроку и заменить ее на новую. Строку, ее подстроку для замены и новую подстроку вводит пользователь.

5. Отфильтровать из строки числа. Вводится строка, содержащая буквы, целые неотрицательные числа и иные символы. Требуется все числа, которые встречаются в строке, поместить в отдельный целочисленный массив. Например, если дана строка "data 48 call 9 read13 blank0a", то в массиве должны оказаться числа 48, 9, 13 и 0.

6. Удаление из строки повторяющихся символов. Вводится строка. Требуется удалить из нее повторяющиеся символы и все пробелы. Например, если было введено "abc cde def", то должно быть выведено "abcdef".

7. Удаление лишних пробелов. Вводится ненормированная строка, у которой могут быть пробелы в начале, в конце и между словами более одного пробела. Привести ее к нормированному виду, т.е. удалить все пробелы в начале и конце, а между словами оставить только один пробел.

8. Самая длинная строка в массиве. Вводятся строки. Определить самую длинную строку и вывести ее номер на экран. Если самых длинных строк несколько, то вывести номера всех таких строк.

9. Самое длинное слово в строке. Вводится строка слов, разделенных пробелами. Найти самое длинное слово и вывести его на экран. Случай, когда самых длинных слов может быть несколько, не обрабатывать.

10. Количество строчных и прописных букв в строке. Посчитать количество строчных (маленьких) и прописных (больших) букв в введенной строке. Учитывать только английские буквы.

11. Количество слов в строке. Вводится строка, состоящая из слов, разделенных пробелами. Требуется посчитать количество слов в ней.

12. Самая длинная строка в массиве. Вводятся строки. Определить самую длинную строку и вывести ее номер на экран. Если самых длинных строк несколько, то вывести номера всех таких строк.

13. Самое длинное слово в строке. Вводится строка слов, разделенных пробелами. Найти самое длинное слово и вывести его на экран. Случай, когда самых длинных слов может быть несколько, не обрабатывать.

Лабораторная работа 19.
Решение задач с использованием массивов символовов и строковых переменных

Пример 1. Дано целое число n, $1 \le n \le 26$. Записать в строку s первые n прописные латинские буквы.

```
#include <stdio.h> void Init(char *s, int n)
{
int i;
for (i = 0; i < n; i++) s[i] = 'A' + i;
```

```
s[i] = '\0';
}

int main( )
{
char s[27]; int n;
printf("n = "); scanf("%d", &n); Init(s, n);
puts(s); return 0;
}
```

Пример 2. Дана строка, изображающая целое положительное число. Найти сумму цифр этого числа.

```
#include  <ctype.h> int Sum(const char *s)
{
int i, sum;
i = sum = 0;
while (isdigit(s[i]))
{
sum += (s[i] - '0'); i++;
}
return sum;
}
```

Пример 3. Записать в строку n случайных латинских символов, среди которых с вероятностью 1/4 должны присутствовать пробелы. Такую строку можно интерпретировать как предложение, состоящее из слов, разделенных пробелами, и использовать для некоторых те- стовых алгоритмов.

```
#include<stdio.h> #include<stdlib.h> #include<time.h> #define N 20

void Init(char *s, int n)
{
int i;
for (i = 0; i < n - 1; i++) if (rand()%4 == 0)
s[i] = ' ';
else s[i] = 'a' + rand()%26;
s[n - 1] = '\0';
}
int main()
{
char s[N]; srand(time(NULL)); Init(s, N);
puts(s); return 0;
}
```

Если строку s необходимо заполнить случайными символами из некоторой строки SET, то функцию Init() из предыдущей программы можно прописать следующим образом:

```
#include<stdio.h> #include<stdlib.h> #include<time.h> #define N 20
#define SET "abcdefghigklmnopqrstuvwxyz .,;:"

void Init(char *s, int n)
```

```
{
int i, len;
len = strlen(SET);
for (i = 0; i < n - 1; i++) s[i] = SET[rand() % len];
s[n - 1] = '\0';
}

int main()
{
char s[N]; srand(time(NULL)); Init(s, N);
puts(s); return 0;
}
```

Пример 4. Пусть имеется целое неотрицательное число n в десятичной системе счисления и некоторое целое число $2 \le p \le 36$.

Требуется написать функцию, которая записывает в строку s представление числа n в p-й системе счисления.

```
#include<stdio.h>
#define SET "0123456789ABCDEFGHIGKLMNOPQRSTUVWXYZ"
void Itoa(int a, int p, char *s)
{
int i, j, buf; i = 0;
/* генерируем цифры в обратном порядке */ do
{
s[i] = SET[a % p];
a /= p; i++;
} while (a != 0);
s[i] = '\0';
/* переворачиваем строку */ j = i – 1;
i = 0;
while (i < j)
{
buf = s[i]; s[i] = s[j]; s[j] = buf; i++; j--;
}
}

int main()
{
char s[100]; Itoa(123, 16, s);
puts(s); return 0;
}
```

Пример 5. Рассмотрим обратную к предыдущей задаче. Пусть строка s содержит представление некоторого целого неотрицательного числа в p-ичной системе счисления, где $2 \le p \le 36$. Требуется из строки s собрать само число.

Пусть s=”$a_t \ldots a_1 a_0$” – представление некоторого числа в p-ичной системе счисления, где все $0 \le a_i < p$. Тогда само число равно такой сумме:

$a_t p^t + \ldots + a_1 p + a_0$.

```
#include<stdio.h>
#define SET "0123456789ABCDEFGHIGKLMNOPQRSTUVWXYZ"
int Atoi(char *s, int p)
{
```

```
int i, a, digit[256] = {0};
for(i = 0; SET[i]; i++)
digit[SET[i]] = i;
for(i = a = 0; s[i]; i++) a = a*p + digit[s[i]];
return a;
}

int main()
{
printf("%d\n", Atoi("FF", 16));
return 0;
}
```

Пример 6. Пусть SET – некоторое множество символов. Требуется в строке s подсчитать количество символов, принадлежащих множеству SET.

```
#include <stdio.h> #include <string.h>
#define N 1024/* размер строки s */ #define SET"ABC0123"

int Count(char *s)
{
int i, n;
for(i = n = 0; s[i]; i++)
if (strchr(SET, s[i])) n++;
return n;
}
int main( )
{
char s[N]; fgets(s, N, stdin);
printf("n = %d\n", Count(s)); return 0;
}
```

Заметим, что для решения данной задачи более целесообразнее применить функцию strpbrk():

```
int Count(char *s)
{
char *ps = s; int count = 0;
while(ps = strpbrk(ps, SET))
{
count++; ps++;
}
return count;
}
```

Замечание. В двух предыдущих функциях Count() используются функции strchr() и strpbrk() из библиотеки <string.h. Сложность алгоритма в среднем случае будет равна O(m*n), где m и n – длины со- ответствующих строк SET и s. Это не очень хорошо. Посмотрим как можно оптимизировать данный алгоритм.

Пусть int flag[256] – логический массив, каждый индекс которого соответствует тому или иному символу из таблицы символов ASCII. Все элементы массива flag, индексы которых соответствуют элементам строки SET, определим единицами, а все остальные элементы – 0:

```
int flag[256] = {0}; for(i = 0; SET[i]; i++)
flag[SET[i]] = 1;
```

Теперь чтобы проверить, является ли некоторый символ с эле- ментом строки SET, достаточно записать:

```
if (flag[c])
```

Такимобразом,функциюCount()можноусовершенствовать следующим образом:

```
int Count(char *s)
{
int i, count, flag[256] = {0}; for(i = 0; SET[i]; i++)
flag[SET[i]] = 1;
for(i = count = 0; s[i]; i++) count += flag[s[i]];
return count;
}
```

Сложность алгоритма в данном случае равна m+n, то есть алгоритм будет работать очень и очень быстро для любых строк.

Пример 7. Дана строка s. Подсчитать количество содержащихся в ней гласных и согласных латинских букв.

```
#include <stdio.h> #include <string.h> #include <ctype.h>
#define N 500/* размер строки s  */
#define VOWEL "AEIOUYaeiouy" /* гласные латинские буквы */

void Count(char *s, int *ng, int *ns)
{
int i;
for(i = *ng = *ns = 0; s[i]; i++) if (isalpha(s[i]))
if (strchr(VOWEL, s[i])) (*ng)++;
else (*ns)++;
}
int main( )
{
char s[N];
int ng,/* количество гласных букв*/ ns;/* количество согласных букв */
fgets(s, N, stdin); Count(s, &ng, &ns);
printf("ng = %d ns = %d\n", ng, ns); return 0;
}
```

Как и в предыдущем примере, можно значительно усовершенствовать алгоритм, введя дополнительный массив int flag[256]:

```
void Count(char *s, int *ng, int *ns)
{
int i, flag[256] = {0};
for(i = 0; VOWEL[i]; i++) flag[VOWEL[i]] = 1;
for(i = *ng = *ns = 0; s[i]; i++) if (isalpha(s[i]))
                                  if (flag[s[i]]) (*ng)++;  else (*ns)++;
                                                    }
```

Пример 8. Дана некоторая строка s. Определить сколько раз- личных символов встречается в строке s, не рассматривая символ ‘\0’. Опять же для большей

скорости решения данной задачи применим дополнительный массив flag, как и в двух предыдущих задачах. Для того, чтобы программа могла работать и с русскими символами, будем все символы приводить к типу unsigned char.

```
typedef unsigned char UCHAR; int Count(char *s)
{
int i, count, flag[256] = {0}; for(i = 0; s[i]; i++)
flag[(UCHAR)s[i]] = 1;
for(i = count = 0; i < 256; i++) count += flag[i];
return count;
}
```

Пример 9. Дана некоторая строка s. Проверить, все ли символы данной строки попарно различны.

```
typedef unsigned char UCHAR; int Check(char *s)
{
int i, flag = 1, count[256] = {0}; for(i = 0; s[i] && flag; i++)
{
count[(UCHAR)s[i]]++;
if (count[(UCHAR)s[i]] > 1) flag = 0;
}
return flag;
}
```

Заметим, что возможности языка Си позволяют записать функ- цию Count() в более компактной и оптимальной форме:

```
int Check(char *s)
{
int i, count[256] = {0};
for(i = 0; s[i] && ++count[(UCHAR)s[i]] < 2; i++)
;
return s[i] == '\0';
}
```

Пример 10. Дана некоторая строка s. Выделить в данной строке все целые неотрицательные числа и напечатать их.

```
#include <stdio.h> #include <ctype.h>

void Print(char *s)
{
int i, a;
/* пропускаем все символы, не являющиеся цифрами: */ for(i = 0; s[i] && !isdigit(s[i]); i++)
;
while (s[i])
{
a = 0;
/* пробегаем по всем подряд идущим цифрам, входящим в состав очередного числа:
```

```
*/
while (s[i] && isdigit(s[i]))
{
a = a*10 + (s[i] - '0'); i++;
}
printf("%d\n", a);
/* пропускаем все символы, не являющиеся цифрами: */ while (s[i] && !isdigit(s[i]))
i++;
}
}

int main( )
{
char s[N]; fgets(s, N, stdin); Print(s);
return 0;
}
```

Пример 11. Пусть SET – некоторое множество символов из таблицы символов ASCII, не содержащее символ ‘\0’. Требуется в строке s удалить все символы, принадлежащие множеству SET.

Пусть, например, SET = {‘A’, ‘B’, ‘a’, ‘b’, ‘!’}. Приведем очень эффективный алгоритм решения данной задачи. Пусть int flag[256] – логический массив, каждый индекс которого соответствует определенному символу из таблицы символов ASCII. Все элементы массива flag, индексы которых соответствуют элементам множества SET, определим единицами, а все остальные элементы – 0. Для написания алгоритма учтем пример 5 пункта 5.2 и замечание к примеру 4 дан- ного пункта.

```
#define SET "ABab!"
typedef unsigned char UCHAR;

void Del(char *s)
{
int i, j, flag[256] = {0};
/* Все элементы массива flag, индексы которых соответствуют элементам множества SET, определяем единицами:
*/
for(i = 0; SET[i]; i++)
flag[(UCHAR)SET[i]] = 1;

/* Пробегаем все символы строки, не принадлежащие множеству SET:
*/
for(j = 0; s[j] && !flag[(UCHAR)s[j]]; j++)
;
/* Переставляем на новые позиции все символы в строке s, не принадлежащие множеству SET:
*/
for(i = j; s[i]; i++)
if (!flag[(UCHAR)s[i]]) s[j++] = s[i];
s[j] = '\0';
}
```

Пример 12. Пусть s и t – некоторые строки. Требуется написать функцию, которая

определяет количество символов, одновременно входящих в обе данные строки без учета символа ‘\0’.

Сложность приведенного ниже алгоритма равна length(s)+length(t).

```
int Count(char *s, char *t)
{
int i, n, flag[256] = {0};
for(i = n = 0; s[i]; i++)
flag[(UCHAR)s[i]] = 1;
for(i = 0; t[i]; i++)
if (flag[(UCHAR)t[i]]) n++;
return n;
}
```

Пример 13. Пусть имеются строки SET_A, SET_B и s. Требуется проверить, чередуются ли в строке s элементы, принадлежащие строкам SET_A и SET_B. Например, если SET_A – строка, состоящая из латинских букв, а SET_B – строка, состоящая из цифр, то требуется проверить, чередуются ли в строке s буквы и цифры. Или если SET_A – строка, состоящая из гласных латинских букв, SET_B – строка, состоящая из согласных латинских букв, то требуется проверить, чередуются ли в строке s гласные и согласные латинские буквы. Рассмотрим очень быстрый и оптимальный алгоритм решения данной задачи, не использующий функцию strchr(), которая, как было отмечено выше, будет тормозить работу алгоритма. Заведем два целочисленных массива flag_a и flag_b, каждый из которых состоит из 256 элементов (количество элементов из таблицы символов ASCII), и пусть i-ый элемент массива flag_a (flag_b) соответствует i-ому эле- менту из таблицы символов. Напротив каждого элемента в массиве flag_a, которые входят в строку SET_A, выставим флаг, равный 1, а все остальные элементы в массиве flag_a определим нулевыми значениями:

```
int flag_a[256] = {0}; for(i = 0; SET_A[i]; i++)
flag_a[SET_A[i]] = 1;
```

Точно так же поступим и с массивом flag_b. После этого очень легко и, самое главное, очень быстро можно проверить, принадлежит ли один из символов s[i-1] и s[i] строки s строке SET_A, а другой – строке SET_B:

```
if (flag_a[s[i-1]] && flag_b[s[i]] || flag_b[s[i-1]] && flag_a[s[i]])
```

Алгоритм решения данной задачи теперь будет иметь такой вид:

```
#define SET_A "ABCD" #define SET_B  "xyz"

int Check(char *s)
{
int i, flag_a[256] = {0}, flag_b[256] ={0}; if (s[0] == 0)
return 1;
                for(i = 0; SET_A[i]; i++) flag_a[SET_A[i]] = 1; for(i = 0; SET_B[i]; i++)
                                                                flag_b[SET_B[i]] = 1;
i = 1;
while(s[i] && (flag_a[s[i-1]] && flag_b[s[i]] || flag_b[s[i-1]] && flag_a[s[i]]))
i++;
return s[i] == '\0';
}
```

Рассмотрим также частный случай данной задачи. Пусть требуется проверить, чередуются ли в строке s гласные и согласные латинские буквы. В этом случае, конечно, можно в строку SET_A записать все гласные латинские буквы, а в строку SET_B – согласные и применить функцию Check(), но все же в этом случае алгоритм можно упростить и немного ускорить.

Пусть VOWEL – строка, содержащая все гласные латинские буквы, а flag[256] – массив, в котором каждой гласной букве соответствует единица, а всем остальным символам – 0. Поэтому если сим- волы s[i-1] и s[i] являются латинскими буквами и одна из них гласная, а другая согласная, то flag[s[i-1]]+flag[s[i]]==1. Используем это свойство в следующем алгоритме:

```
#define VOWEL "AEIOUYaeiouy" int Check(char *s)
{
int i, flag[256] = {0}; if (s[0] == 0)
return 1;
if (!isalpha(s[0])) return 0;
for(i = 0; VOWEL[i]; i++) flag[VOWEL[i]] = 1;
i = 1;
while(s[i] && isalpha(s[i]) && (flag[s[i-1]] + flag[s[i]] == 1)) i++;
return s[i] == '\0';
}
```

Пример 14. Пусть s – некоторая строка, c – некоторый символ. Требуется в строке s каждую серию подряд идущих символов c заменить одним символом c.

```
void Zamena(char *s, char c)
{
int i, j, n; i = j = 0;
while(s[i])
{
n = 0; /* количество подряд идущих символов c */ while (s[i] && s[i] == c)
{
i++; n++;
}
if (n > 0) s[j++] = c;
while (s[i] && s[i] != c)
s[j++] = s[i++];
}
s[j] = '\0';
}

int main( )
{
char s[1024]; fgets(s, 1024, stdin); Zamena(s, 'a');
puts(s); return 0;
}
```

Пример 15. В строке s заменить каждую серию подряд идущих одинаковых символов на один символ данной серии. Например, строку s=”aabbbccd” необходимо преобразовать в s=”abcd”.

```
void Zamena(char *s)
{
int i, j;
```

```
if (s[0] == '\0')
return;
for (i = j = 1; s[i]; i++)
if (s[i] != s[i-1]) s[j++] = s[i];
s[j] = '\0';
}
```

Пример 16. Из строки s удалить все слова, имеющие в своем составе хотя бы одну цифру.

#include <stdio.h> #include <string.h> #include <ctype.h> #define DEL " .,:;\n\t" #define N 1024

```
/* проверка, присутствует ли в массиве s хотя бы одна цифра: */
int Check(char *s, int len)
{
int i;
for(i = 0; i < len && !isdigit(s[i]); i++)
;
return i >= len;
}

int main( )
{
char s[N];
int i, j, k, len; fgets(s, N, stdin); i = k = 0;
while (s[i])
{
while (s[i] && strchr(DEL, s[i])) s[k++] = s[i++];
j = i;
while (s[i] && !strchr(DEL, s[i])) i++;
len = i - j;
if (Check(s + j, len))
{
strncpy(s + k, s + j, len); k += len;
}
}
s[k] = '\0'; puts(s); return 0;
}
```

Индивидуальные задания

1. Подсчитать общее количество цифр и знаков '+', ' ', и '*', входящих во вводимую с клавиатуры строку.
2. Вводится текст из цифр и строчных латинских букв, за которым следует точка. Определить, каких букв гласных (a, e, i, o, u, y) или согласных больше в этом тексте.
3. Вводится текст, за которым следует точка. В алфавитном порядке напечатать (по разу) все строчные русские буквы, входящие в этот текст.
4. Разработать программу, производящую арифметические действия над вводимыми с клавиатуры комплексными числами.
5. Разработать программу, проверяющую, может ли ферзь за один ход перейти с одного заданного поля в другое. Вертикальные координаты поля обозначаются

буквами (a..h), горизонтальные цифрами (1..8).

6. Разработать программу, преобразующую вводимые с клавиатуры строчные буквы в прописные и наоборот, не используя встроенную функцию.

7. Разработать программу, проверяющую, является ли введенное с клавиатуры слово (строка) палиндромом (читается одинаково в обе стороны).

8. Во введенном с клавиатуры тексте подсчитать сумму встречающихся в нем целых чисел, не используя встроенную функцию.

9. Разработать программу, вычисляющую длину строки, не используя встроенную функцию.

10. Написать программу, удаляющую заданную подстроку из строки, не используя встроенную функцию.

11. Определить количество вхождений заданной подстроки в строку. Ввод строки и подстроки организовать с клавиатуры.

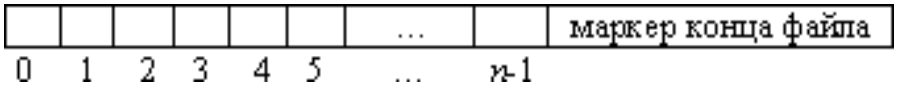
12. Строка содержит несколько слов, между соседними словами не менее одного пробела, за последним словом – точка. Выбрать все слова, имеющие нечетную длину, вывести их на экран в обратном порядке (например, слово «школа» в обратном порядке – «алокш»).

13. Ввести строку, содержащую несколько слов. Выбрать то слово, которое содержит наибольшее количество гласных русских букв.

Лабораторная работа 20.
Решение задач с использованием файлового ввода/вывода

Файл – именованная совокупность произвольных данных, размещенная на внешнем запоминающем устройстве, которая хранится, пересылается и обрабатывается как единое целое. Файл может содержать программу, числовые данные, текст, закодированное изображение и т.д.

В языке Си любой файл рассматривается как последовательный поток байтов, который оканчивается маркером конца файла:



Для начала заметим, что вводимые данные отправляются на диск или печатающее устройство не сразу. Так как запись данных на диск и их считывание с диска требуют довольно много времени, то сначала они поступают в область памяти, называемую буфером, предназначенную для временного хранения информации. Когда буфер заполняется, данные разом направляются на диск или принтер. Считанные с диска данные также порциями поступают сначала в буфер.

Для того чтобы записывать в файл данные или считывать их из файла, необходимо открыть файл с помощью функции fopen из библиотеки <stdio.h>. Открытый файл возвращает указатель (называемый файловым указателем) на структуру FILE, которая содержит информацию для работы с файлом. Данная структура содержит: адрес буфера файла; количество символов, остающихся в буфере; положение текущего символа в буфере; открыт файл для чтения или записи; были ли ошибки при работе с файлом; не встретился ли маркер конца файла.

При записи в файл или чтении из файла программа получает необходимую информацию из структуры FILE. Указатель на данную структуру определяется следующим образом:

FILE *f;

Функция fopen() возвращает указатель на файл: FILE *fopen(char *name, char *mode)

Обращение к fopen() выглядит следующим образом: f = fopen (name, mode);

Первый аргумент (name) – это строка, содержащая имя файла. Второй аргумент несет информацию о том, в каком режиме файл открывается. Ниже приведены режимы открытия файла:

Режим	Описание
"r"	Открыть файл для чтения. Этот файл должен существовать
"w"	Открыть файл для записи; если этот файл ранее существовал, его содержимое уничтожается
"a"	Открыть файл для записи (добавления) в конец файла. Создать файл, если его нет
"r+"	Открыть файл одновременно для чтения и записи. Файл должен существовать
"w+"	Открыть файл для чтения и записи. Если этот файл существует, то его содержимое уничтожается
"a+"	Открыть файл для чтения и добавления. Создать файл заново, если его нет

Попытка открыть несуществующий файл для чтения сразу обернется ошибкой. Могут иметь место и другие ошибки, например попытка чтения файла, защищенного от чтения и т.д. Если при открытии файла произошла та или иная ошибка, то функция fopen возвратит значение NULL. Например, следующие инструкции

FILE *f;

if ((f = fopen(“c:\\a.data”, "r")) == NULL) printf(“File could not be opened\n”);

означают попытку открытия файла “c:\\a.data” для чтения. Если при открытии файла произошла ошибка (файловый указатель равен NULL), тогда на экран выводится сообщение “ File could not be opened”.

Рассмотрим основные библиотечные функции (библиотека <stdio.h>) для работы с файлами.

FILE *fopen (const char *filename, const char* mode)

Функция fopen() открывает файл с заданным именем и возвращает указатель на файл или NULL, если файл не удалось открыть.

int fflush(FILE *f)

Производит дозапись всех оставшихся в буфере незаписанных данных. Для потока ввода эта функция неопределена. Функция возвращает EOF в случае возникновения ошибки, в противном случае – 0. Инструкция fflush(NULL) выполняет указанные действия для всех потоков вывода.

int fclose(FILE *f)

Закрывает файл f. Перед этим происходит запись еще незаписанных данных из буфера либо считывание еще непрочитанных из буфера данных. При закрытии потока, буферы, захваченные системой, освобождаются. Функция возвращает EOF в случае ошибки и 0 – в противном случае.

int feof(FILE *f)

Определяет, достигнут ли конец файла. При его достижении функция возвращает ненулевое значение, в противном случае возвращается 0.

int remove(const char *filename)

Удаляет файл с указанным именем. Возвращает ненулевое значение в случае

ошибки, в противном случае возвращается 0.

int rename(const char *oldname, const char* newname)

Функция rename() заменяет старое имя файла oldname новым newname; если попытка изменить имя оказалась неудачной, то воз- вращает ненулевое значение, в противном случае – 0.

int fputc(int c, FILE *f)

Записывает символ c в файл f. Функция возвращает записанный символ при успешном исходе, иначе – EOF.

int fgetc(FILE *f)

Читает очередной символ из файла f, на который указывает файловый указатель. Функция возвращает прочитанный символ при успешном исходе; если же достигнут конец файла – EOF.

int fputs(const char *s, FILE *f)

Записывает строку символов s в файл f. Возвращает количество записанных символов; в случае ошибки – EOF.

char *fgets(char *s, int n, FILE *f)

Читает строку символов из файла в массив символов s. При этом читается не более n-1 символов, прекращая чтение, если встретился символ ‘\n’. В строку s также добавляется нуль-символ ‘\0’. Функция возвращает s; если же файл исчерпан, тогда – NULL.

int fprintf(FILE *f, const char *format, argument_1, …, argument_n) Записывает символы в файл f под управлением формата. Возвращает количество записанных символов; в случае ошибки – EOF.

int fscanf(FILE *f, const char *format, &argument_1, …, &argument_n) Читает данные из файла f под управлением формата. Возвращает ко- личество преобразованных элементов; в случае ошибки – EOF. Дан- ная функция работает аналогично функции scanf() (см. п. 2.2. Фор- матный ввод-вывод).

int fseek(FILE *f, long count, int origin)

Устанавливает указатель файла f на определенный байт. Последующее чтение или запись будет проводиться с данного байта (с данной позиции). В случае двоичного файла позиция устанавливается со смещением count относительно:

- начала файла, если значение переменной origin равно SEEK_SET;
- текущей позиции, если значение переменной origin равно SEEK_CUR;
- конца файла, если значение переменной origin равно SEEK_END.

int rewind(FILE *f)

Перемещение указателя в начало файла. Данный оператор эквивалентен записи

fseek(f, 0, SEEK_SET); long ftell(FILE *f)

Возвращает текущую позицию указателя файла f; в случае ошибки функция возвращает -1.

int ferror(FILE *f)

Возвращает ненулевое значение, если при очередной операции при работе с файлом f произошла ошибка, в противном случае возвращается 0.

Текстовые файлы

Текстовый файл – последовательность строк, причем каждая строка заканчивается символом ‘\n’. Текстовый файл состоит из обычных печатных символов (в кодах ASCII), включая пробелы, символы новой строки, символы табуляции. Например, в текстовом файле целое число 123 представлено тремя символами-цифрами ‘1’, ‘2’ и ‘3’, которые в файле расположены друг за другом, образуя слово “123”.

Приведем примеры использования библиотечных функций для работы с текстовыми файлами.

Пример 1 (функция fputc()). Ниже приводится функция посимвольной записи в

файл с именем fileName, которая продолжается до тех пор, пока в стандартном потоке не встретится имитация признака конца файла EOF.

Многие операционные системы позволяют имитировать условие конца файла с помощью клавиатуры. В системе MS-DOS для этого используется комбинация клавиш Ctrl+z, после чего следует нажать клавишу Enter. В среде Windows для имитации конца файла используется комбинация клавиш Ctrl+z, а в среде UNIX – Ctrl+d.

```
int CreateTextFile(char *fileName)
{
FILE *f;
char c;
if ((f = fopen(fileName, "w")) == NULL) return 1;
while ((c = getchar()) != EOF) fputc(c, f);
fclose(f); return 0;
}
```

Пример 2 (функция fgetc()). Вывод содержимого текстового файла на экран. Функция считывает из файла по одному символу и выводит считанные символы на экран.

```
int ReadTextFile(char *fileName)
{
FILE *f;
char c;
if ((f = fopen(fileName, "r")) == NULL) return 1;
while ((c = fgetc(f)) != EOF) putchar(c);
fclose(f); return 0;
}
```

Пример 3 (функция fputs()). Построчное заполнение файла. Как и в примере 1, в следующей программе ввод строк совершается до тех пор, пока в стандартном потоке не встретится имитация признака конца файла EOF.

```
int CreateTextFile(char *fileName)
{
FILE *f;
char s[500]; int i;
if ((f = fopen(fileName, "w")) == NULL) return 1;
while (fgets(s, 500, stdin) != NULL)
fputs(s, f);/* записываем строку s в файл*/
fclose(f);
return 0;
}
```

Пример 4 (функция fgets()). Построчное чтение файла.

```
int ReadTextFile(char *fileName)
{
FILE *f;
char line[500];
if ((f = fopen(fileName, "r")) == NULL) return 1;
while (fgets(line, 500, f) != NULL) printf(line);
fclose(f); return 0;
}
```

Функции, работающие с символами и строками, предназначены только для чтения и записи текста. Если требуется записать или считать из файла данные, содержащие числовые значения, то нужно использовать функции fprintf() и fscanf().

Пример 5 (функция fprintf()). Следующая функция записывает в текстовый файл сведения о работниках некоторой организации: фамилия работника, год рождения, заработная плата. Ввод данных с клавиатуры прекращается в том случае, когда на запрос ввести очередную фамилию работника пользователь введет строку “000”.

```
int CreateFile(char *fileName)
{
FILE *f;
char name[50];/* фамилия*/
int year;/* год рождения*/
float earnings;/* заработная плата */
if ((f = fopen(fileName, "w")) == NULL)
return 1; printf("name: ");
scanf("%s", name);
while (strcmp(name, "000"))
{
printf("year: ");
scanf("%d", &year); printf("earnings: ");
scanf("%f", &earnings);
fprintf(f, "%s %d %.2f\n", name, year, earnings); printf("name: ");
scanf("%s", name);
}
fclose(f); return 0;
}
```

Пример 6 (функция fscanf()). Чтение записей из файла, содержащего сведения о работниках. Создание данного файла приводится в предыдущем примере.

```
int ReadFile(char *fileName)
{
FILE *f;
char name[50];/* фамилия*/
int year;/* год рождения*/
float earnings;/* заработная плата */ if ((f = fopen(fileName, "r")) == NULL)
return 1;
while (fscanf(f, "%s%d%f", name, &year, &earnings) == 3)
{
printf("%s %d %.2f\n", name, year, earnings);
}
fclose(f); return 0;
}
```

Пример 7 (функция fscanf()). Следующая функция выделяет все слова из текстового файла, которые разделены пробельными символами (‘ ’, ‘\t’, ‘\n’), и выводит данные слова на экран.

```
int ReadWords(char *fileName)
{
```

```
FILE *f;
char s[500];
if ((f = fopen(fileName, "r")) == NULL) return 1;
while (fscanf(f, "%s", s) == 1) puts(s);
fclose(f); return 0;
}
```

Пример 8. В текстовом файле заменить все подряд идущие символы со значением x одним символом x.

```
int Zamena(char *fname1, char *fname2, char x)
{
FILE *f, *g; char c;
int n;
if ((f = fopen(fname1, "r")) == NULL) return 1;
if ((g = fopen(fname2, "w")) == NULL)
{
fclose(f); return 1;
}
c = fgetc(f); while (c != EOF)
{
n = 0;
while (c != EOF && c == x)
{
n++;
c = fgetc(f);
}
if (n > 0) fputc(x, g);
while (c != EOF && c != x)
{
fputc(c, g); c = fgetc(f);
}
}
fclose(f);
fclose(g); return 0;
}
```

Индивидуальные задания

1. Заполнить файл последовательного доступа f целыми числами, полученными с помощью генератора случайных чисел. Получить в файле g те компоненты файла f, которые являются четными.
2. Записать в файл последовательного доступа n действительных чисел. Вычислить произведение компонентов файла и вывести на печать.
3. Заполнить файл последовательного доступа f целыми числами, полученными с помощью генератора случайных чисел. Получить в файле g все компоненты файла f, которые делятся на 3.
4. Записать в файл последовательного доступа n целых чисел, полученных с помощью генератора случайных чисел. Подсчитать количество пар противоположных чисел среди компонентов этого файла.
5. Заполнить файл последовательного доступа f целыми числами, полученными с помощью генератора случайных чисел. Из файла f получить файл g, исключив повторные вхождения чисел. Вывести файл g на печать.

6. Записать в файл последовательного доступа n произвольных натуральных чисел. Переписать в другой файл последовательного доступа те элементы, которые кратны 2. Вывести полученный файл на печать.

7. Заполнить файл последовательного доступа n действительными числами, полученными с помощью датчика случайных чисел. Найти сумму минимального и максимального элементов этого файла.

8. 8 Записать в файл f последовательного доступа n натуральных чисел. Получить в другом файле последовательного доступа все компоненты файла f, кроме тех, которые кратны 4. Вывести полученный файл на печать.

9. Дан текстовый файл f, содержащий сведения о сотрудниках учреждения, записанные по следующему образцу: Фамилия Имя Отчество. Записать эти сведения в файле g, используя образцы: 1) Имя Отчество Фамилия; 2) Фамилия И.О. Выбор варианта преобразования задается в начале работы программы

10. Заполнить файл последовательного доступа f целыми числами, полученными с помощью генератора случайных чисел. Получить в файле g все компоненты файла f, которые делятся на 4

11. Записать в файл последовательного доступа n произвольных натуральных чисел. Переписать в другой файл последовательного доступа те элементы, которые кратны 5. Вывести полученный файл на печать.

12. Записать в файл последовательного доступа n действительных чисел. Вычислить сумму компонентов файла и вывести на печать.

13. Заполнить файл последовательного доступа f целыми числами, полученными с помощью генератора случайных чисел. Получить в файле g все компоненты файла f, которые делятся на 2.

14. Заполнить файл последовательного доступа f целыми числами, полученными с помощью генератора случайных чисел. Получить в файле g те компоненты файла f, которые являются нечетными.

15. Заполнить файл последовательного доступа f целыми числами, полученными с помощью генератора случайных чисел. Получить в файле g все компоненты файла f, которые делятся на 3.

Лабораторная работа № 21
Сравнительный анализ стратегий разработки программного обеспечения

Цель. Изучить стратегии разработки программного обеспечения, выполнить анализ и сравнение.

Теоретическая часть

Базовые стратегии разработки программных средств и систем

На начальном этапе развития вычислительной техники ПС разрабатывались по принципу «кодирование – устранение ошибок». Модель такого процесса разработки ПС иллюстрирует рисунок 1.

Очевидно, что недостатками данной модели являются:

- неструктурированность процесса разработки ПС;
- ориентация на индивидуальные знания и умения программиста;
- сложность управления и планирования проекта;
- большая длительность и стоимость разработки;
- низкое качество программных продуктов;
- высокий уровень рисков проекта.

Для устранения или сокращения вышеназванных недостатков созданы и широко используются **три базовые стратегии** разработки ПО: каскадная, инкрементная, эволюционная.

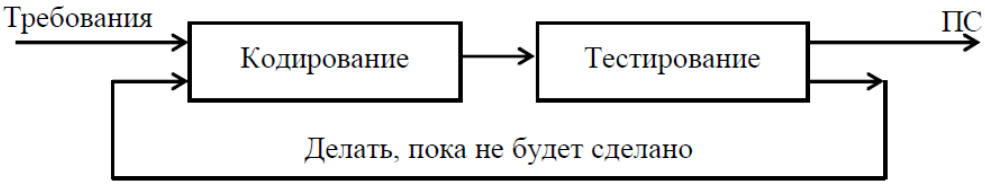


Рисунок 1 – Модель «Делать, пока не будет сделано»

Некоторые характеристики каскадной, инкрементной и эволюционной стратегий разработки ПС и предъявляемые к ним требования приведены в стандарте **ГОСТ Р ИСО/МЭК ТО 15271–2002 – Информационная технология – Руководство по применению ГОСТ Р ИСО/МЭК 12207 (Процессы жизненного цикла программных средств)**.

Выбор той или иной стратегии определяется характеристиками: проекта, требований к продукту, команды разработчиков, команды пользователей. Три базовые стратегии могут быть реализованы с помощью различных моделей ЖЦ.

Каскадная стратегия разработки программных средств и систем

Каскадная стратегия представляет собой однократный проход этапов разработки. Данная стратегия основана на полном определении всех требований к разрабатываемому программному средству или системе в начале процесса разработки. Каждый этап разработки начинается после завершения предыдущего этапа. Возврат к уже выполненным этапам не предусматривается. Промежуточные продукты разработки в качестве версии программного средства (системы) не распространяются. Представителями моделей, реализующих каскадную стратегию, являются каскадная и V-образная модели.

Инкрементная стратегия разработки программных средств и систем

Инкрементная стратегия представляет собой многократный проход этапов разработки с запланированным улучшением результата.

Данная стратегия основана на полном определении всех требований к разрабатываемому программному средству (системе) в начале процесса разработки. Однако полный набор требований реализуется постепенно в соответствии с планом в последовательных циклах разработки.

Результат каждого цикла называется **инкрементом**.

Эволюционная стратегия разработки программных средств и систем

Эволюционная стратегия представляет собой многократный проход этапов разработки. Данная стратегия основана на частичном определении требований к разрабатываемому программному средству или системе в начале процесса разработки. Требования постепенно уточняются в последовательных циклах разработки. Результат каждого цикла разработки обычно представляет собой очередную поставляемую версию программного средства или системы.

Практическая часть

Задание 1. Запишите определение каскадной стратегии разработки ПО.

Задание 2. Выделите основные достоинства и недостатки каскадной стратегии.

Задание 3. Приведите область применения каскадной модели.

Задание 4. Запишите определение инкрементной стратегии разработки ПО.

Задание 5. Запишите основные достоинства и недостатки инкрементной стратегии.

Задание 6. Выделите область применения инкрементной модели.

Задание 7. Запишите определение эволюционной стратегии разработки ПО.

Задание 8. Выделите основные достоинства и недостатки эволюционной стратегии.

Задание 9. Приведите область применения эволюционной модели.

Задание 10. Выполните сравнение стратегий и запишите результаты в сводную таблицу.

Характеристика проекта	**Стратегия**		
	Каскадная	**Инкрементная**	**Спиральная**
Новизна разработки и обеспеченность ресурсами			
Масштаб проекта			
Срок выполнения проекта			
Заключение отдельных договоров на отдельные версии			
Определение основных требований в начале проекта			
Изменение требований по мере развития проекта			
Разработка итерациями			
Распространение промежуточного ПС			

Задание 11. Результаты выполнения практического задания запишите в отчет.

Контрольные вопросы

8. Дайте определение понятию «программная инженерия».
9. Дайте определение понятию «жизненный цикл».
10. Что такое макетирование? Изобразите схему данного процесса.
11. Выберите подходящий процесс разработки для перечисленных ниже программных приложений. Обоснуйте свой выбор.

а) Система решения квадратных уравнений.

б) Система определения оценки по результатам ответа на три экзаменационных вопроса.

в) Информационная система института.

Лабораторная работа 22.
Работа с ЕСКД и ГОСТ

Каждое требование к программному обеспечению, представленное в спецификации требований к программному обеспечению (СТПО), является определением некоторой возможности программного обеспечения, которая должна быть реализована. Стандарт СТ РК 1090-2002 «ЕДИНАЯ СИСТЕМА ПРОГРАММНОЙ ДОКУМЕНТАЦИИ. Спецификация требований к программному обеспечению» распространяется на спецификацию требований к программному обеспечению и относится к основополагающим общетехническим стандартам. Стандарт устанавливает необходимое содержание и характеристики СТПО, методы, используемые для описания требований к программному обеспечению. Стандарт применяется для спецификации требований к вновь разрабатываемому программному обеспечению.

В результате анализа требований к будущему программному продукту получают его спецификации: выполняют декомпозицию и содержательную постановку решаемых задач, уточняют их взаимодействие и определяют эксплуатационные ограничения.

Спецификациями называют точное формализованное описание функций и ограничений разрабатываемого программного продукта.

Функциональная спецификация в разработке программного обеспечения — это документ, описывающий требуемые характеристики системы (функциональность).

Эксплуатационная спецификация — это описание правил использования программного обеспечения. Эксплуатационные спецификации определяют требования к техническим средствам, надежности, безопасности и т.д. Совокупность спецификаций представляет собой общую логическую модель проектируемого программного обеспечения.

В целом в процессе определения спецификаций строят общую модель предметной области как некоторой части реального мира, с которой будет тем или иным способом взаимодействовать разрабатываемое программное обеспечение, и конкретизируют его основные функции.

Функциональные спецификации должны быть полными и точными. Требование полноты означает, что спецификации должны содержать всю существенную информацию, чтобы ничего важного не было упущено, и не должны содержать несущественной информации, например деталей реализации, чтобы не препятствовать разработчику в выборе наиболее эффективных решений. Требование точности означает, что спецификации должны однозначно восприниматься как заказчиком, так и разработчиком.

Функциональная спецификация состоит из трех частей.

- 1. Описание внешней информационной среды, с которой будет взаимодействовать разрабатываемое программное обеспечение. Должны быть определены все способы взаимодействия с внешней средой, все информационные объекты, к которым будет применяться разрабатываемое ПО, а также существенные связи между этими информационными объектами.
- 2. Определение функций программного обеспечения, определенных на множестве состояний этой информационной среды. Вводятся обозначения всех определяемых функций, специфицируются их входные данные и результаты выполнения с указанием типов данных и заданий всех ограничений, которым должны удовлетворять эти данные и результаты. Определяется содержание каждой из этих функций.
- 3. Описание исключительных ситуаций, если таковые могут возникнуть при выполнении программ, и реакций на эти ситуации, которые должны обеспечить соответствующие программы. Должны быть перечислены все существенные случаи, когда программное обеспечение не сможет нормально выполнить ту или иную свою функцию. Для каждого такого случая должна быть определена реакция программы.

Все функциональные спецификации разрабатываемого программного обеспечения описывают перечень функций и состав обрабатываемых данных. Они различаются только системой приоритетов

(акцентов), которая используется разработчиком в процессе анализа требований и определения спецификаций.

Утверждение требований исполнителем и заказчиком определяет момент достижения соглашения между ними по всем пунктам спецификаций требований. Заказчик может потребовать предоставить ему прототип или иной пример, иллюстрирующий проверку некоторых требований к программному обеспечению, может определить множество проверочных примеров или многократно изменить параметры отдельных требований к ПП. В этом случае такие пожелания заказчика необходимо оформлять в приложении к спецификациям требований.

При составлении спецификаций не следует употреблять слова и словосочетания, допускающие неоднозначное толкование. Точные спецификации разрабатываемого программного обеспечения можно определить, только разработав некоторую формальную модель этого программного обеспечения.

Задание 1: изучите предлагаемый стандарт и ответьте на контрольные вопросы.

Контрольные вопросы

1. Как могут быть представлены требования?
2. Для реализации данного подхода «Модели спецификаций» какие используются типы моделей?
3. Что из себя представляет математическая модель?
4. функциональная модель?
5. Что из себя представляет временная модель?
6. Что должно включать определение модели спецификации?
7. Для чего нужна аннотация?
8. Какие основные вопросы, рассматриваемые при разработке требований?

Задание 2: напишите какое приложение или программный продукт вы хотели бы разработать. Определите требования, которые вы предъявили к программному продукту.

Пример.

Разработать техническое задание на программный продукт, предназначенный для наглядной демонстрации школьникам графиков функций одного аргумента y =/(x).

Разрабатываемая программа должна рассчитывать таблицу значений и строить график функций на заданном отрезке по заданной формуле и менять шаг аргумента и границы отрезка. Кроме этого, программа должна запоминать введенные формулы.

1. Введение

Настоящее техническое задание распространяется на разработку программы сортировки одномерного массива методами пузырька, прямого выбора, Шелла и быстрой сортировки, предназначенной для использования школьниками старших классов при изучении курса школьной информатики.

2. Основание для разработки

2.1. Программа разрабатывается на основе учебного плана кафедры «Информатика и программное обеспечение вычислительных систем».

2.2. Наименование работы:

«Программа сортировки одномерного массива».

2.3. Исполнитель: компания BestSoft.

2.4. Соисполнители: нет.

3. Назначение

Программа предназначена для использования школьниками

при изучении темы «Обработка одномерных массивов» в курсе «Информатика».

4. Требования к программе или программному изделию

4.1. Требования к функциональным характеристикам

4.1.1. Программа должна обеспечивать возможность выполнения следующих функций:

- ввод размера массива и самого массива;
- хранение массива в памяти;
- выбор метода сортировки;
- вывод текстового описания метода сортировки;

• вывод результата сортировки.

4.1.2. Исходные данные:

• размер массива, заданный целым числом;

• массив.

4.1.3. Организация входных и выходных данных

Входные данные поступают с клавиатуры.

Выходные данные отображаются на экране и при необходимости выводятся на печать.

4.2. Требования к надежности

Предусмотреть контроль вводимой информации.

Предусмотреть блокировку некорректных действий пользователя при работе с системой.

4.3. Требования к составу и параметрам технических средств.

Система должна работать на IBM-совместимых персональных компьютерах.

Минимальная конфигурация:

• тип процессора. Pentium и выше;

• объем оперативного запоминающего устройства 32 Мб и более;

• объем свободного места на жестком диске 40 Мб.

Рекомендуемая конфигурация:

• тип процессора. Pentium II 400;

• объем оперативного запоминающего устройства 128 Мб;

• объем свободного места на жестком диске 60 Мб.

4.4. Требования к программной совместимости.

Программа должна работать под управлением семейства операционных систем Win 32 (Windows 95/98/2000/ME/XP и т. п.).

5. Требования к программной документации

5.1. Разрабатываемые программные модули должны быть самодокументированы, т. е. тексты программ должны содержать все необходимые комментарии.

5.2. Разрабатываемая программа должна включать справочную информацию о работе программы, описания методов сортировки и подсказки учащимся.

5.3. В состав сопровождающей документации должны входить:

5.3.1. Пояснительная записка на пяти листах, содержащая описание разработки.

5.3.2. Руководство пользователя.

Лабораторная работа 23.
Ручная отладка программного обеспечения.

Цель работы:

Изучить методы тестирования логики программы.

Составить тесты для проверки правильности работы программы.

Теоретический материал

Тест — это набор контрольных входных данных совместно с ожидаемыми результатами. Ключевой вопрос — полнота тестирования: какое количество каких тестов гарантирует, возможно, более полную проверку программы?

Исчерпывающая проверка на всем множестве входных данных недостижима.

Пример: программа, вычисляющая функцию двух переменных:

Y=f(X, Z).

Если X, Y, Z — вещественные, то полное число тестов $(2^{32})^2 = 2^{64} = 10^{31}$ Если на каждый тест тратить 1 мс, то 2^{64} мс = = 800 млн лет.

Следовательно:

• в любой нетривиальной программе на любой стадии ее готовности содержатся необнаруженные ошибки;

• тестирование — технико-экономическая проблема, основанная на компромиссе время — полнота. Поэтому нужно стремиться к возможно меньшему количеству хороших тестов с желательными свойствами.

Существуют разные подходы к проектированию тестов.

Первый состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей либо спецификаций сопряжения модуля с другими модулями, программа при этом рассматривается как «черный ящик». Смысл теста заключается в том, чтобы проверить, соответствует ли программа внешним спецификациям. При этом содержание модуля не имеет значения. Такой подход получил название —- стратегия **«черного ящика»**.

Второй подход — стратегия **«белого ящика»**, основан на анализе логики программы. При таком подходе тестирование заключается в проверке каждого пути, каждой ветви алгоритма.

При этом внешняя спецификация во внимание не принимается.

Ни один из этих подходов не является оптимальным.

Реализация тестирования методом «черного ящика» сводится к проверке всех возможных комбинаций входных данных.

Невозможно протестировать программу, подавая на вход бесконечное множество значений, поэтому ограничиваются определенным набором данных. При этом исходят из максимальной отдачи теста по сравнению с затратами на его создание. Она измеряется вероятностью того, что тест выявит ошибки, если они имеются в программе. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста.

Тестирование методом «белого ящика» также не дает 100%-ной гарантии того, что модуль не содержит ошибок. Даже если предположить, что выполнены тесты для всех ветвей алгоритма, нельзя с полной уверенностью утверждать, что программа соответствует ее спецификациям. Например, если требовалось написать программу для вычисления кубического корня, а программа фактически вычисляет корень квадратный, то реализация будет совершенно неправильной, даже если проверить все пути.

Вторая проблема — отсутствующие пути. Если программа реализует спецификации не полностью (например, отсутствует такая специализированная функция, как проверка на отрицательное значение входных данных программы вычисления квадратного корня), никакое тестирование существующих путей не выявит такой ошибки.

И наконец, проблема зависимости результатов тестирования от входных данных. Одни данные будут давать правильные результаты, а другие нет. Например, если для определения равенства трех чисел программируется выражение вида: if ((A + B+C)/3 = =A), то оно будет верным не для всех значений A, B и C (ошибка возникает в том случае, когда из двух значений B или C одно больше, а другое на столько же меньше A). Если концентрировать внимание только на тестировании путей, нет гарантии, что эта ошибка будет выявлена.

Таким образом, полное тестирование программы невозможно, т. е. никакое тестирование не гарантирует полное отсутствие ошибок в программе. Поэтому необходимо проектировать тесты таким образом, чтобы увеличить вероятность обнаружения ошибки в программе.

Стратегия «белого ящика»

Существуют следующие методы тестирования по принципу «белого ящика»:

• покрытие операторов;
• покрытие решений;
• покрытие условий;
• покрытие решений/условий;
• комбинаторное покрытие условий.

Метод покрытия операторов

Целью этого метода тестирования является выполнение каждого оператора программы хотя бы один раз.

Пример.

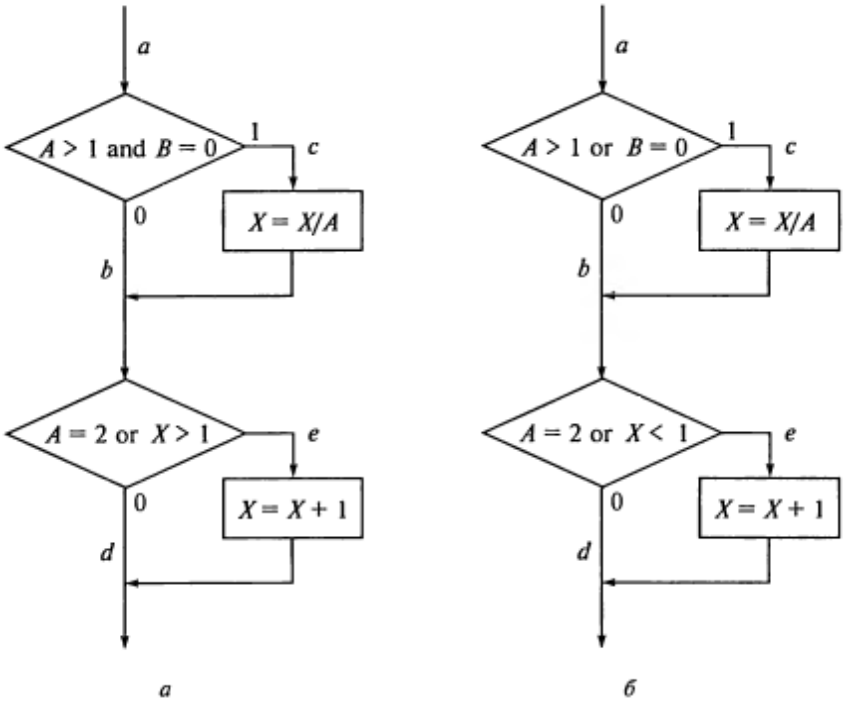
Если для тестирования задать значения переменных A = 2, B=0, X=3, будет реализован путь ace, т. е. каждый оператор программы выполнится один раз (рис. 1, а). Но если внести в алгоритм ошибки — заменить в первом условии and на or, а во втором X> 1 на X< 1 (рис. 1, б), ни одна ошибка не будет обнаружена (табл..1). Кроме того, путь abd вообще не будет охвачен тестом, и если в нем есть ошибка, она также не будет обнаружена.

В табл. .1 ожидаемый результат определяется по блок-схеме на рис. 1, а, а фактический — по рис. 1, б.

Как видно из этой таблицы, ни одна из внесенных в алгоритм ошибок не будет обнаружена.

Таблица 1 Результат тестирования методом покрытия операторов

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A = 2,\ B = 0,\ X = 3$	$X = 2{,}5$	$X = 2{,}5$	Неуспешно



а б

Рис. 1. Пример алгоритма программы:
а — правильный; б — с ошибкой

Метод покрытия решений (покрытия переходов)

Согласно методу покрытия решений каждое направление перехода должно быть реализовано, по крайней мере, один раз.

Этот метод включает в себя критерий покрытия операторов, так как при выполнении всех направлений переходов выполнятся все операторы, находящиеся на этих направлениях.

Для программы, приведенной на рис. 1, покрытие решений может быть выполнено двумя тестами, покрывающими пути {ace, abd), либо {acd, abe).

Для этого выберем следующие исходные данные:

{A = 3, B = 0, X=3) — в первом случае и {A = 2, B= 1, X= 1} — во втором.

Однако путь, где X не меняется, будет проверен с вероятностью 50 %: если во втором условии вместо условия X> 1 записано X< 1, то ошибка не будет обнаружена двумя тестами.

Результаты тестирования приведены в табл. 2.

Таблица 2. Результат тестирования методом покрытия решений

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A = 3,\ B = 0,\ X = 3$	$X = 1$	$X = 1$	Неуспешно
$A = 2,\ B = 1,\ X = 1$	$X = 2$	$X = 1{,}5$	Успешно

Метод покрытия условий

Этот метод может дать лучшие результаты по сравнению с предыдущими. В соответствии с методом покрытия условий записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись, по крайней мере, один раз.

В рассматриваемом примере имеем четыре условия:

{A>1, B= 0}, {A = 2, X> 1}. Следовательно, требуется достаточное число тестов, такое, чтобы реализовать ситуации, где A > 1, A < =1, B= 0, B≠ 0 в точке a и A = 2, A≠ 2, X> 1 и X<= 1 в точке b.

Тесты, удовлетворяющие критерию покрытия условий (табл. Л5.3), и соответствующие им пути:

а) A= 2, B=0, X=4 ace;

б) A= 1, B=l, X=0 abd.

Таблица 3. Результаты тестирования методом покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A = 2,\ B = 0,\ X = 4$	$X = 3$	$X = 3$	Неуспешно
$A = 1,\ B = 1,\ X = 0$	$X = 0$	$X = 1$	Успешно

Метод покрытия решений/условий

Критерий покрытия решений/условий требует такого достаточного набора тестов, чтобы все возможные результаты каждого условия выполнялись по крайней мере один раз, все результаты каждого решения выполнялись по крайней мере один раз и, кроме того, каждой точке входа передавалось управление по крайней мере один раз.

Недостатки метода:

• не всегда можно проверить все условия;

• невозможно проверить условия, которые скрыты другими условиями;

• недостаточная чувствительность к ошибкам в логических выражениях.

Так, в рассматриваемом примере два теста метода покрытия условий

а) A = 2, B = 0, X=4 ace;

б) A = 1, B = 1, X=0 abd

отвечают и критерию покрытия решений/условий. Это является следствием того, что одни условия приведенных решений скрывают другие условия в этих решениях. Так, если условие A> 1 будет ложным, транслятор может не проверять условия B=0, поскольку при любом результате условия B=0 результат решения ((A> 1)&(B=0)) примет

значение ложь. То есть в варианте на рис. 1 не все результаты всех условий выполнятся в процессе тестирования.

Рассмотрим реализацию того же примера на рис. 2.

Наиболее полное покрытие тестами в этом случае осуществляется так, чтобы выполнялись все возможные результаты каждого простого решения. Для этого нужно покрыть пути aceg (тест A= 2, B=0, X=4), acdfh (тест A = 3, B = 1, X=0), abfh (тест A = 0, B = 0, X= 0), abfi (тест A = 0, B=0, X= 2).

Протестировав алгоритм на рис. 2, нетрудно убедиться в том, что критерии покрытия условий и критерии покрытия решений/условий недостаточно чувствительны к ошибкам в логических выражениях.

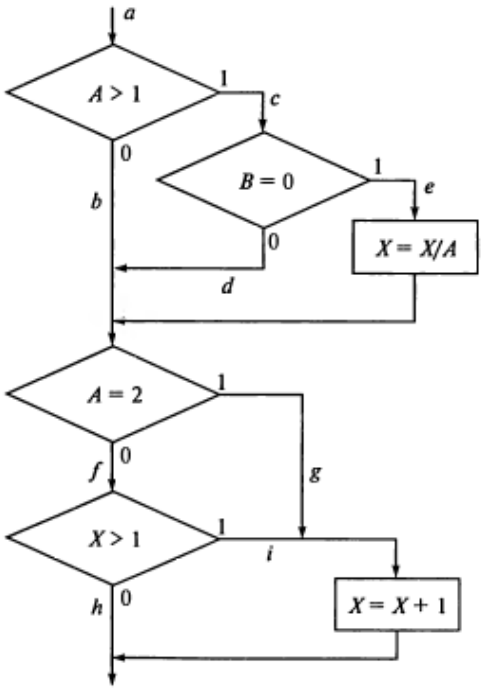


Рис. 2. Пример алгоритма программы

if (A>1 && B==0) X=X/A; if (A==2 !! X>1) X=X+1; Правильная программа

if (A>1 !! B==0) X=X/A; if (A==2 !! X>1) X=X+1; Программа с ошибкой

Метод комбинаторного покрытия условий

Критерий комбинаторного покрытия условий удовлетворяет также и критериям покрытия решений, покрытия условий и покрытия решений/условий.

Этот метод требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении выполнялись по крайней мере один раз. По этому критерию в рассматриваемом примере должны быть покрыты тестами следующие восемь комбинаций:

1. A> 1, B=0. 5.A = 2, X> 1.
2. A>1, B ≠ 0. 6.A = 2, X<=1
3. A<=1, B=0. 7.A≠ 2, X>1.
4. A<=1, B≠ 0. 8.A≠ 2, X< 1.

Для того чтобы протестировать эти комбинации, необязательно использовать все 8 тестов. Фактически они могут быть покрыты четырьмя тестами (табл. 4):

• A = 2, B= 0, X= 4 {покрывает 1, 5};
• A = 2, B= 1, X- 1 {покрывает 2, 6};
• A = 0,5, B=0, X= 2 {покрывает 3, 7};
• A = 1, B= 0, X= 1 {покрывает 4, 8}.

Таблица 4. Результаты тестирования методом комбинаторного покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A=2,\ B=0,\ X=4$	$X=3$	$X=3$	Неуспешно
$A=2,\ B=1,\ X=1$	$X=2$	$X=1{,}5$	Успешно
$A=0{,}5,\ B=0,\ X=2$	$X=3$	$X=5$	Успешно
$A=1,\ B=0,\ X=1$	$X=1$	$X=1$	Неуспешно

Теоретическая часть

Отладка – это процесс локализации и исправления ошибок, обнаруженных при тестировании программного обеспечения.

Локализацией называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо определить ее причину, т.е. определить оператор или фрагмент, содержащие ошибку. Причины ошибок могут быть как очевидны, так и очень глубоко скрыты. В целом сложность отладки обусловлена следующими причинами:

– требует от программиста глубоких знаний специфики управления используемыми техническими средствами, операционной системы, среды и языка программирования, реализуемых процессов, природы и специфики различных ошибок, методик отладки и соответствующих программных средств;

– психологически дискомфортна, так как необходимо искать собственные ошибки и, как правило, в условиях ограниченного времени;

– возможно взаимовлияние ошибок в разных частях программы, например, за счет затирания области памяти одного модуля другим из-за ошибок адресации;

– отсутствуют четко сформулированные методики отладки.

Классификация ошибок

В соответствии с этапом обработки, на котором появляются ошибки, различают:

– синтаксические ошибки – ошибки, фиксируемые компилятором (транслятором, интерпретатором) при выполнении синтаксического и частично семантического анализа программы;

– ошибки компоновки – ошибки, обнаруженные компоновщиком (редактором связей) при объединении модулей программы;

– ошибки выполнения – ошибки, обнаруженные операционной системой, аппаратными средствами или пользователем при выполнении программы.

Методы отладки программного обеспечения

Отладка программы в любом случае предполагает обдумывание и логическое осмысление всей имеющейся информации об ошибке. Большинство ошибок можно обнаружить по косвенным признакам посредством тщательного анализа текстов программ и результатов тестирования без получения дополнительной информации. При этом используют различные методы:

– ручного тестирования;

– индукции;

– дедукции;

– обратного прослеживания.

Метод ручного тестирования

Это – самый простой и естественный способ данной группы. При обнаружении ошибки необходимо выполнить тестируемую программу вручную, используя тестовый набор, при работе с которыми была обнаружена ошибка. Метод очень эффективен, но не применим для больших программ, программ со сложными вычислениями и в тех случаях, когда ошибка связана с неверным представлением программиста о выполнении некоторых операций. Данный метод часто используют как составную часть других методов отладки.

Общая методика отладки программных продуктов, написанных для выполнения в операционных системах MS DOS и Win32:

1 этап – изучение проявления ошибки;

2 этап – определение локализации ошибки;

3 этап – определение причины ошибки;

4 этап – исправление ошибки;

5 этап – повторное тестирование.

Процесс отладки можно существенно упростить, если следовать основным рекомендациям структурного подхода к программированию:

−программу наращивать «сверху-вниз», от интерфейса к обрабатывающим подпрограммам, тестируя ее по ходу добавления подпрограмм;

−выводить пользователю вводимые им данные для контроля и проверять их на допустимость сразу после ввода;

−предусматривать вывод основных данных во всех узловых точках алгоритма (ветвлениях, вызовах подпрограмм).

Спецификация программы, программная спецификация (program specification) - точная и полная формулировка определенной задачи или группы задач, содержащая сведения, необходимые для построения алгоритма их решения. Содержит описание результата, который должен быть достигнут с помощью конкретной программы, а также действий, выполняемых программой для достижения конечного результата без упоминания того, как указанный результат достигается

Практическая часть

Задание 12. Запишите вариант в отчет.

Задание 13. Согласно поставленной задаче выполните ручную отладку:

– Запишите алгоритм программы;
– Составить тесты для тестирования.
– Рассчитать результат вручную или на ПК.
– Выполните отладку логики программы;
– Составьте тестовые наборы для проверки функционала системы.

Задание 14. Результаты выполнения практического задания запишите в отчет.

Контрольные вопросы

12. В чем заключается ручная отладка ПО?
13. На каком этапе проводится ручная отладка?
14. Опишите методы отладки.
15. В чем состоит суть тестирования?
16. Методы тестирования «белого ящика».
17. Как определить количество тестов?
18. Какова эффективность тестирования логических выражений разными типами тестов?

Варианты заданий

Составить программу, реализующие линейный и разветвляющийся алгоритмы, которые размещены на разных вкладках окна формы. В качестве f(x) использовать по

выбору: cos(x) или x^2 или e^x.

1. $t = \frac{2\,cos\left(x-\frac{\pi}{6}\right)}{0.5+sin^2 y}\left(1+\frac{z^2}{3-\frac{z^2}{5}}\right)$.

2. $u = \frac{\sqrt[3]{8+|x-y|^2+1}}{x^2+y^2+2} - e^{|x-y|} * (tg^2 z + 1)^x$.

3. $v = \frac{1+sin^2(x+y)}{\left|x-\frac{2y}{1+x^2y^2}\right|} * x^{|y|} + cos^2\left(arctg\frac{1}{z}\right)$.

4. $w = |cos\,x - cos\,y|^{1+2\,sin^2 y} * \left(1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4}\right)$.

5. $\alpha = ln\left(y^{-\sqrt{|x|}}\right) * \left(x - \frac{y}{2}\right) + sin^2\,arctg(z)$.

6. $\beta = \sqrt{10\left(\sqrt[3]{x} + x^{y+2}\right)} * (arcsin^2 z - |x-y|)$.

7. $\gamma = 5arctg(x) - \frac{1}{4}arccos(x) * \frac{x+3|x-y|+x^2}{|x-y|z+x^2}$.

8. $a = \begin{cases} ln(y+2) + f(x), x/y > 0 \\ ln|y| - tg\left(f(x)\right), x/y < 0 \\ f(x) * y^3, \quad \text{иначе} \end{cases}$ 9. $b = \begin{cases} ln|y + f(x)| +, 3 < xy < 8 \\ cos\left(f(x)\right) - y, \quad xy > 12 \\ sh(f(x) + cs(y)), \quad \text{иначе} \end{cases}$

10. $c = \begin{cases} f^3(x) + ctg(y), \quad xy > 12 \\ sh\left(f^3(x)\right) + y^2, \; xy < 7 \\ cos\left(x - f^3(x)\right), \; \text{иначе} \end{cases}$ 11. $d = \begin{cases} ctg(y) + f(x), \quad x/y > 0 \\ ln|y| + tg\left(f(x)\right), x/y < 0 \\ f(x) * y^3, \quad \text{иначе} \end{cases}$

12. $k = \begin{cases} (f(x)+y)^2, \; 4 > xy > 1 \\ f(x) * tg(y), \; 8 < xy < 10 \\ f(x) + y, \quad \text{иначе} \end{cases}$ 13. $l = \begin{cases} f^2(x) + arctg\left(f(x)\right), \quad 1 \le x < 5 \\ \left(y - f(x)\right)^2 + arctg\left(f(x)\right), \; y > x \\ \left(y + f(x)\right)^3 + 0.5, \quad \text{иначе} \end{cases}$

14. $m = \begin{cases} f^3(x) + sin(y), \quad xy < 5 \\ ch\left(f^3(x)\right) + y^2, \; xy > 7 \\ cos\left(x + f^3(x)\right), \; \text{иначе} \end{cases}$ 15. $l = \begin{cases} e^{f(x)-|y|}, \quad 0.5 < xy < 5 \\ \sqrt{|f(x)+y|}, \; 0.1 < xy < 0.5 \\ 2f^2(x), \quad \text{иначе} \end{cases}$

Лабораторная работа 24
Оценочное тестирование программного продукта.

Цель работы: изучить классификацию видов тестирования,практически закрепить эти знания путем генерации тестов различных видов, научиться планировать тестовые активности в зависимости от специфики поставляемой на тестирование функциональности.

Теоретические сведения

Тестирование – процесс, направленный на оценку корректности, полноты и

качества разработанного программного обеспечения.

Тестирование можно классифицировать по очень большому количеству признаков. Далее приведен обобщенный список видов тестирования по различным основаниям.

Типы тестов по покрытию (по глубине)

Smoke test – тестирование системы для определения корректной работыбазовых функций программы в целом, без углубления в детали. При проведении теста определяется пригодность сборки для дальнейшего тестирования.

Minimal Acceptance Test (MAT, Positive test): тестирование системы или ее части только на валидных данных (валидные данные – это данные, которые необходимо использовать для корректной работы модуля/функции). При тестировании проверяется правильной работы всех функций и модулей свалидными данными.

Для крупных и сложных приложений используется ограниченный набор сценариев и функций.

Acceptance Test (AT): полное тестирование системы или ее части как на корректных, так и на некорректных данных/сценариях. Вид теста, направленный на подтверждение того, что приложение может использоваться по назначению при любых условиях.

Тест на этом уровне покрывает все возможные сценарии тестирования: проверку работоспособности модулей при вводе корректных значений; проверку при вводе некорректных значений; использование форматов данных отличных от тех, которые указаны в требованиях; проверку исключительных ситуаций, сообщений об ошибках; тестирование на различных комбинациях входных параметров; проверку всех классов эквивалентности; тестирование граничных значений интервалов; сценарии не предусмотренные спецификацией и т.д.

Тестовые активности (типы тестов по покрытию (по ширине)):

Defect Validation – проверка результата исправления дефектов. Включает в себя проверку на воспроизводимость дефектов, которые были

исправлены в новой сборке продукта, а также проверку того, что исправление не повлияло на ранее работавшую функциональность

New Feature Test (NFT, AT of NF) – определение качества поставленной на тестирование новой функциональности, которая ранее не тестировалась. Данный тип тестирования включает в себя: проведение полного теста (AT) непосредственно новой функциональности; тестирование новой функциональности на соответствие документации; проверку всевозможных взаимодействий ранее реализованной функциональности с новыми модулями и функциями.

Regression testing (регрессионное тестирование) – проводится с целью оценки качества ранее реализованной функциональности. Включает в себя проверку стабильности ранее реализованной функциональности после внесения изменений, например добавления новой функциональности, исправление дефектов, оптимизация кода, разворачивание приложения на новом окружении. Регрессионное тестирование может быть проведено науровне Smoke, MAT или AT.

Типы тестов по знанию коду

Черный ящик – тестирование системы, функциональное илинефункциональное, без знания внутренней структуры и компонентов системы. У тестировщика нет доступа к внутренней структуре и коду приложения либо в процессе тестирования он не обращается к ним.

Белый ящик – тестирование основанное на анализе внутренней структуры компонентов или системы. У тестировщика есть доступ к внутренней структуре и коду приложения.

Серый ящик – комбинация методов белого и черного ящика, состоящая в том,

что к части кода архитектуры у тестировщика есть, а кчасти кода – нет.

Типы тестов по степени автоматизации

Ручное – тестирование, в котором тест-кейсы выполняются тестировщиком вручную без использования средств автоматизации.

Автоматизированное – набор техник, подходов и инструментальных средств, позволяющий исключить человека из выполнения некоторых задач в процессе тестирования. Тест-кейсы частично или полностью выполняет специальное инструментальное средство.

Типы тестов по изолированности компонентов

Unit/component (модульное) – тестирование отдельных компонентов(модулей) программного обеспечения.

Integration (интеграционное) – тестируется взаимодействие между интегрированными компонентами или системами.

System (системное) – тестируется работоспособность системы в целомс целью проверки того, что она соответствуе установленным требованиям.

Типы тестов по подготовленности.

Интуитивное тестирование выполняется без подготовки к тестам, без определения ожидаемых результатов, проектирования тестовых сценариев.

Исследовательское тестирование – метод проектирования тестовых сценариев во время выполнения этих сценариев. Тестировщик совершаетпроверки, продумывает их, придумывает новые проверки, часто использует для этого полученную информацию.

Тестирование по документации – тестирование по подготовленным тестовым сценариям, руководству по осуществлению тестов.

Титы тестов по месту и времени проведения

User Acceptance Testing (UAT) (приемочное тестирование) – формальное тестирование по отношению к потребностям, требованиям и бизнес процессам пользователя, проводимое с целью определения соответствия системы критериям приёмки и дать возможность пользователям, заказчикам или иным авторизованным лицам определить, принимать систему.

Alpha Testing (альфа-тестирование) – моделируемое или действительное функциональное тестирование, выполняется в организации, разрабатывающей продукт, но не проектной командой (это может быть независимая команда тестировщиков, потенциальные пользователи, заказчики). Альфа тестирование часто применяется к коробочному программному обеспечению в качестве внутреннего приемочного тестирования.

Beta Testing (бета-тестирование) – эксплуатационное тестирование потенциальными или существующими клиентами/заказчиками на внешней стороне (в среде, где продукт будет использоваться) никак связанными с разработчиками, с целью определения действительно ли компонент или система удовлетворяет требованиям клиента/заказчика и вписывается в бизнес-процессы. Бета-тестирование часто проводится как форма внешнего приемочного тестирования готового программного обеспечения для того, чтобы получить отзывы рынка.

Типы тестов по объекту тестирования

Functional testing (функциональное тестирование) – это тестирование, основанное на анализе спецификации, функциональности компонента или системы. Функциональным можно назвать любой вид тестирования, который согласно требованиям проверяет правильную работу.

Safety testing (тестирование безопасности) – тестирование программного продукта с целью определить его безопасность (безопасность

– способность программного продукта при использовании оговоренным образом оставаться в рамках приемлемого риска причинения вреда здоровью,бизнесу, программам, собственности или окружающей среде.

Security testing (тестирование защищенности) – это тестирование с целью оценить защищенность программного продукта. Тестирование защищенности проверяет фактическую реакцию защитных механизмов, встроенных в систему, на проникновение.

Compatibility testing (тестирование совместимости) – процесс тестирования для определения возможности взаимодействия программного продукта, проверка работоспособности приложения в различных средах (браузеры и их версии, операционные системы, их типа, версии и разрядность)

Виды тестов:

- ✓ кросс-браузерное тестирование (различные браузеры или версии браузеров)
- ✓ кросс-платформенноетестирование(различныеоперационныесистемы или версии операционных систем)

Нефункциональное тестирование – это проверка характеристик программы. Иначе говоря, когда проверяется не именно правильность работы, а какие-либо свойства (внешний вид и удобство пользования, скорость работы и т.п.).

1. **Тестирование пользовательского интерфейса (GUI)** – тестирование, выполняемое путем взаимодействия с системой через графический интерфейс пользователя.

- ✓ навигация
- ✓ цвета, графика, оформление
- ✓ содержание выводимой информации
- ✓ поведение курсора и горячие клавиши
- ✓ отображениеразличногоколичестваданных(нетданных,минимальное и максимальное количество)
- ✓ изменение размеров окна или разрешения экрана

2. **Тестирование удобства использования (Usability Testing)** – тестирование с целью определения степени понятности, легкости в изучении и использовании, привлекательности программного продукта для пользователя при условии использования в заданных условиях эксплуатации.

- ✓ визуальное оформление
- ✓ навигация
- ✓ логичность

3. **Тестирование доступности (Accessibility testing)** – тестирование, которое определяет степень легкости, с которой пользователи с ограниченными способностями могут использовать систему или еекомпоненты.

4. **Тестирование интернационализации** – тестирование способности продукта работать в локализованных средах (способность изменять элементы интерфейса в зависимости от длины и направления текста, менять сортировки/форматы под различные локали и т.д.). (Максим Черняк).

Интернационализация – это процесс, упрощающий дальнейшую адаптацию продукта к языковым и культурным особенностям региона, отличного от того, в котором разрабатывался продукт. Это адаптация продукта для потенциального использования практически в любом месте, Интернационализация производится на начальных этапах разработки, в то время как локализация — для каждого целевого языка.

5. **Тестирование локализации (Localization testing)** – тестирование, проводимое с целью проверить качество перевода продукта с одного языка на другой.

6. **Тестирование производительности или нагрузочное тестирование –**

процесс тестирования с целью определения производительности программного продукта.

Виды тестов:

- ✓ нагрузочное тестирование (Performance and Load testing) – вид тестирования производительности, проводимый с целью оценки поведения компонента или системы при возрастающей нагрузке, например количестве параллельных пользователей и/или операций, а также определения какуюнагрузку может выдержать компонент или система;
- ✓ объемное тестирование (Volume testing) – позволяет получить оценку производительности при увеличении объемов данных в базе данных приложения;
- ✓ тестирование стабильности и надежности (Stability / Reliabilitytesting) – позволяет проверять работоспособность приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки.
- ✓ стрессовое тестирование (Stress testing) – вид тестирования производительности, оценивающий систему или компонент на граничных значениях рабочих нагрузок или за их пределами, или же в состоянии ограниченных ресурсов, таких как память или доступ к серверу.

7. **Тестирование требований (Requirements testing)** – проверка требований на соответствие основным характеристикам качества.

8. **Тестирование прототипа (Prototyte testing)** – метод выявления структурных, логических ошибок и ошибок проектирования на ранней стадии развития продукта до начала фактической разработки.

9. **Тестирование установки (Installability testing) и лицензирования** – процесс тестирования устанавливаемости программного продукта.

Виды тестов:

- ✓ формальный тест программы установки приложения (проверка пользовательского интерфейса, навигации, удобства пользования, соответствия общепринятым стандартам оформления);
- ✓ функциональный тест программы установки;
- ✓ тестирование механизма лицензирования и функций защиты от пиратства;
- ✓ проверка стабильности приложения после установки.

10. **Тестирование на отказ и восстановление (Failover and Recovery Testing)** – тестирование при помощи эмуляции отказов системы или реально вызываемых отказов в управляемом окружении.

Тестирование программного продукта включает следующие этапы:

1. Изучение и анализ предмета тестирования.
2. Планирование тестирования.
3. Исполнение тестирования.

Изучение и анализ предмета тестирования начинается еще до утверждения спецификации и продолжается на стадии разработки (кодирования) программного обеспечения. Конечной целью этапа изучение ианализ предмета тестирования является получение ответов на два вопроса:

- какие функциональности предстоит протестировать,
- как эти функциональности работают.

Планирование тестирования происходит на стадии разработки (кодирования) программного обеспечения. На стадии планирования тестирования перед тестировщиком стоит задача поиска компромисса между объемом тестирования, который возможен в теории, и объемом тестирования, который возможен на практике. На данной стадии необходимо ответить на вопрос: как будем тестировать? Результатом планирования тестирования является тестовая документация.

Выполнение тестирования происходит на стадии тестирования и представляет

собой практический поиск дефектов с использованием тестовой документации, составленной ранее.

Для всех программных продуктов выполняют следующие типы тестов и их композиции.

Для первого билда рекомендуется проводить Smoke+AT готовой функциональности: поверхностное тестирование (Smoke Test) выполняется для определения пригодности сборки для дальнейшего тестирования; полное тестирование системы или ее части как на корректных, так и на некорректных данных/сценариях (Acceptance Test, AT) позволяет обнаружить дефекты и внести запись о них в багтрэкинговую систему.

Для последующих билдов композиции тестов могут быть следующими:

- Если не была добавлена новая функциональность, то: DV+MAT. Т.е., выполняется проверка исправления дефектов программистом (Defect Validation, DV), а также проверка работоспособности остальной функциональности после исправления дефектов на позитивных сценариях(Minimal Acceptance Test, MAT).
- Если была добавлена новая функциональность, то: Smoke+DV+NFT+Regression Test. В частности, выполняется поверхностное тестирование (Smoke Test), проверка исправления дефектов программистом (Defect Validation, DV), тестирование новых функциональностей (New Feature Testing, NFT), проверка старых функциональностей, т.е.регрессионное тестирование (Regression Test).
- Если была добавлена новая функциональность, то возможен также вариант: DV+NFT+Resression test, т.е. без выполнения Smoke Test.

В зависимости от типа и специфики приложения (web, desktop, mobile) выполняют специализированные тесты (например, кроссбраузерное или кроссплатформенноетестирование,тестированиелокализациии интернационализации и др.).

Содержание отчета

1. Изучить материал
2. Ответить на контрольные вопросы.
3. Выбрать объект реального мира, в примере ниже «Карандаш»(на последней странице)
4. Сгенерированные тесты различных видов для выбранного объекта реального мира.
5. Тестовые активности для сформулированных задач.
6. Создать кластер по созданным тестам, можно использовать приложение для создания интеллектуальных карт, например https://www.mindmeister.com/
7. .Выводы по работе.

Контрольные вопросы

1. Что такое тестирование?
2. Какиесуществуюттипытестовпопокрытию?Дайтехарактеристику каждому.
3. Какие существуют тестовые активности? Дайте характеристикукаждому.
4. Какие существуют типы тестов знанию кода? Дайте характеристику каждому.
5. Какие существуют типы тестов по степени автоматизации? Дайте характеристику каждому.
6. Какие существуют типы тестов по изолированности компонентов?Дайте характеристику каждому.
7. Какиесуществуюттипытестовпоподготовленности?Дайте характеристику каждому.
8. Какие существуют типы тестов по месту и времени проведения?Дайте

характеристику каждому.

9. Какие существуют типы тестов по объекту тестирования? Дайте характеристику каждому.

10. Какиесуществуюттипыфункциональныхтестов?Дайте характеристику каждому.

11. Какиесуществуюттипынефункциональныхтестов?Дайте характеристику каждому.

12. Какие этапы составляют процесс тестирования?

13. Чтопроисходитнаэтапеизученияианализапредметатестирования?

14. Что происходит на этапе планирования тестирования?

15. Что происходит на этапе исполнения тестирования?

16. Какие типы тестов выполняют для первой поставки программного продукта?

17. Какиетипытестоввыполняютдляпоследующихпоставок программного продукта?

Лабораторная работа 25.
Знакомство с разнообразием реализации UML сервисов

Цель работы: является знакомство с базовыми приёмами проектирования системи процессов с использованием универсального языка моделирования (UML)

Справочная информация:

Диаграмма прецедентов, представляет собой простой способ визуального представления основных возможностей разрабатываемого программного обеспечения или процесса.

> Для выполнения лабораторной работы может быть использован свободно распространяемые редакторы, доступные по адресу: http://staruml.io, Draw.io, https://creately.com/ или любой другой редактор UML диаграмм.

Пример диаграммы прецедентов, описывающий функции текстового редактора.

Для описания функций используются действующее лицо (обозначенное человекоподобной фигурой) прецеденты (обозначенные овалами) и ассоциативные связи.

Следует отметить, что действующим лицом может являться не только пользователь, но и программа, программист, специалист службы технической поддержки и т.д. А одной диаграмме, может присутствовать множество действующих лиц.

Помимо ассоциативных связей, существуют:

- направленные ассоциации (линия со стрелкой) – в явной форме указывают характер отношений между прецедентами
- зависимости – указывают на зависимости между прецедентами

- обобщения – указывают на вхождение частного прецедента в более общий

пример 1

В данном примере описан прецедент расширенной функции “сохранить файл”. Для указания того, что сохранение файла с выбором формата содержит в себе выбор форматаи является расширением стандартной функции сохранения файла, используется отношение зависимости.

На данной диаграмме показано что ввод текста является частным случаем редактированияфайла, а ввод связан с выбором шрифта (без указания типа связи).

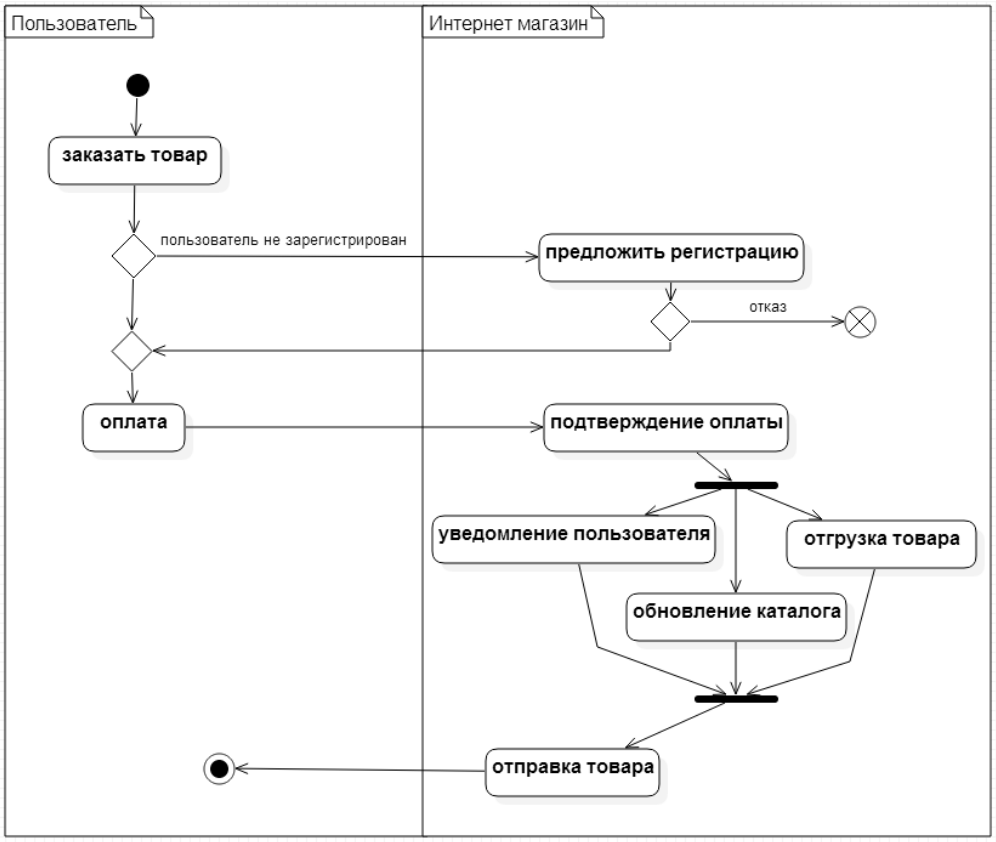
Следует помнить, что диаграмма прецедентов должна описывать общие функциональные возможности разрабатываемой программы или системы, а также фундаментальные зависимости между прецедентами, не акцентируя внимания на деталях реализации процессов и действующих лиц.

Диаграмма деятельности является альтернативой представлению процессов в виде блок схем и используется для описания последовательности действий и выборов.

Состоит из следующих элементов:

- начало процесса – обозначает старт описываемого процесса, может не совпадать с началом работы программы или глобального процесса.
- действие – содержит в себе описание действий на текущем этапе выполнения алгоритма.
- решение – как и на блок схемах, обозначается ромбом, однако не содержит в себе текста. Текс условий ветвления указывается на исходящих из решения управляющих потоках.
- управляющий поток – указывает последовательность выполнения действий.
- разделение – начало блока независимых операций.
- соединение – завершение блока независимых операций.
- завершение процесса – окончание описываемого процесса, может не совпадать сокончанием работы программы или глобального процесса.

пример 2



Пример описания процесса заказа товара через интернет-магазин, при помощи диаграммы деятельности.

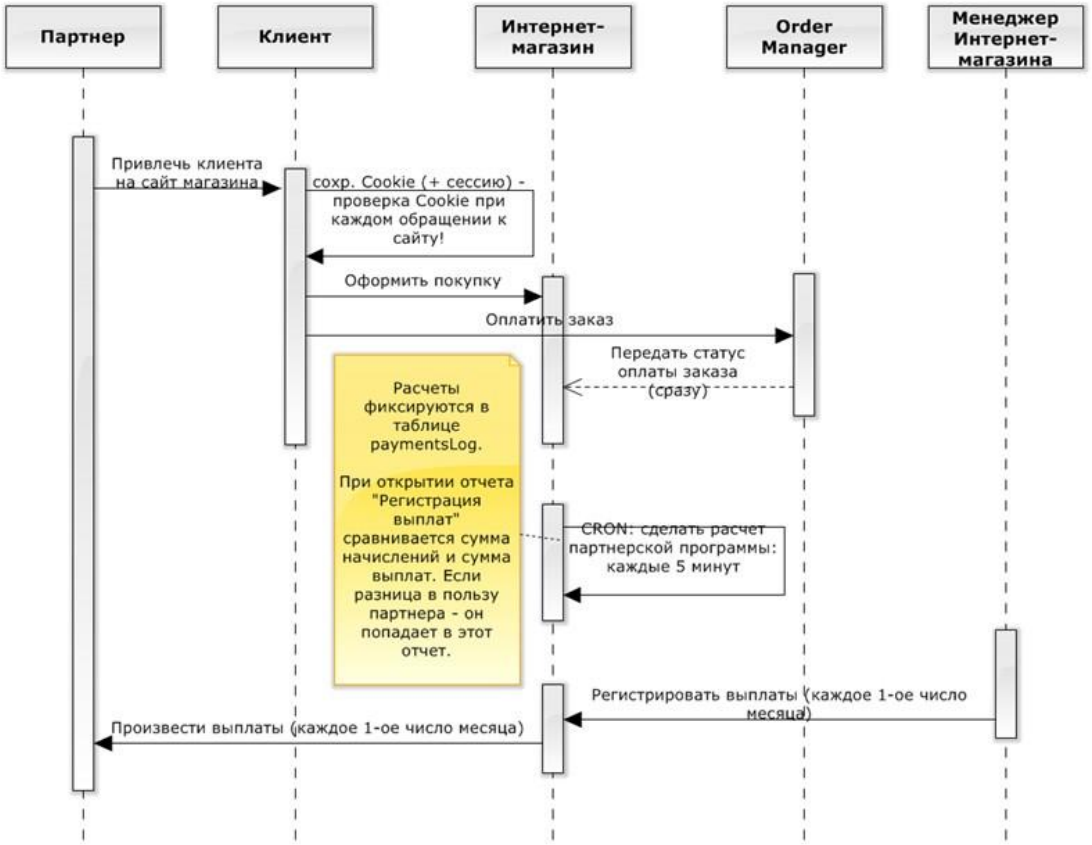
На представленной диаграмме, действия расположены в двух областях, обозначающих действующих лиц участвующих в процессе заказа. Подобное представление не является обязательным при составлении диаграммы деятельности, однако, в дальнейшем, может упростить создание диаграммы последовательности.

Предполагается что операции по уведомлению пользователя, обновлению каталога и отгрузки товара, могут осуществляться одновременно, поскольку независимы друг от друга.

Действие “отправка товара” является обобщённым и может быть представлено в виде отдельной диаграммы, если это необходимо для понимания моделируемых процессов.

Следует помнить, что диаграмма деятельности должна описывать последовательностьдействий и выборов в процессе выполнения некоего процесса, не акцентируя внимания на классах, полях и методах.

Пример 3



Задание:

Посмотреть видео

Попробовать создать диаграммы прецедентов (пример 1) и деятельности (пример 2), последовательности (пример 3) в различных сервисах(2-3). Напишите какой сервис для вас более удобен и почему.

Лабораторная работа 26. Создание диаграммы вариантов использования

Цель работы:

1. Знакомство с основными понятиями **UML**
2. Изучение компонентов модели
3. Построение модели вариантов использования

Диаграмма вариантов использования – это тип поведенческой диаграммы UML, который часто используется для анализа различных систем. Они позволяют визуализировать различные типы ролей в системе и то, как эти роли взаимодействуют с системой.

Важность использования диаграммы прецедентов

Как уже упоминалось ранее, диаграммы прецедентов используются для сбора требований к использованию системы. В зависимости от ваших требований вы можете использовать эти данные различными способами. Ниже приведены несколько способов их использования.

- **Идентификация функций и как с ними взаимодействуют роли** – основное назначение диаграмм сценариев использования.
- **Для представления системы на высоком уровне** – Особенно полезно при представлении**е** руководителям или заинтересованным сторонам. Вы можете выделить роли, которые взаимодействуют с системой, и функциональные возможности, предоставляемые системой, не углубляясь во внутреннюю работу системы.

- **Идентификация внутренних и внешних факторов** – Это может показаться простым, но в больших сложных проектах система может быть идентифицирована как внешняя роль в другом случае использования.

Объекты диаграммы прецедентов

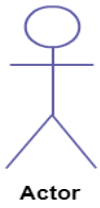
Использовать диаграммы корпуса состоят из 4 объектов.

- Актер
- Случай использования
- Система
- Пакет

Объекты более подробно описаны ниже.

Актер

Актер в использует диаграмму прецедентов – это **любая сущность, которая выполняет роль** в одной данной системе. Это может быть человек, организация или внешняя система и обычно рисуется как скелет, показанный ниже.

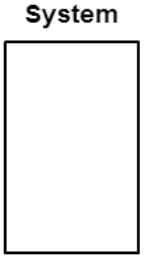


Случай использования

Случай использования **представляет собой функцию или действие внутри системы**. Она нарисована как овал и названа функцией.

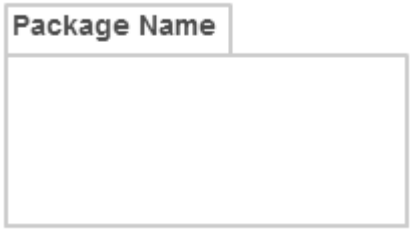
Система

Система используется для **определения сферы применения** и нарисована в виде прямоугольника. Это необязательный элемент, но полезный при визуализации больших систем. Например, вы можете создать все случаи использования, а затем использовать системный объект для определения области применения вашего проекта. Или вы даже можете использовать его, чтобы показать различные области, охваченные в разных релизах.



Пакет

Пакет является еще одним дополнительным элементом, который чрезвычайно полезен в сложных диаграммах. Подобно диаграммам классов, пакеты **используются для группировки случаев использования**. Они нарисованы, как показано на рисунке ниже.



Рекомендации по диаграммам прецедентов

Несмотря на то, что диаграммы использования могут быть использованы для различных целей, существуют некоторые общие рекомендации, которым необходимо следовать при рисовании примеров использования.

К ним относятся стандарты именования, направления стрелок, размещение вариантов использования, использование системных блоков, а также правильное использование отношений.

Отношения в диаграммах вариантов использования

Существует пять типов отношений на диаграмме прецедентов. Они

- Ассоциация между актером и случаем использования
- Обобщение актера
- Расширить отношения между двумя случаями использования
- Включить взаимосвязь между двумя случаями использования
- Обобщение случая использования

Как создавать диаграммы прецедентов использования различные процессы на примере банковской системы.

Выявление актеров

Актеры – это внешние объекты, взаимодействующие с системой. Это может быть человек, другая система или организация. В банковской системе наиболее очевидным действующим лицом является клиент. Другие актеры могут быть банковскими служащими или кассирами в зависимости от роли, которую пытаетесь показать в случае использования.

Примером внешней организации может служить налоговый орган или центральный банк. Кредитный процессор является хорошим примером внешней системы, связанной в качестве агента.

Определение случаев использования

Теперь пришло время идентифицировать случаи использования. Хороший способ сделать это – определить, что нужно участникам системы. В банковской системе клиенту необходимо будет открывать счета, вводить и выводить средства, запрашивать чековые книги и выполнять аналогичные функции. Так что все это можно рассматривать как случаи использования.

Случаи использования верхнего уровня всегда должны обеспечивать полную функцию, требуемую для агента. В зависимости от сложности системы вы можете расширить или включить случаи использования.

После того, как вы определили актёров и верхний уровень использования, у вас есть базовое представление о системе. Теперь вы можете точно настроить его и добавить к нему дополнительные слои деталей.

Ищите общие функциональные возможности для использования Включать

Ищите общие функциональные возможности, которые могут быть повторно использованы в системе. Если вы найдете два или более случаев использования, которые имеют общую функциональность, вы можете извлечь общие функции и добавить его в отдельный случай использования. Затем вы можете подключить его через include relationship, чтобы показать, что он всегда вызывается, когда выполняется исходный сценарий использования. (см. диаграмму для примера).

Возможно ли обобщение актеров и случаев использования

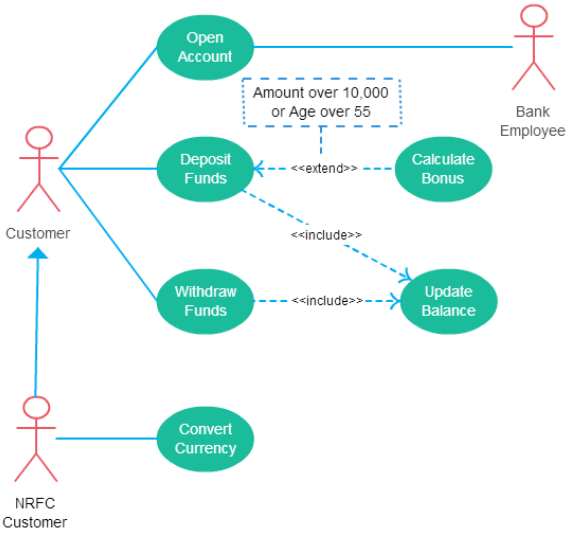
Могут быть случаи, когда агенты ассоциируются с аналогичными случаями использования, в то время как запускается несколько случаев использования, уникальных только для них. В таких случаях можно обобщить агент, чтобы показать наследование функций. Аналогичную вещь можно сделать и в случае использования.

Одним из лучших примеров этого является случай использования “Оплатить” в платежной системе. Вы можете далее обобщить его до “Оплатить кредитной картой”, “Оплатить наличными”, “Оплатить чеком” и т.д. Все они имеют атрибуты и функциональность оплаты со специальными уникальными для них сценариями.

Необязательные функции или дополнительные функции

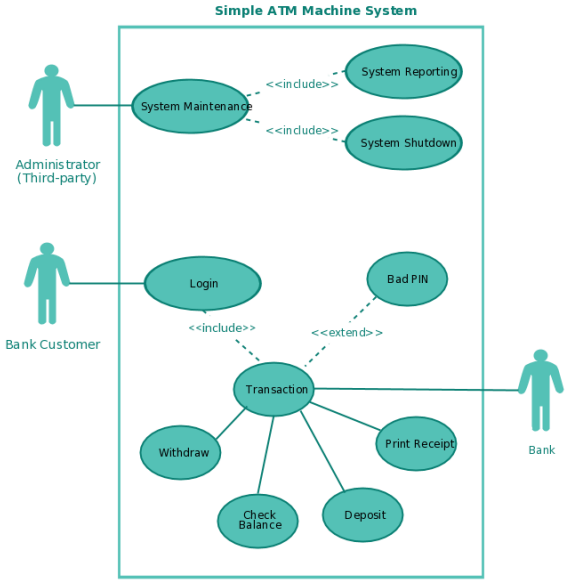
Есть некоторые функции, которые срабатывают опционально. В таких случаях можно использовать отношения расширения и прикрепить к ним правило расширения. В приведенном ниже примере банковской системы “Рассчитать бонус” является необязательным и срабатывает только при выполнении определенного условия.

Продление не всегда означает, что оно необязательно. Иногда вариант использования, связанный с удлинением, может дополнять вариант использования базы. Следует помнить, что базовый сценарий использования должен быть способен выполнять функцию самостоятельно, даже если сценарий использования расширения не вызывается.



Случай использования с большинством сценариев, найденных в диаграмма прецедентов

Шаблоны диаграмм прецедентов использования



Шаблон варианта прецедентов для системы банкомата

Сценарий (scenario) - специально написанный текст, который описывает поведение моделируемой системы в форме последовательности выполняемых действий актеров и самой системы.

Один из шаблонов для написания сценария отдельного варианта использования рассматривается в табл. 1.

Для удобства записи и чтения сценария часто используют модификацию данного шаблона, располагая разделы последовательно сверху вниз. При этом написание сценариев модели начинают с базовых или основных вариантов использования, после чего рассматриваются сценарии второстепенных или включаемых вариантов использования. Сценарии расширяющих вариантов использования обычно помещаются в разделе исключений базового варианта использования.

Таблица 1.

Шаблон для написания сценария отдельного варианта использования

<table>
<tr><th>Главный раздел</th><th>Раздел «Типичный ход событий»</th><th>Раздел «Исключения»</th><th>Раздел «Примечания»</th></tr>
<tr><td rowspan="4">Имя варианта использования

Актеры

Цель

Краткое описание

Тип

Ссылкана другие варианты использования</td><td rowspan="4">Типичный ход событий,приводящий к успешному выполнению данного варианта использования</td><td>Исключение № 1</td><td>Примечание № 1</td></tr>
<tr><td>Исключение № 2</td><td>Примечание № 2</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Исключение № n</td><td>Примечание № n</td></tr>
</table>

Пример 1.

Построим модель вариантов использования для системы продажи товаров в интернет-магазине.

Посетитель интернет-магазина может просматривать список товаров интернет-магазина, помещать товар в виртуальную корзину и изменятьсодержимое этой корзины.

Посетитель может стать покупателем, если он принимает решение обоформлении заказа на покупку выбранных им товаров.

Менеджер может изменять список товаров и специфицировать условия для предоставления бонусной скидки, а бухгалтер - принимать оплату за выбранный покупателем товар.

При оформлении заказа на покупку товара необходима регистрация покупателя.

При оформлении заказа постоянному покупателю может быть предоставлена специальная бонусная скидка.

В рамках рассматриваемой системы продажи товаров в интернет-магазине возможна оплата выбранных покупателем товаров, как наличными, так и по кредитной карточке.

Выделение актеров и вариантов использования

В соответствии с определением терминов «актер» и «вариантиспользования», для данной предметной области актерами будут люди, участвующие в работе интернет-магазина, а вариантами использования те действия, которые они выполняют в интернет-магазине.

1. В качестве основного актера описываемой системы можно рассматривать актера «Посетитель интернет-магазина». В соответствии с теми действиями, которые может выполнять посетитель интернет-магазина, можно выделить два варианта использования, свзанных с актером

«Посетитель интернет-магазина» отношением ассоциации: «Просмотр списка товаров» и «Изменение содержания корзины».

2. Т.к. посетитель интернет-магазина может стать покупателем, то можно выделить актера «Покупатель», связанного отношением обобщения с актером «Посетитель интернет-магазина», при этом в качестве родителя будет выступать актер «Посетитель интернет-магазина», а в качестве потомка «Покупатель», т.к. последний наследует все свойства поведения посетителя интернет-магазина, но и обладает собственными. Вариантами использования, связанными отношением ассоциации с актером

«Покупатель», будут «Оформление заказа на покупку товаров» и «Оплата выбранного товара».

3. В качестве других актеров рассматриваемой системы могут выступать «Менеджер» и «Бухгалтер». При этом «Менеджер» будет связанотношением ассоциации с вариантами использования «Изменение списка товаров» и «Предоставление бонусной скидки», а бухгалтер - с вариантом использования «Оплата выбранного товара».

4. Поскольку при оформлении заказа на покупку товара необходима регистрация покупателя, и эта функциональность выполняется всегда, она может быть выделена в отдельный вариант использования «Регистрация покупателя», который будет связан с базовым отношением включения. С другой стороны, при оформлении заказа постоянному покупателю может быть предоставлена специальная бонусная скидка. Это требование может быть также представлено в качестве отдельного варианта использования

«Предоставление бонусной скидки», который будет связан с базовым отношением расширения.

5. Дальнейшая детализация модели может быть выполнена на основе установления дополнительного отношения обобщения для варианта использования «Оплата выбранного товара». Если в рамках рассматриваемой системы продажи товаров в интернет-магазине возможнаоплата выбранных покупателем товаров, как наличными, так и по кредитной карточке, то в этом случае диаграмма может быть дополнена соответствующими вариантами использования. При этом варианты использования «Оплата товара наличными» и «Оплата товара по кредитнойкарточке» будут связаны с вариантом использования «Оплата выбранного товара» отношением обобщения.

В результате разрабатываемая диаграмма должна содержать 9 вариантов использования и 4-х актеров, между которыми установлены соответствующие отношения ассоциации, включения, расширения и обобщения. Все варианты использования желательно заключить в прямоугольник, который служит для обозначения субъекта проектируемойсистемы.

Пример 2.

Будем проектировать систему для предметной области "Предприятиепо сборке и продаже компьютеров".

Для нашей предметной области мы выделили следующих актеров:

Актер	**Краткое описание**
Менеджер по работе с клиентами	Сотрудник, который общается с заказчиком и работает с заказом

Менеджер по снабжению	Сотрудник, который занимается закупкой необходимых комплектующих
Инженер по сборке настольных компьютеров	Сотрудник, который занимается сборкой настольных компьютеров
Инженер по сборке ноутбуков	Сотрудник, который занимается сборкой ноутбуков
Инженер по тестированию	Сотрудник, который занимается тестированием собранных компьютеров
Завскладом	Сотрудник, который заведует складом комплектующих

Рассмотрим теперь, какие возможности должна предоставлять нашасистема:

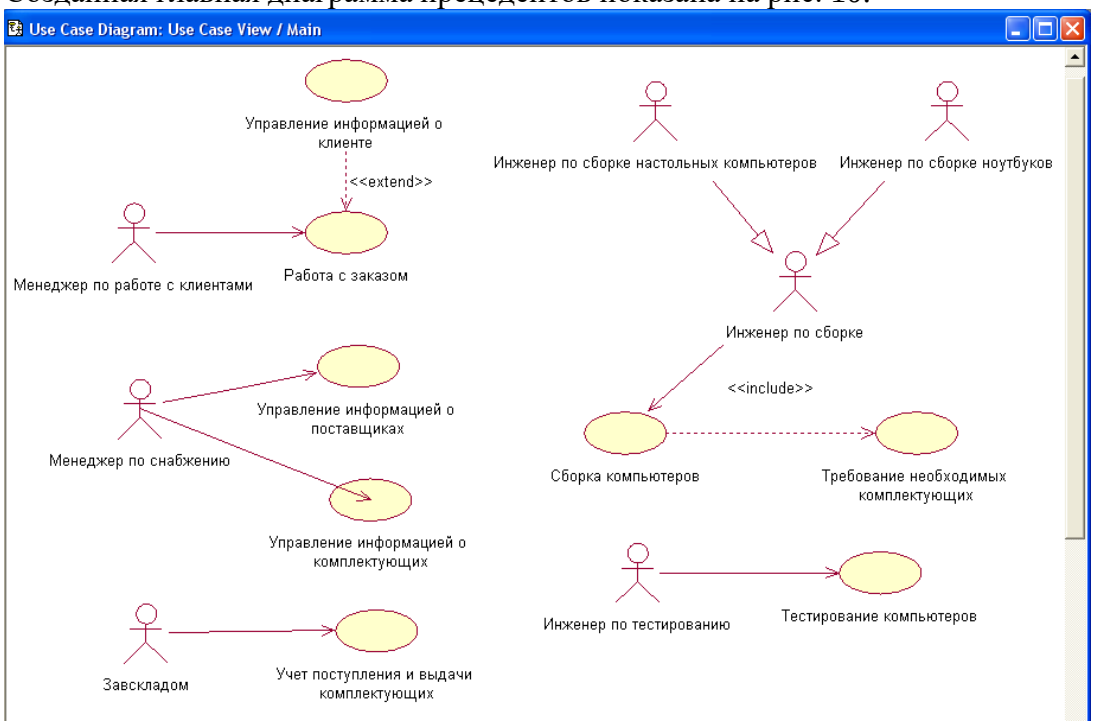
- актер Менеджер по работе с клиентами использует систему для оформления, редактирования заказов и управления информацией оклиентах предприятия;
- актер Менеджер по снабжению использует систему для просмотра перечня необходимых для закупки комплектующих и ведения информации о снабжении;
- актер Инженер по сборке настольных компьютеров использует систему для просмотра нарядов на сборку персональных компьютеров, для заказа комплектующих со склада и отметки о ходевыполнения работы;
- актер Инженер по сборке ноутбуков использует систему для просмотра нарядов на сборку ноутбуков, для заказа комплектующихсо склада и отметки о ходе выполнения работы;
- актер Инженер по тестированию использует систему для просмотранарядов на тестирование собранной продукции и отметки о ходе выполнения работы;
- актер Завскладом использует систему для учета поступления и выдачи комплектующих.

Наосновааниивышеизложенногоможновыделитьследующиепрецеденты:

Прецедент	**Краткое описание**
Работа с заказом	Запускается менеджером по работе с клиентами. Позволяет вносить, изменять, удалять или просматривать заказ.
Управление информацией о клиенте	Запускается менеджером по работе с клиентами. Позволяет добавлять, изменять или удалять клиентов, а также просматривать информацию о клиентах.
Управление информацией о поставщиках	Запускается менеджером по снабжению. Позволяет добавлять, изменять или удалять поставщиков, а также просматривать информацию о поставщиках.
Управление информацией о комплектующих	Запускается менеджером по снабжению. Позволяет просматривать информацию о комплектующих, производить анализ их расходования, прогнозировать необходимое их количество и делать заказ.
Сборка компьютеров	Запускается инженером по сборке. Позволяет просматривать наряды на сборку компьютеров и делать отметки о ходе выполнения работы.
Требование необходимых комплектующих	Запускается инженером по сборке. Предназначено для затребования необходимых комплектующие со склада.
Тестированиекомпьютеров	Запускается инженером по тестированию. Позволяет просмотреть список компьютеров, подлежащих тестированию и сделать отметки о ходе выполнения работ.
Учет поступления и выдачи комплектующих	Запускается завскладом. Позволяет вести учет поступления

	и выдачи запчастей и комплектующих.

Созданная главная диаграмма прецедентов показана на рис. 10:



Главная диаграмма прецедентов

Задание: С помощью сервисов по созданию UML диаграмм (Любой из предложенных http://staruml.io, Draw.io, https://creately.com/) подготовить сценарии (таблицы) и диаграммы вариантов использования (use case):

1. Авиакасса Пассажир бронирует билет на рейс у агента.
2. Автосервис. Клиент сдает автомобиль в автосервис.

Ответить на контрольные вопросы

1. Что такое вариант использования?
2. Что такое диаграмма вариантов использования?
3. Какие существуют ассоциации?

Лабораторная работа 27.
Создание диаграммы последовательностей.

Диаграммы последовательности, обычно используемые разработчиками, моделируют взаимодействия между объектами в одном варианте использования. Они иллюстрируют, как различные части системы взаимодействуют друг с другом для выполнения функции, а также порядок, в котором происходит взаимодействие при выполнении конкретного варианта использования.

Проще говоря, диаграмма последовательности показывает, что различные части системы работают в «последовательности», чтобы что-то сделать.

Обозначения диаграмм последовательности

Диаграмма последовательности структурирована таким образом, что представляет собой временную шкалу, которая начинается сверху и постепенно спускается, отмечая последовательность взаимодействий. У каждого объекта есть столбец, а сообщения, которыми они обмениваются, представлены стрелками.

Краткий обзор различных частей диаграммы последовательности

Обозначение линии жизни

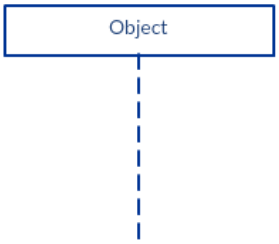


Диаграмма последовательности состоит из нескольких таких обозначений линий жизни, которые должны быть расположены горизонтально в верхней части диаграммы. Никакие два обозначения жизненного пути не должны перекрывать друг друга. Они представляют различные объекты или части, которые взаимодействуют друг с другом в системе во время последовательности.

Обозначение линии жизни с символом элемента актера используется, когда конкретная диаграмма последовательности принадлежит прецеденту.

Линия жизни с элементом сущности представляет системные данные. Например, в приложении обслуживания клиентов объект Customer будет управлять всеми данными, связанными с клиентом.

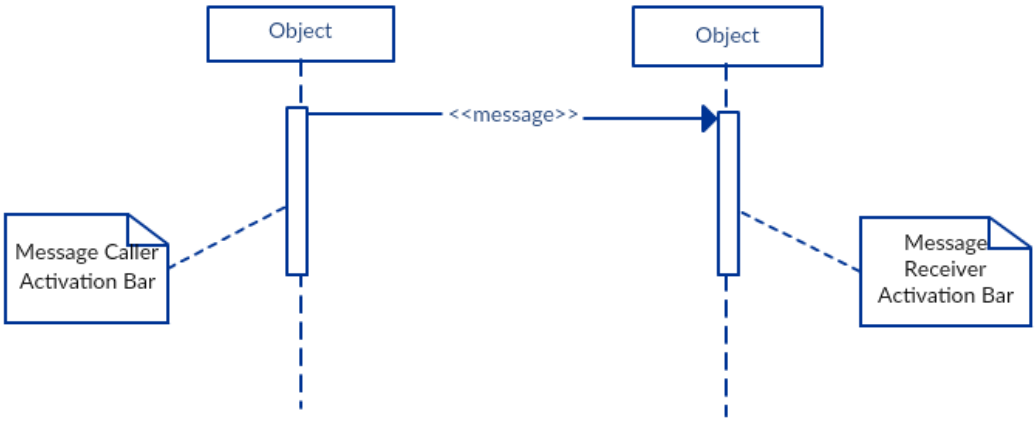
Линия жизни с граничным элементом указывает границу системы / программный элемент в системе; например, экраны пользовательского интерфейса, шлюзы баз данных или меню, с которыми взаимодействуют пользователи, являются границами.

А линия жизни с элементом управления указывает на контролирующую организацию или менеджера. Он организует и планирует взаимодействия между границами и объектами и служит посредником между ними.

Полоски активации

Панель активации — это прямоугольник, помещенный на спасательный круг. Он используется, чтобы указать, что объект активен (или создан) во время взаимодействия между двумя объектами. Длина прямоугольника указывает на продолжительность пребывания объектов в активном состоянии.

На диаграмме последовательности взаимодействие между двумя объектами происходит, когда один объект отправляет сообщение другому. Использование панели активации на линиях жизни вызывающего сообщения (объект, который отправляет сообщение) и получателя сообщения (объект, который получает сообщение) указывает на то, что оба активны / создаются во время обмена сообщением.



Стрелки сообщений

Стрелка от вызывающего сообщения к получателю сообщения указывает сообщение на диаграмме последовательности. Сообщение может течь в любом направлении; слева направо, справа налево или назад к самому сообщению вызывающего абонента. Хотя вы можете описать сообщение, отправляемое от одного объекта к другому, с помощью стрелки, с помощью разных стрелок вы можете указать тип отправляемого или получаемого сообщения.

Стрелка сообщения снабжена описанием, которое называется подписью сообщения. Формат подписи этого сообщения приведен ниже. Все части, кроме message_name, необязательны.

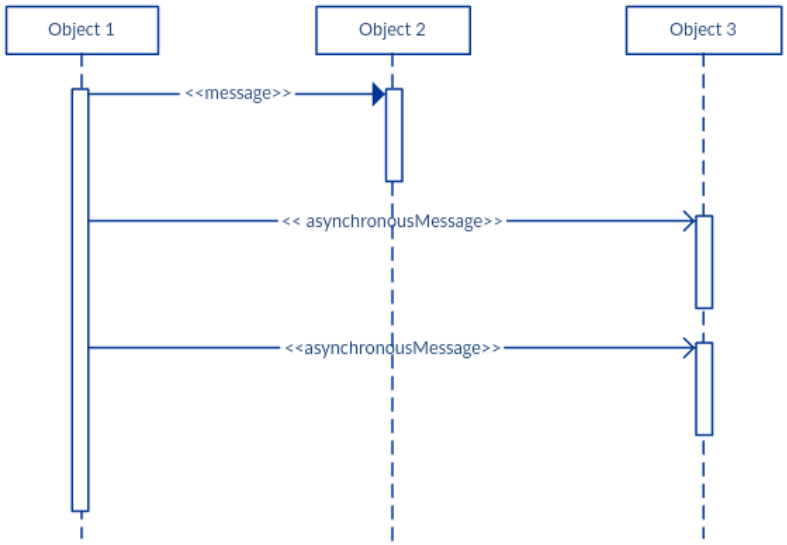
attribute = имя_сообщения (аргументы): return_type

- **Синхронное сообщение**

Как показано в примере панели активации, синхронное сообщение используется, когда отправитель ожидает, пока получатель обработает сообщение и вернется, прежде чем продолжить с другим сообщением. Стрелка, используемая для обозначения этого типа сообщения, сплошная, как показано ниже.

- **Асинхронное сообщение**

Асинхронное сообщение используется, когда вызывающий сообщение не ждет, пока получатель обработает сообщение и вернется, прежде чем отправлять другие сообщения другим объектам в системе. Стрелка, используемая для отображения этого типа сообщения, представляет собой линейную стрелку, как показано в примере ниже.

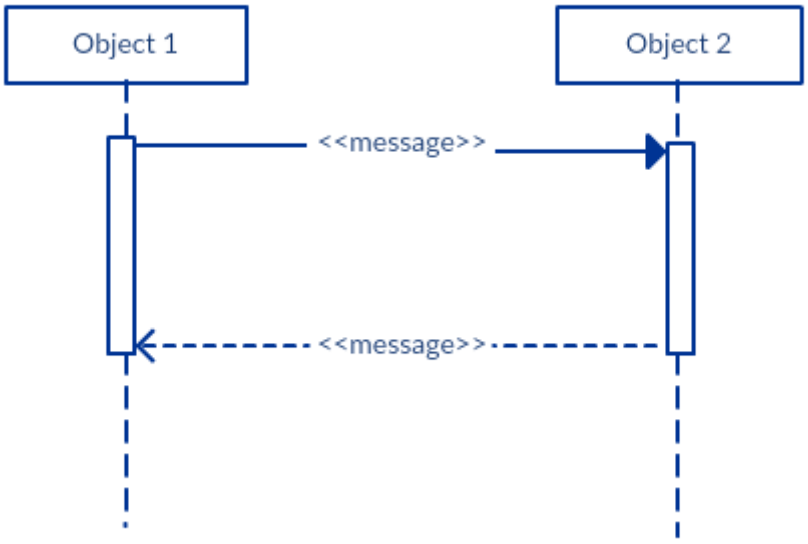


- **Обратное сообщение**

Ответное сообщение используется, чтобы указать, что получатель сообщения завершил обработку сообщения и возвращает управление вызывающей стороне сообщения. Сообщения возврата являются необязательными элементами обозначения, поскольку панель активации, которая запускается синхронным сообщением, всегда подразумевает ответное сообщение.

Совет: вы можете избежать загромождения ваших диаграмм, минимизируя использование возвращаемых сообщений, поскольку возвращаемое значение может

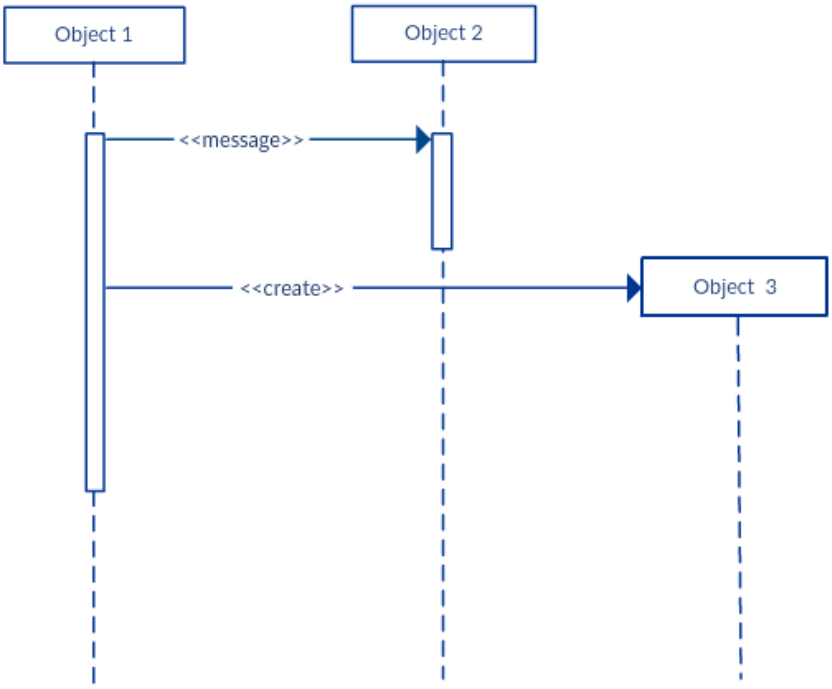
быть указано в самой стрелке исходного сообщения.



- **Сообщение о создании участника**

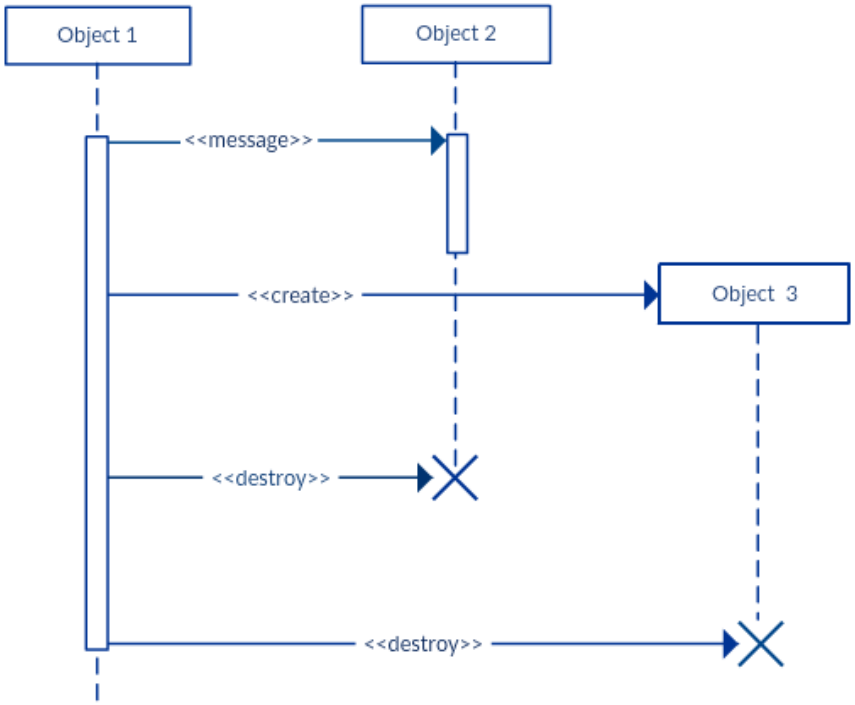
Объекты не обязательно живут на протяжении всей последовательности событий. Объекты или участники могут быть созданы в соответствии с отправляемым сообщением.

Обозначение поля отброшенного участника можно использовать, когда вам нужно показать, что конкретный участник не существовал до тех пор, пока не был отправлен вызов создания. Если созданный участник что-то делает сразу после своего создания, вы должны добавить поле активации прямо под полем участника.



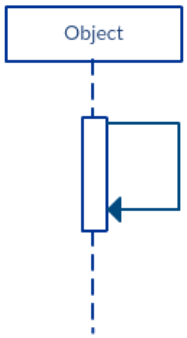
- **Сообщение об уничтожении участника**

Точно так же участники, когда они больше не нужны, также могут быть удалены из диаграммы последовательности. Это делается путем добавления «X» в конце линии жизни указанного участника.



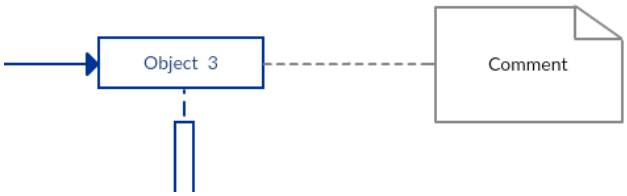
- **Рефлексивное сообщение**

Когда объект отправляет сообщение самому себе, это называется рефлексивным сообщением. На это указывает стрелка сообщения, которая начинается и заканчивается на той же линии жизни, как показано в примере ниже.



Комментарий

Диаграммы UML обычно допускают аннотацию комментариев во всех типах диаграмм UML. Объект комментария представляет собой прямоугольник с загнутым углом, как показано ниже. Комментарий можно связать со связанным объектом с помощью пунктирной линии.



Примечание: Ознакомьтесь с рекомендациями по работе с диаграммой последовательности, чтобы узнать о фрагментах последовательности.

Рекомендации по диаграммам последовательностей

• **Управляйте сложными взаимодействиями с фрагментами последовательностей**

Фрагмент последовательности представлен в виде прямоугольника, который обрамляет раздел взаимодействий между объектами (как показано в примерах ниже) на диаграмме последовательности.

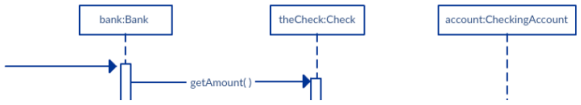
Он используется для более структурированного отображения сложных взаимодействий, таких как альтернативные потоки и циклы. В верхнем левом углу фрагмента сидит оператор. Это - оператор фрагмента - указывает, что это за фрагмент.

Альтернативы

Фрагмент альтернативной комбинации используется, когда необходимо сделать выбор между двумя или более последовательностями сообщений. Он моделирует логику «если бы еще».

Альтернативный фрагмент представлен большим прямоугольником или рамкой; он указывается путем упоминания «alt» внутри поля имени фрейма (он же оператор фрагмента).

Чтобы показать две или более альтернативы, большой прямоугольник затем делится на так наываемые операнды взаимодействия с помощью пунктирной линии, как показано в приведенном выше примере диаграммы последовательности. У каждого операнда есть защита для проверки, и она помещается в верхнем левом углу операнда.



Параметры

Фрагмент комбинации опций используется для обозначения последовательности, которая будет иметь место только при определенных условиях, в противном случае последовательность не появится. Он моделирует утверждение «если, то».

Подобно альтернативному фрагменту, фрагмент option также представлен в прямоугольной рамке, где 'opt' помещается внутри поля имени.

В отличие от альтернативного фрагмента, фрагмент опции не делится на два или более операндов. Охранник опции находится в верхнем левом углу.

(Найдите пример диаграммы последовательности с фрагментом опции в разделе «Шаблоны и примеры диаграмм последовательности»).

Петли

Фрагмент цикла используется для представления повторяющейся последовательности. Поместите слова «петля» в поле имени и условие защиты в верхнем левом углу рамки.

В дополнение к логическому тесту, для защиты в фрагменте цикла могут быть проверены два других особых условия. Это минимальные итерации (записываются как minint = [число] и максимальные итерации (записываются как maxint = [число]).

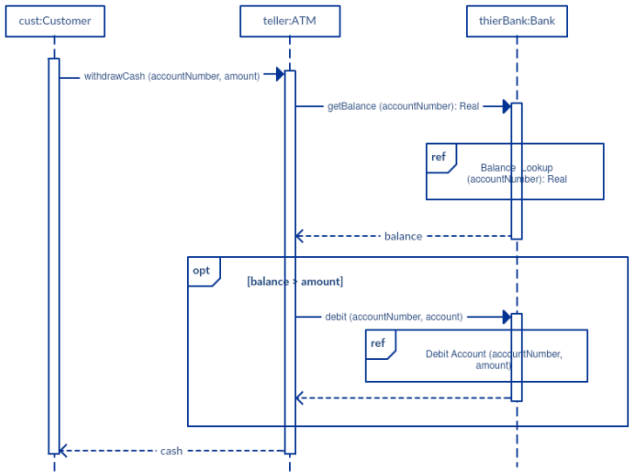
Если это минимальная защита итераций, цикл должен выполняться не меньше указанного числа, а если это максимальная защита итераций, цикл не должен выполняться больше указанного числа.

(Найдите пример фрагмента цикла ниже в шаблонах диаграмм последовательности и в разделе примеров)

Справочный фрагмент

Вы можете использовать ref-фрагмент для управления размером больших диаграмм последовательностей. Это позволяет вам повторно использовать часть одной диаграммы последовательности в другой, или, другими словами, вы можете ссылаться на часть диаграммы на другой диаграмме, используя фрагмент ссылки.

Чтобы указать ссылочный фрагмент, вы должны упомянуть «ref» в поле имени фрейма и имя диаграммы последовательности, на которую имеется ссылка внутри фрейма.



- **Нарисуйте более мелкие диаграммы последовательности, отражающие суть варианта использования.**

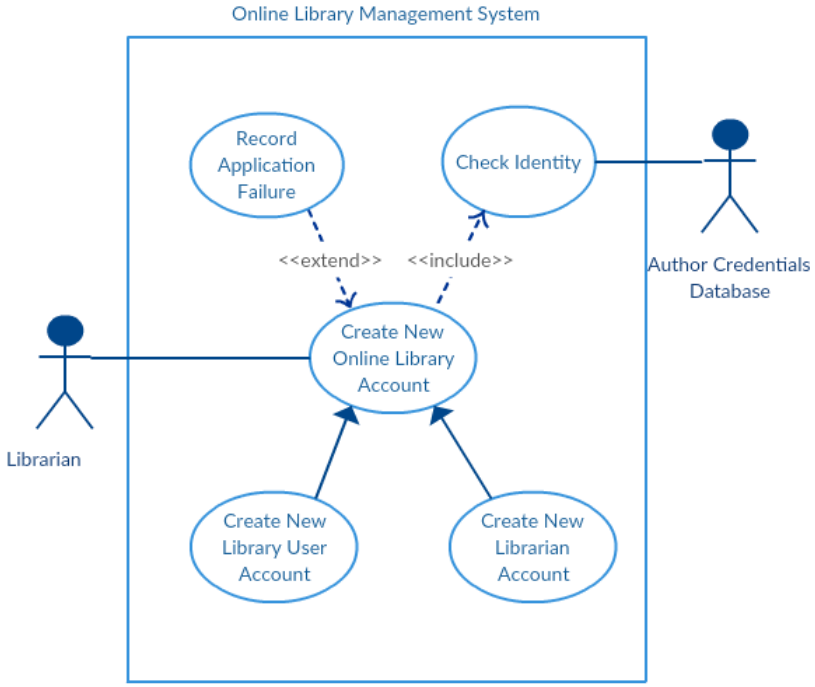
Вместо того, чтобы загромождать диаграмму последовательности несколькими объектами и группами сообщений, которые запутают читателя, нарисуйте несколько диаграмм последовательности меньшего размера, которые точно объясняют, что делает ваша система. Убедитесь, что диаграмма умещается на одной странице и оставляет место для пояснительных примечаний.

Кроме того, вместо того, чтобы рисовать десятки диаграмм последовательностей, выясните, что общего среди сценариев, и сосредоточьтесь на этом. И если код выразительный и может стоять сам по себе, вообще нет необходимости рисовать диаграмму последовательности.

Как нарисовать диаграмму последовательности

Диаграмма последовательности представляет сценарий или поток событий в одном единственном варианте использования. Поток сообщений диаграммы последовательности основан на описании конкретного варианта использования.

Затем, прежде чем вы начнете рисовать диаграмму последовательности или решить, какие взаимодействия должны быть включены в нее, вам нужно нарисовать диаграмму варианта использования и подготовить исчерпывающее описание того, что делает конкретный вариант использования.



Из приведенного выше примера диаграммы варианта использования «Создать новую учетную запись онлайн-библиотеки» мы сосредоточимся на прецеденте под

названием «Создать новую учетную запись пользователя», чтобы нарисовать наш пример диаграммы последовательности.

Прежде чем рисовать диаграмму последовательности, необходимо определить объекты или действующих лиц, которые будут задействованы в создании новой учетной записи пользователя. Это было бы;

- Библиотекарь
- Система управления онлайн-библиотекой
- База данных учетных данных пользователей
- Система электронной почты

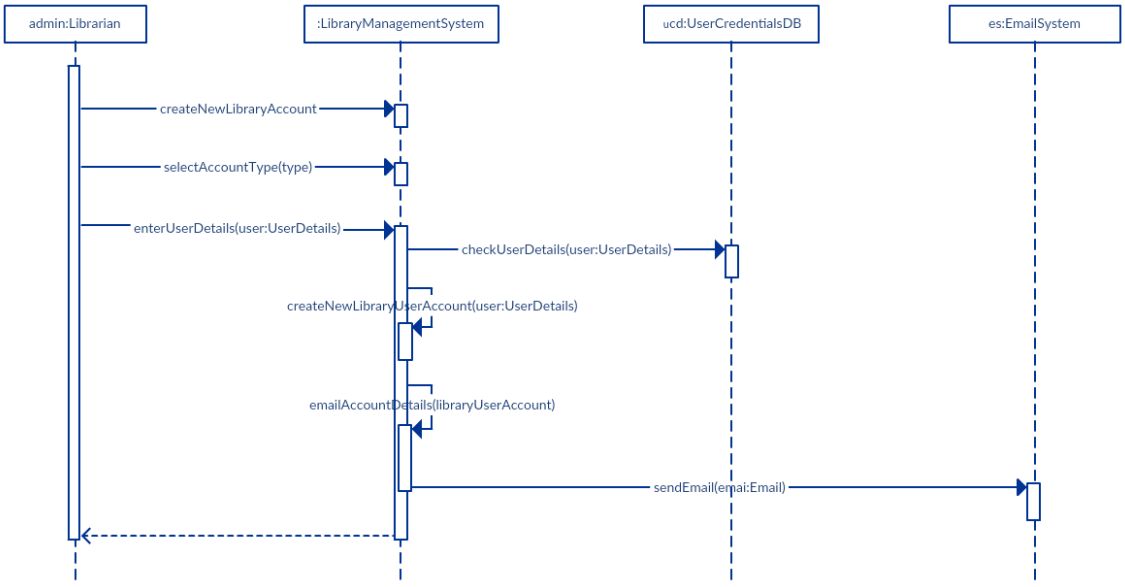
После идентификации объектов важно написать подробное описание того, что делает вариант использования. Из этого описания вы можете легко определить взаимодействия (которые должны идти на диаграмме последовательности), которые будут происходить между указанными выше объектами после выполнения варианта использования.

Вот шаги, которые выполняются в варианте использования под названием «Создать новую учетную запись пользователя библиотеки».

- Библиотекарь запрашивает у системы создание новой учетной записи онлайн-библиотеки.
- Затем библиотекарь выбирает тип учетной записи пользователя библиотеки.
- Библиотекарь вводит данные пользователя
- Данные пользователя проверяются с помощью базы данных учетных данных пользователя.
- Создана новая учетная запись пользователя библиотеки.
- Сводка деталей новой учетной записи затем отправляется пользователю по электронной почте.

На каждом из этих шагов вы можете легко указать, какими сообщениями следует обмениваться между объектами на диаграмме последовательности. Как только все станет ясно, вы можете приступить к рисованию диаграммы последовательности.

На схеме ниже показано, как объекты в системе управления онлайн-библиотекой взаимодействуют друг с другом для выполнения функции «Создать новую учетную запись пользователя библиотеки».



Распространенные ошибки в диаграммах последовательностей

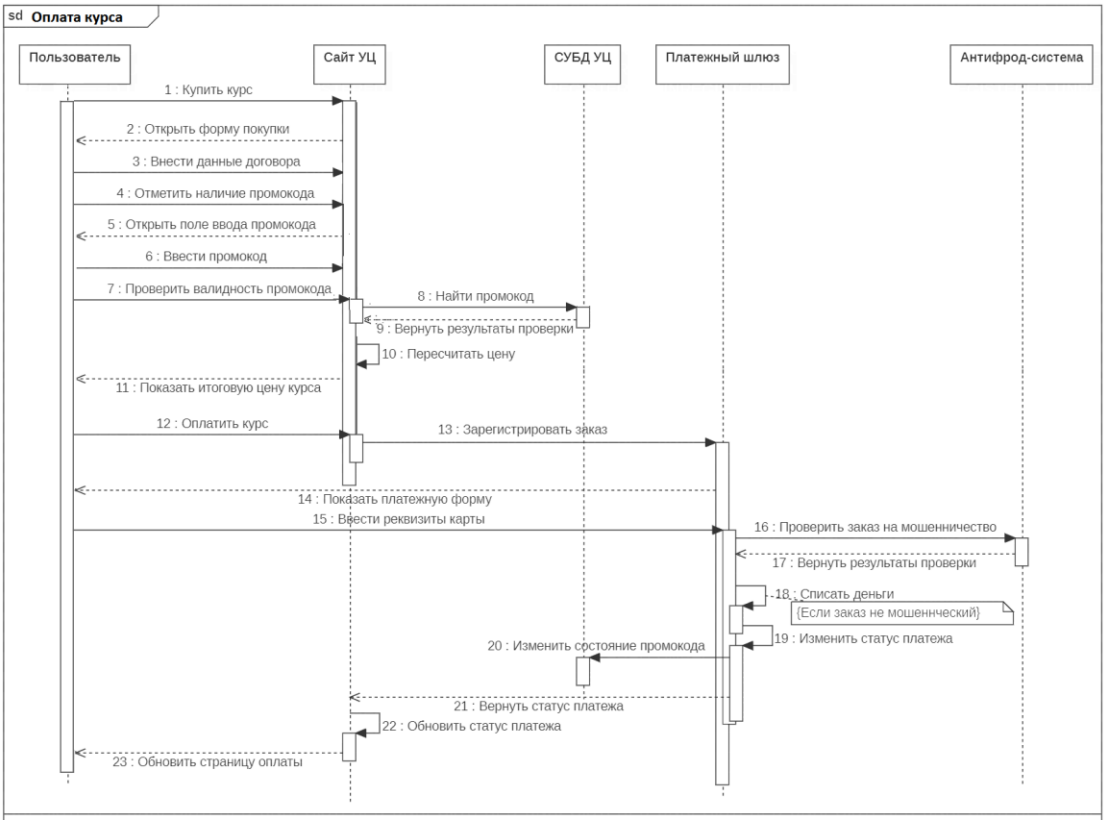
При рисовании схем последовательностей дизайнеры часто допускают эти распространенные ошибки. Избегая этих ошибок, вы можете гарантировать качество вашей диаграммы.

- Добавление слишком большого количества деталей. Это загромождает диаграмму и затрудняет чтение.

- Устаревшие и устаревшие диаграммы последовательности, которые не имеют отношения к интерфейсам, реальной архитектуре и т. Д. Системы. Не забудьте заменить или изменить их.
- Не оставлять пробелов между текстом варианта использования и стрелкой сообщения; это затрудняет чтение диаграммы.
- Не обращая внимания на происхождение стрелок сообщений.

См. Подробное объяснение этих типичных ошибок в Руководстве по диаграммам последовательностей: общие ошибки, которых следует избегать при построении диаграмм последовательности.

Задание: С помощью сервисов по созданию UML диаграмм (Любой из предложенных http://staruml.io, Draw.io, https://creately.com/) подготовить следующие диаграммы последовательностей:



Алгоритм диаграммы последовательности

1. пользователь выбирает курс по бизнес-анализу и нажимает кнопку «Купить»;
2. открывается страница оплаты курса на сайте учебного центра с формой договора покупки курса и чек-боксом о наличии промокода;
3. пользователь вводит свои данные в форму договора покупки курса;
4. пользователь отмечает наличие промокода в чек-боксе;
5. открывается поле ввода промокода и кнопка проверки его валидности;
6. пользователь вводит промкод;
7. пользователь проверяет его валидность, кликнув на кнопку «Проверить»;
8. выполняется проверка валидности промокода с учетом данных по нему, которые имеются в СУБД УЦ.
9. Если промокод валиден (т.е. привязан к выбранному курсу и дата действия его еще не истекла) и находится в состоянии «выдан», цена курса меняется с учетом скидки по промокоду. Иначе цена курса остается прежней и пользователю показывается сообщение о невозможности применить этот код по одной из следующих причин: уже использован, не применим к выбранному курсу или закончился срок его действия.
10. соглашаясь с ценой (пересчитанной или прежней), пользователь нажимает кнопку «Оплатить». При этом в сторону платежного шлюза посылается запрос со всеми

параметрами заказа, где сумма списания равна итоговой цене, рассчитанной с учетом скидки по промокоду, если ее удалось применить.

11. открывается веб-страница платежного шлюза с формой ввода данных банковской карты пользователя и суммой списания;

12. пользователь вводит реквизиты своей банковской карты и нажимает кнопку «Оплатить». При этом на сервер банка в систему «Антифрод» отправляются детали заказа и данные карты, чтобы проверить заказ на мошенничество.

13. в платёжный шлюз возвращаются результаты проверки заказа на мошенничество. Если заказ признан мошенническим, оплата отклоняется и платеж считается неуспешным. Иначе платёжный шлюз списывает деньги со счёта клиента.

14. Платёжный шлюз возвращает сайту УЦ статус платежа (успешный или неуспешный).

15. На странице оплату сайта УЦ отображается статус платежа.

16. При успешном платеже в СУБД УЦ меняется состояние промокода на «Использован».

Задание: выполнить диаграмму последовательности согласно алгоритма:

1. Продавец получает заявку клиента
2. Продавец формирует заказ и передает его Изготовителю продукта.
4. Изготовитель изготавливает продукт.
5. Изготовитель отправляет продукт на Склад и сообщает о готовности Продавцу.
6. Продавец сообщает Клиенту о готовности продукта и принимает от Клиента оплату.
7. Продавец сообщает Отправителю адрес клиента и заказывает транспорт.
8. Отправитель получает продукт со склада и доставляет его клиенту.

Ответить на контрольные вопросы

1. Что такое диаграмма последовательностей?
2. Перечислите основные элементы диаграммы последовательности

Лабораторная работа 28.
Создание диаграммы классов

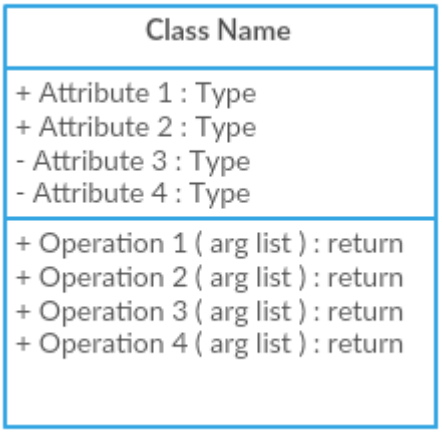
Диаграмма классов – это UML-диаграмма, которая описывает систему, визуализируя различные типы объектов внутри системы и виды статических связей, которые существуют между ними. Он также иллюстрирует операции и атрибуты классов.

Обычно они используются для изучения концепций области, понимания требований к программному обеспечению и описания подробных проектов.

Нотации диаграммы класса с примерами

Существует несколько обозначений диаграмм классов, которые используются при рисовании диаграмм классов UML. Мы перечислили ниже наиболее распространенные нотации диаграммы классов.

Класс



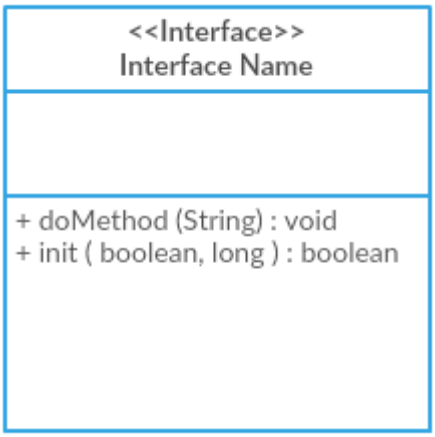
Классы представляют собой центральные объекты в системе. Он представлен прямоугольником с 3 отсеками.

Первый показывает имя класса, а средний – атрибуты класса, которые являются характеристиками объектов. В нижнем списке перечислены операции класса, которые представляют собой поведение класса.

Простой класс

Последние два отсека являются необязательными. Нотация класса без последних двух отделений называется простым классом и содержит только имя класса.

Интерфейс



Символ интерфейса на диаграммах классов обозначает набор операций, которые детализируют ответственность класса.

Пакет

Символ пакета используется для группировки классов или интерфейсов, которые либо похожи по своей природе, либо связаны. Группировка этих элементов дизайна с использованием символов упаковки улучшает читабельность диаграммы

Отношения в диаграмме классов

Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	

Как нарисовать диаграмму классов

Классовые диаграммы идут рука об руку с объектно-ориентированным дизайном. Поэтому знание его основ – ключевая часть умения рисовать хорошие классовые диаграммы.

При необходимости описания статического вида системы или ее функциональных возможностей, потребуется нарисовать диаграмму классов. Вот шаги, которые необходимо выполнить для создания диаграммы классов.

Шаг 1: Определите имена классов

Первым шагом является идентификация первичных объектов системы.

Шаг 2: Различные отношения

Следующий шаг – определить, как каждый из классов или объектов связан друг с другом. Обратите внимание на общие черты и абстракции среди них; это поможет вам сгруппировать их при рисовании диаграммы классов.

Шаг 3: Создать структуру

Сначала добавьте имена классов и свяжите их с соответствующими коннекторами. Добавить атрибуты и функции/методы/операции можно позже.

Классовая диаграмма Лучшие практики

- Классовые диаграммы могут иметь тенденцию к бессвязности по мере их расширения и роста. Лучше всего избегать создания больших диаграмм и разбивать их на более мелкие, которые позже можно будет связать друг с другом. Ты можешь очень легко сделать это с Крейли. Это поможет вам улучшить читабельность ваших диаграмм.
- Используя простую нотацию класса, вы можете быстро создать высокоуровневый обзор вашей системы. Подробная диаграмма может быть создана отдельно по мере необходимости, и даже связана с первой диаграммой для удобства пользования.
- Чем больше линий перекрываются на диаграммах классов, тем более загроможденными они становятся. Читатель только запутается, пытаясь найти путь. Убедитесь, что две линии не пересекаются.
- Используйте цвета для группировки общих модулей. Различные цвета на разных классах помогают читателю различать различные группы.

Примеры диаграмм классов / Шаблоны

Диаграмма класса Пример 1

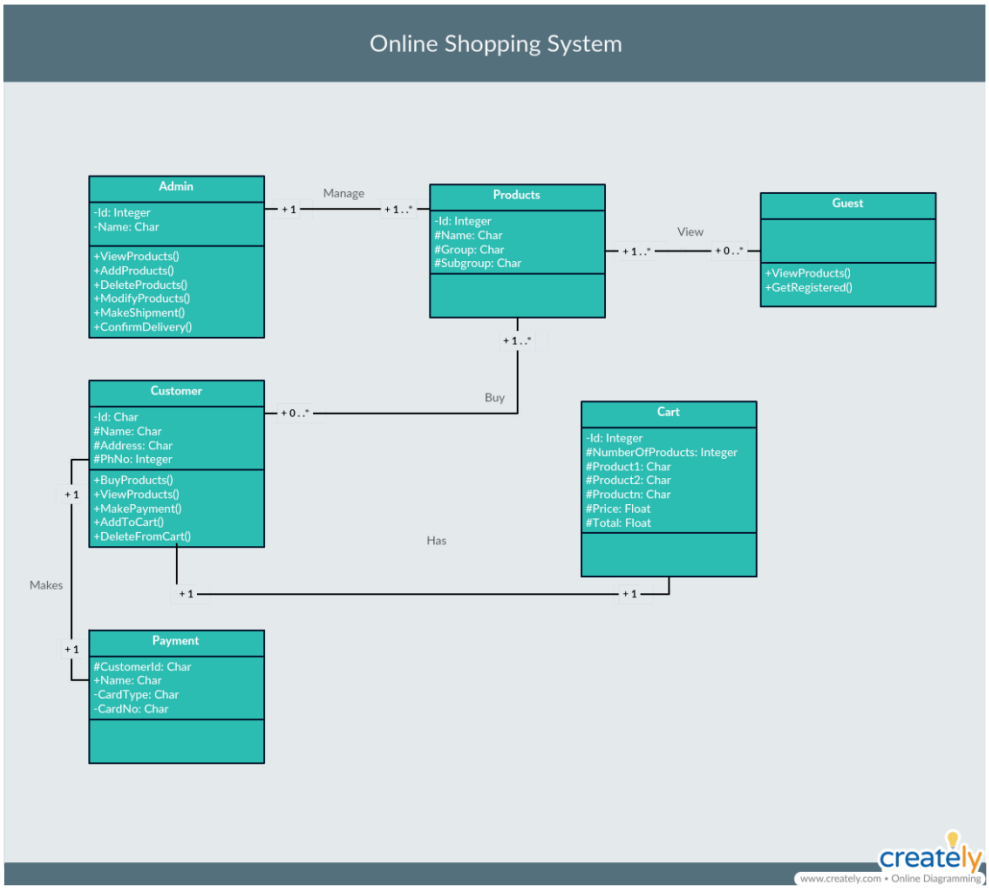


Диаграмма класса Пример 2

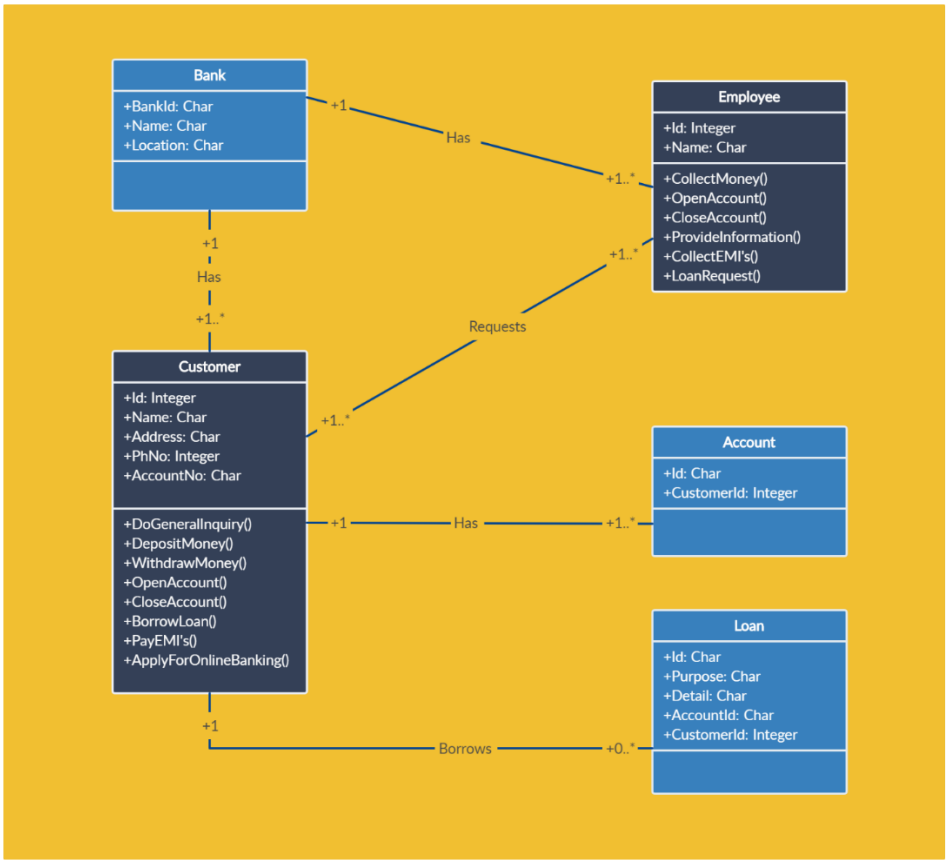


Диаграмма класса Пример 3

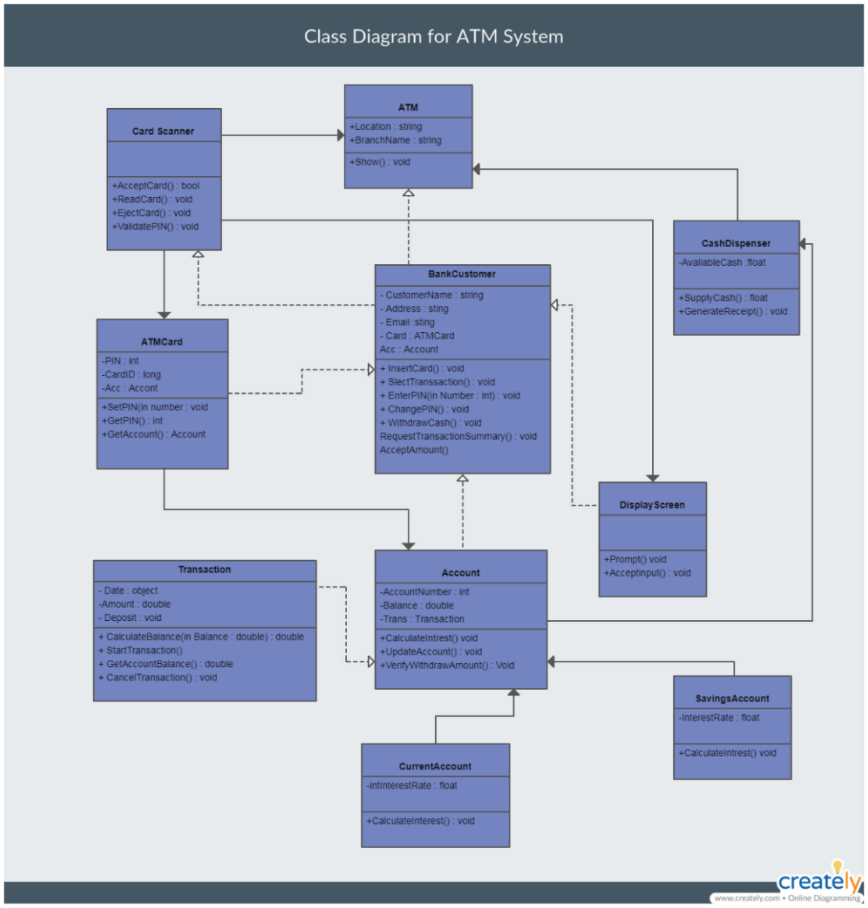


Диаграмма классов для системы банкоматов банка \

Задание: Для заданной предметной области «Туристическое агентство» построить диаграмму классов.

Постановка задачи

Задана предметная область: туристическое агентство. Клиент может выбрать тур на веб-сайте агентства.

Описание бизнес-процессов туристического агентства. Клиент является потенциальным покупателем туристического продукта, взаимодействует с информационной системой через интернет. Турагент реализует клиенту сформированный туроператором тур на тех условиях, которые предлагаются туроператором. Туроператор осуществляет деятельность по формированию, продвижению и реализации туристического продукта. Формирование туристического продукта складывается из бронирования и оплаты отеля, заказа авиарейса, обеспечения услуг по предоставлению транспорта, экскурсионных услуг и т. д. Кроме того, туроператор определяет цены на сформированный им тур и политику скидок. Информационная система предоставляет каталог всех туров. Каталог содержит полную информацию о туре (страна, дата вылета и прилета, количество дней пребывания, стоимость). Клиент может забронировать только тот тур, который присутствует в каталоге, а также оставить пожелания на сайте. Туроператор имеет возможность добавить новый тур в каталог. Турагент оформляет все необходимые документы с клиентом (составляет договор).

Лабораторная работа 29.
Использование стандартного ввода-вывода (iostream)

Программа на C++ состоит из набора инструкций. Каждая инструкция (statement) выполняет определенное действие. В конце инструкции в языке C++ ставится точка с запятой (;). Данный знак указывает компилятору на завершение инструкции. Например:

```
std::cout << "Hello World!";
```

Данная строка выводит на консоль строку "Hello world!", является инструкцией и поэтому завершается точкой с запятой.

Набор инструкций может представлять блок кода. Блок кода заключается в фигурные скобки, а инструкции помещаются между открывающей и закрывающей фигурными скобками:

```
{
   std::cout << "Hello World!";
   std::cout << "Bye World!";
}
```

В этом блоке кода две инструкции, которые выводят на консоль определенную строку.

Функция main

Каждая программа на языке C++ должна иметь как минимум одну функцию - функцию main(). Именно с этой функции начинается выполнение приложения. Ее имя main фиксировано и для всех программ на Си всегда одинаково.

Функция также является блоком кода, поэтому ее тело обрамляется фигурными скобками, между которыми определяется набор инструкций.

В частности, при создании первой программы использовалась следующая функция main:

```
#include <iostream>          // подключаем заголовочный файл iostream
int main()                   // определяем функцию main
```

```
{                       // начало функции
   std::cout << "Hello World!";   // выводим строку на консоль
   return 0;                 // выходим из функции
}                       // конец функции
```

Определение функии main начинается с возвращаемого типа. Функция main в любом случае должна возвращать число. Поэтому ее определение начинается с ключевого слова int.

Далее идет название функции, то есть main. После названия в скобках идет список параметров. В данном случае функция main не принимает никаких параметров, поэтому после названия указаны пустые скобки. Однако есть другие варианты определения функции main, которые подразумевают использование параметров. В частности, нередко может встречаться следующее определение функции main, использующей параметры:

```
int main (int argc, char *argv[])
{

}
```

Директивы препроцессора

В примере выше на консоль выводится строка, но чтобы использовать вывод на консоль, необходимо в начале файла с исходным кодом подключать библиотеку iostream с помощью директивы include.

Директива include является директивой препроцессора. Каждая директива препроцессора размещается на одной строке. И в отличие от обычных инструкцией языка C++, которые завершаются точкой с запятой; , признаком завершения препроцессорной директивы является перевод на новую строку. Кроме того, директива должна начинаться со знака решетки #. Непосредственно директива "include" определяет, какие файлы и библиотеки надо подключить в данном месте в код программы.

Комментарии

Исходный код может содержать комментарии. Комментарии позволяют понять смыл программы, что делают те или иные ее части. При компиляции комментарии игнорируются и не оказывают никакого влияние на работу приложения и на его размер.

В языке C++ есть два типа комментариев: однострочный и многострочный. Однострочный комментарий размещается на одной строке после двойного слеша **//,** Многострочный комментарий заключается между символами **/* текст комментария */**. Он может размещаться на нескольких строках.

Арифметические операции

Арифметические операции производятся над числами. Значения, которые участвуют в операции, называются операндами. В языке программирования C++ арифметические операции бинарными (производятся над двумя операндами) и унарными (выполняются над одним операндом). К бинарным операциям относят следующие:

+ Операция сложения возвращает сумму двух чисел:

```
int a = 10;
int b = 7;
int c = a + b;  // 17
int d = 4 + b;  // 11
```

- Операция вычитания возвращает разность двух чисел:

```
int a = 10;
int b = 7;
int c = a - b;  // 3
int d = 41 - b; // 34
```

*** Операция умножения возвращает произведение двух чисел:**

```
int a = 10;
int b = 7;
int c = a * b;  // 70
int d = b * 5;  // 35
```

/ Операция деления возвращает частное двух чисел:

```
int a = 20;
int b = 5;
int c = a / b;  // 4
double d = 22.5 / 4.5;  // 5
```

При делении стоит быть внимательным, так как если в операции участвуют два целых числа, то результат деления будет округляться до целого числа, даже если результат присваивается переменной float или double:

```
double k = 10 / 4;     // 2
std::cout << k;
```

Чтобы результат представлял число с плавающей точкой, один из операндов также должен представлять число с плавающей точкой:

```
double k = 10.0 / 4;     // 2.5
std::cout << k;
```

% Операция получения остатка от целочисленного деления:

```
int a = 33;
int b = 5;
int c = a % b;  // 3
int d = 22 % 4; // 2 (22 - 4*5 = 2)
```

Также есть две унарные арифметические операции, которые производятся над одним числом: ++ (инкремент) и -- (декремент). Каждая из операций имеет две разновидности: префиксная и постфиксная:

Префиксный инкремент.

Увеличивает значение переменной на единицу и полученный результат используется как значение выражения ++x

```
int a = 8;
int b = ++a;
std::cout << a << "\n"; // 9
std::cout << b << "\n"; // 9
```

Постфиксный инкремент.

Увеличивает значение переменной на единицу, но значением выражения x++ будет то, которое было до увеличения на единицу

```
int a = 8;
int b = a++;
std::cout << a << "\n"; // 9
std::cout << b << "\n"; // 8
```

Префиксный декремент.

Уменьшает значение переменной на единицу, и полученное значение используется как значение выражения --x

```
int a = 8;
int b = --a;
```

```
std::cout << a << "\n"; // 7
std::cout << b << "\n"; // 7
```

Постфиксный декремент.

Уменьшает значение переменной на единицу, но значением выражения x-- будет то, которое было до уменьшения на единицу

```
int a = 8;
int b = a--;
std::cout << a << "\n"; // 7
std::cout << b << "\n"; // 8
```

Условные выражения

Условные выражения представляют собой некоторое условие и возвращают значение типа bool, то есть значение true (если условие истинно), либо значение false (если условие ложно). К условным выражениям относятся операции отношения и логические операции.

Операции отношения

В языке программирования C++ есть следующие операции отношения:

== Операция "равно". Возвращает true, если оба операнда равны, и false, если они не равны:

```
int a = 10;
int b = 4;
bool c = a == b;    // false
bool d = a == 10;       // true
```

> Операция "больше чем". Возвращает true, если первый операнд больше второго, и false, если первый операнд меньше второго:

```
int a = 10;
int b = 4;
bool c = a > b;  // true
```

< Операция "меньше чем". Возвращает true, если первый операнд меньше второго, и false, если первый операнд больше второго:

```
bool c = 10 < 4; // false
```

<= Операция "меньше или равно". Возвращает true, если первый операнд меньше или равен второму, и false, если первый операнд больше второго:

```
bool c = 10 <= 4;    // false
bool d = 10 <= 14;   // true
```

>= Операция "больше или равно". Возвращает true, если первый операнд больше или равен второму, и false, если первый операнд меньше второго:

```
bool c = 10 >= 4;    // true
bool d = 10 >= 14;   // false
```

!= Операция "не равно". Возвращает true, если первый операнд не равен второму, и false, если оба операнда равны:

```
bool c = 10 != 4;   // true
bool d = 4 != 4;    // false
```

Обычно операции отношения применяются в условных конструкциях типа if...else, которые будут рассмотрены далее.

Логические операции

Логические операции обычно объединяют несколько операций отношения. К логическим операциям относят следующие:

! (операция отрицания) - Унарная операция, которая возвращает true, если операнд равен false. Если операнд равен true, операция возвращает false.

```
bool a = true;
bool b = !a;    // false
bool c =!10<5;   // true
```

&& (конъюнкция, логическое умножение) - Возвращает true, если оба операнда не равны false. Возвращает false, если хотя бы один операнд равен false.

```
bool a = true;
bool b = false;
bool c = a && b;    // false
bool d = a && true; // true
```

|| (дизъюнкция, логическое сложение) - Возвращает true, если хотя бы один операнд равен true. Возвращает false, если оба операнда равны false.

```
bool a = true;
bool b = false;
bool c = a || b;        // true
bool d = b || false;    // false
```

Операции присваивания

Все остальные операции присваивания являются сочетанием простой операции присваивания с другими операциями:

+=: присваивание после сложения. Присваивает левому операнду сумму левого и правого операндов: A += B эквивалентно A = A + B

-=: присваивание после вычитания. Присваивает левому операнду разность левого и правого операндов: A -= B эквивалентно A = A - B

*=: присваивание после умножения. Присваивает левому операнду произведение левого и правого операндов: A *= B эквивалентно A = A * B

/=: присваивание после деления. Присваивает левому операнду частное левого и правого операндов: A /= B эквивалентно A = A / B

%=: присваивание после деления по модулю. Присваивает левому операнду остаток от целочисленного деления левого операнда на правый: A %= B эквивалентно A = A % B

<<=: присваивание после сдвига разрядов влево. Присваивает левому операнду результат сдвига его битового представления влево на определенное количество разрядов, равное значению правого операнда: A <<= B эквивалентно A = A << B

>>=: присваивание после сдвига разрядов вправо. Присваивает левому операнду результат сдвига его битового представления вправо на определенное количество разрядов, равное значению правого операнда: A >>= B эквивалентно A = A >> B

&=: присваивание после поразрядной конъюнкции. Присваивает левому операнду результат поразрядной конъюнкции его битового представления с битовым представлением правого операнда: A &= B эквивалентно A = A & B

|=: присваивание после поразрядной дизъюнкции. Присваивает левому операнду результат поразрядной дизъюнкции его битового представления с битовым представлением правого операнда: A |= B эквивалентно A = A | B

^=: присваивание после операции исключающего ИЛИ. Присваивает левому операнду результат операции исключающего ИЛИ его битового представления с битовым представлением правого операнда: A ^= B эквивалентно A = A ^ B

Пространства имен и using

При чтении и записи в предыдущих темах использовались объекты std::cout и std::cin соответственно. Причем они использовались с префиксом std::. Этот префикс указывает, что объекты cout, cin, endl определены в пространствен имен std. А само двойное двоеточие :: представляет оператор области видимости (scope operator), который позволяет указать, в каком пространсте имен определен объект. И без префикса эти объекты по умолчанию мы использовать не можем.

Однако подобная запись может показаться несколько громоздкой. И в этом случае можно использовать оператор using, который позволяет ввести в программу объекты из различных пространств имен.

Использование оператора using имеет следующй формат:

using пространство_имен::объект

Практическая работа

1. Скачать и установить Dev-C++
2. Изучить основные типы данных.
3. Решить следующие задачи: Задание выбрать по порядковому номеру студента в списке журнала группы

Задание №1

Вариант 1. Составить программу, находящую периметр равнобедренного

треугольника по его основанию, a и высоте h, проведенной к основанию (a и h — вещественные). Для нахождения боковой стороны b треугольника использовать теорему Пифагора: b2 = (a/2)2 + h2.

Вариант 2. Длина выражена в сантиметрах. Выразить ее в дюймах. (1 дюйм=2.5 см)

Вариант 3. Составить программу, находящую площадь кольца, заключенного между двумя окружностями с общим центром и радиусами R1 и R2 (R1 и R2 — вещественные, R1 > R2). Воспользоваться формулой площади круга радиуса R: S = π*R2.

Вариант 4. Составить программу, находящую величину угла в радианах R, если дана его величина D в градусах (D — вещественное число, O < D < 360). Воспользоваться следующим соотношением: R= π D/180° .

Вариант 5. Приобрели A шт. книг по цене B руб. за шт. и C шт. тетрадей по цене D руб. за шт. Определить стоимость всего товара.

Вариант 6. Дано, a и в - стороны прямоугольника. Найти его площадь и периметр. Найти площадь прямоугольного треугольника, построенного на катетах, a и в (площадь прямоугольного треугольника рассчитывается как половина произведения сторон катетов).

Вариант 7. Найти среднеарифметическое трех чисел x,y, z . Известен объем информации в байтах. Выразить его в мегабайтах и гигабайтах.

Вариант 8. Дано R - радиус окружности. Найти длину окружности (длина

окружности прямо пропорциональна удвоенному произведению Пи на радиус окружности).

Вариант 9. На базу завезли С кг арбузов по цене 2 руб. за килограмм и В кг дынь по цене 5 руб. за килограмм. Определить, сколько всего стоят арбузы и дыни.

Вариант 10.Дано а и в - стороны прямоугольника. Найти его площадь и периметр. Дано значение веса. Перевести значение веса, выраженное в граммах, в унции (1 унция =28.3 г)

Вариант 11.Составить программу, находящую площадь круга радиуса R (R — вещественное). по заданному радиусу. Площадь круга радиуса R вычисляется по формуле S = π*R2.

Вариант 12.На базу завезли А кг яблок по цене 20 руб. за килограмм и В кг груш по цене 30 руб. за килограмм. Определить, сколько всего стоят яблоки и груши.

Вариант 13.Составить программу , вычисляющую среднее арифметическое AMeaп = (X+ У)/2 и среднее геометрическое GMeaп = Sqrt(X*У) двух положительных чисел X и У

Вариант 14.Составить программу, вычисляющую по стороне а равностороннего треугольника его периметр P = 3*a и площадь S = a2*Sqrt(3)/4

Вариант 15.Составить программу, определяющую по времени T (в секундах) содержащееся в нем количество часов H, минут M и секунд S (T — входная, H, M и S — выходные переменные целого типа).

Задание № 2

Вариант 1. Переставить местами значения 3-х вещественных переменных x,y,z, так чтобы в x оказалось значение y , в y - значение z, а z - прежнее значение x.

Вариант 2. Ввести в 3 переменные символьного типа c1,c2,c3 аббревиатуру вашей специальности. Переставить местами значения 3-х переменных, так чтобы в C1оказалось значение c2 , в c2 - значение c3, а c3 - прежнее значение c1.

Вариант 3. Ввести 3 переменные символьного типа c1,c2,c3 . Переставить местами значения 3-х переменных, так чтобы в C1 оказалось значение c2 , в c2 - значение c3, а c3 - прежнее значение c1.

Вариант 4. Ввести 2 переменные строкового типа s1,s2 . Переставить местами значения 2-х переменных, так чтобы в s1 оказалось значение s2 , в s2 – прежнее значение s1.

Вариант 5. Ввести в 3 переменные строкового типа s1,s2,s3 ваши фамилию, имя, отчество. Переставить местами значения 3-х переменных, так чтобы в s1 оказалось значение s2 , в s2 - значение s3. , в s3 - прежнее значение s1.

Вариант 6. Ввести в 3 переменные символьного типа c1,c2,c3 ваши инициалы. Переставить местами значения 3-х переменных, так чтобы в C1 оказалось значение c2, в c2 - значение c3, а c3 - прежнее значение c1.

Вариант 7. Ввести в 3 переменные символьного типа c1,c2,c3 аббревиатуру DVD. Переставить местами значения 3-х переменных, так чтобы в C1 оказалось значение c2 , в c2 - значение c3, а c3 - прежнее значение c1.

Вариант 8. Переставить местами значения 4-х вещественных переменных

x,y,z,q так чтобы в x оказалось значение y , в y - значение z, в z - значение q, а в q - прежнее значение x.

Вариант 9. Ввести в 2 переменные целого типа a, в ваш возраст и ваш рост.

Переставить местами значения 2-х переменных, так чтобы в a оказалось значение в, в в - прежнее значение a.

Вариант 10.Ввести в 3 переменные вещественного типа c1,c2,c3 ваш год рождения, число и месяц рождения. Переставить местами значения 3-х переменных, так чтобы в c1 оказалось значение c2 , в c2 - значение c3, а c3 - прежнее значение c1.

Вариант 11.Ввести в 2 переменные символьного типа s1,s2 название вашей группы и университета. Переставить местами значения 2-х переменных, так чтобы в s1 оказалось значение s2 , в s2 - прежнее значение s1.

Вариант 12.Ввести в 3 переменные вещественного типа a,b,c ваши число, месяц и год рождения. Переставить местами значения 3-х переменных, так чтобы в a оказалось значение b, в b - значение c , а c - прежнее значение a.

Вариант 13.Объявить 2 переменные логического типа b1, b2. С Помощью оператора присвоения присвоить b1 – значение «истина», b2 – значение «ложь». Переставить их значения между собой.

Вариант 14.Переставить местами значения 4-х целых переменных x,y,z,q так чтобы в x оказалось значение y , в y - значение z, в z - значение q, а в q – прежнее значение x.

Вариант 15.Переставить местами значения 3-х переменных x,y,z, так чтобы в x оказалось значение y , в y - значение z, а z - прежнее значение x.

Задание № 3

Вариант 1. Ввести два числа целого типа a и в. Найти остаток от деления a на в.

Вариант 2. Ввести два числа целого типа a и в. Поделить нацело в на a (целочисленное деление).

Вариант 3. Ввести число целого типа a . (a - не менее чем двухзначное) Найти остаток от деления a на 10.

Вариант 4. Поделить нацело сумму a и в на разность c и d.

Вариант 5. Дано Q целое. Найти остаток от деления Q на 3.

Вариант 6. Дано Q целое. Найти целую часть от деления Q на 3.

Вариант 7. Дано R целое. Найти целую часть от деления R на 2.

Вариант 8. Ввести 3 числа целого типа a ,c и в Поделить нацело c на сумму чисел в и a. (целочисленное деление)

Вариант 9. Найти целую часть от деления суммы целых чисел A и B на число D.

Вариант 10.Ввести числа целого типа A,B,C Найти остаток от деления суммы целых чисел A и B на число C.

Вариант 11.Найти остаток от деления суммы a и в на сумму c и d.

Вариант 12.Дано Q целое. Найти остаток от деления Q на 2.

Вариант 13. Ввести числа целого типа A,B,C Найти целую часть от деления числа A на сумму чисел B и D.

Вариант 14.Ввести числа целого типа A,B,C Найти остаток от деления разности чисел A и B на число C.

Вариант 15.Ввести 2 числа C и D. Найти остаток от деления целого числа C на число D.

Задание № 4

Вариант 1. Ввести 3 числа вещественного типа x и y, z. Найти сумму дробных частей этих 3-х чисел.

Вариант 2. Ввести 3 числа вещественного типа x и y, z. Найти сумму целых частей этих 3-х чисел.

Вариант 3. Ввести 2 числа вещественного типа x и y. Найти округленную сумму этих 2-х чисел.

Вариант 4. Ввести число x - вещественное. Округлить x до ближайшего целого.

Вариант 5. Ввести число x - вещественное. Отбросить дробную часть от x.

Вариант 6. Ввести число x - вещественное. Отбросить целую часть от x.

Вариант 7. Найти дробную часть среднего арифметического трех заданных положительных чисел.

Вариант 8. Ввести 3 числа вещественного типа x и y,z. Найти целую часть суммы этих 3-х чисел. Найти дробную часть среднего геометрического трех заданных положительных чисел.

Вариант 9. Ввести 3 числа вещественного типа x и y,z. Найти дробную часть суммы этих 3-х чисел.

Вариант 10.Ввести 3 числа вещественного типа x и y,z. Округлить до ближайшего целого сумму этих 3-х чисел.

Вариант 11.Ввести число z - вещественное. Найти дробную часть от числа z , а также округлить до ближайшего целого число z.

Вариант 12.Ввести число x - вещественное. Найти отдельно целую и дробную части числа x.

Вариант 13.Ввести 3 числа вещественного типа x и y,z. Отбросить дробную часть от суммы этих 3-х чисел.

Вариант 14.Ввести 2 числа вещественного типа x и y. . Найти целую часть разности этих 2-х чисел.

Вариант 15.Ввести 2 числа вещественного типа x и y. . Найти дробную часть разности этих 2-х чисел.

Лабораторная работа 30.
Решение задачи с использованием операторов ветвления и цикла

Для программирования ветвящихся алгоритмов в языке Си имеется несколько различных средств. К ним относятся рассмотренная выше операция условия ?:, условный оператор if и оператор выбора switch.

Условный оператор. Формат условного оператора, следующий:

if (выражение) оператор1; else оператор2;

Это полная форма оператора, программирующая структуру полного ветвления. Обычно выражение — это некоторое условие, содержащее операции отношения и логические операции. Значение выражения приводится к целому и интерпретируется в соответствии с правилом: равно нулю — ложь, не равно нулю — истина. Если выражение истинно, выполняется оператор 1, если ложно — оператор 2.

Необходимо обратить внимание на следующие особенности синтаксиса условного оператора:

- выражение записывается в круглых скобках;
- точка с запятой после оператора 1 ставится обязательно.

Возможно использование неполной формы условного оператора

if (выражение) оператор;

Вот пример использования полной формы условного оператора для нахождения большего значения из двух переменных a и b:

if(a>b) max=a; else max=b;

Та же самая задача может быть решена с использованием неполного ветвления следующим образом:

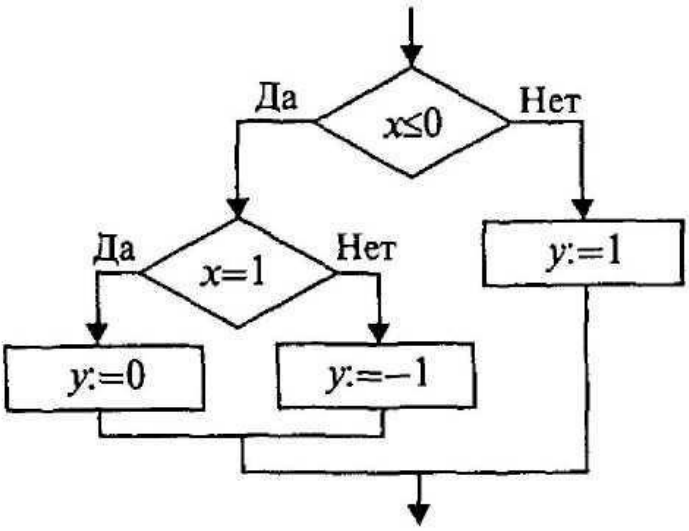
max=a; if(b>a) max=b;

Теперь рассмотрим примеры программирования вложенных ветвящихся структур. Требуется вычислить функцию sign(x) — знак x, которая определена следующим образом:

$$\text{sign}(x) = \begin{cases} -1, & \text{если } x < 0, \\ 0, & \text{если } x = 0, \\ 1, & \text{если } x > 0. \end{cases}$$

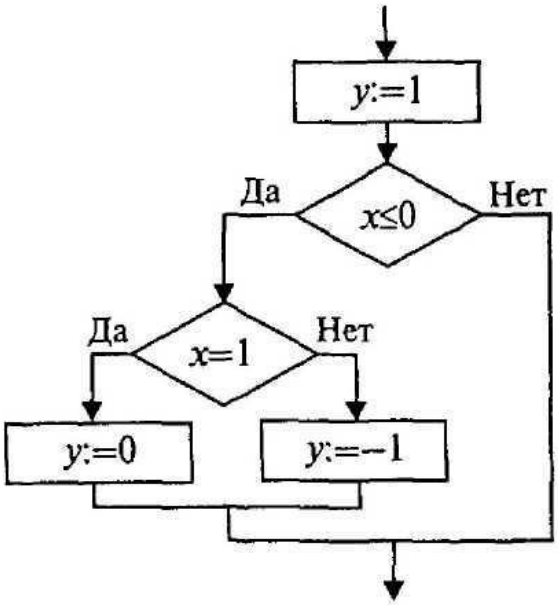
Пример 1. Алгоритм с полными вложенными ветвлениями:

```
if(x<=0)
if(x==0)   y=0;
else  y=-1; else  y=l;
```



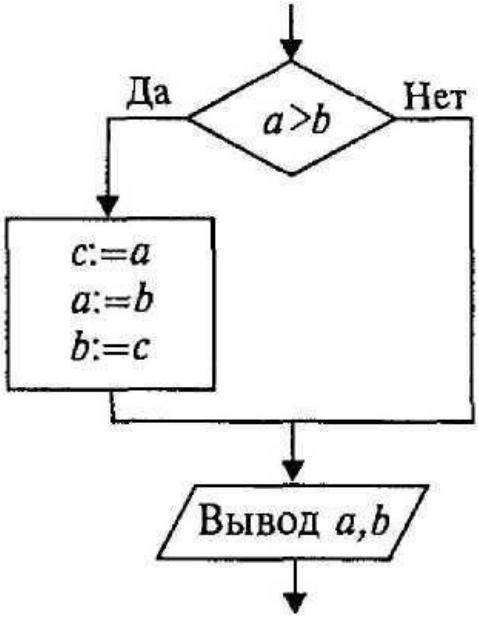
Пример 2. Алгоритм с неполным ветвлением:

```
if(x<=0)
if(x==0)   y=0;
else  y=-l;
```



Пример 3. Упорядочить по возрастанию значения в двух переменных a, b:

```
if(a>b)
{c=a;   a=b;   b=c; }
cout<<"a="<<a<<"b="<<b;
```



В данном примере использован составной оператор — последовательность операторов, заключенная в фигурные скобки. В С фигурные скобки выполняют роль операторных скобок по аналгии с Begin, End в Паскале.

Обратите внимание на то, что перед закрывающей фигурные скобкой точку с запятой надо ставить обязательно, а после скобки точка с запятой не ставится.

В следующем примере вернемся к задаче вычисления площади треугольника по длинам трех сторон. Добавим в программу проверку условия правильности исходных данных: a, b, c должны быть положительными, а сумма длин каждой пары сторон треугольника должна быть больше длины третьей

Пример 4.

```
// Площадь треугольника
#include <iostream.h>
#include <math.h>
void main()
{float a,b,c,p,s;
cout<<"\na="; cin>>a;
cout<<"\nb="; cin>>b;
cout<<"\nc="; cin>>c;
if (a>0 && b>0 && c>0 && a+b>c && a+c>b && b+c>a)
{ p=(a+b+c)/2;
s=sqrt(p*(p-a)*(p-b)*(p-c));
cout<<"\nrПлощадь  треугольника="<<s;
}
else cout("\n Неверные исходные данные.");
}
```

Оператор выбора (переключатель). Формат оператора выбора:

```
switch (целочисленное_выражение)
 { case константа1: список___операторов;
   case константа2: список_операторов;
……..
default: список_операторов; }
```

Последняя строка (default) может отсутствовать. Выполнение оператора происходит в следующем порядке:

1. Вычисляется выражение.

2. Полученное значение последовательно сравнивается с константами, помещенными после служебного слова case; при первом совпадении значений выполняются операторы, стоящие после двоеточия.

3. Если ни с одной из констант совпадения не произошло, то выполнятся операторы после слова default.

Для того чтобы «обойти» выполнение операторов на последующих ветвях, нужно принять специальные меры, используя операторы выхода или перехода.

Рассмотрим фрагмент программы, который переводит числовую оценку знаний ученика в ее словесный эквивалент. Согласно вузовской системе: 5 — «отлично», 4 — «хорошо», 3 — «удовлетворительно», 2 — «неудовлетворительно».

Пример 5.

```
#include <iostream.h>
 void main()
{
int ball;
cout<<"\nВведите оценку:  ";
cin>>ball;
 switch  (ball)
{ case 2: cout<<"\tЭто неудовлетворительно!\n"; break;
case 3: cout<<"\tЭто удовлетворительно!\n"; break;
case 4: cout<<"\t Это хорошо !\n"; break;
case 5: cout<<"\tЭто отлично!\n"; break;
default: cout<<"\tНеТ такой оценки!\n";
}
}
```

Здесь используется еще один новый для нас оператор break — оператор выхода. Его исполнение завершает работу оператора выбора, т.е. происходит «обход» других ветвей. Вот два варианта результатов выполнения этой программы:

```
Введите оценку: 3          Это удовлетворительно!
Введите оценку: 7          Нет такой оценки!
```

Если на всех ветвях убрать оператор break, то результат может выглядеть следующим образом:

```
Введите оценку: 3          Это удовлетворительно!
Это хорошо!
Это отлично!
Нет такой оценки!
```

В этом случае выполнились операторы на всех ветвях, начиная с той, которая помечена константой 3.

Возможны задачи, в которых такой порядок выполнения ветвей оператора выбора может оказаться полезным. В следующем фрагменте программы происходит возведение вещественного числа x в целую степень n, где n изменяется в диапазоне от 1 до 5.

```
y=1.0;
switch(n)
{ case 5:y=y*x;
case 4: y=y*x;
case 3: y=y*x;
case 2: y=y*x;
case 1: y=y*x; cout<<"y="<<y; break;
default: cout<<"Степень больше 5";
}
```

Операторы цикла с постусловием (do…while)

Иногда возникает необходимость, чтобы тело цикла выполнилось хотя бы один раз, но условие не позволяет этому случится. Вот именно тут нам поможет цикл с постусловие **do…while.** Который реализован следующей конструкцией:

do оператор **while (**выражение**);**

Цикл с постусловием — цикл, в котором условие проверяется после выполнения тела цикла. Отсюда следует, что тело всегда выполняется хотя бы один раз.

Работает цикл следующим образом. В начале выполняется оператор, затем вычисляется значение выражения. Если оно истинно, оператор тела цикла выполняется еще раз.

Рассмотрим код на примере прошлой программы, изменив некоторые строки.

C++

```
    #include "stdafx.h"
#include <iostream>
using namespace std;
int main ()
{
int x, y, i, pen; //описаваем нужные переменные
x=0; //стартовое значение переменной
y=56; //определяем пенсионный возраст
do
{
cout<<"How old are you? \n";
cin>>i;
pen=y-i;
cout<<"Remained until retirement (years) = "<<pen<<endl;
x++; //изменение параметра цикла (аналогично x=x+1)
}
while (x<=-1); //условие ложно, но цикл выполняется 1 раз
system ("pause");
return 0;
}
```

Задание 1. Составить программу на языке C++ согласно варианта

Задачи для индивидуального решения:

1) $y = \begin{cases} m^2 + 4n, m > n \\ mn, m \le n \end{cases}$

2) $y = \begin{cases} m + 2n, m > n \\ m^2 + 4, m \le n \end{cases}$

3) $y = \begin{cases} m^3 + 2, m > 0 \\ m^2, m \le 0 \end{cases}$

4) $y = \begin{cases} m - n, m > n \\ n - m, m \le n \end{cases}$

5) $y = \begin{cases} n^5, m > n \\ mn, m \le n \end{cases}$

6) $y = \begin{cases} a^2 - b, a > b \\ b^4 - a, a \le b \end{cases}$

7) $y = \begin{cases} 4n^4, m > n \\ m + n, m \le n \end{cases}$

8) $y = \begin{cases} a + 3b, a > b \\ a^2 b, a \le b \end{cases}$

9) $y = \begin{cases} a + b, a > b \\ a + b^2, a \le b \end{cases}$

10) $y = \begin{cases} a^2 + b, a > b \\ a^3 + b, a \le b \end{cases}$

11) $y = \begin{cases} mn^2, m > 0 \\ n^4 - m, m \le 0 \end{cases}$

12) $y = \begin{cases} a^3 - 4ab, a > 0 \\ b^2 + 4a, a \le 0 \end{cases}$

13) $y = \begin{cases} a - b, a > b \\ b - a^3, a \le b \end{cases}$

14) $y = \begin{cases} m^3, m > n \\ (m + n)^3, m \le n \end{cases}$

15) $y = \begin{cases} m - \frac{n}{2}, m > n \\ mn, m \le n \end{cases}$

Задание 2. Составить программу с использованием оператора **switch** на языке C++ Задачи повышенной сложности

1. Написать алгоритм, позволяющий получить словесное наименование школьных оценок.
2. Написать алгоритм, классифицирующий треугольники (остроугольные, прямоугольные, тупоугольные), если даны углы.
3. Написать алгоритм, который по номеру дня недели - целому числу от 1 до 7 выдавать в качестве результата количество уроков в классе в соответствующий день.
4. Написать алгоритм нахождения числа дней в месяце, если даны: Номер месяца n - целое число a, равное 1 для високосного года и равное 0 в противном случае.
5. По номеру дня недели вывести его название.
6. В зависимости от того введена ли открытая скобка или закрытая, напечатать "открытая круглая скобка" или "закрытая фигурная скобка". (Учитывать круглые, квадратные, фигурные скобки).
7. В зависимости от введённого символа L, S, V программа должна вычислять длину окружности; площадь круга; объём цилиндра.
8. По введенному числу от 0 до 15 вывести название цвета, соответствующего этому коду.
9. Определить, является ли введенная буква русского алфавита гласной.
10. Написать программу, которая бы предлагала меню для вывода графических объектов, и рисовала бы соответствующий выбору.
11. Придумайте шуточный тест с выдачей шуточных результатов.
12. Напишите программу, которая по введенному числу из промежутка 0..24, определяет время суток.
13. Напишите программу, которая по введенному номеру месяца високосного или невисокосного года, выводит количество дней в месяце.

Задание 3. Решить следующие задачи с использованием оператора цикла постусловия:

1.Вводятся десять натуральных чисел больше 2. Посчитать, сколько среди них нечетных чисел.

2. Вывести на экран таблицу умножения (от 1 до 9)

Задание 4. Выбрать индивидуальное задание в соответствии с номером в журнале. Решить следующие задачи с использованием оператора цикла постусловия

1. Составить программу нахождения суммы натуральных двухзначных четных чисел.
2. Написать программу, вычисляющую сумму и среднее арифметическое 50 первых натуральных чисел.
3. Составить программу, выводящую на экран квадраты первых 20 натуральных чисел.
4. Составить программу, определяющую количество элементов кратных 5, в последовательности 100 натуральных чисел.
5. Составить программу нахождения среднего арифметического чисел, кратных 3 для ряда из 100 первых натуральных чисел
6. В введенном промежутке натуральных чисел найти те, количество делителей у которых не меньше введенного значения. Для найденных чисел вывести на экран количество делителей и все делители.
7. Вывести на экран, из каких простых множителей состоит введенное натуральное число.
8. Найти все совершенные числа до 10000. Совершенное число - это такое число, которое равно сумме всех своих делителей, кроме себя самого. Например, число 6 является совершенным, т.к. кроме себя самого делится на числа 1, 2 и 3, которые в сумме дают 6.
9. Вывести какой-либо символ по диагоналям воображаемого квадрата.
10. Среди натуральных чисел, которые были введены, найти наибольшее по сумме цифр. Вывести на экран это число и сумму его цифр.
11. Вывести на экран "прямоугольник", образованный из двух видов символов. Контур прямоугольника должен состоять из одного символа,
12. а в "заливка" - из другого.
13. Доказать гипотезу Сиракуз на диапазоне чисел. Гипотеза Сиракуз утверждает, что любое натуральное число сводится к единице в результате повторения следующих действий над самим числом и результатами этих действий. • Если число четное следует разделить его на 2. • Если нечетное, то умножить его на 3, прибавить 1 и разделить на 2.
14. Посчитать, сколько раз встречается определенная цифра в введенной последовательности чисел. Количество вводимых чисел и цифра, которую необходимо посчитать, задаются вводом с клавиатуры.
15. В первый день спортсмен пробежал x километров, а затем он каждый день увеличивал пробег на 10% от предыдущего значения. По данному числу y определите номер дня, на который суммарный пробег спортсмена составит не менее y километров. Например, при вводе 10 100 программа должна вывести 8.

Лабораторная работа 31.
Решение задач с использованием одномерных и двумрных массивов

Цель работы: Получение навыков в работе с массивами и выполнения над ними базовых операций

Пояснения к работе

Массив – это коллекция переменных одинакового типа, обращение к которым происходит по общему для всех имени. Язык C++ позволяет организовывать массивы различных размерностей: одномерные и многомерные (двухмерные, трехмерные и т.д.).

Одномерные массивы

Одномерный массив - это список связанных переменных, занимающий непрерывную область памяти. Для объявления одномерного массива используется

следующая форма записи:

тип имя_массива [размер];

Тип определяет тип каждого элемента массива. **Размер** определяет количество элементов массива. Например, согласно описанию

int A[10];// резервируется область памяти для 10 элементов типа int

объявлен массив с именем **A**, содержащий 10 элементов целого типа.

При объявлении массива в качестве размера можно указывать только целую константу. Индекс в [ ] описывает позицию элемента в массиве. Так, первым элементом является **A[0],** а последним **A[9].**

При выполнении программы границы массива не контролируются. Если мы ошиблись и вместо **A[9]=50** напишем **A[10]=50** (для массива объявленного выше) компилятор не выдаст ошибку, но это приведет к изменению значений других переменных и результат выполнения программы будет не предсказуем.

Все элементы массива можно также инициализировать одним значением:

int MyNumbers [5] = {100}; // инициализировать все элементы значением 100

Размер массива может явно не указываться, если при его объявлении производится инициализация значений элементов. Например:

int p[]={2, 4, 6, 10, 1};

В этом случае создается массив из пяти элементов со следующими значениями:

p[0]=2, p[1]=4, p[2]=6, p[3]=10, p[4]=1.

Двухмерные массивы

При объявлении двухмерного массива для каждой размерности (количество строк и столбцов) используются отдельные квадратные скобки. Например:

int num[3][4];

При инициализации многомерных массивов каждая размерность должна быть заключена в фигурные скобки:

```
double temp[2][3] = {    { 3.2, 3.3, 3.4 },
    { 4.1, 3.9, 3.9 } };
```

Последовательность выполнения работы

1. Создать новое приложение
2. Ввести текст программы для следующего задания:

В массиве хранится 8 целых чисел. Поменять местами значения первого и последнего элемента массива Массив вывести.

3. Проверить работу программы с различными входными числовыми значениями.

```
#include <iostream>
int main()
{ using namespace std;
const n=8;
int a[n]={25,3,16,-2,1,10,12,10 };
int i, k;
for (i=0; i<n; i++)
cout<<a[i]<<"  ";
cout<<endl;
k=a[0];a[0]=a[n-1];a[n-1]=k;
for (i=0; i<n; i++) cout<<a[i]<<"  ";
cout<<endl;
return 0;
}
```

Практические задания

1	Одномерный массив из 10 элементов заполнить числовыми значениями. Ввести число К. Определить количество элементов массива, равных К. Определить, является ли массив унимодальным, (т.е. содержит ли он элементы только одного знака).
	Двухмерный массив из 4 строк и 3 столбцов заполнить числовыми значениями. Ввести А (номер строки) и В (номер столбца). Заменить на ноль все элементы
2	Одномерный массив из 9 элементов заполнить числовыми значениями. Все отрицательные значения элементов массива заменить нулем. Все положительные значения – увеличить на 1.
	Двухмерный массив из 3 строк и 4 столбцов заполнить числовыми значениями. Вывести первый положительный элемент для каждого столбца двумерного массива.
3	Одномерный массив из 12 элементов заполнить числовыми значениями. Все элементы массива, значения которых кратны 5, увеличить в 2 раза. Определить сумму положительных элементов массива
	Двухмерный массив из 4 строк и 4 столбцов заполнить числовыми значениями. Поменять местами значения первой строки и первого столбца.
4	Одномерный массив из 11 элементов заполнить числовыми значениями. Определить количество нулевых значений в массиве. Определить сколько раз в массиве меняется знак.
	Двухмерный массив из 4 строк и 5 столбцов заполнить числовыми значениями. Найти сумму элементов для каждой строки
5	Одномерный массив из 10 элементов заполнить числовыми значениями. Все элементы массива увеличить в 3 раза. Определить в какой половине массива находится максимальный элемент.
	Двухмерный массив из 5 строк и 3 столбцов заполнить числовыми значениями. Найти минимальное значение для каждого столбца.
6	Одномерный массив из 12 элементов заполнить числовыми значениями. Определить сумму всех элементов массива. Определить количество элементов, значения которых больше суммы соседних элементов (предыдущего и последующего).
	Двухмерный массив из 4 строк и 3 столбцов заполнить числовыми значениями. Вывести «yes», если значение первого столбца совпадают со значениями последнего столбца.
7	Одномерный массив из 15 элементов заполнить числовыми значениями. Определить количество элементов массива, значения которых = первому элементу. Определить произведение четных по значению элементов массива.
	Двухмерный массив из 4 строк и 4 столбцов заполнить числовыми значениями. Определить является ли массив симметричным относительно побочной диагонали
8	Одномерный массив из 12 элементов заполнить числовыми значениями. Найти среднее арифметическое значение отрицательных элементов массива. Элементы с положительными значениями обнулить
	Двухмерный массив из 3 строк и 3 столбцов заполнить числовыми значениями. Содержимое массива повернуть на 90^0 по часовой стрелке
9	Одномерный массив из 10 элементов заполнить числовыми значениями. Определить произведение положительных значений. Вывести номера локальных максимумов, т.е. таких A_i, что $A_{i-1} < Ai > A_{i+1}$
	Двухмерный массив из 3 строк и 3 столбцов заполнить 0 и 1, это будет результат игры в крестики-нолики. Вывести «да», если игра закончилась победой крестиков.
10	Одномерный массив из 14 элементов заполнить числовыми значениями. Поменять местами минимальный и максимальный элементы массива.

	Двухмерный массив из 4 строк и 4 столбцов заполнить числовыми значениями. Вывести минимальный элемент главной диагонали
11	Одномерный массив из 12 элементов заполнить числовыми значениями. Ввести число К. Определить количество элементов массива, значения которых больше К. Первый отрицательный элемент заменить нулем.
	Двухмерный массив из 3 строк и 4 столбцов заполнить числовыми значениями Известно, что среди его элементов два и только два равных между собой. Вывести их индексы.
12	Одномерный массив из 15 элементов заполнить числовыми значениями. Определить количество нечетных значений в массиве. Определить, каких элементов больше положительных или отрицательных элементов массива.
	В массиве хранится информация о проданных (1) и свободных (0) местах на киносеанс. Кинозал состоит из 10 рядов по 12 мест в каждом. Для введенного № ряда вывести номера свободных кресел.
13	Одномерный массив из 12 элементов заполнить числовыми значениями. Все отрицательные элементы массива заменить значением первого элемента. Определить количество положительных значений.
	Заполнить массив 2 на 5 числами от 0 до 1. Вывести «да», если существует столбец, состоящий только из нулей
14	Одномерный массив из 12 элементов заполнить числовыми значениями. Вывести первый и второй максимум. Определить среднее арифметическое значение положительных элементов массива.
	Двухмерный массив из 3 строк и 4 столбцов заполнить числовыми значениями. Определить, в какой строке сумма элементов больше: во второй или в третьей.
15	Одномерный массив из 10 элементов заполнить числовыми значениями. Найти номера первого и последнего нулевого элемента. Определить сумму отрицательных элементов массива
	Массив хранит оценки сессии 7-ми студентов по 5-ти предметам. Заполнить массив значениями от 2 до 5. Определить количество двоек по каждому предмету.
16	Одномерный массив из 8 элементов заполнить числовыми значениями. Поменять местами первый элемент массива с максимальным элементом Все отрицательные элементы массива заменить его абсолютным значением
	Двухмерный массив из 3 строк и 4 столбцов заполнить числовыми значениями. Для каждого столбца массива вывести первое нечетное значение.
17	Одномерный массив из 10 элементов заполнить числовыми значениями. Сдвинуть элементы массива на одну позицию вправо, последний элемент переместить на первое место. Определить количество нулевых значений.
	Массив хранит оценки экзаменов 10_ти студентов по 5_ти предметам. Организовать заполнение массива значениями от 2 до 5. Определить количество предметов, по которым были получены только оценки «5» и «4».
18	Одномерный массив из 10 элементов хранит оценки, выставленные спортсмену судьями. Определить балл, идущий спортсмену в зачет по олимпийской системе (средний балл без учета максимального и минимального).
	Двухмерный массив из 5 строк и 5 столбцов заполнить числовыми значениями Вывести элементы массива, располагающиеся ниже главной диагонали
19	Одномерный массив из 12 элементов заполнить числовыми значениями. Определить количество элементов массива, значения которых = последнему элементу. Все положительные элементы увеличить на 1

	Двухмерный массив из 3 строк и 5 столбцов заполнить числовыми значениями. Ввести число А. Последний нулевой элемент каждой строки заменить на А
20	Одномерный массив из 14 элементов заполнить числовыми значениями. Определить среднее арифметическое четных значений. Вывести номер минимального элемента
	В массиве С(6,6) поменять местами элементы симметрично относительно главной диагонали.
21	Одномерный массив из 10 элементов заполнить числовыми значениями. Определить содержит ли массив элементы, равные максимальному элементу.
	Двухмерный массив из 3 строк и 5 столбцов заполнить числовыми значениями. Определить количество четных значений в каждой строке.
22	Одномерный массив из 10 элементов заполнить числовыми значениями. Определить, является ли массив пилообразным, т.е. выполняется ли условие А1 < А2 > А3 < А4 > А5… Каждый элемент, значение которого <10, заменить его квадратом.
	Двухмерный массив из 5 строк и 5 столбцов заполнить числовыми значениями. Вывести на экран максимальный элемент побочной диагонали
23	Одномерный массив из 8 элементов заполнить числовыми значениями. Поменять местами две половинки массива (1-5, 2-6, 3-7, 4-8). Определить количество четных значений
	Заполнить массив 3 на 3 числовыми значениями. Определить, является ли он магическим квадратом, т.е. таким, в котором суммы во всех строках и столбцах одинаковы.
24	Одномерный массив из 12 элементов заполнить числовыми значениями. Ввести число К. Все положительные элементы массива увеличить на К. Определить, содержит ли массив два соседних четных элемента
	Заполнить массив 3 на 5 числовыми значениями. Вывести «ДА» , если сумма элементов первого столбца совпадает с суммой элементов последнего столбца.
25	Одномерный массив из 12 элементов заполнить числовыми значениями. Определить среднее арифметическое значение нечетных элементов массива. Четные значения увеличить в 3 раза
	Заполнить массив 3 на 5 числовыми значениями. Определить количество элементов массива, значения которых попадают в интервал от А до В.

Лабораторная работа 32.
Создание классов и объектов

Цель:
- познакомиться с созданием классов;
- изучить работу класса на практических примерах;
- уметь использовать полученные знания при решении сложных задач.

Краткая теория:

Class имя_класса {
закрытые функции и переменные класса
public:

открытые функции и переменные класса
} список_объектов
Например:

```
Class Myclass  {
//закрытый элемент класса
int a;
public:
void set_a(int num)
int get_a();
};
```

Этот класс имеет одну закрытую переменную a и две открытые функции, set_a() и get_a().

Функции, которые объявляются внутри класса, называются функциями-членами.

Поскольку a является закрытой переменной класса, она не доступна для любой функции вне myclass. Однако поскольку set_a() и get_a() являются членами myclass, они имеют доступ к a. Более того, set_a() и get_a(), являясь открытыми членами myclass, могут вызываться из любой части программы, использующей myclass.

Хотя функции set_a() и get_a() объявлены в myclass, они еще на определены. Для определения функции-члена необходимо связать имя класса, частью которого является функция-член, с именем функции. Это достигается путем написания имени функции вслед за именем класса с двумя двоеточиями. Два двоеточия называются оператором расширения области видимости. Например:

```
Void myclass::set_a(int num)  {
A=num;
}
int myclass::get_a()  {
return a;
}
```

set_a() и get_a() имеют доступ к переменной a, которая для myclass является закрытой. При определении функции-члена пользуются следующей основной формой:

```
тип_возвр_значения имя_класса::имя_функции(список_параметров)  {
//тело функции
}
```

имя_класса – это имя того класса, которому принадлежит определяемая функция.

Объявление класса myclass не задает ни одного объекта типа myclass, оно определяет только тип объекта, который будет создан при его фактическом объявлении. Чтобы создать объект, используется имя класса, как спецификатор типа данных. Например, в этой строке объявляются два объекта типа myclass:

```
Myclass ob1, ob2; //это объекты
```

После того, как объект класса создан, можно обращаться к открытым членам класса, используя оператор точка (.), аналогично тому, как осуществляется доступ к членам структуры. Предположим, что ранее объекты были объявлены, тогда следующие инструкции вызывают set_a() для объектов ob1, ob2:

```
ob1.set_a(10);  //установка версии a объекта ob1 равной 10
ob2.set_a(99);  //установка версии a объекта ob2 равной 99
```

Каждый объект содержит собственную копию всех данных, объявленных в классе. Это значит, что a в ob1 отлично от a в ob2.

1. Рассмотрим программу, в которой используется myclass для задания значений a для ob1 и ob2 и вывода на экран этих значений для каждого объекта:

```
#include <iostream>
using namespace std;
class myclass  {
//закрытый элемент класса
```

```
int a;
public:
void set_a(int num)
int get_a();
};
void myclass::set_a(int num)  {
a=num;
}
int myclass::get_a()  {
return a;
}
int main()  {
myclass ob1, ob2;
ob1.set_a(10);
ob2.set_a(99);
cout<<ob1.get_a()<<"\n";
cout<<ob2.get_a()<<"\n";
return 0;
}
```

Данная программа выводит на экран величины 10 и 99.

2. В предыдущем примере переменная a в myclass является закрытой. Это означает, что она непосредственно доступна только для членов myclass. Если попытаться обратиться к закрытому члену класса из этой части вашей программы, которая не является членом этого класса, то результатом будет ошибка при компиляции. Следующий фрагмент демонстрирует ошибку.

```
#include <iostream>
using namespace std;
int main()  {
myclass ob1, ob2;
ob1.set_a(10);
ob2.set_a(99);
cout<<ob1.get_a()<<"\n";
cout<<ob2.get_a()<<"\n";
return 0;
}
```

3. Если бы a была объявлена в открытой секции myclass, тогда к ней, как показано ниже, можно было бы обратиться из любой части программы:

```
#include <iostream>
using namespace std;
class myclass  {
public:
int a;  //теперь a открыта и здесь на нужны функции set_a() и get_a()
};
int main()  {
myclass ob1, ob2;
ob1.a=10;
ob2.a=99;
cout<<ob1.a<<"\n";
cout<<ob2.a<<"\n";
return 0;
}
```

Задания:

1. Создать класс card, который поддерживает каталог библиотечных карточек. Этот класс должен хранить заглавие книги, имя автора и выданное на руки число экземпляров книги. Заглавие и имя автора храните в виде строки символов, а количество экземпляров – в виде целого числа. Используйте открытую функцию-член store() для запоминания информации о книгах и открытую функцию-член show() для вывода информации на экран. В функцию main() включите краткую демонстрацию работы созданного класса.

2. Создайте структуру с именем train, содержащую поля: название пункта назначения, номер поезда, время отправления. Ввести данные в массив из пяти элементов типа train, упорядочить элементы по номерам поездов. Добавить возможность вывода информации о поезде, номер которого введен пользователем. {Добавить возможность сортировки массив по пункту назначения, причем поезда с одинаковыми пунктами назначения должны быть упорядочены по времени отправления.}

3. Определить класс Children, который содержит такие поля (члены класса): закрытые – имя ребенка, фамилию и возраст , публичные – методы ввода данных и отображения их на экран. Объявить два объекта класса, внести данные и показать их.

4. Создайте класс Phone, который содержит переменные number, model. Создайте три экземпляра этого класса. Выведите на консоль значения их переменных. Добавить в класс Phone методы: receiveCall, имеет один параметр – имя звонящего. Выводит на консоль сообщение “Звонит {name}”. Метод getNumber – возвращает номер телефона. Вызвать эти методы для каждого из объектов. Добавить конструктор в класс Phone, который принимает на вход три параметра для инициализации переменных класса - number, model.

Лабораторная работа 33.
Введение в наследование. Виртуальные функции.

Цель:

- познакомиться с созданием классов;
- изучить работу класса на практических примерах;
- уметь использовать полученные знания при решении сложных задач.

Краткая теория:

```
Class имя_класса  {
закрытые функции и переменные класса
public:
открытые функции и переменные класса
} список_объектов
```

Например:

```
Class Myclass  {
//закрытый элемент класса
int a;
public:
void set_a(int num)
int get_a();
};
```

Этот класс имеет одну закрытую переменную a и две открытые функции, set_a() и get_a().

Функции, которые объявляются внутри класса, называются функциями-членами.

Поскольку a является закрытой переменной класса, она не доступна для любой функции вне myclass. Однако поскольку set_a() и get_a() являются членами myclass, они

имеют доступ к a. Более того, set_a() и get_a(), являясь открытыми членами myclass, могут вызываться из любой части программы, использующей myclass.

Хотя функции set_a() и get_a() объявлены в myclass, они еще на определены. Для определения функции-члена необходимо связать имя класса, частью которого является функция-член, с именем функции. Это достигается путем написания имени функции вслед за именем класса с двумя двоеточиями. Два двоеточия называются оператором расширения области видимости. Например:

```
Void myclass::set_a(int num)  {
A=num;
}
int myclass::get_a()  {
return a;
}
```

set_a() и get_a() имеют доступ к переменной a, которая для myclass является закрытой. При определении функции-члена пользуются следующей основной формой:

```
тип_возвр_значения имя_класса::имя_функции(список_параметров)  {
//тело функции
}
```

имя_класса – это имя того класса, которому принадлежит определяемая функция.

Объявление класса myclass не задает ни одного объекта типа myclass, оно определяет только тип объекта, который будет создан при его фактическом объявлении. Чтобы создать объект, используется имя класса, как спецификатор типа данных. Например, в этой строке объявляются два объекта типа myclass:

```
Myclass ob1, ob2; //это объекты
```

После того, как объект класса создан, можно обращаться к открытым членам класса, используя оператор точка (.), аналогично тому, как осуществляется доступ к членам структуры. Предположим, что ранее объекты были объявлены, тогда следующие инструкции вызывают set_a() для объектов ob1, ob2:

```
ob1.set_a(10);  //установка версии a объекта ob1 равной 10
ob2.set_a(99);  //установка версии a объекта ob2 равной 99
```

Каждый объект содержит собственную копию всех данных, объявленных в классе. Это значит, что a в ob1 отлично от a в ob2.

4. Рассмотрим программу, в которой используется myclass для задания значений a для ob1 и ob2 и вывода на экран этих значений для каждого объекта:

```
#include <iostream>
using namespace std;
class myclass  {
//закрытый элемент класса
int a;
public:
void set_a(int num)
int get_a();
};
void myclass::set_a(int num)  {
a=num;
}
int myclass::get_a()  {
return a;
}
int main()  {
myclass ob1, ob2;
ob1.set_a(10);
ob2.set_a(99);
```

```
cout<<ob1.get_a()<<"\n";
cout<<ob2.get_a()<<"\n";
return 0;
}
```

Данная программа выводит на экран величины 10 и 99.

5. В предыдущем примере переменная a в myclass является закрытой. Это означает, что она непосредственно доступна только для членов myclass. Если попытаться обратиться к закрытому члену класса из этой части вашей программы, которая не является членом этого класса, то результатом будет ошибка при компиляции. Следующий фрагмент демонстрирует ошибку.

```
#include <iostream>
using namespace std;
int main()  {
myclass ob1, ob2;
ob1.set_a(10);
ob2.set_a(99);
cout<<ob1.get_a()<<"\n";
cout<<ob2.get_a()<<"\n";
return 0;
}
```

6. Если бы a была объявлена в открытой секции myclass, тогда к ней, как показано ниже, можно было бы обратиться из любой части программы:

```
#include <iostream>
using namespace std;
class myclass  {
public:
int a;  //теперь a открыта и здесь на нужны функции set_a() и get_a()
};
int main()  {
myclass ob1, ob2;
ob1.a=10;
ob2.a=99;
cout<<ob1.a<<"\n";
cout<<ob2.a<<"\n";
return 0;
}
```

Задания:

1. Создать функцию min(), которая возвращает наименьший из двух численных аргументов, используемых при вызове функции. Перегрузите функцию min() так, чтобы она воспринимала в качестве аргументов символы, целые и действительные двойной точности.
2. Создать класс card, который поддерживает каталог библиотечных карточек. Этот класс должен хранить заглавие книги, имя автора и выданное на руки число экземпляров книги. Заглавие и имя автора храните в виде строки символов, а количество экземпляров – в виде целого числа. Используйте открытую функцию-член store() для запоминания информации о книгах и открытую функцию-член show() для вывода информации на экран. В функцию main() включите краткую демонстрацию работы созданного класса.

Лабораторная работа 34
Полиморфизм. Родовые функции. Родовые классы.

Цель работы: 1) изучить возможности наследования классов на языке С++; 2) получить основные навыки программирования с использованием наследования классов. Теоретические сведения

Язык С++ позволяет классу наследовать данные-элементы и функции-элементы одного или нескольких других классов. Новый класс называют производным классом. Класс, элементы которого наследуются производным классом, называется базовым классом. В свою очередь производный класс может служить базовым для другого класса. Наследование дает возможность заключить некоторое общее или схожее поведение различных объектов в одном базовом классе.

Наследование позволяет также изменить поведение существующего класса. Производный класс может переопределить некоторые функции-элементы базового, наследуя, тем не менее, основной объем свойств и атрибутов базового класса. Общий вид наследования: class Base { // ….. }; class Derived: <ключ доступа> Base { // …… };

Ключ доступа может быть private, protected, public. Если ключ не указан, то по умолчанию он принимается private. Наследование позволяет рассматривать целые иерархии классов и работать со всеми элементами одинаково, приводя их к базовому. Правила приведения следующие: Наследуемый класс всегда можно привести к базовому;

Базовый класс можно привести к наследуемому только если в действительности это объект наследуемого класса. Ошибки приведения базового класса к наследуемому отслеживаются программистом.

Доступ к элементам класса

При наследовании ключ доступа определяет уровень доступа к элементам базового класса внутри производного класса. В таблице описаны возможные варианты доступа.

Наследование	**Доступ в базовом классе**	**Доступ в производном классе**
public	public protected private	public protected private
protected	public protected private	protected protected private
private	public protected private	private private private

Конструкторы и деструкторы при наследовании

Конструкторы не наследуются. Если конструктор базового класса требует спецификации одного или нескольких параметров, конструктор производного класса должен вызывать базовый конструктор, используя список инициализации элементов. Пример 1.\

```
#include <string>
class Base
{   public:
   Base(int, float);
};
class Derived: Base
{   public:
```

```
   Derived(char* lst, float amt);
};
   Derived:: Derived(char* lst, float amt) : Base(strlen(lst),amt)
   {   }
```

В деструкторе производного класса компилятор автоматически генерирует вызовы базовых деструкторов, поэтому для удаления объекта производного класса следует сделать деструктор в базовых классах виртуальным. Для вызова используется delete this либо operator delete.

Виртуальные функции

Функция-элемент может быть объявлена как virtual. Ключевое слово virtual предписывает компилятору генерировать некоторую дополнительную информацию о функции. Если функция переопределяется в производном классе и вызывается с указателем (или ссылкой) базового класса, ссылающимся на представитель производного класса, эта информация позволяет определить, какой из вариантов функции должен быть выбран: такой вызов будет адресован функции производного класса.

Для виртуальных функций существуют следующие правила: виртуальную функцию нельзя объявлять как static. спецификатор virtual необязателен при переопределении функции в производном классе. виртуальная функция должна быть определена в базовом классе и может быть переопределена в производном.

Пример программирования Пример 2. Написать программу с наследованием класса стек от класса массив.

```
#include <iostream.h>
#include <stdlib.h>
class massiv
{   int *num;
   int kol;
   public:
   massiv(int n);
   void print();
   virtual int kolich(){return kol;}
   void put(int k,int n){num[k]=n;}
   ~massiv(){delete num;}
};
massiv::massiv(int n)
{   num=new int[n];
   kol=n;
   for(int i=0;i<kol;i++)  num[i]=random(100)-50;
}
void massiv::print()
{   for(int i=0;i<kolich();i++)  cout<<num[i]<<" ";
   cout<<endl;
}
class stec:public massiv
{   int top;
   public:
   stec(int);
   virtual int kolich() {return top;}
   void pop(int k);
};
stec::stec(int n):massiv(n)
```

```
{   top=0;
}
void stec::pop(int k)
{   put(top++,k);  }
void main()
{   randomize();
    massiv a(10);
    a.print();
    stec b(10);
    b.pop(random(100)-50);
    b.pop(random(100)-50);
    b.pop(random(100)-50);
    b.print();
}
```

Главное отличие виртуальной функции от просто перегруженной в том, какая функция будет вызываться при рассмотрении производного класса как базового. Пример 3.

```
#include <iostream>
class Base
{   public:
    Base(){};
    Print(){   cout<<"I'm a Base print"<<endl;}
    virtual View(){ cout<<"I'm a Base view"<<endl;}
};
class Derived: public Base
{   public:
    Derived(){};
    Print(){   cout<<"I'm a Derived print"<<endl;}
    View(){ cout<<"I'm a Derived view"<<endl;}
};
void main(void)
{   Base *A=new Base;
    Derived *B=new Derived;
    Base *C;
    A->Print();
    A->View();
    B->Print();
    B->View();
    C=(Base *)B;
    C->Print();
    C->View();
}
```

Результат: "I'm a Base print" "I'm a Base view" "I'm a Derived print" "I'm a Derived view" "I'm a Base print" "I'm a Derived view"

Таким образом, мы видим, что виртуальные функции позволяют нам всегда работать с теми функциями, которые специфичны именно для используемого класса, даже когда мы рассматриваем его как базовый.

Контрольные вопросы

1. Какой класс называется базовым?
2. Какой класс называется производным?
3. Какие ключи доступа используются при наследовании?

4. Наследуются ли конструкторы?
5. Наследуются ли деструкторы?
6. Что собой представляет виртуальная функция?
7. Можно ли виртуальную функцию объявить как static?

Варианты заданий

1. Разработать программу с использованием наследования классов, реализующую классы: графический объект; круг; квадрат. Используя виртуальные функции, не зная с объектом какого класса вы работаете, выведите на экран его размер и координаты.
2. Разработать программу с использованием наследования классов, реализующую классы: железнодорожный вагон; вагон для перевозки автомобилей; цистерна. Используя виртуальные функции, не зная с объектом какого класса вы работаете, выведите на экран его вес и количество единиц товара в вагоне.
3. Разработать программу с использованием наследования классов, реализующую классы: массив; стек; очередь. Используя виртуальные функции, не зная с объектом какого класса вы работаете, выведите на экран количество элементов и добавьте элемент.
4. Разработать программу с использованием наследования классов, реализующую классы: воин; пехотинец(винтовка); матрос(кортик). Используя виртуальные функции, не зная с объектом какого класса вы работаете, выведите на экран его возраст и вид оружия.
5. Разработать программу с использованием наследования классов, реализующую классы: точка; линия; круг. Используя виртуальные функции, не зная с объектом какого класса вы работаете, выведите на экран координаты и размер.
6. Разработать программу с использованием наследования классов, реализующую классы: работник больницы; медсестра; хирург. Используя виртуальные функции, не зная с объектом какого класса вы работаете, выведите на экран возраст и название должности.
7. Разработать программу с использованием наследования классов, реализующую классы: точка; квадрат пирамида. Используя виртуальные функции, не зная с объектом какого класса вы работаете, выведите на экран его размер и координаты.
8. Разработать программу с использованием наследования классов, реализующую классы: реагент; углерод; железо. Используя виртуальные функции, не зная с объектом какого класса вы работаете, выведите на экран его количество и свойства (форма кристаллической решетки для углерода и чистота выработки руды для железа).
9. Разработать программу с использованием наследования классов, реализующую классы: работник фирмы; стажер; руководящий сотрудник; директор. Используя виртуальные функции, не зная с объектом какого класса вы работаете, выведите на экран целое число - уровень допуска, и название должности.
10. Разработать программу с использованием наследования классов, реализующую классы: молодой человек; студент; военнослужащий; военный курсант. Используя виртуальное наследование и виртуальные функции, не зная с объектом какого класса вы работаете, выведите на экран сведения о военнообязанности.

Лабораторная работа 35.
Обработка исключений

В данной лабораторной работе вам нужно написать класс исключения и программу, способную генерировать и обрабатывать определенный вид исключения (программа должна содержать блоки try, catch, точку throw). Позаботьтесь о том, чтобы исключение действительно могло возникнуть, продемонстрируйте работу перехватчика и обработчика исключений.

Итак, мы увидели, как новый метод обработки ошибок удобен и прост. Блок try-

catch может содержать вложенные блоки try-catch и если не будет определено соответствующего оператора catch на текущем уровен вложения, исключение будет поймано на более высоком уровне. Единственная вещь, о которой вы должны помнить, - это то, что операторы, следующие за throw, никогда не выполнятся.

```
try

{

throw;

// ни один оператор, следующий далее (до закрывающей скобки)

// выполнен не будет

}

catch (...)

{

cout << "Исключение!" << endl;

}
```

try-блок — так называемый блок повторных попыток. В нем надо располагать код, который может привести к ошибке и аварийному закрытию программы;

throw генерирует исключение. То что остановит работу try-блока и приведет к выполнению кода catch-блока. Тип исключения должен соответствовать, типу принимаемого аргумента catch-блока;

catch-блок — улавливающий блок, поймает то, что определил throw и выполнит свой код. Этот блок должен располагаться непосредственно под try-блоком. Никакой код не должен их разделять.

если в try-блоке исключение не генерировалось, catch-блок не сработает. Программа его обойдет.

```
try
{   if  (b==0)
throw  0;
double   d=a /b;
cout    <<  d;     }
catch  (int)
{
cout<<”ERROR:   division by  zero.”;
}
```

Блок try реагирует не только на исключения, возбуждаемые непосредственно внутри него, но и на все не перехваченные исключения, возбуждаемые внутри функций прямо или косвенно вызываемых из блока try.

Варианты

Обработать следующие исключения:

1. Взятие квадратного корня из отрицательного числа.

2. Ввод пользователем отрицательного возраста.
3. Ввод пользователем вещественного числа вместо целого.
4. Попытка записи в файл, открытый только для чтения.
5. Ввод пользователем строки вместо числа.
6. Ввод пользователем специального символа вместо целого числа.
7. Ввод пользователем несуществующей даты, например, «31 февраля».
8. Ввод несуществующего времени, например, 56 час. 335 мин. 98 сек.
9. Ввод пользователем отрицательной зарплаты.
10. Ввод числа, спецсимволов или русских букв вместо символов латинского алфавита.
11. Ввод числа, спецсимволов или латинских букв вместо символов русского алфавита.
12. Превышение при вводе максимально возможной длины строки.
13. Выход индекса за пределы массива.
14. Ввод пользователем строки вместо числа.
15. Нехватка памяти при динамическом ее выделении.

Лабораторная работа 36
Создание простого приложения

Настройка формы

Пустая форма в правом верхнем углу имеет кнопки управления, предназначенные для свертывания формы  , для разворачивания формы на весь экран  , для возвращения к исходному размеру  и для закрытия формы  . С помощью мыши, «захватывая» одну из кромок формы или выделенную строку заголовка, можно регулировать размеры формы и ее положение на экране.

Для изменения заголовка после вызова окна инспектора объектов (F11) выбирается свойство Caption, и в выделенном окошке вместо стандартного текста **Form1** набирается нужный текст, например, «Лаб.раб.1. Гр.610101 Иванова А.».

Внимание! Свойства Name (имя) и Caption (заголовок) у компонент совпадают, но имя менять не рекомендуется, т.к. оно входит в текст программы.

Компоненты, предназначенные для ввода-вывода

Если необходимо ввести из формы в программу или вывести на форму информацию, которая вмещается в одну строку, используют окно однострочного редактора текста, представляемого компонентой **Edit**, для чего в меню компонент Standard выбирается пиктограмма  и щелчком кнопкой мыши устанавливается в нужном месте формы. Мышью регулируются размер окошка и его положение на форме.

В заголовочный файл Unit1.h автоматически вставляется переменная Edit* (1,2,…) класса TEdit. В поле Text (Edit1–>Text) такой переменной будет содержаться строка символов (тип AnsiString) и отображаться в соответствующем окне Edit*.

Основные функции преобразования строк

StrToFloat(St) – преобразует строку St в вещественное число;

StrToInt(St) – преобразует строку St в целое число.

FloatToStr (W) – преобразует вещественное число W в строку символов;

FloatToStrF (W, формат, n1, n2) – вещественное число W в строку символов под управлением формата:

ffFixed – фиксированное положение разделителя целой и дробной частей, n1 – общее количество цифр числа, n2 – количество цифр в дробной части, причем число округляется с учетом первой отбрасываемой цифры;

fFfExponent – n1 задает общее количество цифр мантиссы, n2 – количество цифр порядка XX (число округляется);

ffGeneral – универсальный формат, использующий наиболее удобную для чтения форму представления вещественного числа; соответствует формату ffFixed, если количество цифр в целой части ≤ n1, а само число больше 0,00001, в противном случае соответствует формату ffExponent.

FormatFloat (формат, W) – преобразует вещественное число W в строку;

IntToStr (W) – преобразует целое число W в строку символов.

Например, если значения вводимых из Edit1 и Edit2 переменны x и y имеют целый и действительный типы, соответственно, то следует записать:

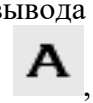
x = StrToInt(Edit1–>Text);

y = StrToFloat(Edit2–>Text);

Внимание! При записи числовых значений в окошках Edit* не должно быть пробелов, а разделителем целой и дробной частей обычно является «**запятая**»!

В инспекторе объектов с помощью свойства **Font** устанавливается стиль, отражаемого в строке Edit* текста.

Компонента Label

Данная компонента используется для вывода надписей, для чего в меню компонент Standard выбирается пиктограмма  , и в нужном месте формы устанавливается надпись Label1, регулируется размер, место положения, изменяется свойство Caption инспектора объектов, в котором вводится нужный текст, например, строка “Значение X = ”, а также выбирается стиль (свойство Font).

При установке таких компонент в текст Unit1.h вставляются переменные типа TLabel, в которых хранятся пояснительные строки. Эти строки можно изменять в процессе работы программы, например:

Label1->Caption = “”;– «очистка» строки;

Label1->Caption = “Не выполняется!”;– вывод строки.

Компонента Memo

Для вывода результатов работы программы обычно используется окно многострочного редактора текста, представленое компонентой Memo, для чего выбирается пиктограмма  , помещается на форму, регулируется ее размер и местоположение. После установки с помощью инспектора свойства ScrollBars – SSBoth в окне появятся вертикальная и горизонтальная полосы прокрутки.

При установки данной компонеты в Unit1.h прописывается переменная Memo1 типа TMemo. Информация, выводимая построчно в окне Memo1, находится в массиве строк Memo1–>Lines, каждая из которых имеет тип String.

Для очистки окна используется метод Memo1–>Clear( ).

Для добавления новой строки используется метод Memo1–>Lines–>Add( ).

Если нужно вывести числовое значение, то его надо преобразовать к типу AnsiString (см. прил. 4) и добавить в массив Memo1–>Lines, например, вывестиint u = 100;

double w = –256.38666;

в результате записей

Memo1–>Lines–>Add (“ Значение u = ”+IntToStr(u));

Memo1–>Lines–>Add (“ Значение w = ”+FloatToStrF(w,ffFixed,8,2));

появятся строки

Значение u = 100

Значение w = –256.39

При этом под все число отводится восемь позиций, из которых две позиции занимает его дробная часть.

Если выводимая информация превышает размер окна Memo1, то для просмотра используются полосы прокрутки.

Обработка событий

Напомним, что программа в среде Builder представляет собой набор функций, выполняющих обработку событий, связанных с формой, например, щелчок кнопкой мыши – событие OnClick, создание формы – событие OnCreate.

Функция-обработчик FormCreate

При запуске программы возникает событие «создание формы» (OnCreate). Оформим функцию-обработчик этого события, которая обычно используется для инициализации начальных установок, таких, как, например, занести начальные значения исходных данных в соответствующие окна Edit*, очистить окно Memo.

Для этого делаем двойной щелчок кнопкой мыши на любом **свободном** месте формы, после чего в листинг программы (Unit1.cpp) автоматически вносится заготовка для создания функции:ее заголовок … **FormCreate (…)** и фигурные скобки.

Между символами { }, которые обозначают начало и конец функции, соответственно, вставляем нужный текст программы (см. пример, п. 1.4.1).

Внимание! Не набирайте заголовки функций-обработчиков вручную.

Функция-обработчик нажатия кнопки (Button*Click)

Выбрав в меню Standard пиктограмму 

, помещаем на форму компоненту Button1 (2,3,…). С помощью инспектора объектов изменяем заголовок (Caption) на текст, например, «Выполнить», регулируем положение и размер кнопки. Двойным щелчком кнопкой мыши по компоненте Button1 в текст программы вставляем заготовку ее функции-обработчика **… Button1Click (…) { }**. Между фигурными скобками набираем соответствующий код.

Запуск и работа с программой

Перед запуском программы на обработку следует сохранить программу, для чего нужно выбрать в меню File пункт Save All.

Запустить программу можно, нажав Run в главном меню Run, или клавишу F9, или пиктограмму  . При этом происходит трансляция и, если нет ошибок, компоновка программы и создание единого загружаемого файла с расширением .**exe**. На экране появляется активная форма программы (см. рис.1.2).

Завершить работу программы можно, нажав кнопку  на форме или выбрав ProgramReset в главном меню Run.

Пример выполнения задания

Составить программу вычисления арифметического выражения для заданных значений x, y, z :

$$u = tg^2(x+y) - e^{y-z}\sqrt{\cos x^2 + \sin z^2}\,.$$

1.4.1. Пример создания оконного приложения

В оконном режиме панель диалога программы создать в виде, представленном на рис. 1.2.

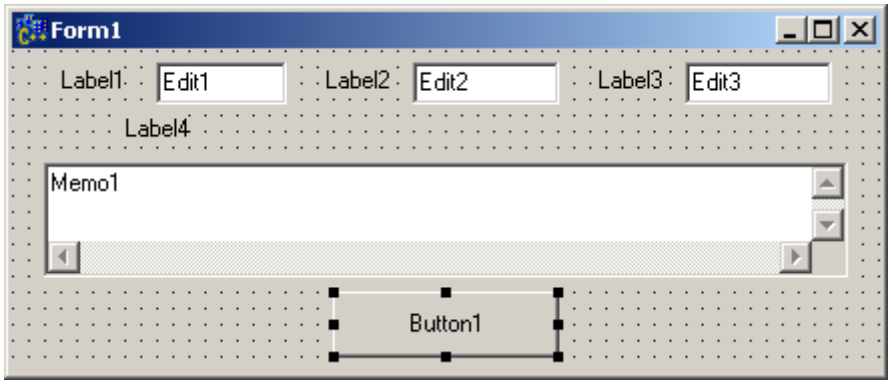


Рис. 1.2

Для создания проекта необходимо выполнить следующие действия.

1. Запускаем C++ Builder. Создаем в разрешенной для Пользователя папке (d:\work\ или c:\work\) папку с номером группы (\610101\), открыв ее, сохраняем предлагаемые файлы Unit1.cpp и Project1.cpp (рекомендуем без изменения).

2. Оформляем окно формы, заменив заголовок Form1 на нужный текст. Помещаем на форму необходимые компоненты Label1, Label2, Label3, Label4 (вставляя в Caption соответствующие тексты), Edit1, Edit2, Edit3, Memo1 с полосами прокрутки (см. п. 1.3), Button1 (заменив в Caption текст).

Используя свойство Font, выбираем стили выводимых текстов.

3. Оформляем листинг программы (Unit1.cpp). Двойным щелчком кнопкой мыши по свободному месту формы создаем функцию FormCreate и заполняем ее (см. пример). Переходим на форму (F12), щелкаем дважды по кнопке «ВЫПОЛНИТЬ» и заполняем созданную функцию Button1Click (см. пример).

4. Перед запуском программы на обработку, **сохраняем все.**

5. Запускаем проект на выполнение, исправляем ошибки.

Текст программы может иметь следующий вид (наклонным мелким шрифтом выделен текст, редактировать который не рекомендуется):

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "math.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Edit1->Text = "3,4";
Edit2->Text = "7,4e-2";
Edit3->Text = "1,943e2";
Memo1->Clear();
Memo1->Lines->Add("Лабораторная работа № 1");
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
double x, y, z, a, b ,c, rez;
x = StrToFloat(Edit1->Text);
y = StrToFloat(Edit2->Text);
z = StrToFloat(Edit3->Text);
a = pow(tan(x+y),2);
b = exp(y-z);
c = sqrt(cos(x*x)+sin(z*z));
rez = a-b*c;
```

```
Memo1->Lines->Add("При x = "+FloatToStrF(x,ffFixed,7,3)
        + "; y = "+FloatToStrF(y,ffFixed,7,3)+"; z = "+FloatToStrF(z,ffFixed,7,3));
Memo1->Lines->Add("Результат = "+FloatToStr(rez));
}
```

Внимание! В строковых константах разделитель целой и дробной частей – запятая: Edit1->Text = "3,4"; в отличие от числовых констант в тексте программы.

В результате должно получиться рабочее окно (рис. 1.3). Если щелкнуть мышью по кнопке «ВЫПОЛНИТЬ», в окне Memo1 появится соответствующий текст (результат). Далее в окошках Edit* можно изменять исходные значения и, нажимая кнопку «ВЫПОЛНИТЬ», получать новые результаты.

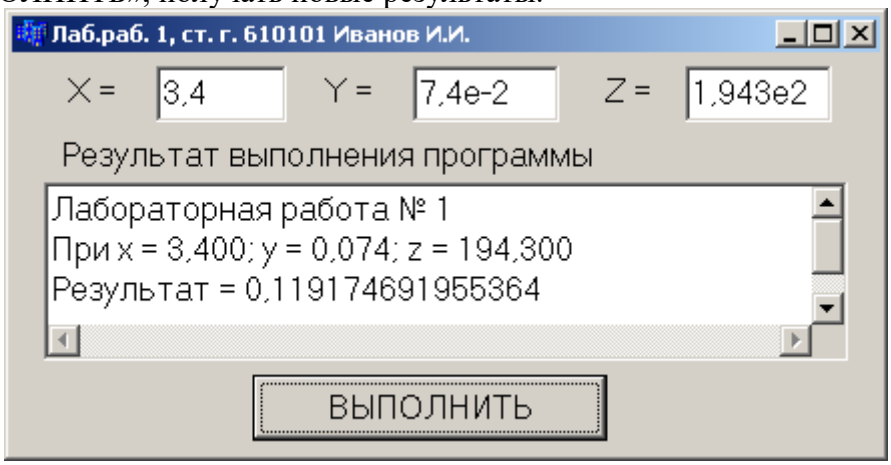
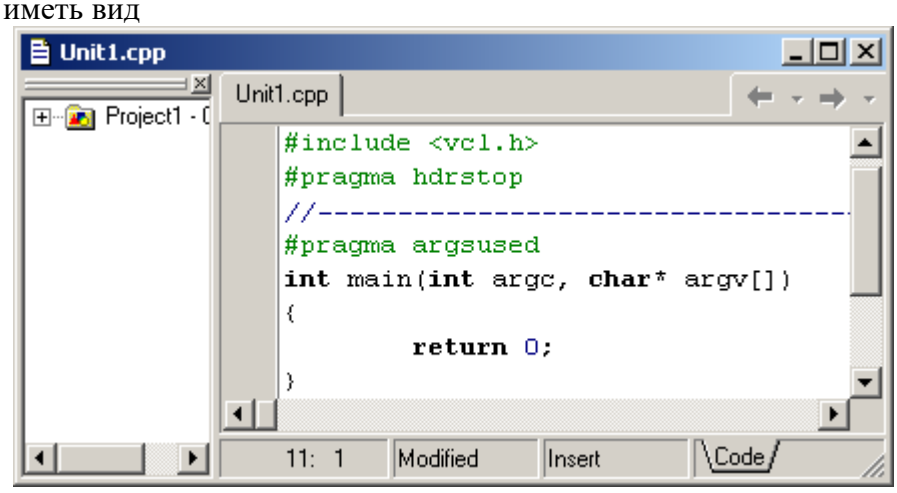


Рис. 1.3

1.4.2. Создание консольного приложения

Чтобы создать проект в консольном приложении, выполняем следующую последовательность действий: File → Close All → File → New → Other → Console Wizard → Ok. Закрываем все окошки, кроме 5 (см. рис 1.1), которое в консольном приложении будет иметь вид

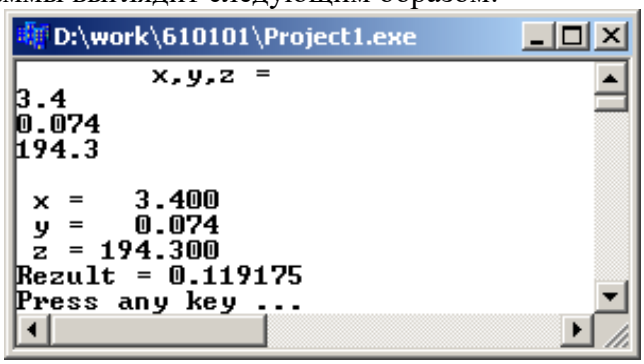


Текст программы может иметь следующий вид:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#pragma hdrstop
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
```

```
{
double x, y, z, a, b ,c, rez;
puts("\n\tx,y,z = ");
scanf("%lf%lf%lf", &x, &y, &z);
a = pow(tan(x+y),2);
b = exp(y-z);
c = sqrt(cos(x*x)+sin(z*z));
rez = a-b*c;
printf("\n x = %7.3lf\n y = %7.3lf\n z = %7.3lf\nRezult = %lf\n", x, y, z, rez);
puts("Press any key ... ");
getch();
return 0;
}
```

Для исходных данных x = 3,4; y = $7{,}4\cdot10^{-2}$; z = $1{,}943\cdot10^{2}$, результат выполнения программы выглядит следующим образом:



1.5. Индивидуальные задания

Создать программу вычисления указанной величины. Результат проверить при заданных исходных значениях.

1. $t = \dfrac{2\cos(x-\pi/6)}{0{,}5+\sin^2 y}\left(1+\dfrac{z^2}{3-z^2/5}\right)$.

При x = 14.26, y = –1.22, z = 3.5×10^{-2} : **0.564846.**

2. $u = \dfrac{\sqrt[3]{8+|x-y|^2+1}}{x^2+y^2+2} - e^{|x-y|}(tg^2 z+1)^x$.

При x = –4.5, y = 0.75×10^{-4}, z = 0.845×10^{2} : **–55.6848.**

3. $v = \dfrac{1+\sin^2(x+y)}{\left|x-\dfrac{2y}{1+x^2y^2}\right|}x^{|y|} + \cos^2[arctg(1/z)]$.

При x = 3.74×10^{-2}, y = –0.825, z = 0.16×10^{2} : **1.0553.**

4. $w = \left|\cos x - \cos y\right|^{(1+2\sin^2 y)}\left(1+z+\dfrac{z^2}{2}+\dfrac{z^3}{3}+\dfrac{z^4}{4}\right)$.

При x = 0.4×10^{4}, y = –0.875, z = -0.475×10^{-3} : **1.9873**.

5. $\alpha = \ln(y^{-\sqrt{|x|}})(x-y/2)+\sin^2 arctg(z)$.

При x = –15.246, y = 4.642×10^{-2}, z = 20.001×10^{2} : –**182.036**.

6. $\beta = \sqrt{10(\sqrt[3]{x}+x^{y+2})}\cdot(\arcsin^2 z - |x-y|)$

При x = 16.55×10^{-3}, y = –2.75, z = 0.15 : **40.630694**.

7.

$$\gamma = 5arctg(x) - \frac{1}{4}\arccos(x)\frac{x+3|x-y|+x^2}{|x-y|z+x^2}.$$

При x = 0.1722, y = 6.33, z = 3.25×10^{-4} : – **205.305571**.

8. $$\varphi = \frac{e^{|x-y|}|x-y|^{x+y}}{arctg\ x + arctg\ z} + \sqrt[3]{x^6 + \ln^2 y}.$$

При x = -2.235×10^{-2}, y = 2.23, z = 15.221 : **39.374**.

9.

$$\psi = |\, x^{y/x} - \sqrt[3]{y/x}\,| + (y-x)\frac{\cos y - z/(y-x)}{1+(y-x)^2}.$$

При x = 1.825×10^{2}, y = 18.225, z = -3.298×10^{-2} : **1.2131**.

10. $$a = 2^{-x}\sqrt{x + \sqrt[4]{|y|}}\ \sqrt[3]{e^{x-1/\sin z}}.$$

При x = 3.981×10^{-2}, y = – 1.625 × $\times10^{3}$, z = 0.512 : **1.26185**.

11.

$$b = y^{\sqrt[3]{|x|}} + \cos^3 y\ \frac{|x-y|\cdot\left(1+\frac{\sin^2 z}{\sqrt{x+y}}\right)}{e^{|x-y|} + x/2}.$$

При x = 6.251, y = 0.827, z = 25.001 : **0.7121**.

12. $$c = 2^{y^x} + (3^x)^y - \frac{y\cdot(arctgz - \pi/6)}{|x| + \frac{1}{y^2+1}}.$$

При x = 3.251, y = 0.325, z = 0.466×10^{-4} : **4.251433**.

13. $$f = \frac{\sqrt[4]{y + \sqrt[3]{x-1}}}{|x-y|(\sin^2 z + tg\ z)}.$$

При x = 17.421, y = 10.365×10^{-3}, z = 0.828×10^{5} : **0.33056**.

14.

$$g = \frac{y^{x+1}}{\sqrt[3]{|y-2|}+3} + \frac{x+y/2}{2|x+y|}(x+1)^{-1/\sin z}.$$

При x = 12.3×10^{-1}, y = 15.4, z = 0.252×10^{3} : **82.825623**.

15.

$$h = \frac{x^{y+1} + e^{y-1}}{1 + x|y - tgz|}(1 + |y-x|) + \frac{|y-x|^2}{2} - \frac{|y-x|^3}{3}.$$

При x = 2.444, y = 0.869×10^{-2}, z = -0.13×10^{3} : –**0.49871**.

16. $$w = \sqrt[3]{x^6 + \ln^2 y} + \frac{e^{|x-y|}|x-y|^{x+y}}{arctg(x) + arctg(z)}.$$

При x = -2.235×10^{-2}, y = 2.23, z = 15.221 : **39.374**.

17. $$a = 2^{-x}\sqrt{x + \sqrt[4]{|y|}}\ \sqrt[3]{e^{x-1/\sin z}}.$$

При x = 3.981×10-2, y = –1.625 × ×103, z = 0.512 : 1.26185.

18.

$$b = y^{\sqrt[3]{|x|}} + \cos^3 y\ \frac{|x-y|\cdot\left(1+\frac{\sin^2 z}{\sqrt{x+y}}\right)}{e^{|x-y|} + x/2}.$$

При x = 6.251, y = 0.827, z = 25.001 : 0.7121.

$$c = 2^{y^x} + (3^x)^y - \frac{y\cdot(arctgz - \pi/6)}{|x| + \frac{1}{y^2+1}}.$$

При x = 3.251, y = 0.325, z = 0.466×10-4 : 4.251433.

19.

20. $f = \dfrac{\sqrt[4]{y+\sqrt[3]{x-1}}}{|x-y|(\sin^2 z + tg\ z)}$.

При x = 17.421, y = 10.365×10-3, z = 0.828×105 : 0.33056.

21. $\beta = \sqrt{10(\sqrt[3]{x} + x^{y+2})} \cdot (\arcsin^2 z - |x-y|)$

При x = 16.55×10^{-3}, y = –2.75, z = 0.15 : **40.630694**.

22. $\gamma = 5arctg(x) - \dfrac{1}{4}\arccos(x)\dfrac{x+3|x-y|+x^2}{|x-y|z+x^2}$.

При x = 0.1722, y = 6.33, z = 3.25×10^{-4} : **–205.305571**.

23. $\varphi = \dfrac{e^{|x-y|}|x-y|^{x+y}}{arctg\ x + arctg\ z} + \sqrt[3]{x^6 + \ln^2 y}$.

При x = -2.235×10^{-2}, y = 2.23, z = 15.221 : **39.374**.

24. $u = \dfrac{\sqrt[3]{8+|x-y|^2+1}}{x^2+y^2+2} - e^{|x-y|}(tg^2 z + 1)^x$.

При x = –4.5, y = 0.75×10-4, z = 0.845×102 : –55.6848.

25. $v = \dfrac{1+\sin^2(x+y)}{\left|x - \dfrac{2y}{1+x^2y^2}\right|} x^{|y|} + \cos^2[arctg(1/z)]$.

При x = 3.74×10-2, y = –0.825, z = 0.16×102 : 1.0553.

Лабораторная работа 37
Добавление меню и панели инструментов. Графические возможности среды. Создание текстового редактора

Запустите Borland C++ Builder и создайте новый проект с именем edit. Поскольку текстовый редактор предназначен для работы с текстовыми строками, нам понадобится компонент **Memo** из вкладки **Standard**. Поместите его в центре формы проекта. Теперь поместите на форму компонент **MainMenu** из той же вкладки. Наконец, добавьте на форму три стандартные кнопки **BitButton** из вкладки **Additional**. Они понадобятся для формирования в программе органов управления для открытия и сохранения файлов и для других функций. Поскольку редактор должен при открытии и сохранении файлов обращаться к дисковым накопителям, необходимо добавить на форму диалоги **Open Dialog** и **Save Dialog** из вкладки **Dialogs**. Кроме того, добавьте на форму диалог **Font Dialog** из этой же вкладки. Он облегчит нам задачу настройки шрифтов из программы. Эти компоненты являются невидимыми при работе программы, поэтому место их размещения на форме не имеет значения. Но все же лучше располагать такие компоненты без наложения, для того чтобы они не скрывали друг друга. Последний компонент, который необходимо поместить на форму, — **PopupMenu** из вкладки **Standard**. Он позволит создать в программе контекстное меню, появляющееся при нажатии правой кнопки мыши.

Теперь разместим выбранные компоненты на форме и настроим их свойства. Начнем с главного компонента — формы приложения. Выберите в инспекторе объектов в поле селектора объектов **Form1**. Измените свойство **Caption** этого объекта на название программы редактора Edit. Размеры формы установите с помощью мыши, захватив любой угол или сторону формы и перемещая их в любом направлении. Размер формы

можно будет поправить при необходимости на любом этапе разработки программы. В свойстве **Position** установите значение **poDesigned**. Это обеспечит центровку нашего приложения при запуске по центру экрана. Щелкните по компоненту **MainMenu** левой кнопкой мыши. При этом откроется окно дизайнера меню. В свойстве **Caption** инспектора объектов введите название **&File** и нажмите клавишу <Enter>. На форме приложения при этом появится строка главного меню. Таким образом, на форме можно будет видеть поле главного меню и учитывать его размеры. Теперь можно временно закрыть окно дизайнера меню и приступить к размещению остальных компонентов на форме. Вначале разместите на форме компонент **Memo** так, чтобы он занимал всю ее свободную область за исключением правой полосы для кнопок. Кнопки расположите друг под другом, не изменяя при этом их размеров. В оставшейся свободной части формы можно расположить оставшиеся невидимые компоненты приложения. Не забывайте, что перемещать компоненты можно группами, предварительно выделив эти компоненты с помощью мыши. В результате после размещения всех компонентов должна получиться форма, подобная приведенной на рис. 21.1.

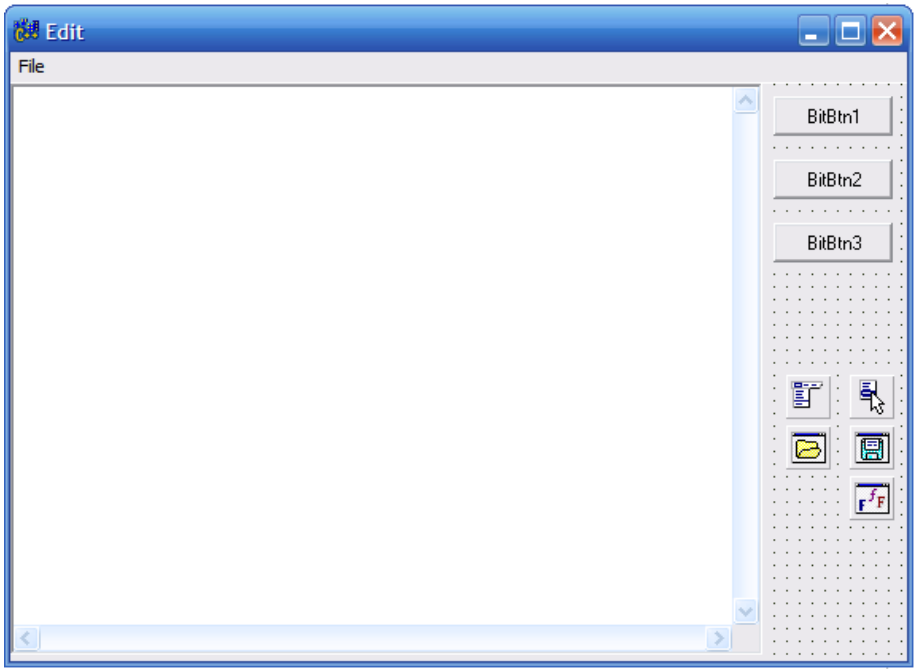


Рис. 21.1. Окно формы программы

Теперь приступим к изменению свойств остальных компонентов. Начнем с компонента **Memo**, выбрав его в инспекторе объектов. Вначале удалим надпись **Memo1** с компонента, для того чтобы она не появлялась при запуске приложения. Для этого изменим свойство **Lines**, щелкнув по кнопке с тремя точками правее надписи **Tstrings** этого свойства. При этом откроется окно редактора строки **String List Editor**, в котором необходимо удалить строку **Memo1** и нажать кнопку **ОК**. Теперь добавим для этого компонента линейки прокрутки для обеспечения возможности просмотра и редактирования больших областей текста с помощью мыши. Для этого в свойстве **ScrollBars** необходимо выбрать значение **ssBoth**. Для того чтобы работало контекстное меню, необходимо в свойстве **PopupMenu** выбрать значение **PopupMenu1**. Это значение стало доступным после помещения на форму проекта диалога **PopupMenu**. Настройку данного меню произведем позже.

Настроим свойства кнопок в соответствии с выполняемыми ими функциями. Для этого изменим поочередно свойство **Caption** для всех трех кнопок

на **Open**, **Save** и **Font** соответственно. Для того чтобы тип курсора изменялся при попадании на эти кнопки, изменим еще одно групповое свойство кнопок. Для этого с помощью мыши выделите все три кнопки, и в появившемся объекте **3 items selected** в инспекторе объектов измените свойство **Cursor** на значение **crHandPoint**.

Настройку главного меню произведем с помощью дизайнера меню, вызываемого двойным щелчком левой кнопки мыши по компоненту **MainMenu**. Ранее мы уже ввели один пункт меню под названием **&File**. Символ "ампер-санд" перед **File** производит выделение первой буквы названия в меню для быстрого доступа, путем ее подчеркивания. Следующий по горизонтали пункт меню назовем **&Edit**. Теперь заполним пункты меню по вертикали. Щелкните левой кнопкой мыши по пункту меню **File**, а затем ниже него по пустому полю. Свойство **Caption** этого пункта меню измените на **Open** и нажмите клавишу <Enter>. Следующий пункт назовите Save. Аналогично расширим по вертикали пункты меню **Edit**, добавив пункты **Font** и **Clear**. На этом этап создания главного меню программы редактора завершен. Все введенные пункты меню теперь будут присутствовать в приложении, но пока не будут выполняться, поскольку не созданы функции для выполнения соответствующих команд. Создадим их. Для этого воспользуемся компонентами диалогов, помещенных ранее на форму приложения. С помощью диалогов работы с файлами можно задать направленность приложения путем определения типов открываемых файлов. Это делается с помощью свойства **Filter**.

Выберите в инспекторе объектов компонент **OpenDialog1** и нажмите кнопку с тремя точками правее свойства **Filter**. При этом появится диалоговое окно **Filter Editor**, в котором производится настройка типов файлов. Окно разбито на два поля. В левом поле **Filter Name** вводятся строки пояснения. В правом поле **Filter** — расширения файлов. Для указания нескольких типов файлов необходимо записать их через точку с запятой. Если задать несколько строк с пояснениями и типами файлов, то при работе приложения их можно будет выбирать с помощью выпадающего списка. Заполните оба поля диалогового окна **Filter Editor** в соответствии с рис. 21.2 и нажмите кнопку **OK**.

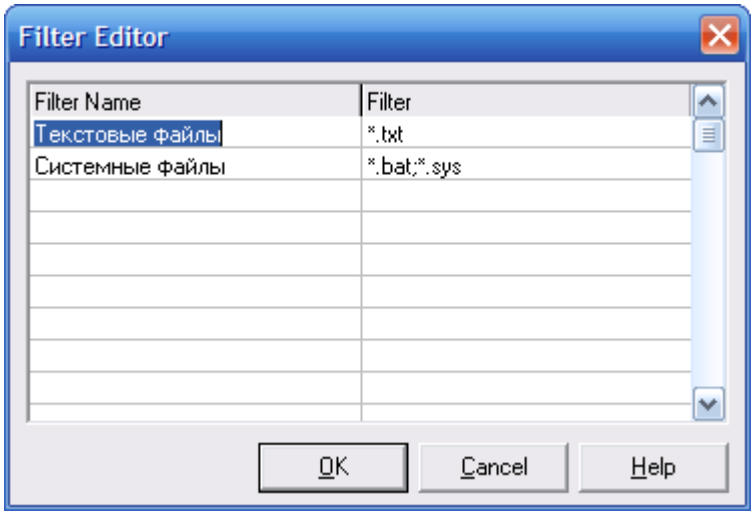


Рис. 21.2. Окно Filter Editor

Аналогично выполните настройку компонента **SaveDialog1**. Для корректности работы приложения оба эти диалога должны иметь одинаковую настройку свойства **Filter**. Чтобы облегчить эту работу и избежать ошибок, можно использовать два способа. Первый способ уже был показан ранее, когда несколько компонентов кнопок выделялось в группу и задавалось свойство одновременно для всех выделенных компонентов. Второй способ заключается в том, что, задав свойство для одного из компонентов, можно скопировать и вставить его для других компонентов с помощью

комбинации клавиш <Ctrl>+<C> и <Ctrl>+<V> соответственно. Так, после задания свойства **Filter** компонента **OpenDialog1** с помощью **Filter Editor** в поле этого свойства появится запись: Текстовые файлы |*.txt| Системные файлы |*.bat; *.sys. Выделив эту строку с помощью клавиш <Home> и <Shift>+<End>, скопируйте ее с помощью клавиш <Ctrl>+<C>. Теперь можно вставить эту строку в свойство **Filter** для компонента **SaveDialog1**. Замечательно то, что работу этих компонентов можно проверить еще до выполнения программы. Дело в том, что после задания свойства **Filter** этих компонентов двойной щелчок по компонентам приводит к открытию диалога на основе заданных свойств. Например, после двойного щелчка по компоненту **SaveDialog1** откроется окно, изображенное на рис. 21.3.

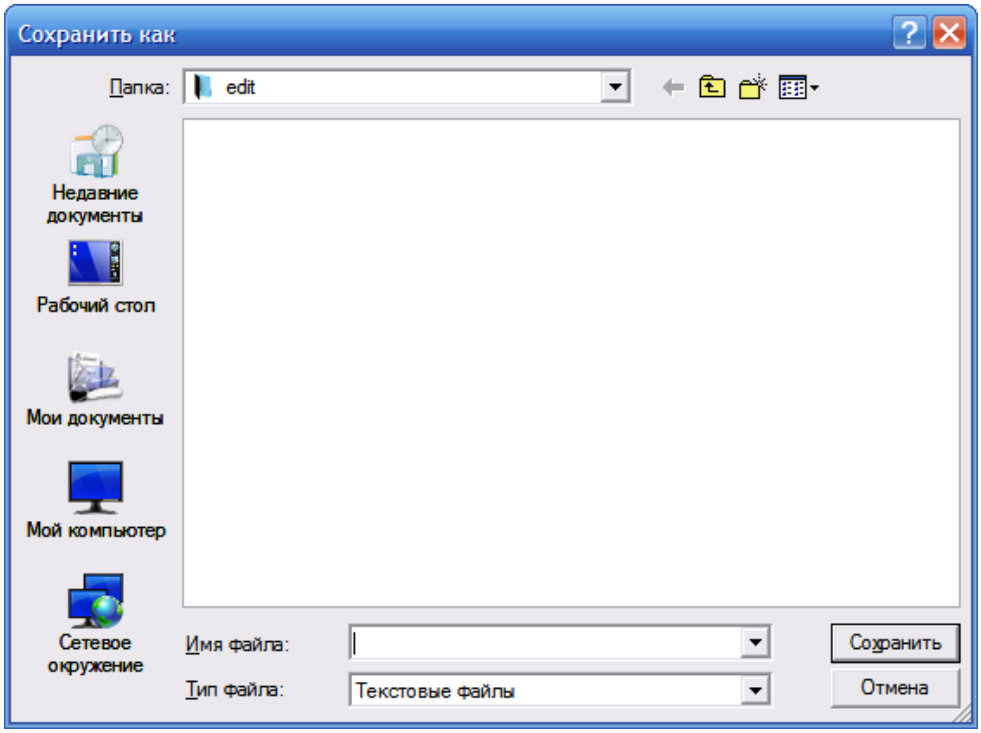


Рис. 21.3. Окно компонента Save Dialog

Теперь настроим компонент **FontDialog1**. Основным свойством данного компонента является **Font**, с помощью которого можно выбрать тип шрифта, его цвет, размер, стиль и т. п. Эти установки будут использоваться в программе редактора по умолчанию. С помощью свойства **Options** можно настроить вид и поведение диалога. Например, можно добавить в него кнопку справки, установив для подсвойства **fdShowHelp** значение **true**.

Нам осталось произвести настройку контекстного меню **PopupMenu**. Она производится аналогично настройке главного меню MainMenu. Добавим в это меню пункты с названиями **Font** и **Clear** через свойство **Caption**, как это показано на рис. 21.4.

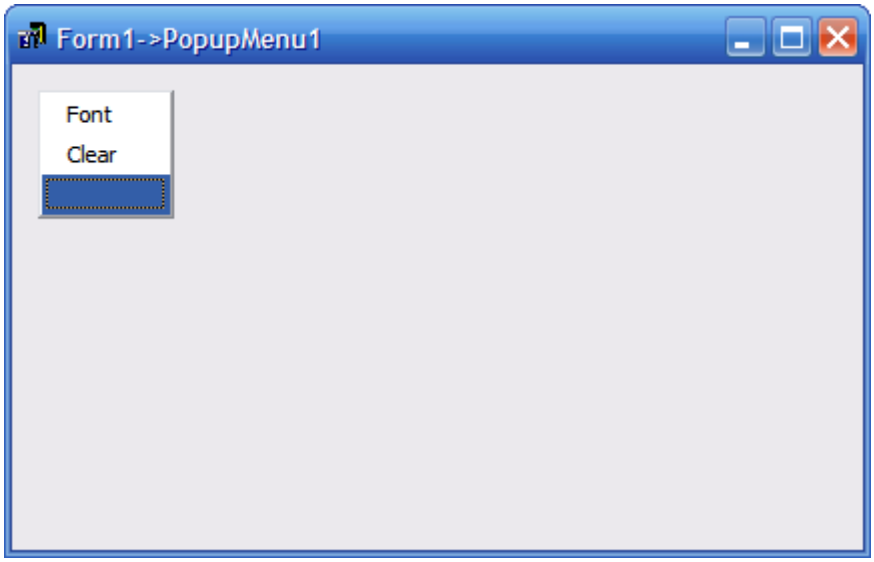


Рис. 21.4. Настройка контекстного меню PopMenu

Кроме того, добавьте название **PopupMenu1** в свойство **PopupMenu** главной формы приложения.

На этом настройка компонентов завершена и получена визуальная оболочка программы редактора. Однако без программных строк это приложение не будет работать. При запуске приложения будут открываться пункты меню, вводиться и редактироваться текст, но только потому, что эти функции заложены в сами компоненты. Нажатие на кнопки и вызов команд из главного меню не будут приводить к каким-либо действиям программы, поскольку не заданы функции обработки событий. Вдохнем в приложение жизнь. В программе редактора имеется четыре типа операций. Это открытие и сохранение файлов, настройка шрифтов и очистка окна редактора. Все эти операции можно вызвать разными способами. Например, с помощью главного меню, кнопок или контекстного меню. Можно написать функции для каждого способа, но это усложнит программу и внесет трудности при ее создании и отладке. Правильнее будет создать такую функцию один раз, а затем задать ее выполнение другим компонентам с помощью инспектора объектов. Создайте функцию для команды главного меню **Open**. Для этого откройте на главной форме пункт меню **File** и щелкните левой кнопкой мыши по команде **Open**. При этом активизируется редактор кода с заготовкой функции обработки. Введите между фигурными скобками следующий текст программы:

```
if (OpenDialog1->Execute() )
Memo1->Lines->LoadFromFile (OpenDialog1->FileName) ;
```

Обратите внимание на то, что среда разработки сама предоставляет выбрать функцию из множества разрешенных для компонента функций после введения символа стрелки. Эти программные строки позволят при запуске диалога открытия файла загрузить содержимое открываемого файла в свойство **Lines** компонента **Memo1** для отображения на экране. Теперь задайте выполнение этой функции кнопке команды **Open** с помощью инспектора объектов. Для этого щелкните левой кнопкой мыши по кнопке **Open** и выберите в инспекторе объектов вкладку **Events** (События). Раскройте список в событии **OnClick** и выберите в нем строку **Open1Click**. Таким образом будет задано, что при нажатии кнопки в программе редактора запустится та же функция открытия файла, что и для команды **Open** из пункта главного меню **File**. Запустите программу с помощью команды **Run** и убедитесь, что команды открытия файла работают и выбирают заданные нами типы файлов. Аналогично создадим функции для оставшихся трех типов команд. Начнем с команды сохранения файлов **Save**. Откройте на главной форме пункт меню **File** и щелкните левой

кнопкой мыши по команде **Save**. При этом активизируется редактор кода с заготовкой функции обработки. Введите между фигурными скобками следующий текст программы:

```
if (SaveDialog1->Execute() )
Memo1->Lines-> SaveToFile (SaveDialog1->FileName) ;
```

Эти программные строки позволят при запуске диалога сохранения файла загрузить содержимое свойство **Lines** компонента **Memo1** в сохраняемый файл. Задайте выполнение этой функции кнопке с командой Save с помощью инспектора объектов. Для этого щелкните левой кнопкой мыши по кнопке **Save** и выберите в инспекторе объектов вкладку Events. Раскройте список в событии **OnClick** и выберите в нем строку **Save1Click**. Таким образом будет задано, что при нажатии кнопки **Save** в программе редактора запустится та же функция сохранения файла, что и для команды **Save** из пункта главного меню **File**.
Для команды **Font** из пункта **Edit** главного меню введите с помощью инспектора кодов строки:

```
if (FontDialog1->Execute() )
Memo1-> Font=FontDialog1-> Font;
```

Это позволит изменять шрифт текста компонента **Memo1** при открытии диалога **FontDialog1** Задайте выполнение этой функции кнопке с командой **Font** с помощью инспектора объектов, для чего раскройте список в событии **OnClick** и выберите в нем строку **Font1Click**.
Теперь нужно добавить вызов этой функции в контекстное меню. Выделите на форме компонент **PopupMenu1** и в инспекторе объектов щелкните по кнопке с тремя точками в строке свойства **Items**. При этом откроется окно **Form1 | PopupMenu**. В этом окне щелкните на поле **Font** и в инспекторе объектов на закладке **Events** выберите для события **OnClick** строку **Font1Click**.
Наконец создадим функцию обработки очистки редактора. Для команды **Clear** из пункта **Edit** главного меню введите с помощью инспектора кодов одну-единственную строку:

```
Memo1->Clear ();
```

Эта программная строка будет производить очистку содержимого компонента **Memo1**. Аналогично предыдущему описанию назначьте выполнение данной функции через контекстное меню, выбрав строку **Clear1Click** для события **OnClick**.
На этом создание программы редактора заканчивается, и можно приступить к его проверке. Предварительно сохраните весь проект в отдельном каталоге, например, с именем **Edit**, а затем запустите его на выполнение. На экране должно появиться окно, аналогичное изображенному на рис. 21.5.

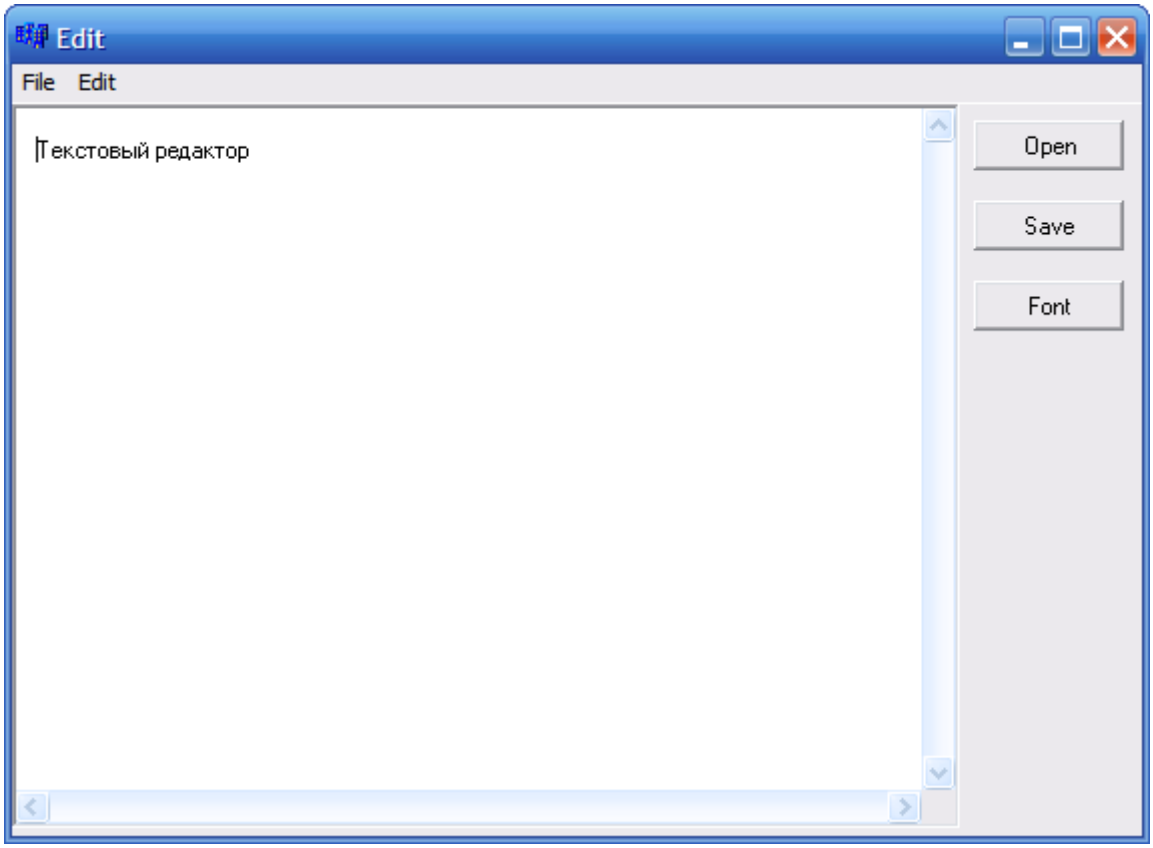


Рис. 21.5. Окно работающей программы

Нажмите кнопку **Open** и выберите какой-либо файл на диске. После чего загрузите его в редактор. Измените с помощью клавиатуры текст файла и сохраните его с новым именем, нажав кнопку **Save**. Нажмите кнопку **Font** и измените шрифт текста редактора. Убедитесь в том, что текст изменился в соответствии с вашими установками. Из главного меню выполните команду **Clear** и убедитесь, что поле редактирования очистилось. Проверьте работу остальных пунктов главного меню. Далее проверьте работу контекстного меню, нажав правую кнопку мыши на поле редактора и выбрав одну из двух команд. Обратите внимание на то, что при наведении курсора на кнопки он должен менять свой вид.
Редактор готов к работе. Создайте для него иконку, как это делали раньше, и используйте редактор как отдельное законченное приложение. Со временем вы сможете добавлять в него новые функции и возможности, повышая тем самым его функциональность и свое мастерство.

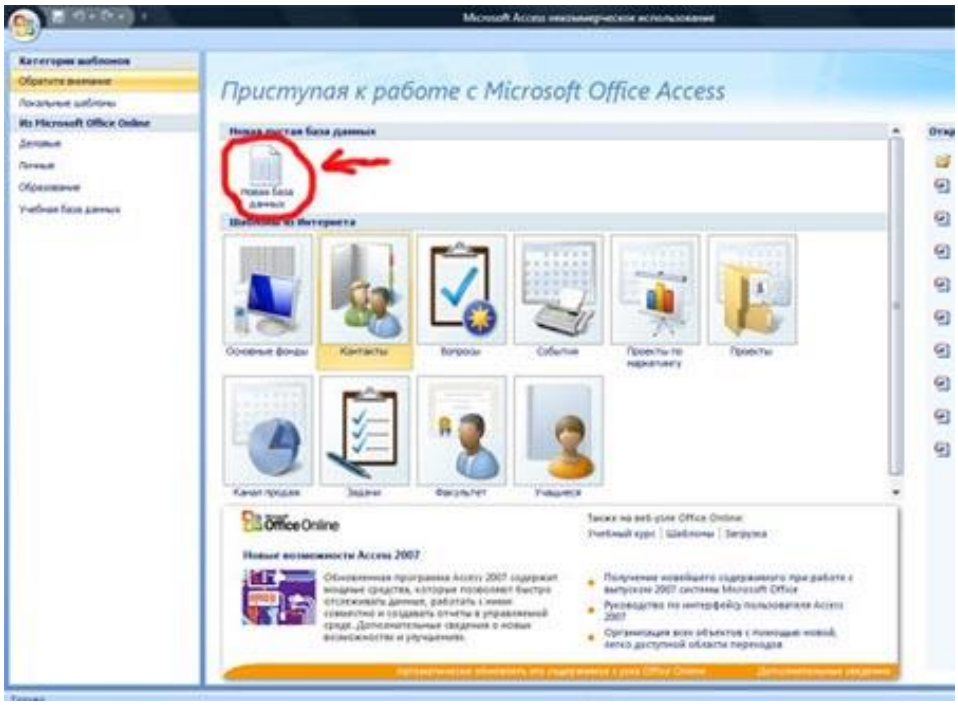
Лабораторная работа 38.
Программирование баз данных C++ Builder
(Создание базы в MS Access 2007)

Будем писать программу учета продуктов на складе по средней цене прихода. База данных будет включать в себя следующие возможности:

1. Справочник продуктов, справочник групп продуктов, справочник поставщиков, справочник единиц измерения.
2. Форму прихода продуктов на склад.
3. Форму расхода продуктов.
4. Возможность во всех формах которые описаны выше добавлять, редактировать и удалять записи.
5. Форму просмотра продуктов на складе.

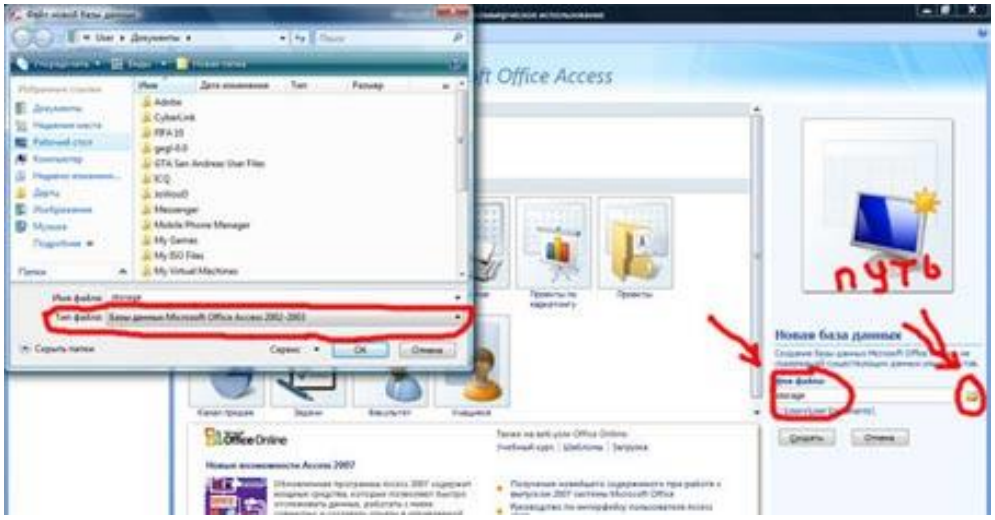
6. Печать данных прихода, расхода и состояния склада.

А для этого нам нужно научиться работать с базами данных. Начнем работу с создания базы в MS Access 2007. Создадим базу с названием storage.mdb, для этого запустим MS Accecss 2007.

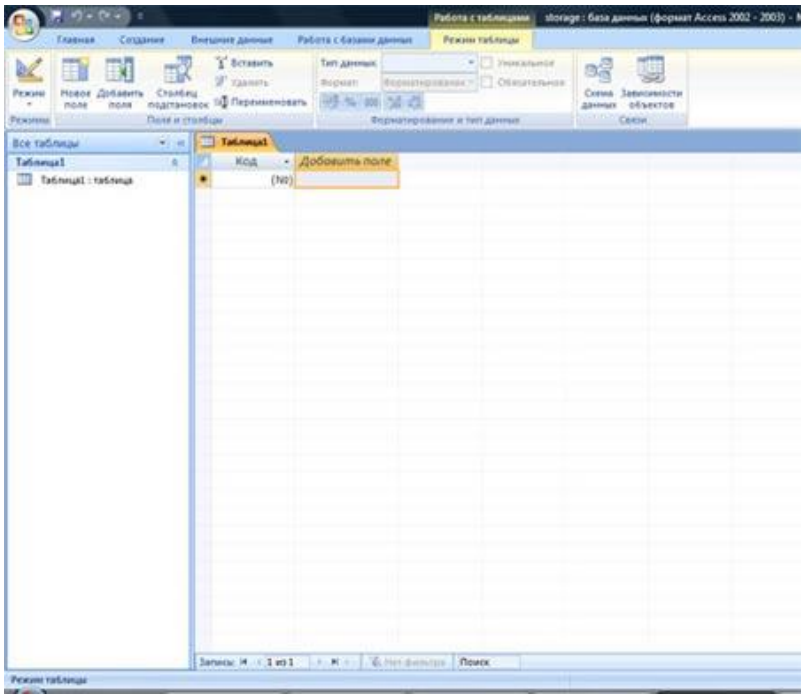


Выбираем, новая база данных и вводим имя файла storage,

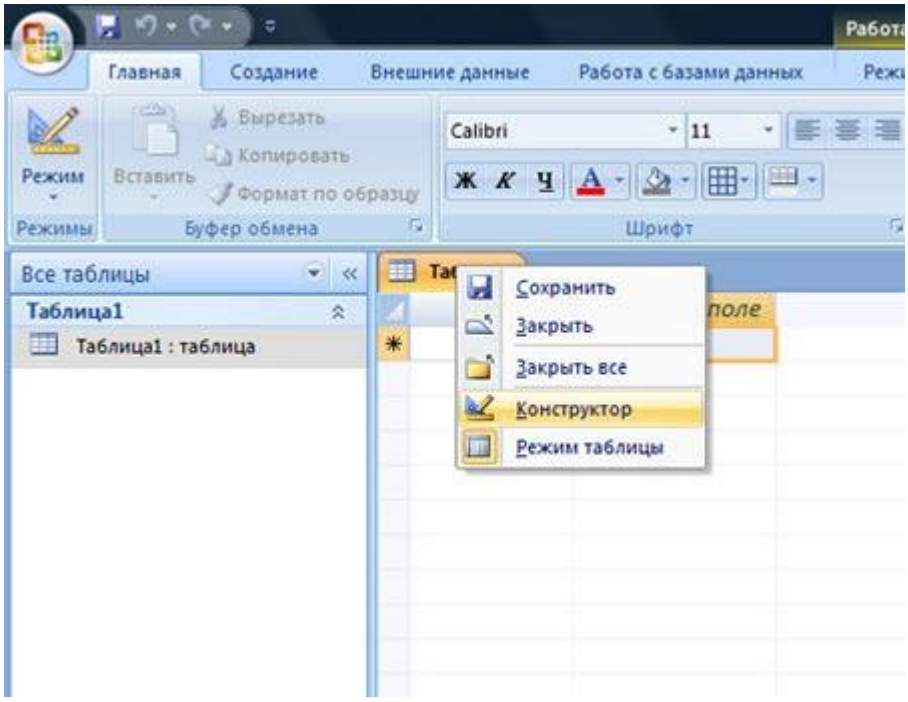
Выбираем путь для сохранения базы данных и тип файла Базы данных Microsoft Office 2002-2003, жмем ОК, а затем



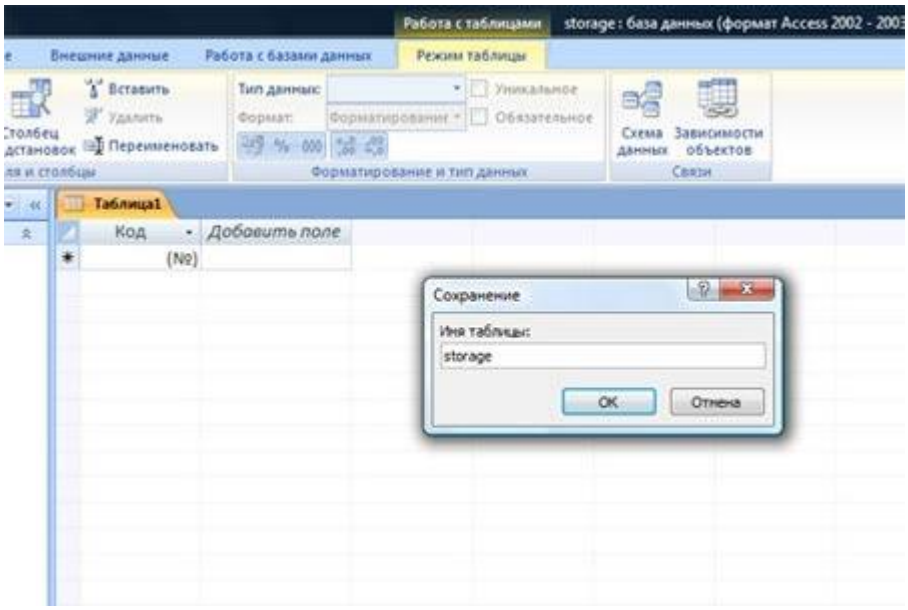
нажимаем кнопку создать и видим следующее окно.



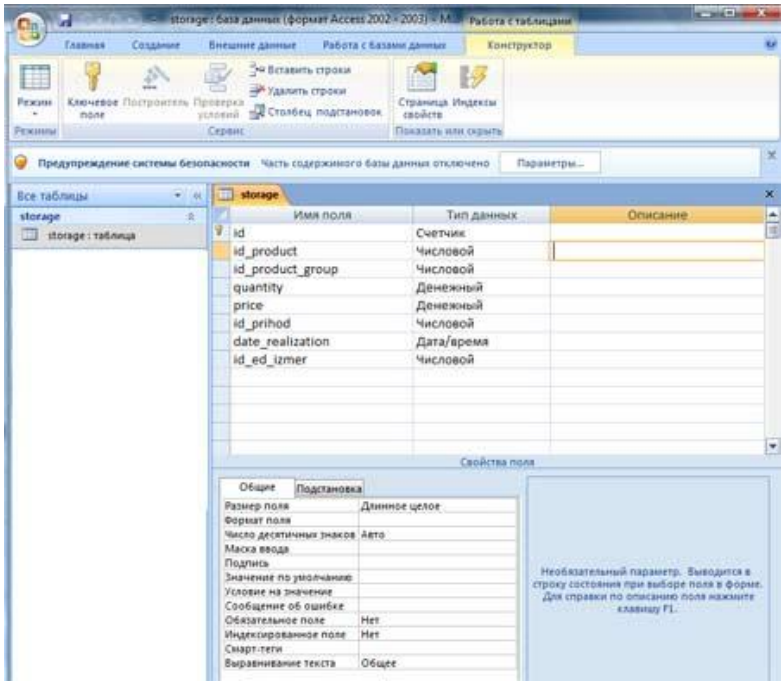
Теперь создадим таблицу storage, для этого щелкнем правой кнопкой мыши на вкладке Таблица1 и выберем Конструктор.



Затем вводим имя таблицы storage и жмем ОК.



Далее создаем следующие поля таблицы



id (счетчик) – нужно задать как ключевое,

id_product (тип данных –числовой, в свойствах поля- размер поля- Длинное целое) – код продукта,

id_product_group (тип данных –числовой, в свойствах поля- размер поля- Длинное целое) – код группы продуктов,

quantity (тип данных -денежный, в свойствах поля- формат поля- основной, число десятичных знаков - 3) – количество,

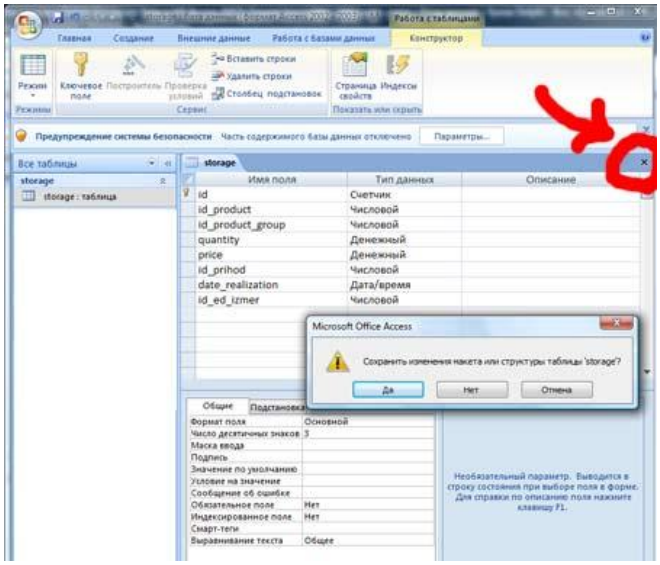
price (тип данных -денежный, в свойствах поля- формат поля- основной, число десятичных знаков - 4) – цена,

(По поводу выбора денежного типа для хранения цены и количества смотрите форум http://basicsprog.ucoz.ru/forum/3-2-1);

id_prihod (тип данных –числовой, в свойствах поля- размер поля- Длинное целое) – код прихода,

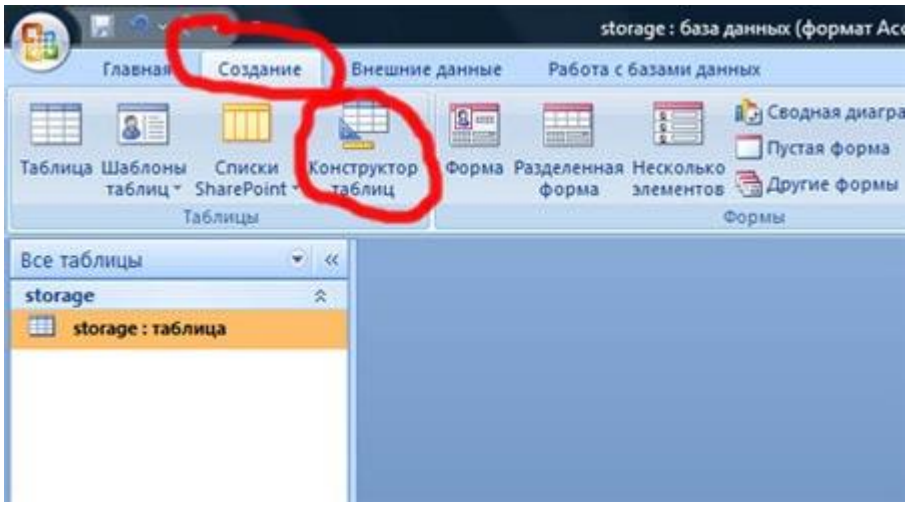
date_realization (тип данных - Дата/время) – дата реализации,

id_ed_izmer (тип данных –числовой, в свойствах поля- размер поля- Длинное целое) – код единицы измерения.

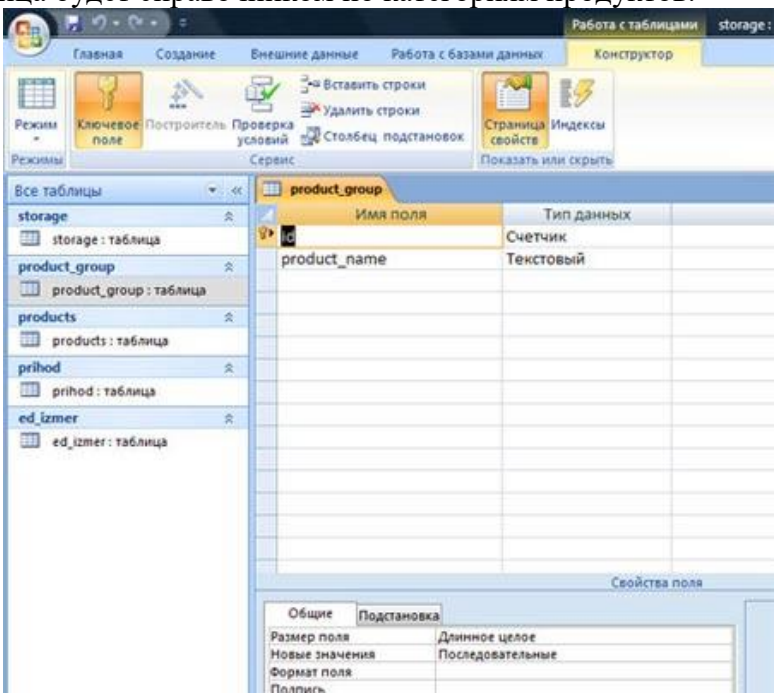


затем нажимаем на крестик и подтверждаем сохранение.

Далее переходим на вкладку создание, выбираем конструктор таблиц и создаем еще пять таблиц.

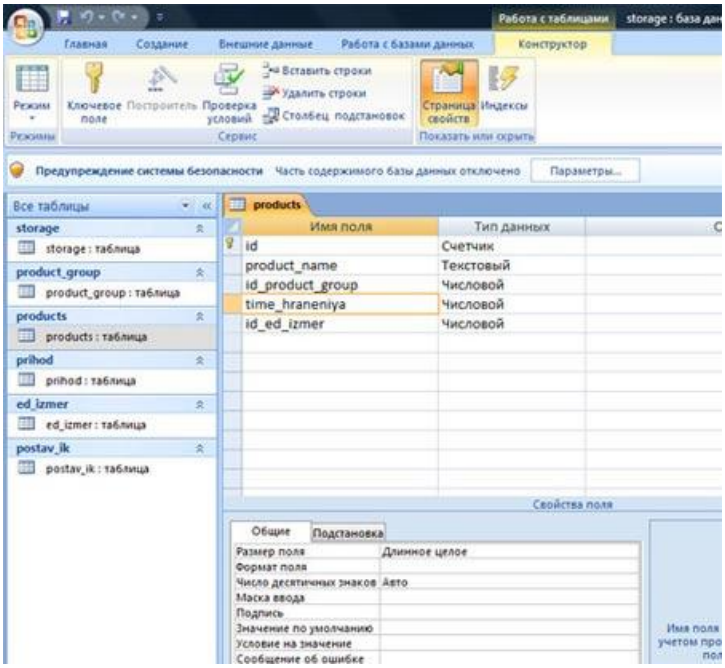


1. Таблица product_group, которая содержит следующие поля:
id (счетчик) – нужно задать как ключевое,
product_name (текстовый) –наименование категории продукта.
Данная таблица будет справочником по категориям продуктов.



2. Таблица products, которая содержит следующие поля:
id (счетчик) – нужно задать как ключевое,
product_name (текстовый) –наименование продукта,
id_product_group (тип данных –числовой, в свойствах поля- размер поля- Длинное целое) – код группы продуктов,
time_hraneniya (тип данных –числовой, в свойствах поля- размер поля- Длинное целое) – время хранения продукта,
id_ed_izmer (тип данных –числовой, в свойствах поля- размер поля- Длинное целое) – код единицы измерения по умолчанию.

Эта таблица будет справочником продуктов.



3. Таблица prihod, которая содержит следующие поля:

id (счетчик) – нужно задать как ключевое,

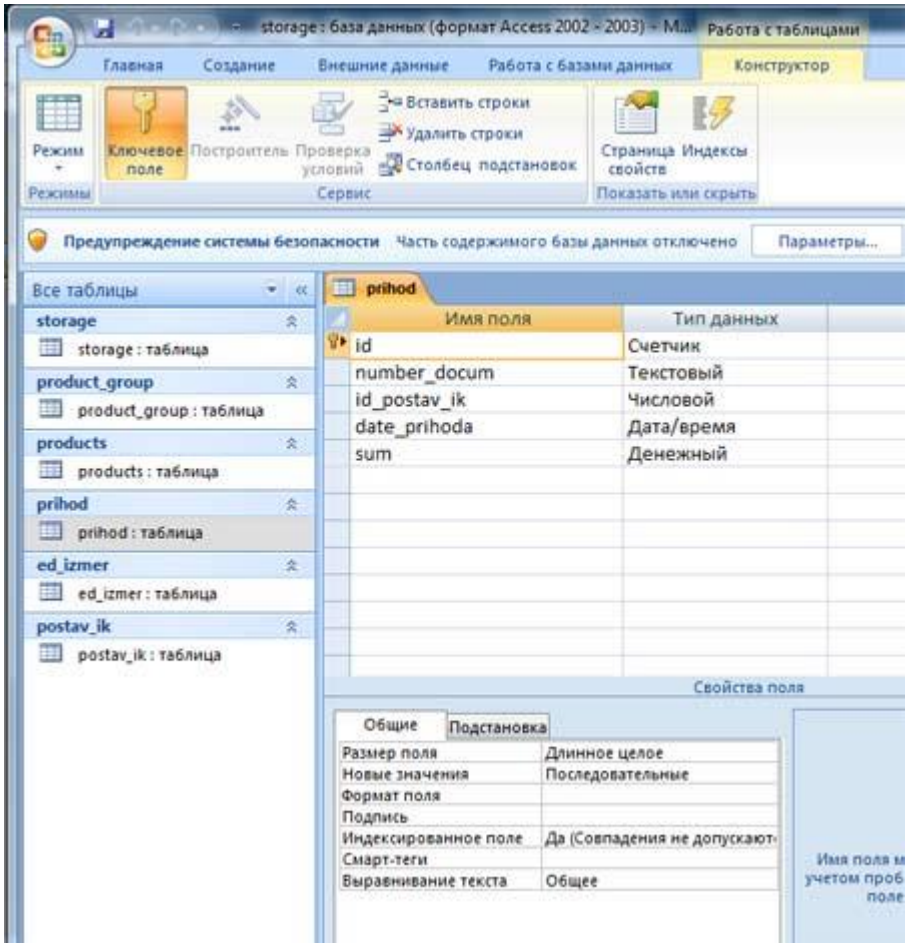
number_docum (текстовый, в свойствах поля- размер поля- 20) – номер накладной

id_postav_ik (тип данных –числовой, в свойствах поля- размер поля- Длинное целое) – код поставщика,

date_prihoda (тип данных - Дата/время) – дата прихода,

sum (тип данных -денежный, в свойствах поля- формат поля- основной, число десятичных знаков - 4) – сумма прихода.

Таблица прихода продуктов на склад.

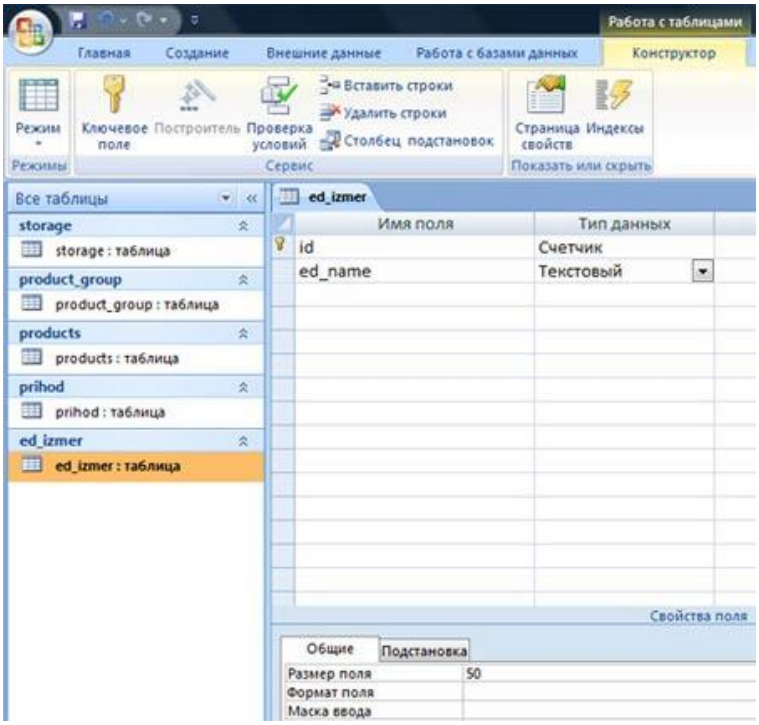


4. Таблица ed_izmer, которая содержит следующие поля:

id (счетчик) – нужно задать как ключевое,

ed_name (текстовый, в свойствах поля- размер поля- 50) – наименование единицы измерения.

Таблица справочник по единицам измерения.



5. Таблица postav_ik, которая содержит следующие поля:

id (счетчик) – нужно задать как ключевое,

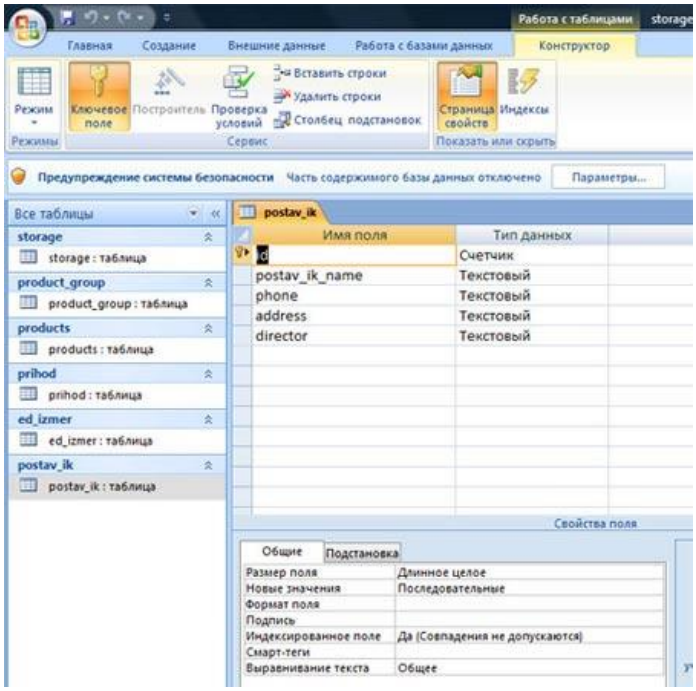
postav_ik_name (текстовый, в свойствах поля- размер поля- 255) – наименование организации поставщика,

phone (текстовый, в свойствах поля- размер поля- 50) – телефон организации,

address (текстовый, в свойствах поля- размер поля- 255) – адрес организации,

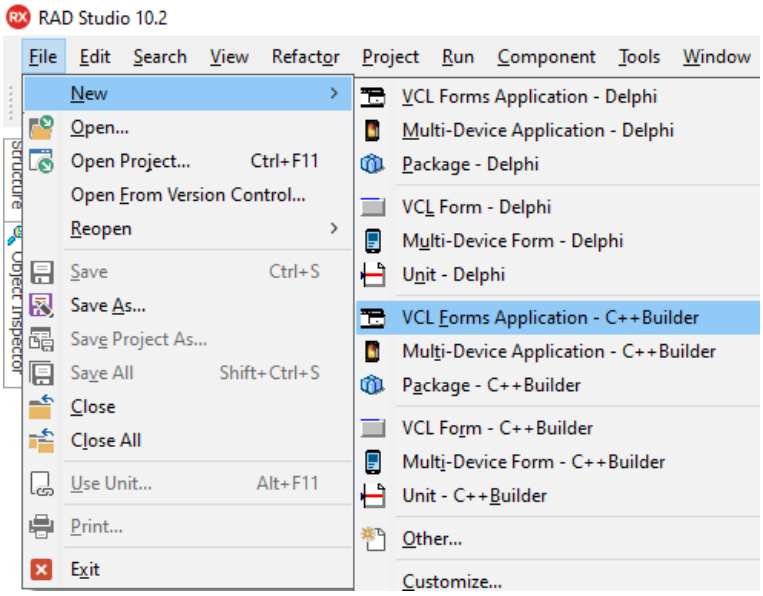
director (текстовый, в свойствах поля- размер поля- 255) – руководитель организации.

Таблица будет справочником поставщиков.

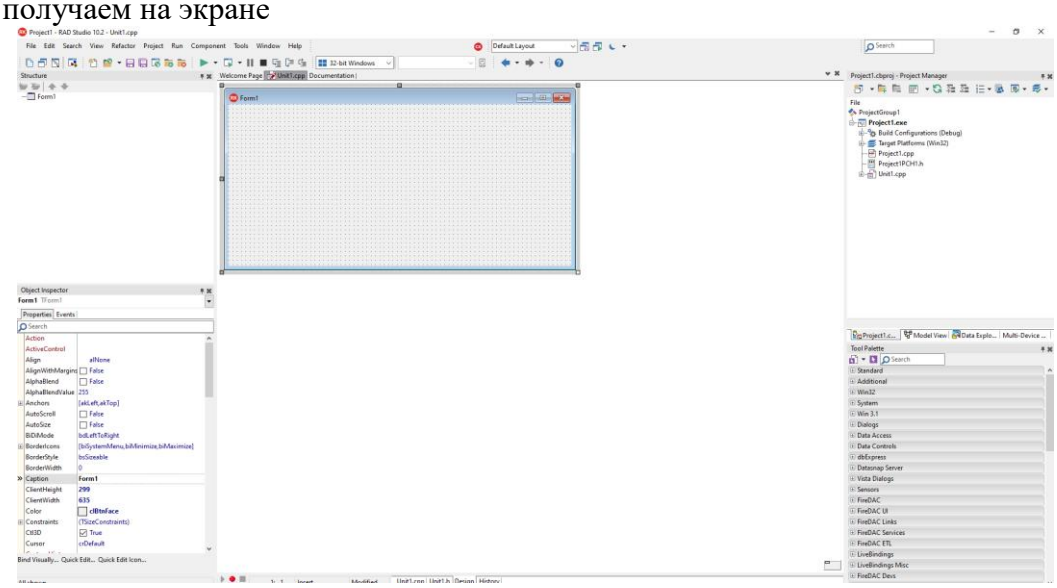


Создаем главную форму программы склад

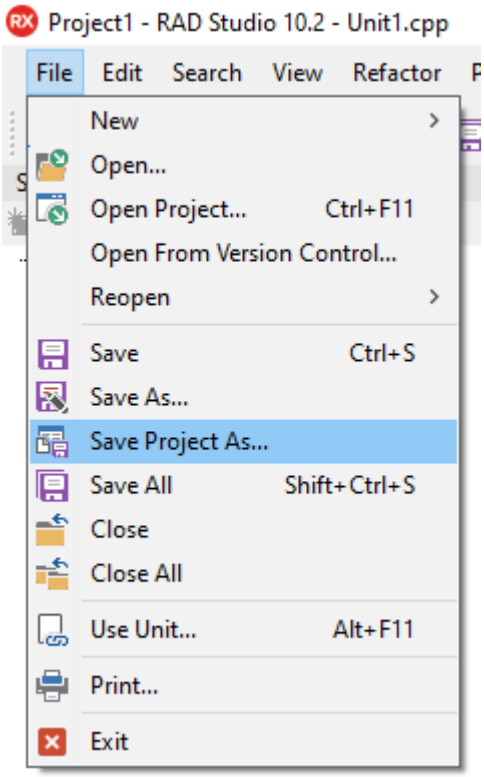
Запускаем среду быстрой разработки Embarcadero Rad Studio.

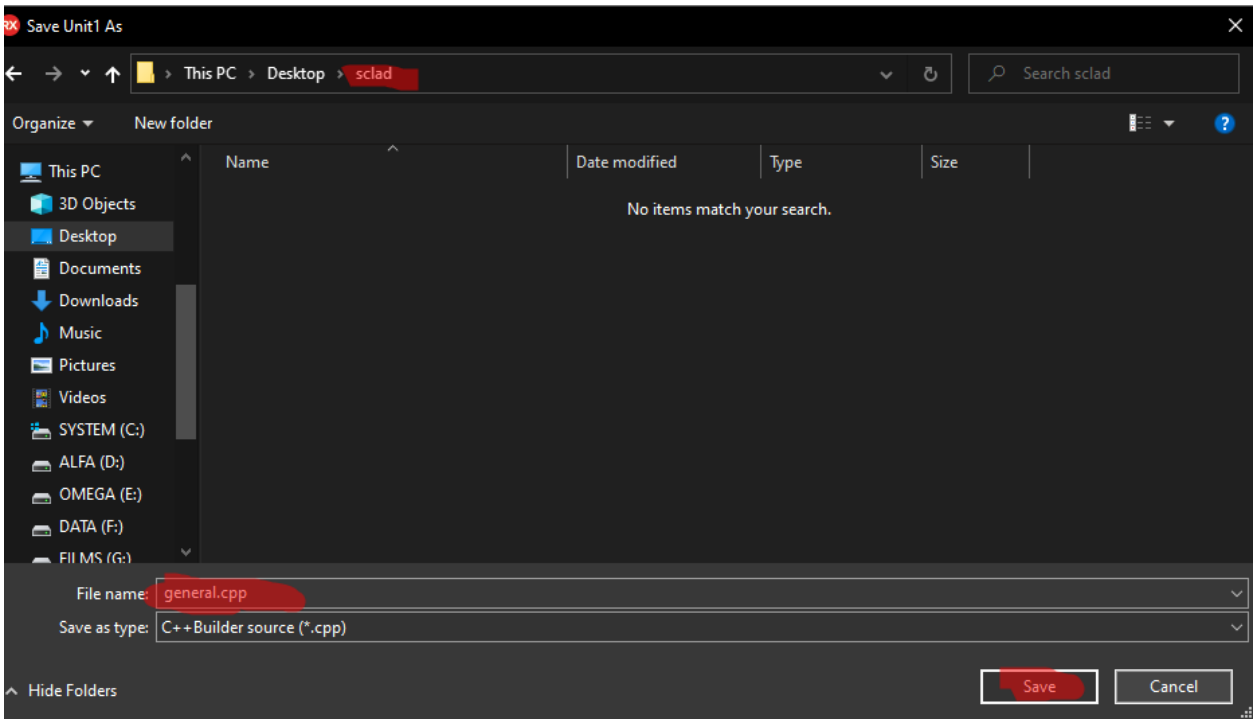


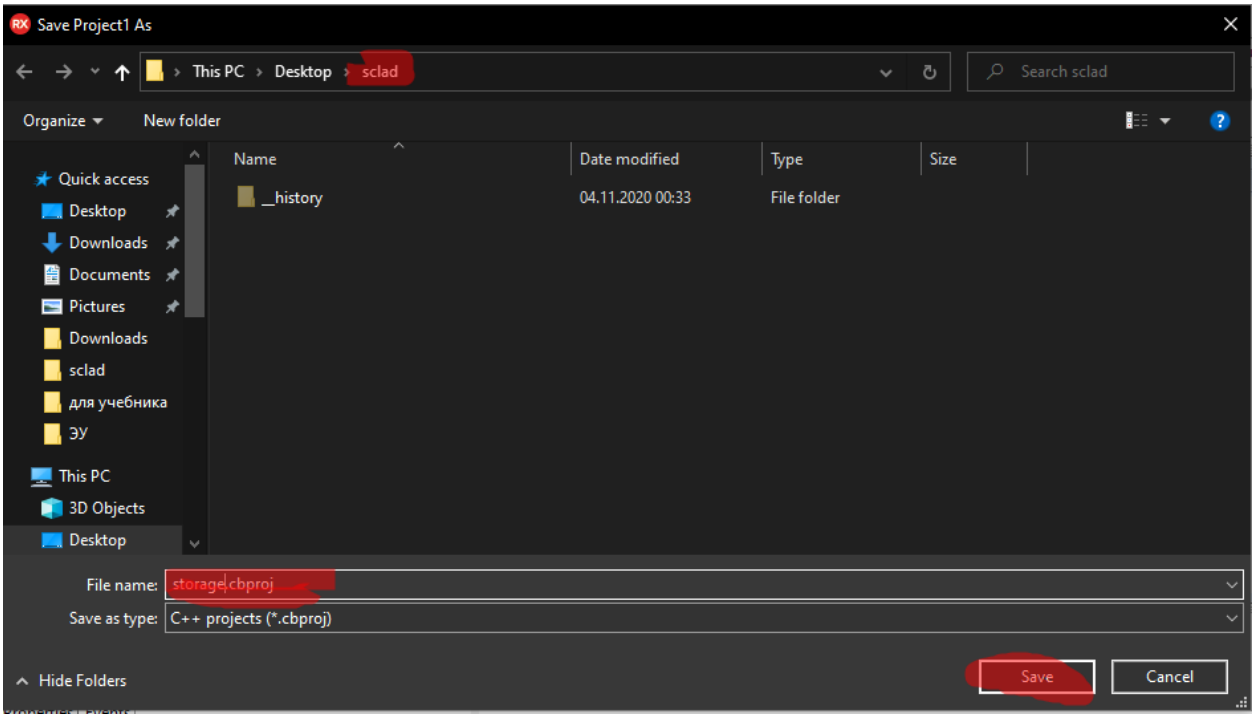
и получаем на экране



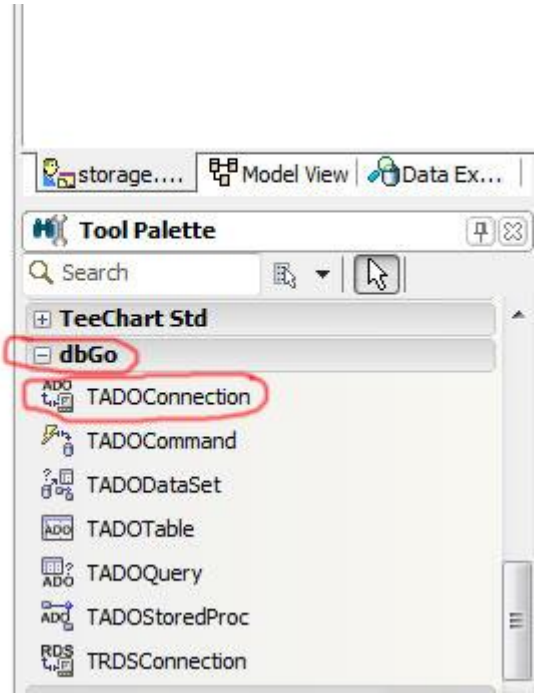
Теперь сохраним наш проект в нужную нам папку (у меня папка **sclad**) и назовем файл (модуля cpp) как **general**, а файл проекта **storage**. Для этого выбираем **File->Save Project As…**


Project1 - RAD Studio 10.2 - Unit1.cpp
File
Edit
Search
View
Refactor
New
Open...
Open Project... Ctrl+F11
Open From Version Control...
Reopen
Save Ctrl+S
Save As...
Save Project As...
Save All Shift+Ctrl+S
Close
Close All
Use Unit... Alt+F11
Print...
Exit


Save Unit1 As
This PC
Desktop
sclad
Search sclad
Organize
New folder
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
SYSTEM (C:)
ALFA (D:)
OMEGA (E:)
DATA (F:)
Name
Date modified
Type
Size
No items match your search.
File name: general.cpp
Save as type: C++Builder source (*.cpp)
Hide Folders
Save
Cancel



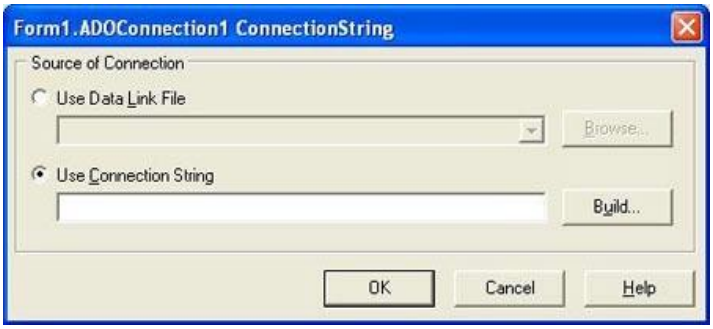
Создадим подключение, к базе, используя компонент TADOConnection. Размещаем на форме компонент **TADOConnection** вкладки **dbGo** (старое название **ADO**).



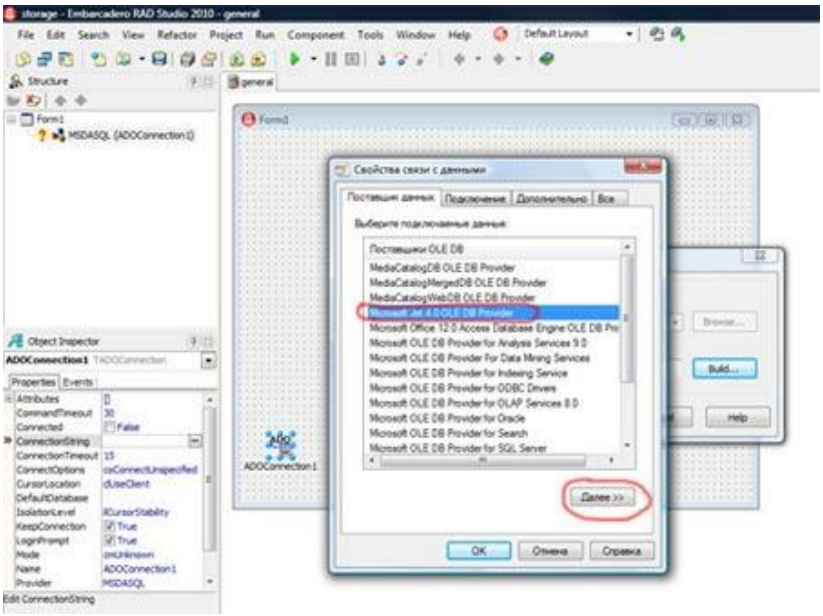
Перед настройкой подключения к базе, необходимо скопировать файл базы данных (**storage.mdb)**, созданный на прошлом уроке в папку с проектом.

Начинаем подключение…

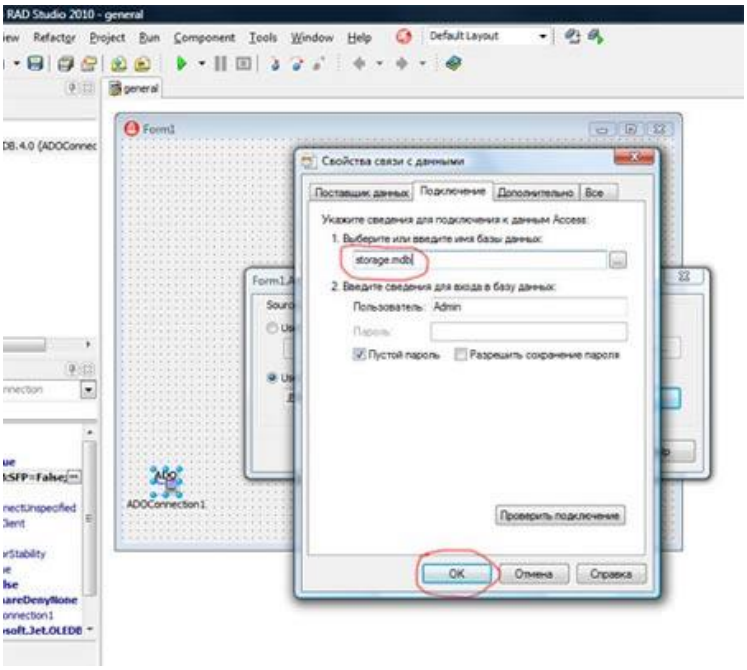
Выделяем компонент **TADOConnection** и в свойстве **LoginPromt** ставим на **False**. Это делаем для того, чтобы при подключении к БД пароль у нас не запрашивался, дальше в свойстве **ConnectionString** нажимаем на кнопку с «…» и видим:



В окне нажимаем на кнопку «**Build…**» и появляется следующее окно:



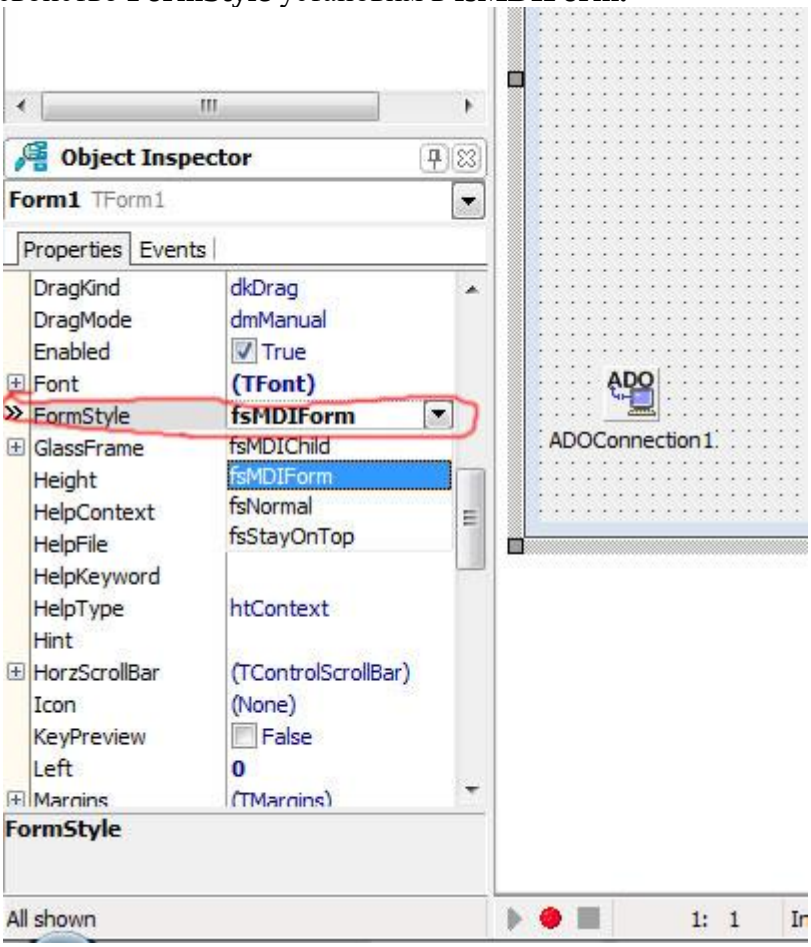
Выбираем провайдера, а именно **Microsoft Jet 4.0 OLE DB Provaider** и нажимаем кнопку «**Далее**».



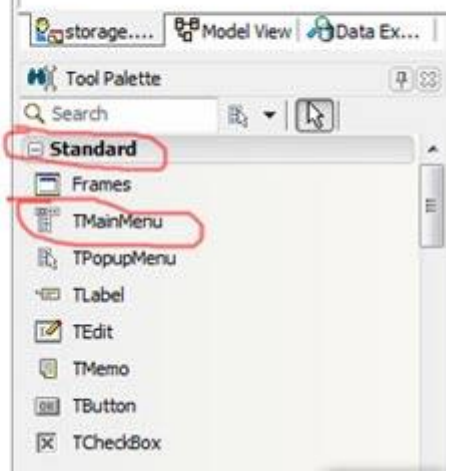
В данном окне мы указываем путь к нашей БД и имя пользователя по умолчанию **Admin ,** жмем кнопку **«Ок»**. Если ваша БД находится в корневом каталоге с программой, то в данном поле достаточно указать ее имя с расширением, примерно так **storage.mdb**.

В свойстве компонента **TADOConnection – Mode** выбрать из выпадающего списка **cmShareDenyNone,** а свойство **TADOConnection – Connected** ставим **True.** Все подключение к базе активировано.

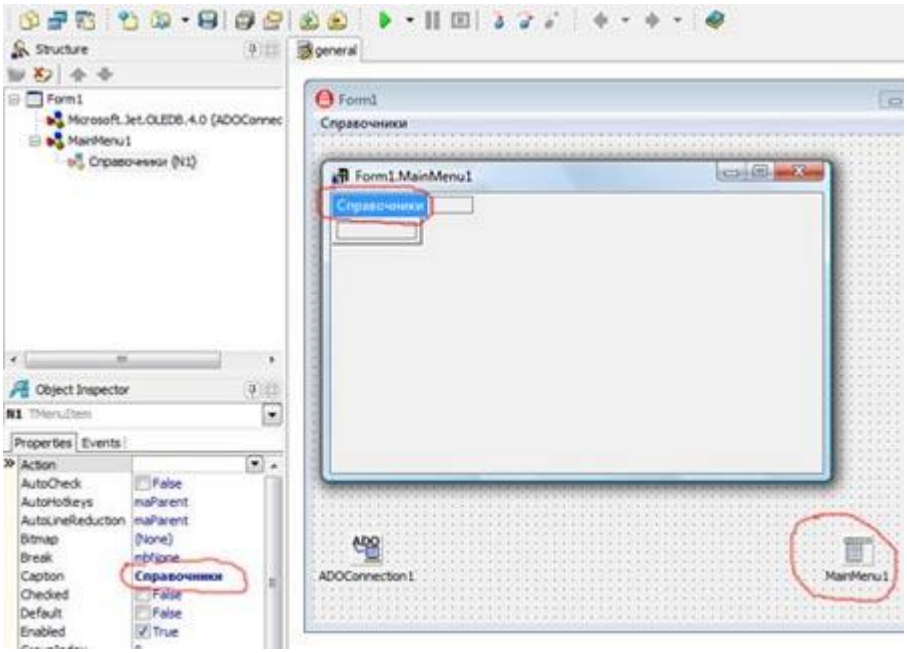
Сделаем нашу форму главной MDI формой, для этого в инспекторе объектов в свойствах **Form1** свойство **FormStyle** установим в **fsMDIForm.**



Добавим на форму компонент **TMainMenu** из вкладки **Standard**



Щелкнем дважды мышкой на компоненте **MainMenu1** и увидим окно для создания меню.



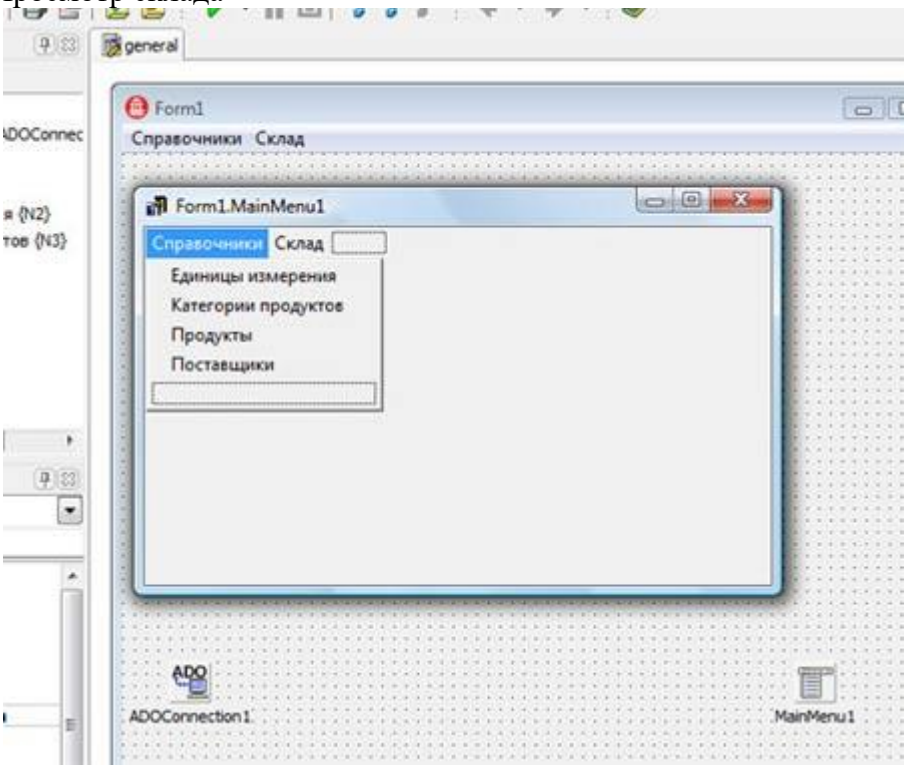
В свойстве Caption инспектора объектов задаем следующие пункты меню:

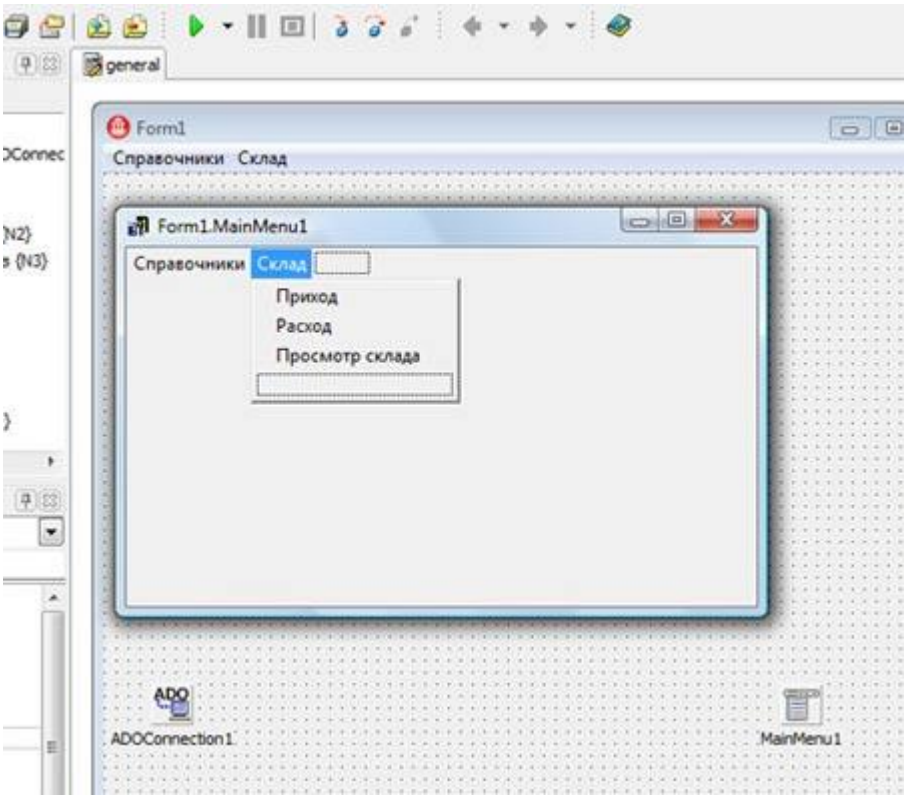
Справочники

- Единицы измерения
- Категории продуктов
- Продукты
- Поставщики

Склад

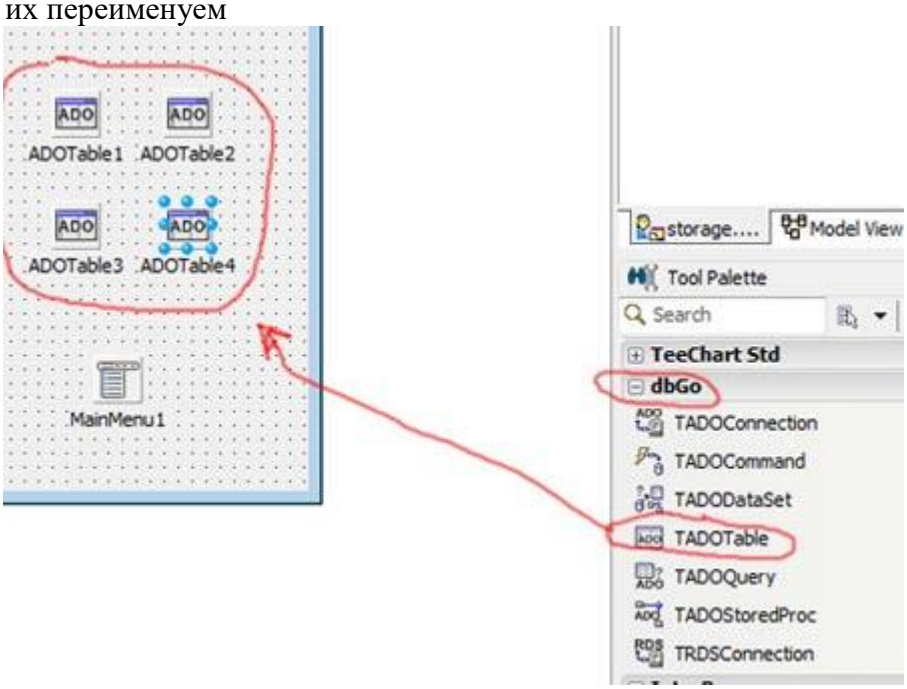
- Приход
- Расход
- Просмотр склада





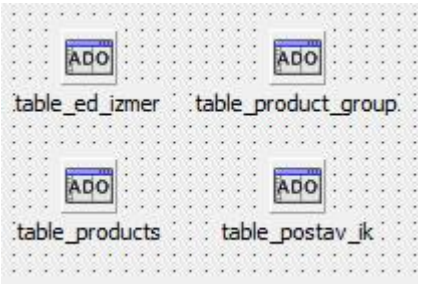
Заканчиваем создание меню и жмем на крестик.

Далее из панели компонентов **dbGo** (ADO) размещаем на форме четыре компонента **TADOTable**, к ним мы подключим наши справочники из базы данных, но для начала мы их переименуем



ADOTable1 – table_ed_izmer,
ADOTable2 – table_product_group,
ADOTable3 – table_products,
ADOTable4 – table_postav_ik.

Для этого в инспекторе объектов **ADOTable1** в свойство Name записываем **table_ed_izmer** и так делаем для **ADOTable2, ADOTable3, ADOTable4**. Должно получиться следующее:



Подключим наши таблицы к компоненту **ADOConnection1** и к одноименным таблицам нашей БД.

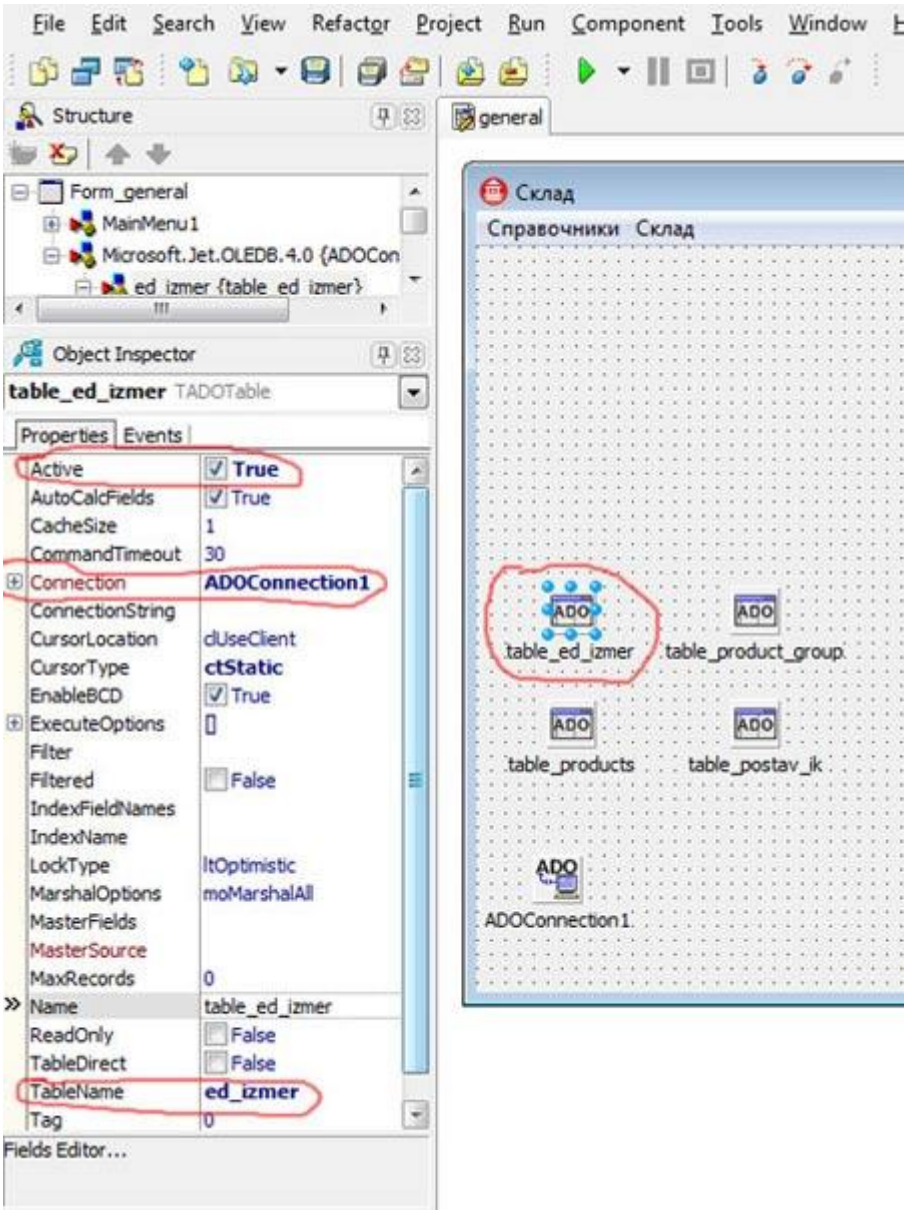
Для этого в инспекторе объектов **для таблицы table_ed_izmer (TADOTable)** свойства **Connection устанавливаем -> ADOConnection1, TableName-> ed_izmer, Active-> True;**

для таблицы table_product_group (TADOTable) свойства **Connection устанавливаем -> ADOConnection1, TableName-> product_group, Active-> True;**

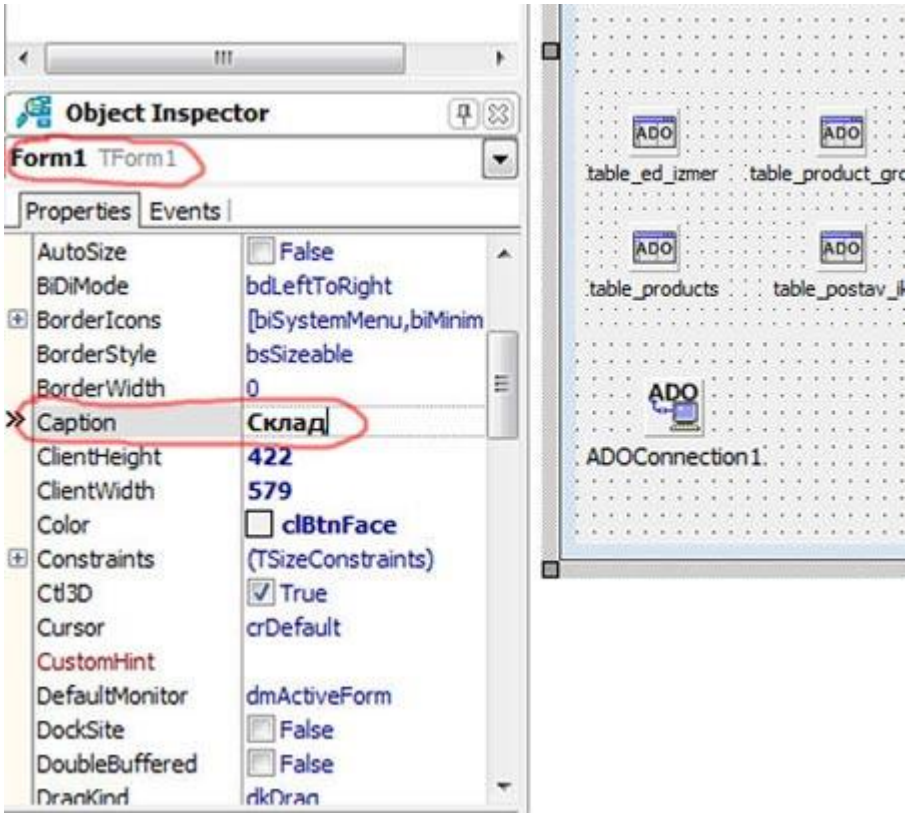
для таблицы table_products (TADOTable) свойства **Connection устанавливаем -> ADOConnection1, TableName-> products, Active-> True;**

для таблицы table_postav_ik (TADOTable) свойства **Connection устанавливаем -> ADOConnection1, TableName-> postav_ik , Active-> True;**

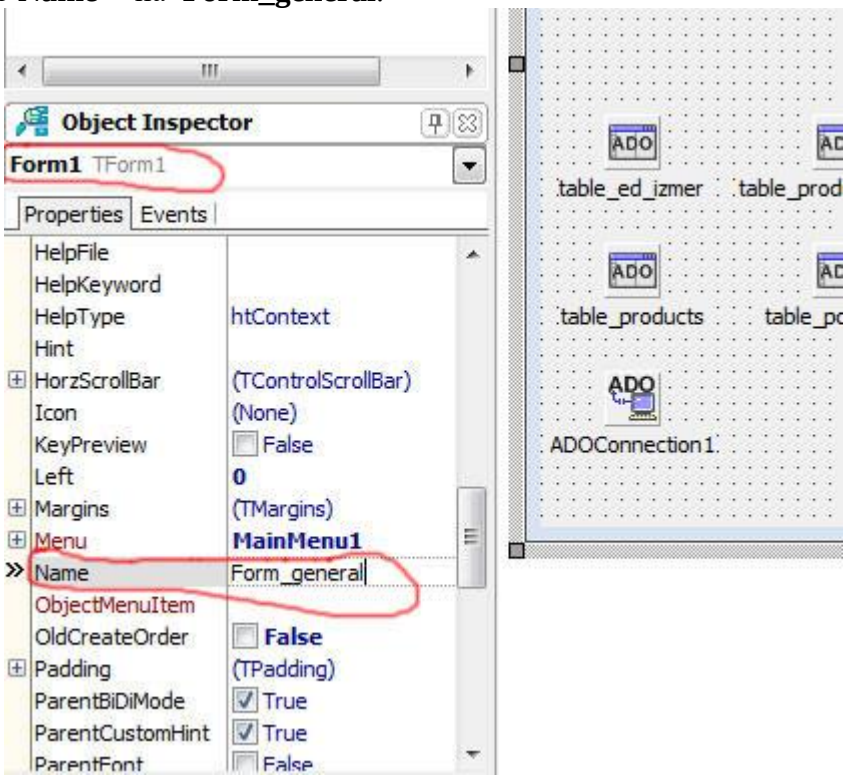
Свойство **Active -> True** устанавливайте в последнюю очередь.



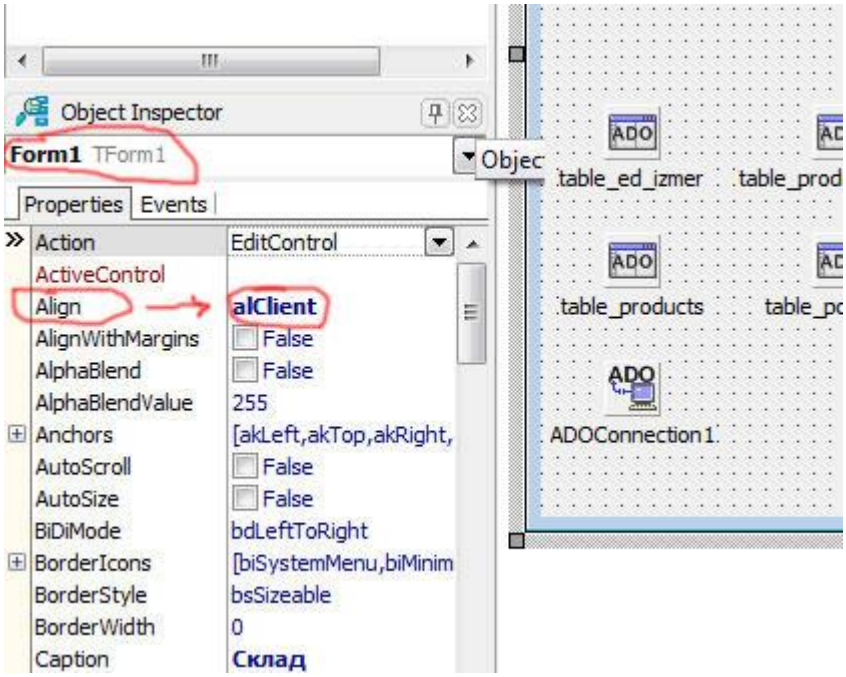
На последок изменим в инспекторе объектов свойство **Caption** главной формы (Form1) на **Склад**,



а свойство **Name** ->на **Form_general**.



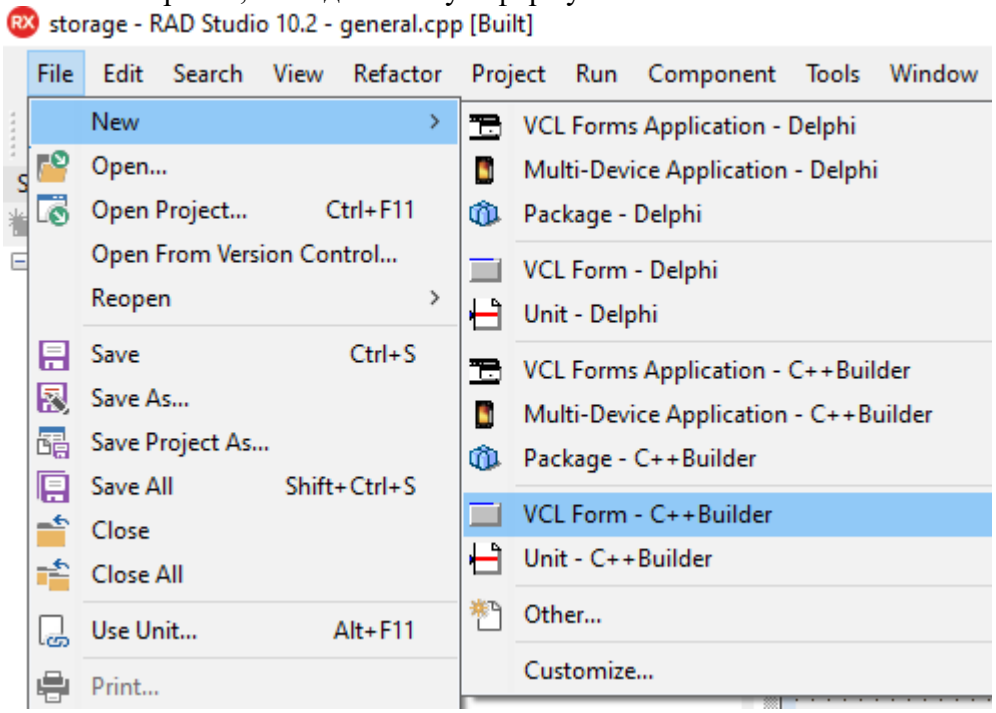
А чтобы форма разворачивалась на весь экран **Align -> alClient**



Лабораторная работа 39.
Создаем подчиненные формы для справочников программы склад

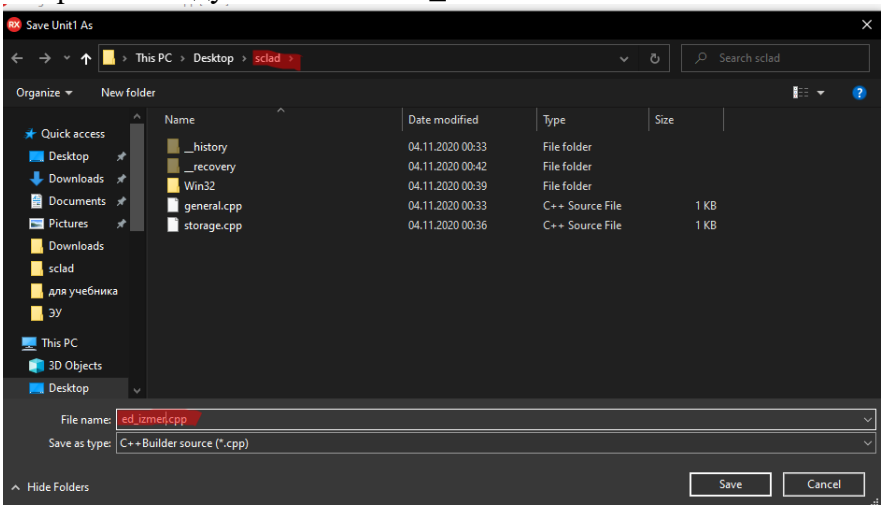
На этом уроке мы создадим форму для справочника единицы измерения нашего склада.

Запускает наш проект, и создаем новую форму File->New->Form -



В инспекторе объектов устанавливаем следующие свойства для новой формы
Caption -> Единицы измерения;
Name -> Form_ed_izmer;

Сохраните модуль с именем **ed_izmer.**



Пропишем **general;** в программном модуле **ed_izmer** как показано на рисунке.

```
//---------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "ed_izmer.h"
#include "general.h"
//---------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm_ed_izmer *Form_ed_izmer;
//---------------------------------------------
__fastcall TForm_ed_izmer::TForm_ed_izmer(TCompo
    : TForm(Owner)
{
}
//---------------------------------------------
```

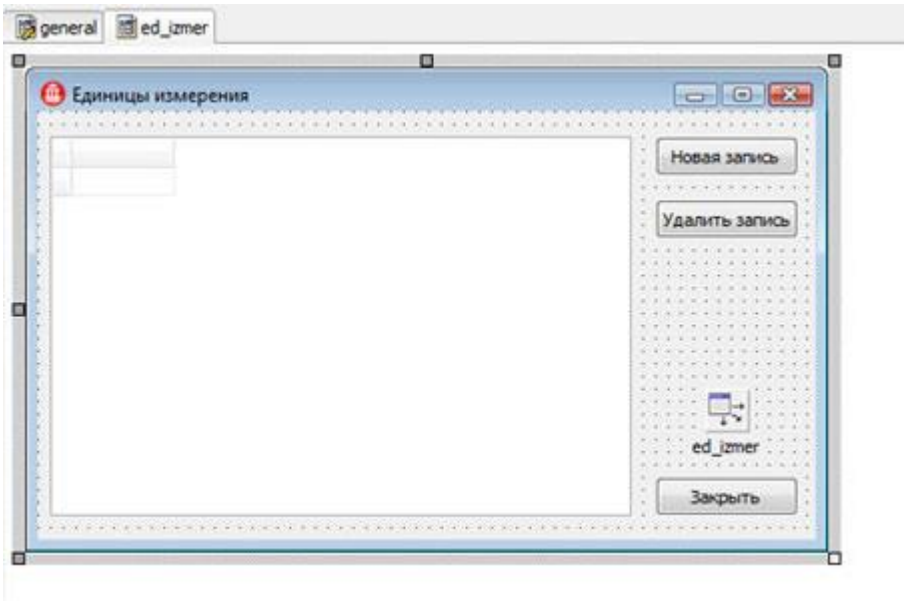
А программном модуле **general** главной формы пропишем **ed_izmer;**

```
#include <vcl.h>
#pragma hdrstop

#include "general.h"
#include "ed_izmer.h"
//---------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm_general *Form_general;
//---------------------------------
__fastcall TForm_general::TForm_ge
    : TForm(Owner)
{
}
//---------------------------------
```

Это делается для того, чтобы установить связь между формами, иначе нельзя будет обращаться из одной формы к компонентам другой формы.

Далее размещаем на форме следующие компоненты

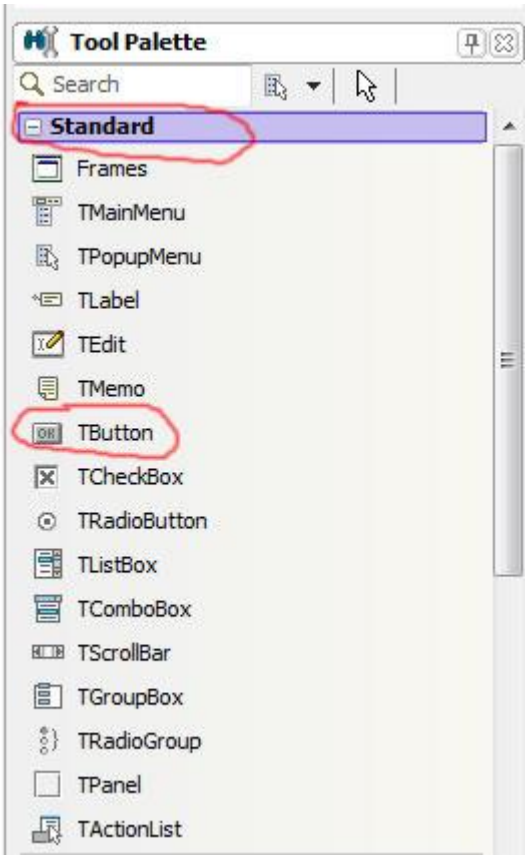


Три кнопки **TButton** из вкладки **Standart**

В инспекторе объектов для **Button1** устанавливаем свойство **Caption -> Новая запись;**

для **Button2** устанавливаем свойство **Caption -> Удалить запись;**

для **Button3** устанавливаем свойство **Caption -> Закрыть;**

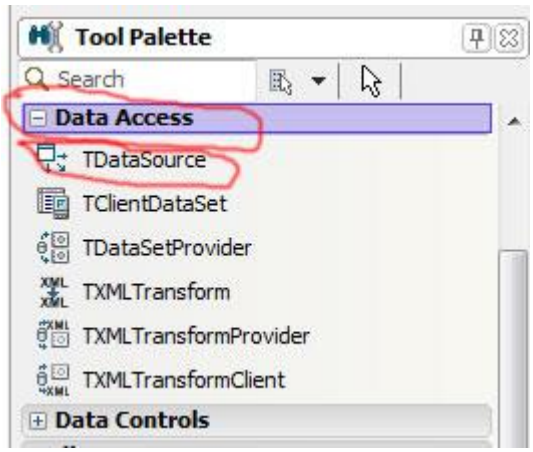


Из вкладки **Data Access** помещаем на форму компонент **TDataSource.**

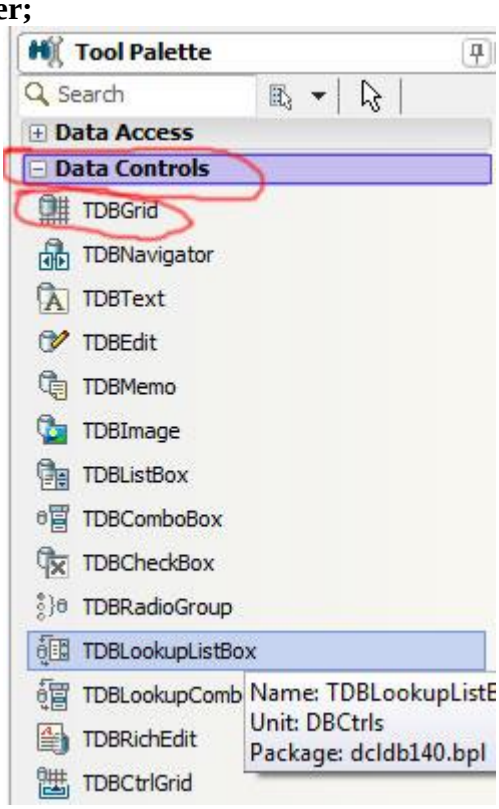
В инспекторе объектов для него устанавливаем следующие свойства:

DataSet -> Form_general.table_ed_izmer;

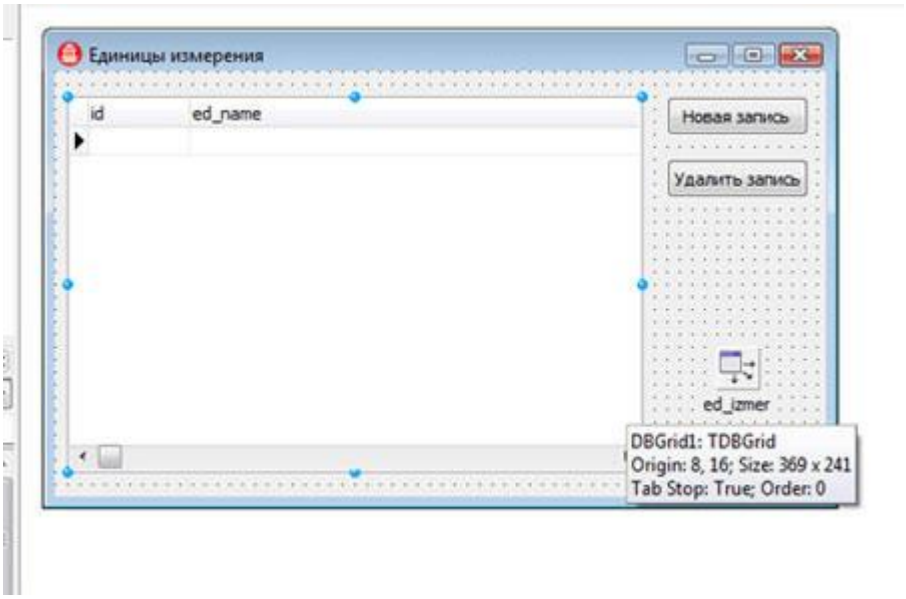
Name -> ed_izmer.



А из вкладки **Data Controls** помещаем на форму компонент **TDBGrid**.
В инспекторе объектов для него устанавливаем следующие свойства:
DataSource -> ed_izmer;



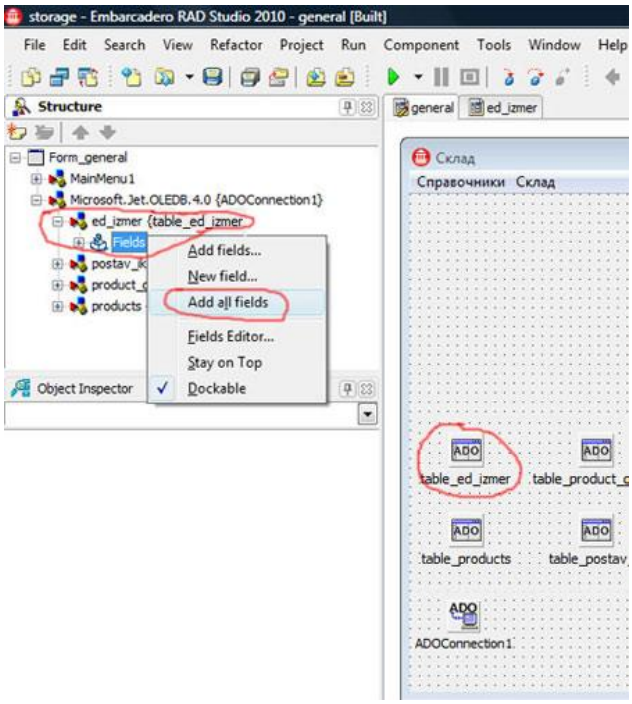
Если вы все правильно сделали то должны на форме увидеть следующее:



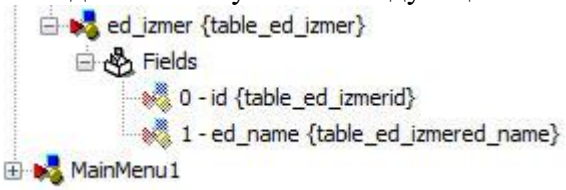
Если поля таблицы не отображаются в **DBGrid,**возможно у вас отключено (**False**) свойство **Active** главной форме в инспекторе объектов компонента **table_ed_izmer (TADOtable),** установите **Active -> True.**

Сейчас мы переименуем поля нашей таблице. Для этого перейдем на главную форму проекта.

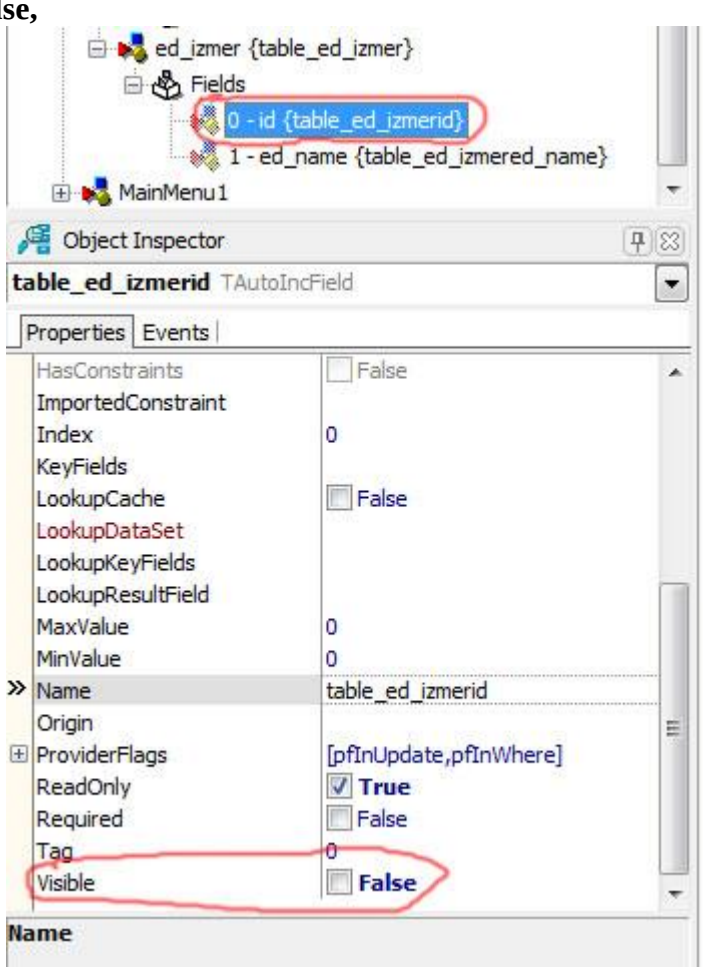
Выберем компонент **table_ed_izmer (ADOTable)** и в структуре (**Structure**) -> **ed_izmer -> Fields** -> щелкнем правой кнопкой мыши и выберем **Add all fields**.



У вас должно получиться следующее:



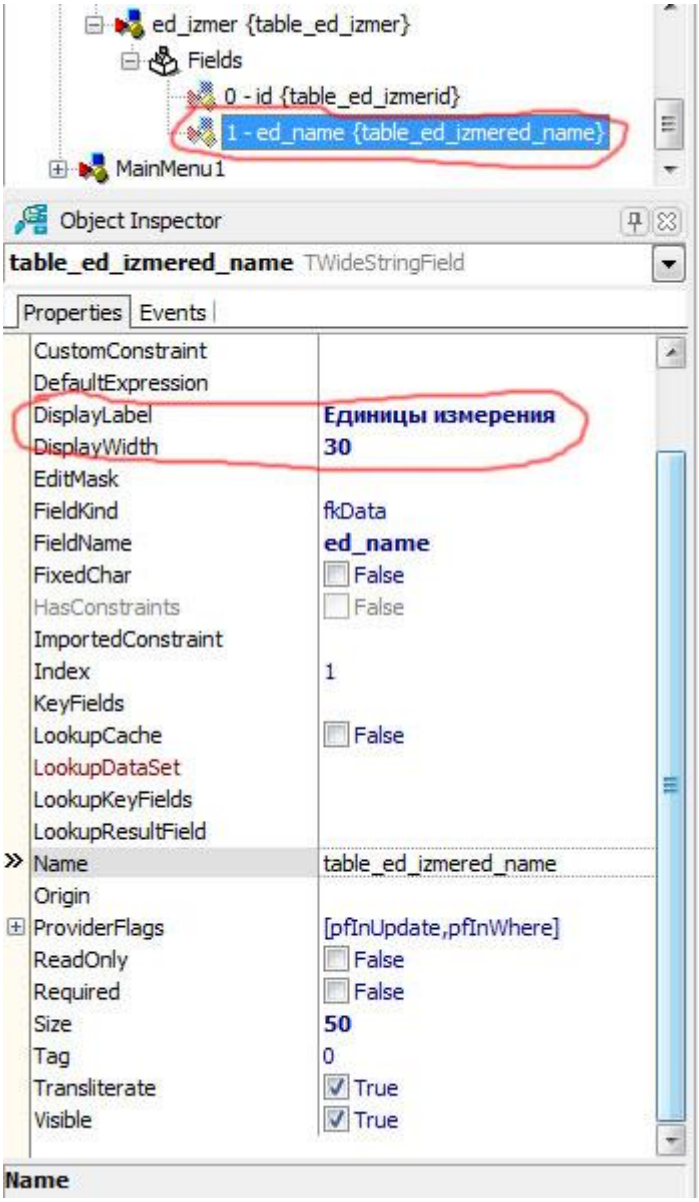
Далее выбираем **поле id** и в инспекторе объектов устанавливаем у него свойство **Visible->False,**



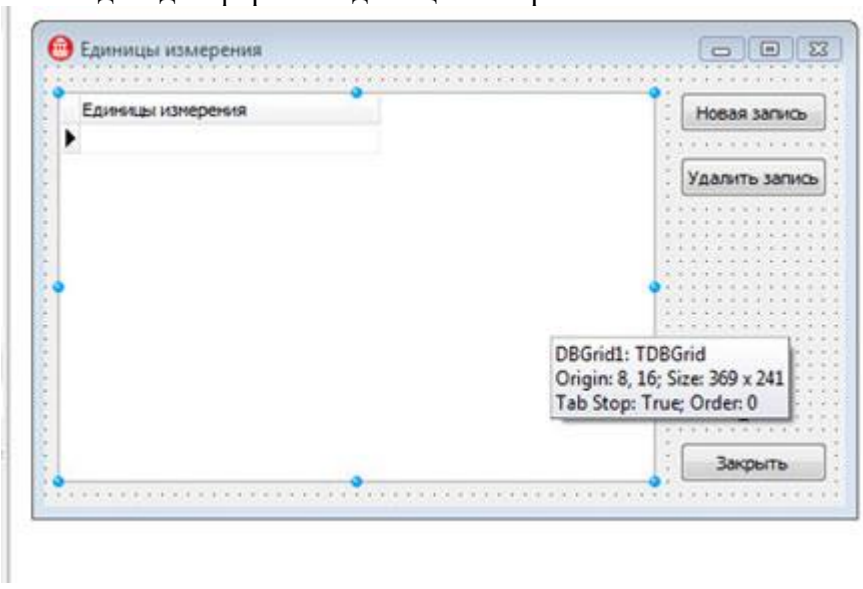
затем выбираем **поле ed_name** и в инспекторе объектов устанавливаем у него свойства

DisplayLabel -> Единица измерения;

DisplayWidth -> 30;



После этого видим для формы «Единицы измерения»:



Теперь пишем обработчики событий для кнопок.

Щелкнем дважды мышкой по кнопке **«Новая запись»** и пишем следующий код:

```
//---------------------------------------------
void __fastcall TForm_ed_izmer::Button1Click(TOb
{
Form_general->table_ed_izmer->Append();
DBGrid1->SetFocus() ;

}
//---------------------------------------------
```

Щелкнем дважды мышкой по кнопке **«Удалить запись»** и пишем следующий код:

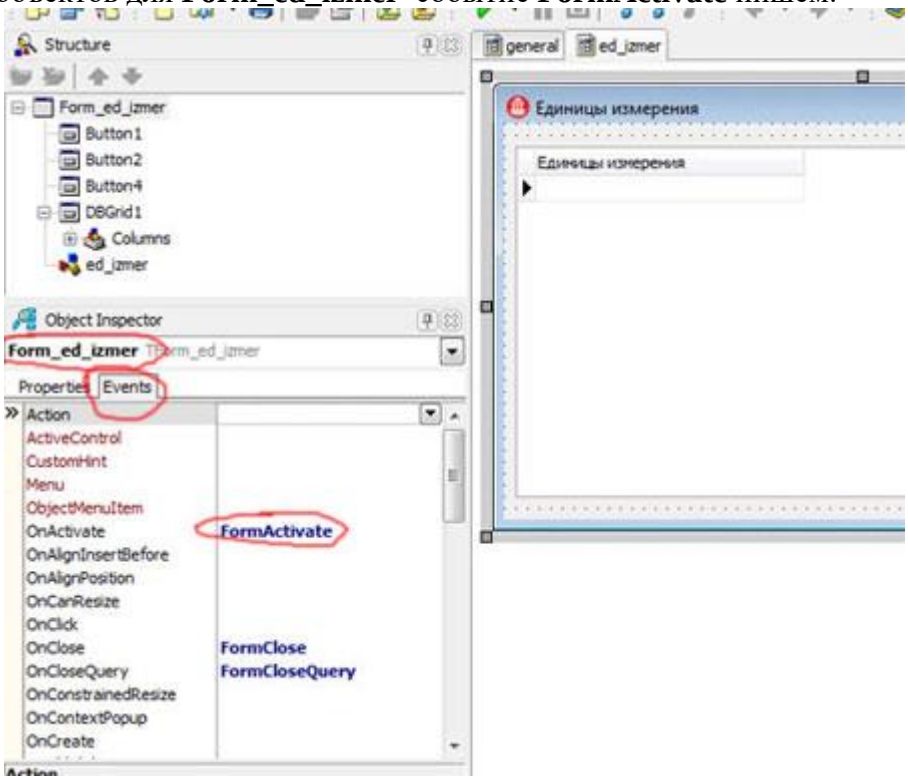
```
{
//удаление записи
//если записи отсутствуют то выводим сообщение, иначе удаляем запись
  if (DBGrid1->DataSource->DataSet->RecordCount!=0)
     { if (MessageDlg("Удаление записей может привести к нарушению работы всей базы данных. Удалить запись?",
     mtConfirmation, TMsgDlgButtons() << mbYes << mbNo,0) == mrYes)
     Form_general->table_ed_izmer->Delete();}
     else ShowMessage("Нет записей для удаления");

}
//---------------------------------------------------------------------------
```

Щелкнем дважды мышкой по кнопке **«Закрыть»** и пишем следующий код:

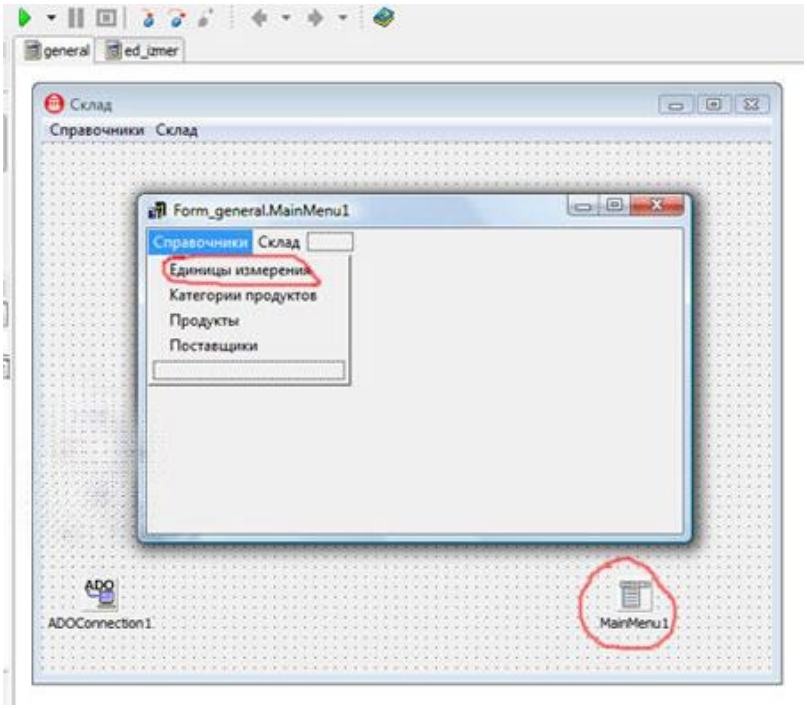
```
void __fastcall TForm_ed_izmer::Button3Click(TObject *Sender)
{
Close();
}
```

В инспекторе объектов для **Form_ed_izmer** событие **FormActivate** пишем:



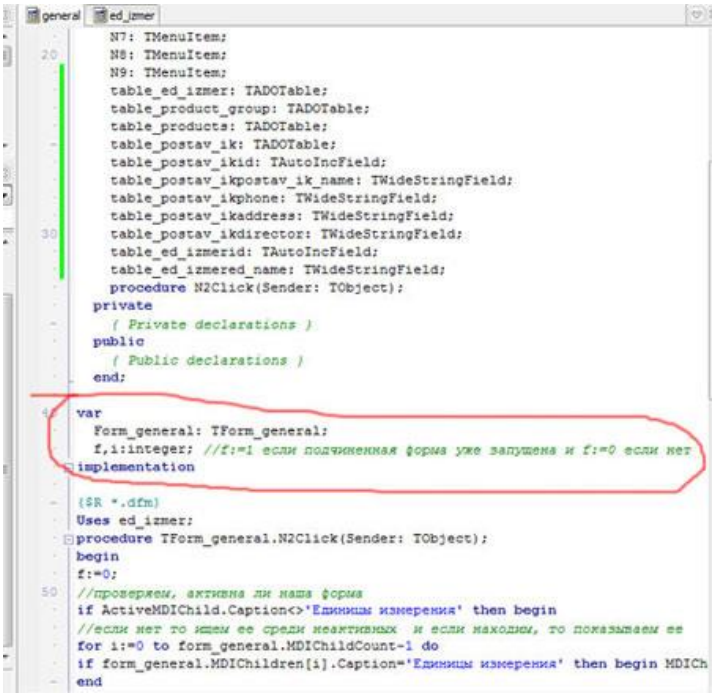

```
void __fastcall TForm_ed_izmer::FormActivate(TObject *Sender)
{
 DBGrid1->SetFocus() ;
}
```

Сейчас сделаем запуск справочника **«Единиц измерения»** из главной формы. Переходим на главную форму, щелкаем мышкой по компоненту MainMenu1, а затем по пункту Единицы измерения

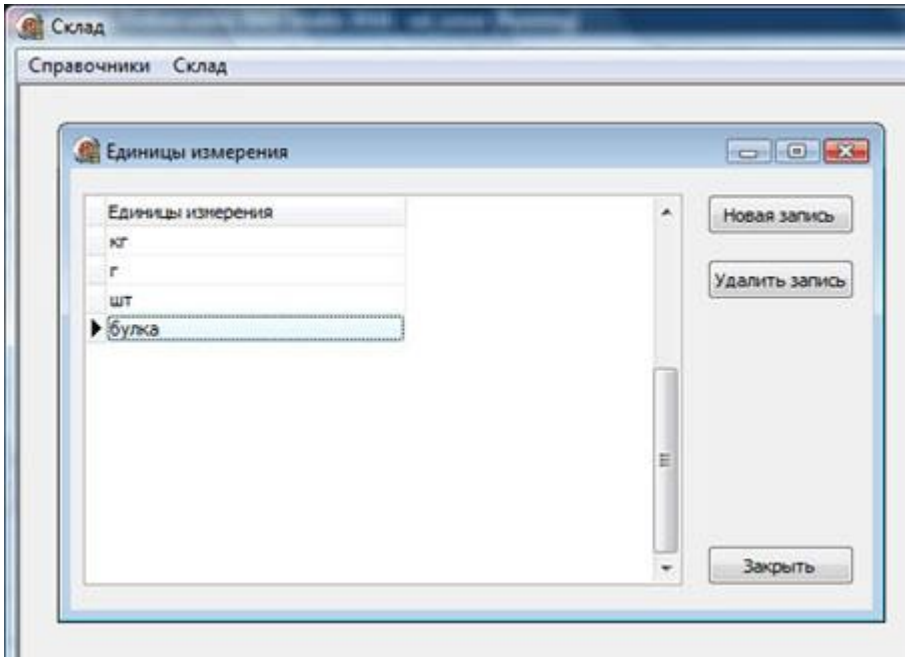


И пишем следующий обработчик события:

```
void __fastcall TForm_general::N2Click(TObject *Sender)
{
Form_ed_izmer->Show();
}
```



Вот и все готово, запускаем проект и заполняем справочник данными, например так:



Лабораторная работа 40 .

Создаем подчиненные формы для справочников программы склад

На этом уроке мы создадим форму для справочника «Категории продуктов» и «Поставщики» нашего склада.

Процесс создания данных форм аналогичен созданию формы для справочника «Единицы измерения», смотри предыдущий урок. Поэтому описывать буду достаточно кратко.

Создаем новую форму **File->New->Form**

В инспекторе объектов устанавливаем следующие свойства для новой формы

Caption -> Категории продуктов;

Name -> Form_product_group;

Сохраните модуль с именем **product_group.**

Пропишем **general** в программном модуле **product_group** как показано на рисунке.

```
#include <vcl.h>
#pragma hdrstop

#include "product_group.h"
#include "general.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm_product_group *Form_product_group;
//---------------------------------------------------------
```

А программном модуле **general** главной формы пропишем **product_group;**

Далее размещаем на форме следующие компоненты

Три кнопки **TButton** из вкладки **Standart**

В инспекторе объектов для **Button1** устанавливаем свойство **Caption -> Новая запись;**

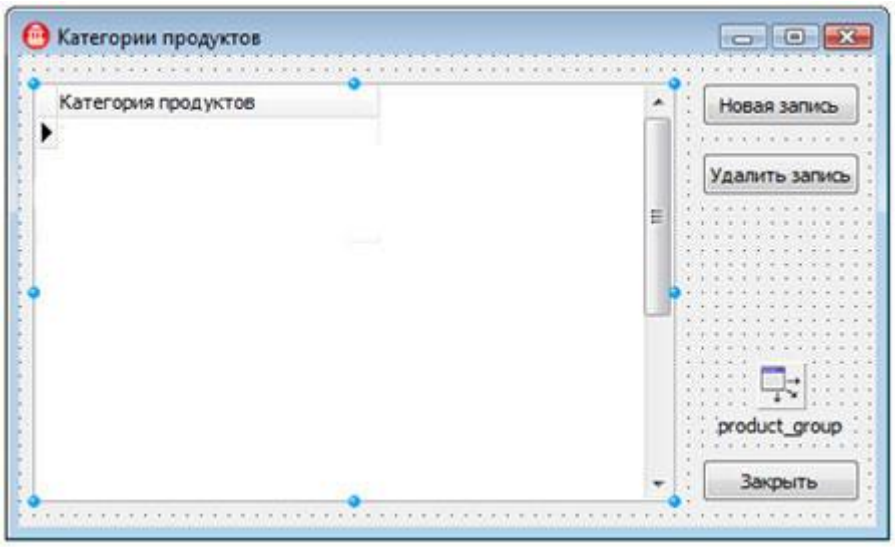
для **Button2** устанавливаем свойство **Caption -> Удалить запись;**

для **Button3** устанавливаем свойство **Caption -> Закрыть.**

Из вкладки **Data Access** помещаем на форму компонент **TDataSource.**

В инспекторе объектов для него устанавливаем следующие свойства:
DataSet -> Form_general.table_product_group;
Name -> product_group.
А из вкладки **Data Controls** помещаем на форму компонент **TDBGrid**.
В инспекторе объектов для него устанавливаем следующие свойства:
DataSource -> product_group.
Если поля таблицы не отображаются в **DBGrid,**возможно у вас отключено (**False**) свойство **Active** главной форме в инспекторе объектов компонента **table_product_group (TADOtable),** установите **Active -> True.**
Сейчас мы переименуем поля нашей таблице. Для этого перейдем на главную форму проекта.
Выберем компонент **table_product_group (ADOTable)** и в структуре (**Structure**) -> **product_group -> Fields** -> щелкнем правой кнопкой мыши и выберем **Add all fields**.
Далее выбираем **поле id** и в инспекторе объектов устанавливаем у него свойство **Visible->False,**
затем выбираем **поле product_name** и в инспекторе объектов устанавливаем у него свойства
DisplayLabel -> Категория продуктов;
DisplayWidth->30;

После этого видим для формы **«Категории продуктов»**:



Теперь пишем обработчики событий для кнопок.
Щелкнем дважды мышкой по кнопке «Новая запись» и пишем следующий код:

```
//---------------------------------------------------------------------------
void __fastcall TForm_product_group::Button1Click(TObject *Sender)
{
Form_general->table_product_group->Append();
DBGrid1->SetFocus() ;
}
//---------------------------------------------------------------------------
```

Щелкнем дважды мышкой по кнопке «Удалить запись» и пишем следующий код:

```
//---------------------------------------------------------------------------
void __fastcall TForm_product_group::Button2Click(TObject *Sender)
{

//если записи отсутствуют то выводим сообщение, иначе удаляем запись
  if (DBGrid1->DataSource->DataSet->RecordCount!=0)
     { if (MessageDlg("Удаление записей может привести к нарушению работы всей базы данных. Удалить запись?",
     mtConfirmation, TMsgDlgButtons() << mbYes << mbNo,0) == mrYes)
     Form_general->table_product_group->Delete();}
     else ShowMessage("Нет записей для удаления");
}

//---------------------------------------------------------------------------
```

Щелкнем дважды мышкой по кнопке «Закрыть» и пишем следующий код:

```
//---------------------------------------------------------------------------
void __fastcall TForm_product_group::Button3Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------------------------
```

В инспекторе объектов для **Form_product_group** событие **FormActivate** пишем:

```
//---------------------------------------------------------------------------
void __fastcall TForm_product_group::FormActivate(TObject *Sender)
{
DBGrid1->SetFocus() ;
}
//---------------------------------------------------------------------------
```

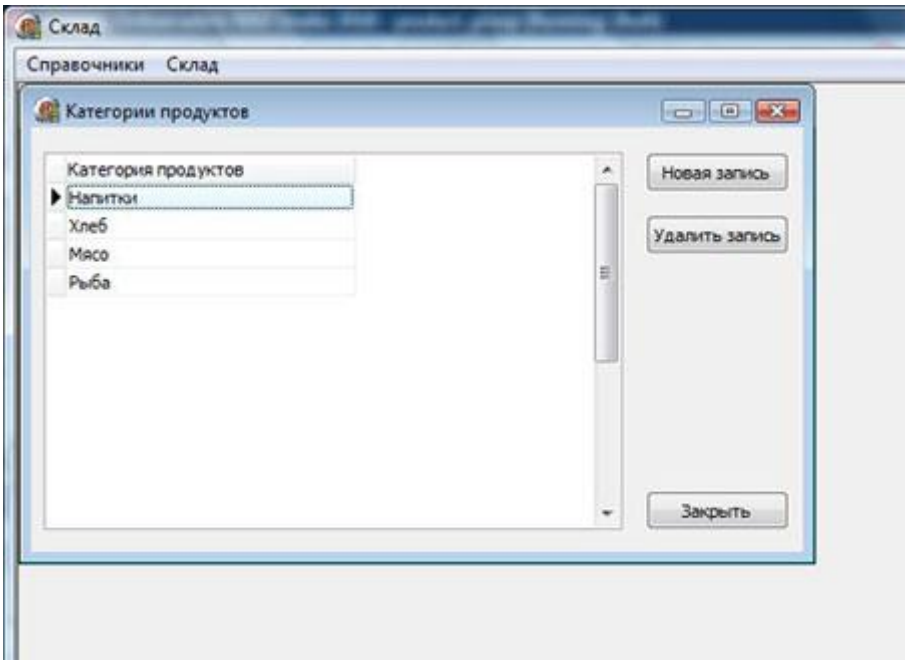
Сейчас сделаем запуск справочника «**Категории продуктов**» из главной формы.

Переходим на главную форму, щелкаем мышкой по компоненту MainMenu1, а затем по пункту **Категории продуктов**

И пишем следующий обработчик события:

```
//---------------------------------------------------------------------------
void __fastcall TForm_general::N3Click(TObject *Sender)
{
    Form_product_group->Show();
}
//---------------------------------------------------------------------------
```

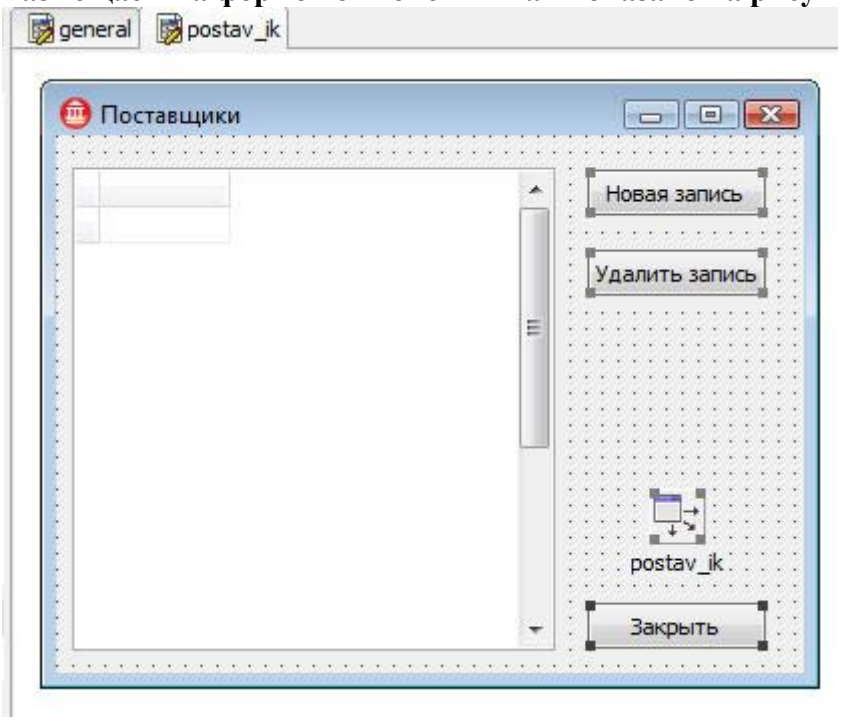
Вот и все готово, запускаем проект и заполняем справочник данными, например так:



Создаем форму для справочника «Поставщики»

Создаем новую форму **File->New->Form**

В инспекторе объектов устанавливаем следующие свойства для новой формы
Caption -> Поставщики;
Name -> Form_postav_ik;
Сохраните модуль с именем **postav_ik.**
Пропишем **general** программном модуле **unit postav_ik,**а в программном модуле **general** главной формы пропишем **postav_ik.**
Размещаем на форме компоненты как показано на рисунке:



Далее все делаем аналогично, как для предыдущего справочника. А я приведу лишь код для обработки событий.
Для кнопки **«Новая запись»**

```
//---------------------------------------------------------------------------
void __fastcall TForm_postav_ik::Button1Click(TObject *Sender)
{
Form_general->table_postav_ik->Append();
DBGrid1->SetFocus() ;
}
//---------------------------------------------------------------------------
```

Для кнопки **«Удалить запись»**

```
}
//---------------------------------------------------------------------------
void __fastcall TForm_postav_ik::Button2Click(TObject *Sender)
{
//если записи отсутствуют то выводим сообщение, иначе удаляем запись
  if (DBGrid1->DataSource->DataSet->RecordCount!=0)
     { if (MessageDlg("Удаление записей может привести к нарушению работы всей базы данных. Удалить запись?",
     mtConfirmation, TMsgDlgButtons() << mbYes << mbNo,0) == mrYes)
     Form_general->table_postav_ik->Delete();}
     else ShowMessage("Нет записей для удаления");
}
//---------------------------------------------------------------------------
```

Для кнопки **«Закрыть»**

```
//---------------------------------------------------------------------------
void __fastcall TForm_postav_ik::Button3Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------------------------
```

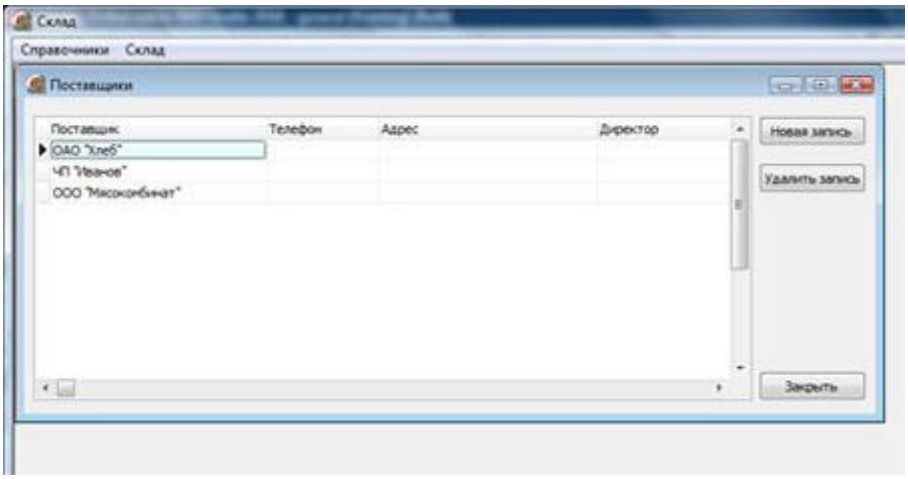
Для события **FormActivate** формы **Form_postav_ik**

```
//---------------------------------------------------------------------------
void __fastcall TForm_postav_ik::FormActivate(TObject *Sender)
{
DBGrid1->SetFocus() ;
}
//---------------------------------------------------------------------------
```

Для главной формы **Form_general** при нажатии на кнопку **Категории** продуктов выпадающего меню

```
//---------------------------------------------------------------------------
void __fastcall TForm_general::N5Click(TObject *Sender)
{
    Form_postav_ik->Show();
}
//---------------------------------------------------------------------------
```

Запускаем проект и заполняем справочник данными.



Критерии для оценки результатов учебных достижений при проведении текущей, промежуточной и итоговой аттестации по результату обучения «Разработка компонентов проектной и технической документации с использованием графических языков спецификаций»

Цифровой эквивалент	Оценка	Критерии
Оценка «Отлично»	Оценка «А»	Творческое осмысление учебного материала, использование дополнительных источников для более глубокого осмысления сущности явлений, видение когнитивной структуры материала, выявление недостающих элементов структуры, дополнение ими; выделение проблемных аспектов изучаемого материала; выполнение заданий на 95-100%; естественная мотивация в изучении предмета; креативное использование полученных знаний для решения проблем в жизненных ситуациях. Самостоятельно и творчески составляет план разработки программных продуктов, разработывает алгоритм действий. Определяет взаимосвязь и ответсвенность. Разрабатывает программный продукт. Тестирует, выявлеят и устраняет ошибки. Разрабатывает собственный проекты. Проводит самооценку проекта.
	Оценка «А-»	Полное овладение учебным материалом и его воспроизведение с собственными дополнениями и аргументами; свободное оперирование учебным материалом различной степени сложности в проблемных и креативных ситуациях; выполнение заданий творческого характера; высокий уровень самостоятельности и творческого подхода; выполнение заданий на 95%; допущение незначительных погрешностей в последовательности действий или оформлении; творческое использование полученных знаний для решения проблем в жизненных ситуациях. Самостоятельно составляет план разработки программных продуктов, разработывает алгоритм действий. Определяет взаимосвязь и ответсвенность. Разрабатывает программный продукт. Тестирует, выявлеят и устраняет ошибки. Разрабатывает собственный проекты. Проводит самооценку проекта.
Оценка «Хорошо»	Оценка «В+»	Освоение учебного материала и самостоятельное применение их в типичных, вариативных и проблемных ситуациях; владение навыками творческого применения полученных знаний; выполнение заданий на 90%, умение исправить собственные ошибки, самокритичность, планирование действий по совершенствованию навыков решения задач; использование полученных знаний для решения проблем в жизненных ситуациях Допускает незначительные погрешности в разработке проекта, которые исправляет по рекомендации педагога.
	Оценка «В»	Овладение изученым материалом, видение взаимосвязи частей изучаемого материала, его родо-видовых и причинно-следственных связей; умение применять знания в типичных, вариативных и иногда проблемных ситуациях; выполнение

		заданий на 85%, активное участие в выполнении творческого задания в группе; самокритичность и умение ставить цель по устранению своих ошибок. Сопоставляет имеющиеся данные по проекту, с целью дальнейшего их применения. Показывает достаточный уровень самостоятельности и творческого подхода при выполнении задания.
	Оценка B-	Овладение программным материалом на основе определения его когнитивной структуры (семантических блоков), умение применять знания в типичных, вариативных и иногда проблемных ситуациях; выполнение заданий на 80%, проявление способности исправить собственные ошибки при указании на них; естественная мотивация в выполнении творческих заданий; самокритичность и умение ставить цель по устранению своих ошибок. Имеет естественную мотивацию при выполнении заданий. Допускает погрешности и ошибки, которые исправляет по рекомендации педагога.
	Оценка C+	Воспроизведение учебного материала на 75%; овладение навыками выполнения задания по образцу в типичных ситуациях; затруднение выполнения задания в вариативных ситуациях; стремление самостоятельно выполнять задания, следствием которого является неполнота, непоследовательность действий, приводящая к ошибкам; стремление выполнять творческую работу в группе; отсутствуют навыки самостоятельного творческого решения задачи. Допускает погрешности и ошибки.
Оценка «Удовлетво-рительно»	Оценка C	Понимание учебного материала, воспроизведение его на 70%; проявление интереса к учебе, приложение усилий как желания достичь поверхностный результат (мотивация учения – получение положительной отметки); владение средним уровнем учебных умений (выполнение заданий на 70%), заключающимся в повторении действий сверстников без глубокого осмысления значимости для дальнейшего познавательного процесса.
	Оценка C-	Механическое освоение учебного материала на репродуктивном уровне без осмысления содержания, отрывистое его воспроизведение на основе наводящих вопросов; неумение применять полученные знания на практике, проявление стремления использовать самостоятельно алгоритм решения задачи и достижение 60% -го выполнения.
	Оценка D+	Отсутствие видения целостной структуры представленного учебного материала, его частичное воспроизведение без освоения родовидовых отношений и причинно-следственных связей; наличие существенных ошибок в ответе; использование алгоритма выполнения с чьей-либо помощью; отсутствие самостоятельных навыков выполнения заданий.
	Оценка D	Изучает готовый план мероприятияя под руководством педагога. Воспроизводит термины и понятия из плана. Воспроизводит должностные обязанности и алгоритм действийс помощью педагога. Частично выполняет алгоритм

		работ с помощью педагога. Затрудняется при выполнении заданий в стандартных и нестандартных ситуациях.
Оценка «Неудовлет-ворительно»	Оценка F	Не осваивает более половины программы модуля. Дополнительные и уточняющие вопросы педагога не приводят к коррекции ответа обучающегося. Имеет пробелы в знании основного материала, предусмотренного программой, в ответах допускает принципиальные ошибки, не выполняет отдельные задания, предусмотренные формами текущего, промежуточного и итогового контроля.

КОНТРОЛЬНО-ИЗМЕРИТЕЛЬНЫЕ МАТЕРИАЛЫ

1 Нұсқа
Вариант

1.Сұрақ
Вопрос
Что означает следующий оператор: Select * from Tovar?
A. Результирующий набор данных будет пустым.
B. В результирующий набор данных попадают все записи, которые содержат символ «*», из таблицы Tovar.
C. В результирующий набор данных попадают все записи из таблицы Tovar, значения числовых полей которой перемножаются.
D. В результирующий набор данных попадают все поля и все записи из таблицы Tovar.
E. В результирующий набор данных попадают все поля, содержащие в своем имени символ «*», из таблицы Tovar

2. Сұрақ
Вопрос
Отладка программ это:
A. алгоритмизация программирование
B. компиляция
C. компоновка
D. локализация и исправление ошибок
E. анализ влияния ошибок на приложение

3. Сұрақ
Вопрос
Цикл с постусловием?
A. 1)do while
B. 2)repeat
C. 3)until
D. 4)while
E. 5)for

4. Сұрақ
Вопрос
Если не будет указан базовый тип, то какай тип будет подразумеваться по умолчанию
A. String
B. Short
C. IntShort
D. Char
E. Long

5. Сұрақ
Вопрос
Язык программирования С++ разработал
A. Дональд Кнут
B. Никлаус Вирт
C. Бьерн Страуструп
D. Кен Томпсон
E. Блез Паскал

6. Сұрақ
Вопрос
Каково будет значение переменной k после выполнения следующего оператора k = ++k; если до его выполнения k равнялось 6?
A. 6
B. 7
C. 14
D. 8
E. 16

7. Сұрақ
Вопрос
Битовая операция инверсии битов (побитовое НЕ) обозначается
A. \~
B. >>
C. <<
D. @
E. ~

8. Сұрақ
Вопрос
Цикл с предусловием?
A. do while
B. repeat
C. while
D. for
E. until

9. Сұрақ
Вопрос
Логическое «не равно» обозначается:
A. <>
B. !
C. ==
D. |

E. =!

10. Сұрақ
Вопрос
Каков результат работы следующего фрагмента кода?
int x = 0;
switch(x)
{case 1: cout << "Один";
case 0: cout << "Нуль";
case 2: cout << "Привет мир";}
A. 1
B. Один
C. НульПривет мир
D. Нуль
E. Привет мир

11. Сұрақ
Вопрос
C++Builder.Какая опция добавляет файл к проекту в пункте меню Project?
A. Editor Options
B. Add To Project
C. Import Type Library
D. Environment Options
E. Remove From Project

12 Сұрақ
Вопрос
C++Builder.Какая опция отображает в окне кода оператор, на котором было прервано выполнение программы в пункте меню Run?
A. Program Pause
B. Run Until Return
C. Show Execution Point
D. Program Reset
E. Inspect

13. Сұрақ
Вопрос
Строковым литералом будет
A. “sq”
B. ‘s’
C. ‘qsqs’
D. <qsqs>
E. %q

14. Сұрақ
Вопрос
Оператор break в Си++ применяется для:
A. преобразования переменной к целому типу
B. прибавления единицы к коду символа
C. убавление единицы к коду символа
D. преобразования переменной к вещественному типу
E. прерывает выполнение операторов while, do, for и switch

15. Сұрақ
Вопрос
Каков результат работы следующего фрагмента кода?
int x = 0;
switch(x)
{case 1: cout << "Один "; break;
case 0: cout << "Нуль ";break;
case 2: cout << "Привет мир ";}
A. Привет мир
B. Нуль Привет мир
C. Нуль
D. 1
E. Один

16. Сұрақ
Вопрос
Какая из следующих записей - правильный комментарий в C++?
A. /* комментарий */
B. */ Комментарии */
C. ** Комментарий **
D. {комментарий}
E. # комментарий#

17. Сұрақ
Вопрос
Название C++ предложил
A. Бьерн Страуструп
B. Блез Паскал
C. Кэн Томпсон
D. Дональд Кнут
E. Рик Масситти

18. Сұрақ
Вопрос
Что такое IDE?
A. интегрированная среда разработки
B. графический редактор
C. шаблон данных

D. формат ввода данных
E. унифицированный язык разработки

19. Сұрақ
Вопрос
Чему будет равна переменная a, после выполнения этого кода?
int a;
for(a = 1; a < 11; a++) {}
cout<<a;
A. 9
B. 11
C. 6
D. 1
E. 10

20. Сұрақ
Вопрос
Самый важный критерий качества программы:
A. эффективность;
B. работоспособность;
C. простота эксплуатации.
D. быстродействие;
E. надежность;

21. Сұрақ
Вопрос
Простые типы данных в C++.
A. целые – bool, вещественные – float или double, символьные – string
B. целые – int, вещественные – float или real, символьные – string
C. целые – int, вещественные – float или real, символьные – char
D. целые – float, вещественные – int или double, символьные – string
E. целые – int, вещественные – float или double, символьные – char

22. Сұрақ
Вопрос
Автономное тестирование это:
A. составление блок-схем;
B. пошаговая проверка выполнения программы.
C. выбор инструментов для реализации проверочных испытаний
D. инструментальное средство отладки;
E. тестирование отдельных частей программы;

23. Сұрақ
Вопрос
Какой из ниже перечисленных операторов является циклом в C++?
A. repeat
B. while
C. else
D. until
E. if

24. Сұрақ
Вопрос
Какой из ниже перечисленных операторов является циклом в C++?
A. until
B. else
C. repeat
D. for
E. if

25. Сұрақ
Вопрос
Для создания теста с произвольным числом правильных вариантов ответа используется элемент управления:
A. ToolBar
B. RadioBox
C. CheckBox
D. CoolBar
E. RadioButton

26. Сұрақ
Вопрос
Компонент DBNavigator - ...
A. компонент, используется как связка данных из таблиц, с отображающими и управляющими компонентами (закладка Data Access);
B. связывается с конкретной таблицей БД (закладка ADO);
C. используется для подключения к БД (закладка палитры ADO, в некоторых версиях dbGO);
D. таблица, позволяющая вывести содержимое таблицы БД на пользовательскую форму (закладка Data Controls);

E. кнопочная панель, способная управлять данными в привязанной к ней таблице (закладка Data Controls).

27. Сұрақ
Вопрос
Процедура поиска ошибки, когда известно, что она есть это:
A. трансляция.
B. транзакция;
C. отладка;
D. компоновка;
E. тестирование;

28. Сұрақ
Вопрос
В языке Си++ программа начинает выполняться с функции:
A. Do
B. Start
C. Main
D. Program
E. Go

29. Сұрақ
Вопрос
Для того, чтобы выводить на экран тип животных и их клички (например, пес Барбос, кот Барсик), эффективно использовать:
A. метод с одним параметром числового типа
B. метод с двумя параметрами строкового и числового типов
C. метод без параметров
D. метод с двумя параметрами строкового типа
E. метод с двумя параметрами числового типа

30. Сұрақ
Вопрос
C++Builder.Какая опция отладочном режиме выполняет программу и останавливается перед выполнением кода в строке с текстовым курсором в пункте меню Run?
A. Run To Cursor
B. Install MTS Objects
C. Trace To Next Source Line
D. Trace Into
E. Step Over

2 Нұсқа
Вариант

1. Сұрақ
Вопрос
Укажите пример класса
A. растения
B. кот
C. журнал «Компас»
D. кот Пушок
E. книга "Война и мир"

2. Сұрақ
Вопрос
Основные принципы объектно-ориентированного программирования:
A. полиморфизм, инкапсуляция, наследование;
B. свойства, события, методы;
C. визуальные, не визуальные компоненты и запросы.
D. объект, класс, полиморфизм
E. предки, родители, потомки;

3. Сұрақ
Вопрос
Если при объявлении поля класса пропущен модификатор доступа, то компьютер сделает это поле
A. закрытым (private)
B. защищенным (protected)
C. скрытый (hidden)
D. открытым (public)
E. внутренний (inside)

4. Сұрақ
Вопрос
A. В идентификаторах можно использовать
B. заглавные и строчные латинские буквы, символ @
C. заглавные и строчные латинские буквы, знак подчеркивания, символ #
D. заглавные и строчные латинские буквы, цифры, символ @

E. заглавные и строчные латинские буквы, цифры и знак подчеркиваниязаглавные и строчные латинские буквы, символы @, #, $

5. Сұрақ
Вопрос
Символьным литералом будет:
A. ‘q’
B. «sq»
C. “q”
D. «q»
E. «sa»

6. Сұрақ
Вопрос
Элемент управления RadioButton используется для:
A. переключения между различными элементами на форме
B. обеспечения выбора пользователем ровно одного из нескольких вариантов
C. создания кнопки круглой формы
D. создания кнопки квадратной формы
E. для создания «горячих клавиш»

7. Сұрақ
Вопрос
Для чего предназначен деструктор?
A. для расширения метода
B. для инициализации объекта
C. для инициализации метода
D. для уничтожения объекта
E. для спецификации метода

8. Сұрақ
Вопрос
Каково будет значение переменной k после выполнения следующего оператора
k = --k; если до его выполнения k равнялось 6?
A. 16
B. 14
C. 6
D. 5
E. 8

9. Сұрақ
Вопрос
Какой метод используется для создания таблицы базы данных?
A. CreateQuery
B. Clear
C. OpenTable
D. Add
E. CreateTable

10. Сұрақ
Вопрос
Какое свойство компонента DBEdit содержит его имя?
A. Text
B. Alignment
C. Align
D. DataField
E. Name

11. Сұрақ
Вопрос
Исходя из данного кода какое высказывание верно?
```
int main()
{
int a,b,c,d;
a=1;
b=2;
c=a+b+p;
cout << p;
}
```
A. приложение выводит на экран переменную c=3
B. код не верен, потому что переменным c и d не присвоены значения
C. приложение выводит на экран переменную p=21
D. код верен, потому что по умолчанию все переменные имеют целочисленный тип
E. код не верен, потому что переменная p не объявлена

12. Сұрақ
Вопрос
Что выведет следующая программа ?
```
#include <iostream>
using namespace std;
int main() {
int l_i ;
for( l_i = 0; l_i < 9; l_i++)
```

cout << l_i +1;
return 0;}
A. программа не будет построена из-за ошибок
B. цифры от 8 до 1
C. цифры от 2 до 9
D. цифры от 1 до 9
E. цифры от 0 до 8

13. Сұрақ
Вопрос
Какая процедура устанавливает указатель в таблице на последнюю запись?
A. MoveBy
B. First
C. Prior
D. Next
E. Last

14. Сұрақ
Вопрос
Какой результат следующего выражения ?
int *a; int b[2]; a = b;
b[1] = 7; b[0] = 10; *a++; cout << *a;
A. 7
B. 11
C. 13
D. 10
E. 8

15. Сұрақ
Вопрос
Какой класс используется для файлового ввода/вывода?
A. cout
B. fstream
C. strstream
D. strout
E. iostream

16 Сұрақ
Вопрос
Свойства – это
A. синтаксическая надстройка, позволяющая осуществлять в форме вызов процедуры
B. переменные, принадлежащие классу или экземпляру класса
C. процедуры и функции класса
D. это способ доступа к внутреннему состоянию объекта класса, имитирующий обращение к закрытому полю класса
E. синтаксическая надстройка, поддерживаемая компилятором и средой, которая позволяет вызывать методы других объектов

17. Сұрақ
Вопрос
C++Builder.Какая опция показывает окно с кодом проекта в пункте меню Project?
A. Import Type Library
B. Add To Project
C. View Source
D. Remove From Project
E. Add To Repository

18. Сұрақ
Вопрос
Чему будет равен результат вычисления:
int i; int k = 2; int m = 12; i = (m)/(m/k - 5);
cout<<i;
A. 12
B. 10
C. 0
D. -12
E. ошибка во время исполнения

19. Сұрақ
Вопрос
Какой результат будет у следующего выражения?
#define CIRC(x) (3 * (x) * (x))
#include <iostream>
int main()
{
int a = 1, b = 2;
std::cout << CIRC(a + b);
}

A. 27
B. 9
C. 16
D. 28
E. 25

20. Сұрақ
Вопрос
Какое свойство компонента DBEdit осуществляет связь с базой данных?
A. DataField
B. Align
C. Alignment
D. DataSource
E. Color

21. Сұрақ
Вопрос
Назначение отладки:
A. поиск причин существующих ошибок;
B. поиск возможных ошибок;
C. анализ влияния ошибок на приложение
D. разработка алгоритма.
E. составление спецификаций;

22. Сұрақ
Вопрос
Какое из приведенных имен является недопустимым в Си++?
A. xb___@
B. OOP
C. r13
D. L_1
E. x03488erJJJ___

23. Сұрақ
Вопрос
Каждый оператор заканчивается....
A. управляющей последовательностью
B. Return
C. точкой
D. Endl
E. точкой с запятой

24. Сұрақ
Вопрос
Что обозначает данная критическая ошибка EConvertError?
A. Ложное выражение, проверяемое процедурой AsserOT
B. «Молчаливое» исключение, предназначенное для намеренного прерывания вычислений и быстрого выхода! глубоко вложенных процедур и функций
C. Ошибка работы с базами данных
D. Попытка доступа к адреса вне памяти, распределенной приложению
E. Ошибка преобразования строк или объектов (в частности, в функциях StrToInt, StrToFloat, StrToDatel

25. Сұрақ
Вопрос
Динамика конкретного класса может быть выражена с помощью …
A. Диаграммы Вариантов использования
B. Диаграммы последовательности
C. Диаграммы классов
D. Кооперативной диаграммы
E. Диаграммы состояний

26. Сұрақ
Вопрос
Что будет выведено в результате
double x = 12.4;
cout << setw(E. << x << setw(C.
<< setfill('*')<< "" << endl;
A. 12.4***
B. 124e2**
C. 12.4
D. 12.40***
E. 12.4 * *

27. Сұрақ
Вопрос
C++Builder.Какой опцией вызывается диалоговое окно настройки параметров среды?
A. Help | Options
B. Tools | Options
C. File | Options
D. Format | Options
E. View | Options

28. Сұрақ
Вопрос
В результате выполнения программы
int x, y;
x = 1;
y = 1;
if (x) {y = 0; } переменная x получит значение

A. -2
B. 2
C. 1
D. 0
E. -1

29. Сұрақ
Вопрос
Вид ошибки с неправильным использованием служебных слов (операторов):
A. символьная
B. семантическая;
C. математическая
D. синтаксическая;
E. логическая;

30.Сұрақ
Вопрос
Какая процедура устанавливает указатель в таблице на предыдущую запись?
A. MoveBy
B. First
C. Next
D. Prior
E. Last

3 Нұсқа
Вариант

1. Сұрақ
Вопрос
Почему данный код не верен?
```
int main()
{
int a,c,p;
a=1;
b=2;
c=a+b+p;
cout << p;
}
```
A. код неверен, потому что переменная b не объявлена
B. код неверен, потому что по умолчанию все переменные имеют целочисленный тип
C. код неверен, потому что переменной c не присвоены значения
D. код неверен, потому что переменная a не объявлена
E. код верен

2. Сұрақ
Вопрос
C++Builder.К какой закладке относятся SpeedSettings, Custom colors & setting, Default colors & settings?
A. Palette
B. Object Inspector
C. Library
D. Designer
E. Preferences

3. Сұрақ
Вопрос
C++Builder.К какой закладке относятся BPL output directory, Browsing Path?
A. Designer
B. Preferences
C. Palette
D. Library
E. Object Inspector

4. Сұрақ
Вопрос
C++Builder.Какая закладка определяет т. н. переменные среды разработки, в том числе тип ОС, положение основных файлов ОС и т. п.
A. Object Inspector
B. Environment variables
C. Library
D. Explorer
E. Palette

5. Сұрақ
Вопрос
C++Builder.Какая закладка может изменять порядок следования компонентов и страниц, переименовывать страницы, добавлять к ним новые компоненты, удалять существующие и т. д.?
A. Library
B. Palette
C. Preferences
D. Designer
E. Object Inspector

6. Сұрақ
Вопрос
C++Builder.К какой закладке относятся Brief Cursor Shapes, Create Backup File, Preserve Line Ends?
A. Display
B. Environment variables
C. Palette
D. Color
E. General

7. Сұрақ
Вопрос
Событие – это...
A. переменные, принадлежащие классу или экземпляру класса
B. это сообщение, которое возникает в различных точках исполняемого кода при выполнении определённых условий.
C. запись о классе
D. процедуры и функции класса
E. программная/синтаксическая структура, определяющая отношение между объектами, которые разделяют определённое поведенческое множество и не связаны никак иначе.

8. Сұрақ
Вопрос
Какой из перечисленных типов данных не является типом данных в C++?
A. double
B. bool
C. float
D. int
E. real

9. Сұрақ
Вопрос
C++Builder.Какая закладка позволяет выбрать цвета для отдельных синтаксических элементов?
A. Environment variables
B. Display
C. Color
D. General
E. Palette

10. Сұрақ
Вопрос
C++Builder.Какая опция прекращает прогон программы и восстанавливает режим конструирования программы в пункте меню Run?
A. Run Until Return
B. Program Pause
C. Program Reset
D. Show Execution Point
E. Inspect

11. Сұрақ
Вопрос
Операция битового «или» обозначается:
A. &
B. –
C. |
D. $
E. \~

12. Сұрақ
Вопрос
C++Builder.Какая закладка определяет используемые интеллектуальные возможности кодового редактора?
A. Key Mappings
B. Color
C. General
D. Code insight
E. Palette

13. Сұрақ
Вопрос
Один из необязательных этапов жизненного цикла программы:
A. проектирование;
B. оптимизация;
C. тестирование;
D. анализ требований
E. программирование;

14. Сұрақ
Вопрос
Наиболее важный критерий качества:
A. быстродействие;
B. надежность;
C. удобный интерфейс;
D. удобство в эксплуатации;
E. эффективность.

15. Сұрақ
Вопрос
Поля – это....
A. синтаксическая надстройка, поддерживаемая компилятором и средой, которая позволяет вызывать методы других объектов
B. процедуры и функции класса
C. запись о классе
D. синтаксическая надстройка, позволяющая осуществлять в форме вызов функции
E. переменные, принадлежащие классу или экземпляру класса

16. Сұрақ
Вопрос
Инкапсуляция это:
A. определение новых структур данных;
B. объединение переменных, процедур и функций в одно целое;
C. разделение переменных, процедур и функций;
D. определение новых типов данных;
E. применение стандартных процедур и функций.

17. Сұрақ
Вопрос
Укажите функцию, используемую в команде SQL для расчета количества значений?
A. AVG()
B. TRIM()
C. SUM()
D. COUNT()
E. MAX()

18. Сұрақ
Вопрос
В языке Си++ литерал - это:
A. процедура
B. строка
C. буква
D. изменяемый объект языка
E. переменная зарезервированного типа

19. Сұрақ
Вопрос
Укажите функцию, используемую в команде SQL для расчета среднего значения?
A. SUM()
B. MAX()
C. TRIM()
D. COUNT()
E. AVG()

20. Сұрақ
Вопрос
Какое свойство компонента DBEdit позволяет делать его невидимым или видимым?
A. Alignment
B. DataSource
C. Visible
D. DataField
E. Align

21. Сұрақ
Вопрос
Вид ошибки с неправильным написанием служебных слов (операторов):
A. синтаксическая;
B. символьная
C. математическая
D. логическая;
E. семантическая;

22. Сұрақ
Вопрос
Какая функция используется для определения конца файла используется
A. bof()
B. finish()
C. last ()
D. end()
E. eof()

23 Сұрақ
Вопрос
.C++Builder.Какая опция вызывает окно настройки параметров редактора C++Builder в пункте меню Tools Tools-> Options?
A. Editor Options
B. Import Type Library
C. Remove From Project

D. Add To Project
E. Environment Options

24. Сұрақ
Вопрос
C++Builder.Какая опция удаляет файл из проекта в пункте меню Project?
A. Import Type Library
B. Editor Options
C. Remove From Project
D. Environment Options
E. Add To Project

25. Сұрақ
Вопрос
Какая из операций не относится к операции сравнения?
A. =
B. ==
C. <
D. >
E. <>

26. Сұрақ
Вопрос
Что выполняет интерпритатор?
A. Выполняет перевод
B. Выполняет покомандную обработку текста программы
C. Выполняет текст программы
D. Выполняет функцию обработки текста программы
E. Выполняет покомандное объединение текста программы

27. Сұрақ
Вопрос
Трассировка это:
A. выбор инструментов для реализации проверочных испытаний
B. проверка пошагового выполнения программы;
C. отладка модуля;
D. тестирование исходного кода;
E. составление блок-схемы алгоритма.

28.C++Builder.Какая опция помещает проект в репозиторий в пункте меню Project?
A. View Source
B. Remove From Project
C. Import Type Library
D. Add To Project
E. Add To Repository

29. Сұрақ
Вопрос
Компонент DataSource - ...
A. связывается с конкретной таблицей БД (закладка ADO);
B. таблица, позволяющая вывести содержимое таблицы БД на пользовательскую форму (закладка Data Controls);
C. кнопочная панель, способная управлять данными в привязанной к ней таблице (закладка Data Controls).
D. компонент, используется как связка данных из таблиц, с отображающими и управляющими компонентами (закладка Data Access);
E. используется для подключения к БД (закладка палитры ADO, в некоторых версиях dbGO);

30. Сұрақ
Вопрос
Область распространения ЕСПД
A. Распространяется на чертежи
B. Распространяются на документы
C. Распространяются на компьютерное оборудование
D. Распространяется на схемы
E. Распространяется на программы

4 Нұсқа
Вариант

1. Сұрақ
Вопрос
Процесс упорядоченного размещения элементов в массиве называется_____
A. Сортировка
B. Проверка
C. Поиск
D. Изменение
E. Фильтрация

2. Сұрақ

Вопрос
C++Builder.На какой странице собраны переключатели, управляющие параметрами процесса компиляции?
A. Application
B. Default
C. Unit
D. Compiler
E. Form

3. Сұрақ
Вопрос
C++Builder.Какая опция позволяет присоединиться в режиме отладки к одному из уже запущенных процессов на другой сетевой машине в пункте меню Run?
A. Unregister ActiveX Server
B. Parameters
C. Run
D. Attach to Process
E. Register ActiveX Server

4. Сұрақ
Вопрос
Каково будет значение переменной k после выполнения следующего оператора
k = --k+2; если до его выполнения k равнялось 6?
A. 5
B. 16
C. 7
D. 8
E. 6

5. Сұрақ
Вопрос
C++Builder.Какая опция в отладочном режиме выполняет текущую строку кода и не прослеживает работу вызываемых подпрограмм в пункте меню Run?
A. Step Over
B. Trace Into
C. Install MTS Objects
D. Run To Cursor
E. Trace To Next Source Line

6. Сұрақ
Вопрос
C++Builder.Какая опция программа выполняется до ближайшего от текущего положения курсора исполняемого оператора в пункте меню Run?
A. Step Over
B. Trace Into
C. Run To Cursor
D. Install MTS Objects
E. Trace To Next Source Line

7. Сұрақ
Вопрос
C++Builder.Какая опция в отладочном режиме выполняет текущую подпрограмму и останавливается в пункте меню Run?
A. Program Reset
B. Run Until Return
C. Show Execution Point
D. Inspect
E. Program Pause

8. Сұрақ
Вопрос
Число, используемое для обращения к отдельному элементу массива называется
A. Тип
B. Индекс
C. Код
D. Значение
E. Запись

9. Сұрақ
Вопрос
C++Builder.Какая закладка служит для управления свойствами редактора библиотеки типов (используется при разработке многозвенных приложений баз данных)?
A. Type Library
B. Library
C. Palette
D. Object Inspector
E. Explorer

10. Сұрақ
Вопрос

C++Builder. Какая опция приостанавливает прогон отлаживаемой программы в пункте меню Run?
A. Run Until Return
B. Inspect
C. Program Pause
D. Show Execution Point
E. Program Reset

11. Сұрақ
Вопрос
C++Builder. Какая опция открывает окно проверки текущего значения в пункте меню Run?
A. Program Pause
B. Run Until Return
C. Program Reset
D. Show Execution Point
E. Inspect

12. Сұрақ
Вопрос
Элементы массива связаны тем, что они имеют один и тот же .....
A. Индекс
B. Значение
C. Значимость
D. Позицию
E. Тип

13. Сұрақ
Вопрос
К каким числам применима операция %?
A. к вещественным
B. не применяется к числам
C. к дробным
D. и к целым и к вещественным
E. к целым

14. Сұрақ
Вопрос
C++Builder.Какая опция вызывает окно настройки параметров среды C++Builder и ее инструментов в пункте меню Tools-> Options?
A. Environment Options
B. Remove From Project
C. Import Type Library
D. Add To Project
E. Editor Options

15. Сұрақ
Вопрос
Наличие комментариев позволяет:
A. улучшить читабельность программы;
B. повысить надежность программы.
C. улучшить эксплуатацию программы;
D. увеличить быстродействие.
E. применять сложные структуры;

16. Сұрақ
Вопрос
Ошибки при написании программы бывают:
A. лексические;
B. орфографические;
C. морфологические.
D. синтаксические;
E. фонетические;

17. Сұрақ
Вопрос
Процесс определения значения ключа, содержащегося в массиве, называется
A. Изменение
B. Проверка
C. Фильтрация
D. Поиск
E. Сортировка

18. Сұрақ
Вопрос
Что такое идентификаторы в языке Си++ ?
A. это последовательность знаков, начинающаяся со знака $
B. это последовательность знаков, начинающаяся с @
C. это последовательность знаков, цифр и символов подчеркивания, которая начинается с буквы или символа подчеркивания.
D. это последовательность знаков, начинающаяся со знака #
E. это последовательность знаков, начинающаяся со знака @ #

19. Сұрақ
Вопрос
Компонент ADOTable - ...

A. связывается с конкретной таблицей БД (закладка ADO);
B. используется для подключения к БД (закладка палитры ADO, в некоторых версиях dbGO);
C. кнопочная панель, способная управлять данными в привязанной к ней таблице (закладка Data Controls).
D. компонент, используется как связка данных из таблиц, с отображающими и управляющими компонентами (закладка Data Access);
E. таблица, позволяющая вывести содержимое таблицы БД на пользовательскую форму (закладка Data Controls);

20. Сұрақ
Вопрос
C++Builder.Какая опция позволяет задать командную строку запуска программы в пункте меню Run?
A. Run
B. Parameters
C. Attach to Process
D. Register ActiveX Server
E. Unregister ActiveX Server

21. Сұрақ
Вопрос
Функция fseek( ) может установить файловый указатель места
A. только в конец файла;
B. в любое место файла
C. только в начало файла
D. функция предназначена для записи из текстового файла
E. функция предназначена для чтения из текстового файла

22. Сұрақ
Вопрос
Тернарное выражение - это:
A. компактный способ записи оператора switch
B. компактный способ записи оператора WHILE/DO
C. выбор одного из нескольких вариантов
D. компактный способ записи оператора case
E. компактный способ записи оператора IF/ELSE

23. Сұрақ
Вопрос
Что является выражениями в языке Си++ ?
A. это переменные объединенные символьными знаками
B. вспомогательные модули
C. это функции объединенные строковыми знаками
D. это константы объединенные знаками операций
E. это упорядоченный набор операций над константами, переменными и функциями

24. Сұрақ
Вопрос
Что означает запись while (false);?
A. бесконечный цикл
B. цикл выполнится один раз
C. ошибка компиляции
D. аварийный выход из программы
E. цикл, который не выполняется ни разу

25. Сұрақ
Вопрос
Какое свойство компонента DBImage осуществляет связь с определенным полем таблицы?
A. DataField
B. Alignment
C. Align
D. DataSource
E. Color

26. Сұрақ
Вопрос
Оператор_____ используется для принятия решений
A. While
B. If
C. Break
D. Cout
E. For

27. Сұрақ
Вопрос
Программа, переводящая входную программу на исходном языке в эквивалентную ей выходную программу на результирующем языке, называется:
A. интерпретатор
B. компилятор
C. интегратор
D. транслятор
E. сканер

28. Сұрақ
Вопрос
Какими знаками заканчивается большинство строк кода в Си++?
A. ,(запятая)
B. . (точка)
C. :(двоеточие)
D. ;(точка с запятой)
E. – (тире)

29. Сұрақ
Вопрос
Какое свойство компонента Query содержит SQL - запрос?
A. DatabaseName
B. SQL
C. Filtered
D. Name
E. DataSource

30. Сұрақ
Вопрос

Процедура открытия файла в языке С++ выполняется функцией
A. openfile()
B. fopen()
C. fpointer( );
D. fseek( )
E. assign()

5 Нұсқа
Вариант

1. Сұрақ
Вопрос
Процесс исполнения программы с целью обнаружения ошибок:
A. тестирование;
B. отладка
C. кодирование;
D. сопровождение;
E. проектирование.

2. Сұрақ
Вопрос
Для чего предназначен конструктор?
A. для инициализации объекта
B. для спецификации метода
C. для инициализации метода
D. для расширения метода
E. для уничтожения объекта

3. Сұрақ
Вопрос
Один из методов автоматизации программирования:
A. структурное программирование;
B. объектно-ориентированное программирование.
C. модульное программирование;
D. процедурное программирование
E. визуальное программирование;

4. Сұрақ
Вопрос
Битовая операция исключающего ИЛИ обозначается:
A. &&
B. ~
C. ||
D. \~
E. ^

5 Сұрақ
Вопрос
Каков будет результат выполнения операторов:
int i,j,s;
i=j=2; /* i и j получают значение 2 */
s=(i++)+(++j);
A. i = 2, j = 2, s = 5
B. i= 3, j = 3, s = 6
C. i= 3,j = 2, s= 5
D. i = 2, j = 3, s = 5
E. i = 3, j = 3, s = 5

6. Сұрақ
Вопрос
Из чего состоит оператор объявления имени?
A. из названия типа и объявляемого имени
B. из названия типа
C. из набора цифр
D. из объявляемого имени
E. из названия имени и значения

7. Сұрақ
Вопрос
До каких пор будут выполняться операторы в теле цикла while (x < 100)?
A. Пока x больше 100
B. Пока x равен 100
C. Пока x не будет равен 101
D. Пока x строго меньше 100
E. Пока x меньше или равен 100

8. Сұрақ
Вопрос
Выберите из списка истинные утверждения, касающиеся диаграммы последовательностей
Выберите один из 5 вариантов ответа:
A. диаграмма последовательностей показывает взаимодействие объектов во времени
B. нет правильного ответа
C. диаграмма последовательностей не предназначена для отображения взаимодействия объектов
D. диаграмма последовательностей описывает статические аспекты системы
E. диаграммы последовательностей - это просто другая форма диаграмм прецедентов

9. Сұрақ
Вопрос
Указатель - это переменная, которая содержит в качестве своего значения _____другой переменной
A. Размерность
B. Индекс
C. Адрес
D. Номер
E. Код

10. Сұрақ
Вопрос
Что обозначает данная критическая ошибка EDatabaseError?
A. Попытка доступа к адреса вне памяти, распределенной приложению
B. Ложное выражение, проверяемое процедурой AsserOT
C. Ошибка работы с базами данных
4)«Молчаливое» исключение, предназначенное для намеренного прерывания вычислений и быстрого выхода! глубоко вложенных процедур и функций
E. Ошибка преобразования строк или объектов (в частности, в функциях StrToInt, StrToFloat, StrToDate

11. Сұрақ
Вопрос
Чему будет равен результат вычисления:
int i, k = 2, m = 10;
i = (m)/(m/k - 5); cout>>i;
Выберите один из 5 вариантов ответа:
A. ошибка во время исполнения
B. 0
C. 18
D. 5
E. 10

12. Сұрақ
Вопрос
Строки в Си++ представляются в виде:
A. массива элементов типа float
B. символического представления ячейки памяти
C. массива элементов типа int
D. массива элементов типа char
E. одного идентификатора

13. Сұрақ
Вопрос
C++Builder.На какой странице указывается подпись под пиктограммой свернутой программы (Title), сама пиктограмма (Icon) и имя Help-файла (Help file)?

A. Compiler
B. Unit
C. Default
D. Application
E. Form

14. Сұрақ
Вопрос
Составной оператор - это:
A. последовательность операторов, заключенная между ковычками « »
B. последовательность операторов, заключенная в операторные скобки begin ... end
C. последовательность операторов, заключенная квадратные скобки [ ]
D. последовательность операторов, заключенная в фигурные скобки { }
E. последовательность операторов, заключенная в круглые скобки ( )

15. Сұрақ
Вопрос
Инкапсуляция....
A. позволяет использовать в дочерних классах функционал родительского класса и, в случае необходимости, дополнять его
2)представляет собой механизм, который объединяет данные и методы,манипулирующие этими данными, и защищает и то и другое от внешнего вмешательства или неправильного использования
C. представляет собой способность к изменению функционала, унаследованного от базового класса
D. использование только тех характеристик объекта, которые с достаточной точностью представляют его в данной системе
E. свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач

16 Сұрақ
Вопрос
Инструментальные средства программирования:
A. BIOS (базовая система ввода-вывода);
B. СУБД (системы управления базами данных);
C. компиляторы, интерпретаторы;
D. Пакеты прикладных программ
E. ОС (операционные системы).

17. Сұрақ
Вопрос
Какие служебные символы используются для обозначения начала и конца блока кода в языке C++?
A. { }
B. < >
C. []
D. ( )
E. begin end

18. Сұрақ
Вопрос
C++Builder.К какой закладке относятся Editor, Desktops, Toolbars?
A. Designer
B. Library
C. Palette
D. Object Inspector
E. View

19. Сұрақ
Вопрос
Что выведет следующая программа ?
```
#include <iostream>
using namespace std;
int main() {
int l_i ;
for( l_i = 0; l_i <= 9; l_i++)
cout << l_i +1;
return 0;}
```
A. цифры от 0 до 8
B. цифры от 1 до 10
C. цифры от 8 до 1
D. программа не будет построена из-за ошибок
E. цифры от 1 до 9

20. Сұрақ
Вопрос
В языке Си++ указатель - это:
A. символическое представление адреса ячейки памяти

B. символ, указывающий на что-либо
C. метка
D. специальный значок, показывающий, что это динамическая переменная
E. константа

21. Сұрақ
Вопрос
Логическое «не» обозначается:
A. ! =
B. !!
C. !
D. ||
E. *

22. Сұрақ
Вопрос
Операция битового «и» обозначается:
A. ||
B. \~
C. ~
D. $
E. &

23. Сұрақ
Вопрос
Что такое деятельность?
A. протяженное во времени составное поведение
B. протяженная во времени составная последовательность деятельностей
C. протяженный во времени составное вычисление (действия, action) и перехода как передачи контроля
D. протяженный во времени составной поток управления
E. протяженная во времени составная передача контроля от объекта к объекту

24. Сұрақ
Вопрос
Текстовый поток - это:
A. логическое понятие, которое система может относить к чему угодно - от дисковых файлов до терминалов
B. это последовательность символов, при передаче символов из потока на экран, часть из них не выводится (например, символ возврата каретки, перевода строки)
C. последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве.
D. комментарий
E. код программы

25. Сұрақ
Вопрос
Какое свойство компонента ADOTable задает параметры
фильтрации данных?
A. Filter
B. DataSource
C. Name
D. DatabaseName
E. Filtered

26. Сұрақ
Вопрос
Формальный аргумент - это:
A. аргумент, передаваемый в функцию при её вызове
B. в языке C++ не используются формальные аргументы
C. переменная, диапазон значений которой состоит из адресов ячеек памяти или специального значения
D. аргумент, указываемый при объявлении или определении функции.
E. объект, указывающий на определенные данные, но не хранящий их

27. Сұрақ
Вопрос
Компонент DBGrid - ...
A. связывается с конкретной таблицей БД (закладка ADO);
B. используется для подключения к БД (закладка палитры ADO, в некоторых версиях dbGO);
C. кнопочная панель, способная управлять данными в привязанной к ней таблице (закладка Data Controls).
D. таблица, позволяющая вывести содержимое таблицы БД на пользовательскую форму (закладка Data Controls);
E. компонент, используется как связка данных из таблиц, с отображающими и

управляющими компонентами (закладка Data Access);

28. Сұрақ
Вопрос
Выражения - это:
A. основные строительные блоки программы
B. набор символов и операций
C. операторы, выполняющие определенные действия с переменными
D. конструкции, включающие константы (литералы), переменные, знаки операций, скобки для управления порядком выполнения операций, обращения к функциям
E. логические операторы, выполняющие определенные действия с переменными

29. Сұрақ
Вопрос
Оператор-переключатель - это:
A. оператор цикла WHILE/DO
B. оператор для выбора одного из нескольких вариантов (SWITCH)
C. строка с меткой DEFAULT
D. оператор условия IF/ELSE
E. BREAK

30. Сұрақ
Вопрос
Наличие нуль-символа (\0) означает, что:
A. логическим значением переменной является «истина»
B. количество ячеек массива должно быть, по крайней мере, на одну больше, чем число символов, которые необходимо размещать в памяти
C. количество ячеек массива должно быть на две больше, чем число символов, которые необходимо размещать в памяти
D. логическим значением переменной является «ложь»
E. количество ячеек массива должно быть на одну меньше, чем число символов, которые необходимо размещать в памяти

6 Нұсқа
Вариант

1. Сұрақ
Вопрос
Оператор цикла WHILE является:
A. конструкцией цикла с перебором значений параметра
B. конструкцией цикла с выбором варианта
C. конструкцией цикла с предусловием
D. конструкцией цикла с постусловием
E. оператор выбора альтернатив

2. Сұрақ
Вопрос
Писать # include <stdio.h> нужно для:
A. замены каждого параметра в строке лексем на соответствующий аргумент
B. позволяет дать в программе макроопределения (или задать макросы)
C. подключения файла, содержащего макроопределения и объявления данных, необходимых для работы функций из стандартной математической библиотеки
D. переопределения не только константы, но и целых программных конструкций
E. подключения файла, содержащего макроопределения и объявления данных, необходимых для работы функций из стандартной библиотеки ввода-вывода

3. Сұрақ
Вопрос
Аналогом какой диаграммы является диаграмма кооперации?
A. диаграммы вариантов
B. диаграммы деятельностей
C. диаграммы прецедентов
D. диаграммы классов
E. диаграммы последовательностей

4. Сұрақ
Вопрос
У какого компонента имеется свойство Items?
A. DBRadioGroup

B. Button
C. Label
D. DBEdit
E. Bevel

5. Сұрақ
Вопрос
Точка с запятой является:
A. частью оператора
B. спецсимвол
C. не используется в языке программирования
D. разделителем операторов
E. ключевым знаком языка Си

6. Сұрақ
Вопрос
Как расшифровывается аббревиатура UML?
A. Unified Method Language
B. Unified Modeling Language
C. Unified Markup Language
D. Unified Methodology Language
E. Universal Modeling Language

7. Сұрақ
Вопрос
Чему будет равна переменная a, после выполнения этого кода?
int a;
for(a = 0; a < 10; a++) {}
cout<<a;
A. 9
B. 11
C. 10
D. 6
E. 1

8. Сұрақ
Вопрос
Самый большой этап в жизненном цикле программы:
A. программирование;
B. эксплуатация;
C. изучение предметной области;
D. тестирование;
E. корректировка ошибок.

9. Сұрақ
Вопрос
В языке Си++ тело функции ограничено операторными скобками:
A. /* */
B. start finish
C. begin end
D. []
E. {}

10. Сұрақ
Вопрос
Что означает символ "кошачий глаз" на диаграмме состояний?
A. принятие решения
B. слияние потоков управления
C. начальное состояние
D. конечное состояние
E. конец потока управления

11. Сұрақ
Вопрос
C++Builder.Какая опция импортирует в проект библиотеку типов элементов ActiveX в пункте меню Project?
A. Environment Options
B. Editor Options
C. Remove From Project
D. Add To Project
E. Import Type Library

12. Сұрақ
Вопрос
Этап, занимающий наибольшее время, в жизненном цикле программы:
A. сопровождение
B. тестирование;
C. программирование;
D. формулировка требований
E. проектирование;

13. Сұрақ
Вопрос
C++Builder.Какая опция в отладочном режиме выполняет текущую строку кода и прослеживает работу вызываемых подпрограмм в пункте меню Run?
A. Trace To Next Source Line
B. Step Over
C. Install MTS Objects
D. Run To Cursor
E. Trace Into

14. Сұрақ
Вопрос
Что такое диаграмма взаимодействия?
A. нет правильного ответа
B. диаграмма, на которой представлено взаимодействие, состоящее из множества объектов одного класса и его подклассов и сообщений, которыми они обмениваются
C. диаграмма, на которой представлено взаимодействие, состоящее из множества объектов и отношений между ними, включая и сообщения, которыми они обмениваются
D. диаграмма, на которой представлено взаимодействие, состоящее из множества объектов одного класса и сообщений, которыми они обмениваются
E. диаграмма, на которой представлено взаимодействие, состоящее из сообщений, которыми обмениваются элементы модели

15. Сұрақ
Вопрос
Метки в операторе Switch должны быть:
A. Указателями
B. Переменной
C. Константой
D. типа Char
E. типа Float

16. Сұрақ
Вопрос
Автоматические объекты:
A. существуют во время выполнения данного блока и теряют свои значения при выходе из него
B. указывают компилятору выделить для сохранения данных объекта не ячейке стека, а внутренние регистры процессора.
C. являются переменными статического класса памяти
D. можно инициализировать только выражениями с константами и с указателями на ранее описанные объекты
E. используется в объединении с так называемыми константными функциями (функциями, которые объявлены с ключевым словом const).

17. Сұрақ
Вопрос
Какой этап выполняется раньше:
Выберите один из 5 вариантов ответа:
A. оптимизация;
B. отладка;
C. сопровождение
D. тестирование.
E. программирование;

18. Сұрақ
Вопрос
Оптимизация программы это:
A. уменьшение сложности программы
B. улучшение работы существующей программы;
C. отладка
D. повышение сложности программы
E. тестирование

19. Сұрақ
Вопрос
Что такое структурное программирование?
A. Методология разработки ПО, в основе которых лежит представление программ в виде иерархической структуре блоков
B. Методология разработки ПО, в основе которых лежит представление программ в виде линейной задачи
C. Программирование по частям одной процедуры
D. Методология разработки ПО, в основе которых лежит представление программ в виде схем
E. Разработки из частей

20. Сұрақ
Вопрос
Правила, которым должна следовать программа это:
A. спецификация;
B. структура;
C. алгоритм;
D. иерархия.

E. состав информации

21. Сұрақ
Вопрос
Макровызов должен состоять:
A. из макроимени и заключенного, в круглые скобки списка аргументов
B. из списка макросов
C. из списка микропеременных
D. из списка макроимени
E. из списка макропеременных

22. Сұрақ
Вопрос
Каков будет результат выполнения операторов:
int x,y,a;
x=5;
y=x*2+7;
a=y/4;
A. x = 5, y = 17, a = 4
B. x = 4, y = 17, a = 5
C. x = 5, y = 32, a = 8
D. x = 5, y = 17, a = 4,25
E. x = 5, y = 10, a = 2,2

23. Сұрақ
Вопрос
Наследование это:
A. определение новых структур данных;
B. передача свойств потомкам;
C. разделение переменных, процедур и функций;
D. применение стандартных процедур и функций.
E. передача свойств предкам;

24. Сұрақ
Вопрос
Каков будет результат выполнения операторов:
int x,y;
x=y=5;
x+=2;
y-=3;
x*=y;
x/=++y;
A. y = 12, x = 12/3
B. y = 3, x = 4
C. y = 4, x = 12
D. y = 4, x = 14
E. y = 3, x = 16

25. Сұрақ
Вопрос
Какой результат следующего выражения ?
int *a; int b[2]; a = b;
b[0] = 7; b[1] = 10; *a++; cout << *a;
A. 11
B. 12
C. 10
D. 7
E. 8

26. Сұрақ
Вопрос
C++Builder.Какая опция компилирует и выполняет программу в пункте меню Run?
A. Run
B. Parameters
C. Unregister ActiveX Server
D. Register ActiveX Server
E. Attach to Process

27. Сұрақ
Вопрос
Какое слово из списка не относится к зарезервированным словам Си++?
A. cast
B. try
C. volatile
D. main
E. union

28. Сұрақ
Вопрос
Каков будет результат выполнения операторов:
int x,y;
y=-4;
x=(y<0)?-y:y;
A. x = -4
B. x = 4
C. x= 8
D. x = 2
E. x = 0

29. Сұрақ

Вопрос
Укажите пример объекта
A. кот
B. книги
C. растения
D. карандаши
E. деревья

30. Сұрақ
Вопрос
Фактический аргумент - это:
A. аргумент, указываемый при объявлении или определении функции
B. переменная, диапазон значений которой состоит из адресов ячеек памяти или специального значения
C. объект, указывающий на определенные данные, но не хранящий их
D. аргумент, передаваемый в функцию при её вызове;
E. в языке C++ не используются фактические аргументы

7 Нұсқа
Вариант

1. Сұрақ
Вопрос
Из приведенных утверждений
а) объект является экземпляром определенного класса
б) класс является экземпляром определенного объекта
A. верно только б
B. верно только а
C. верны и а, и б
D. оба утверждения ложны

2. Сұрақ
Вопрос
Какое свойство компонента DataSource осуществляет связь с компонентом Table?
A. DataSet
B. Tag
C. Enabled
D. Name
E. AutoEdit

3. Сұрақ
Вопрос
Какой операции нет в C++?
A. Префиксный инкремент
B. Тернарная
C. Последовательной
D. Бинарной
E. Унарной

4. Сұрақ
Вопрос
Что такое стиль программирования?
A. Программирование, которое стилизуется при написании программы
B. Набор элементов, которые образуют надежность, дружественность, отличный интерфейс
C. Использование отступов
D. Набор приемов и методов программирования, которые необходимы соблюдать при написании программы
E. Хороший стиль программирования

5. Сұрақ
Вопрос
Полиморфизм....
A. защищает и то, и другое от внешнего вмешательства или неправильного использования
B. позволяет использовать в дочерних классах функционал родительского класса и, в случае необходимости, дополнять его
C. представляет собой способность к изменению функционала, унаследованного от базового класса
D. представляет собой механизм, который объединяет данные и методы, манипулирующие этими данными, и защищает и то и другое от внешнего вмешательства или неправильного использования
E. экземпляр класса

6. Сұрақ
Вопрос

Для создания теста с одним правильным вариантом ответа используется элемент управления:
A. ToolBar
B. CoolBar
C. RadioBox
D. CheckBox
E. RadioButton

7. Сұрақ
Вопрос
В каких единицах можно измерить надежность:
A. отказов/час;
B. Кбайт/сек;
C. операций /минуту
D. операций/сек
E. км/час;

8. Сұрақ
Вопрос
Тип функции определяется:
A. типом переменных в программе
B. типом возвращаемого ею значения
C. типом ее аргументов
D. типом ее описания
E. использованием в программе

9. Сұрақ
Вопрос
Операция вывода в C++ имеет вид
A. cin>>
B. sout <<
C. post <<
D. cout <<
E. output <<

10. Сұрақ
Вопрос
Объект является
A. переменной, описание которой создает программист при создании класса.
B. сообщением, которое возникает в различных точках исполняемого кода при выполнении определённых условий.
C. конкретным экземпляром определенного класса
D. совокупностью средств, методов и правил взаимодействия (управления, контроля и т. д.) между элементами системы
E. способом доступа к внутреннему состоянию объекта, имитирующий переменную некоторого типа.

11. Сұрақ
Вопрос
Методы – это...
A. переменные, принадлежащие классу или экземпляру класса
B. процедуры и функции класса
C. запись о классе
D. синтаксическая надстройка, поддерживаемая компилятором и средой, которая позволяет вызывать методы других объектов
E. синтаксическая надстройка, позволяющая осуществлять в форме вызов функции

12. Сұрақ
Вопрос
Какой из ниже перечисленных операторов является циклом в C++?
A. do while
B. repeat
C. if
D. else
E. until

13. Сұрақ
Вопрос
Какое свойство компонента DBImage задает равнение относительно формы?
A. Alignment
B. Color
C. DataSource
D. DataField
E. Align

14. Сұрақ
Вопрос
Какой класс является корнем иерархии исключительных ситуаций?
A. EAbort
B. Exception
C. EOverflow
D. EStreamError
E. EInvalidOp

15. Сұрақ
Вопрос
Доступ, при котором записи файла обрабатываются в произвольной последовательности, называется:
A. последовательным;
B. дополнительным
C. основным.
D. прямым;
E. простым;

16. Сұрақ
Вопрос
Что такое модель жизненного цикла программного обеспечения?
A. структура, содержащая процессы действия и задачи, которые осуществляются в ходе разработки, использования и сопровождения программного продукта.
B. структура, содержащая процессы действия и задачи, которые осуществляются в ходе разработки.
C. структура, содержащая процессы задачи, которые осуществляются в ходе использования и сопровождения программного продукта.
D. модель содержащая процессы действия и задачи, которые осуществляются в ходе разработки, использования и сопровождения программного продукта.
E. действия содержащие процессы действия и задачи, которые осуществляются в ходе разработки, использования и сопровождения программного продукта.

17. Сұрақ
Вопрос
Писать # include <math.h> нужно для:
A. замены каждого параметра в строке лексем на соответствующий аргумент макровызова
B. переопределения не только константы, но и целых программных конструкций
C. позволяет дать в программе макроопределения (или задать макросы) переопределения не только константы, но и целых программных конструкций
D. подключения файла, содержащего макроопределения и объявления данных, необходимых для работы функций из стандартной математической библиотеки
E. подключения файла, содержащего макроопределения и объявления данных, необходимых для работы функций из стандартной библиотеки ввода-вывода

18. Сұрақ
Вопрос
Какое свойство компонента ADOTable делает активным таблицу?
A. DatabaseName
B. Name
C. DataSource
D. Active
E. Hint

19. Сұрақ
Вопрос
Что означает символ "бриллианта" на диаграмме деятельностей?
A. распараллеливание потоков деятельностей
B. слияние потоков деятельностей
C. конечное состояние
D. принятие решения
E. начальное состояние

20. Сұрақ
Вопрос
Какая это модель жизненного цикла программного обеспечения?
A. Каскадная модель (водопад)
B. Модель кодирования и устранения ошибок
C. V модель
D. Спиральная модель жизненного цикла ПО

E. Каскадная модель с промежуточным контролем

21. Сұрақ
Вопрос
Оператор цикла DO/WHILE является:
A. конструкцией цикла с выбором варианта
B. конструкцией цикла с постусловием
C. оператор выбора альтернатив
D. конструкцией цикла с перебором значений параметра
E. конструкцией цикла с предусловием

22. Сұрақ
Вопрос
Что такое ЕСПД?
A. Единый стандарт проектной документации
B. Единая система проектной документации
C. Единственный стандарт программной документации
D. Единая система программной документации
E. Нет ответа

23. Сұрақ
Вопрос
В языке Си++ лексема - это:
A. набор специальных символов и директив
B. процедура, выполняющая определенные задания
C. множество строк, определяющих состояние программы
D. это логически выделенная единица языка, воспринимаемая как единое целое компилятором и программистом.
E. функция, выполняющая определенные задания

24. Сұрақ
Вопрос
Согласно ГОСТ 19.102-77, в этап: Разработка программы входит
A. Корректировка программы и программной документации по результатам испытаний.
B. Разработка плана мероприятий по разработке и внедрению программ.
C. Программирование и отладка программы.
D. Разработка программных документов в соответствии с требованиями ГОСТ 19.101-77.
E. Разработка, согласование и утверждение порядка и методики испытаний.

25. Сұрақ
Вопрос
Чему будет равен результат вычисления:
int i, k = 3, m = 12;
i = (m)/(m/k - 5);
cout<<i;
A. ошибка во время исполнения
B. 0
C. -12
D. 12
E. 10

26. Сұрақ
Вопрос
Что такое надежность?
A. Соответствие проверяемого объекта некоторому эталонному объекту или совокупности формализованных эталонных характеристик и правил.
B. Предполагает хорошо спроектированные диалоговые окна
C. Хороший интерфейс
D. Качественный код
E. Свойство программы выполнять заданные функции, сохранять свои характеристики в установленных пределах при определенных условиях эксплуатации.

27. Сұрақ
Вопрос
У какого из приведённых компонентов, имеется свойство Text:
A. DBRadioGroup, OleContainer
B. DBRichEdit, OleContainer, DBRadioGroup
C. DBRadioGroup, OleContainer, DBRichEdit
D. есть во всех

E. DBRichEdit

28. Сұрақ
Вопрос
Какое свойство компонента DBEdit осуществляет связь с определенным полем таблицы?
A. DataField
B. Color
C. Alignment
D. Align
E. DataSource

29. Сұрақ
Вопрос
Что такое транслятор?
A. Программа для перевода из машинного кода в язык программирования
B. Программа для переводы с языка программирования на машинные коды
C. Переводит исходный текст в программный код
D. Программа для создания изменений исходных программ
E. Программа для изменения кода

30. Сұрақ
Вопрос
C++Builder.Какая закладка позволяет настроить "горячие" клавиши, используемые в кодовом редакторе для ускорения решения типовых задач?
A. Color
B. General
C. Palette
D. Key Mappings
E. Code insight

8 Нұсқа
Вариант

1. Сұрақ
Вопрос
Каков будет результат выполнения операторов:
int i,j,s;
i=j=2; /* i и j получают значение 2 */
s=(i++)+(--j);
A. i= 2, j = 1, s = 3
B. i = 3, j = 1, s = 3
C. i =1, j = 3, s = 1
D. i = 3, j = 2, s = 5
E. i = 1, j = 1, s = 3

2. Сұрақ
Вопрос
Что обозначает данная критическая ошибка EDivByZero?
A. Попытка доступа к адреса вне памяти, распределенной приложению
B. Ошибка преобразования строк или объектов (в частности, в функциях StrToInt, StrToFloat, StrToDate
C. «Молчаливое» исключение, предназначенное для намеренного прерывания вычислений и быстрого выхода! глубоко вложенных процедур и функций
D. Ошибка работы с базами данных
E. Попытка целочисленного деления на нуль.

3. Сұрақ
Вопрос
Цель структурного программирования:
A. Снижение работоспособности всего процесса создания ПО
B. Снижение трудоемкости описания создания ПО
C. Снижение результативности всего процесса создания ПО
D. Снижение эффективности всего процесса создания ПО
E. Снижение трудоемкости всего процесса создания ПО

4. Сұрақ
Вопрос
Угроза – это…
A. нет правильного ответа
B. потенциальное возможное событие, действие, процесс или явление, которое может привести к нанесению данных о программе
C. возможное событие, действие, процесс или явление, которое может привести к сохранности данных

D. потенциальное возможное событие, действие, процесс или явление, которое может привести к нанесению ущерба.
E. потенциальное возможное событие, действие, процесс или явление, которое может привести удалению данных о программе

5. Сұрақ
Вопрос
Создание исполняемого кода программы без написания исходного кода называется:
A. трассировкой
B. составлением спецификаций;
C. отладкой;
D. автоматизацией программирования;
E. проектированием.

6. Сұрақ
Вопрос
Первый этап в жизненном цикле программы:
A. анализ требований;
B. проектирование;
C. комплексное тестирование.
D. автономное тестирование;
E. формулирование требований;

7. Сұрақ
Вопрос
C++Builder. Какая закладка позволяет настроить общие свойства кодового редактора?
A. General
B. Display
C. Color
D. Environment variables
E. Palette

8. Сұрақ
Вопрос
Что такое дружественность (юзабилити)?
A. свойство технической системы сохранять свою работоспособность после отказа одной или нескольких её составных частей.
B. свойство объекта непрерывно сохранять работоспособное состояние в течение некоторого времени.
C. свойство объекта сохранять во времени в установленных пределах значения всех параметров, характеризующих способность выполнять требуемые функции в заданных условиях применения, технического обслуживания, хранения и транспортирования
D. свойство объекта сохранять работоспособность при отказе отдельных функциональных узлов.
E. способность продукта быть понимаемым, изучаемым, используемым и привлекательным для пользователя в заданных условиях

9. Сұрақ
Вопрос
Какое свойство компонента DBImage содержит его имя?
A. DataField
B. Name
C. Align
D. DataSource
E. Alignment

10. Сұрақ
Вопрос
Что выполняется раньше:
A. программирование;
B. отладка;
C. сопровождение
D. тестирование.
E. проектирование;

11. Сұрақ
Вопрос
В стадии разработки программы не входит:
A. автоматизация программирования;
B. постановка задачи;
C. составление спецификаций;
D. эскизный проект;
E. тестирование.

12. Сұрақ
Вопрос

Способы оценки качества:
A. наличие документации;
B. оптимизация программы;
C. структурирование алгоритма
D. сравнение с аналогами;
E. отладка

13. Сұрақ
Вопрос
Операция ввода в C++ имеет вид
Выберите один из 5 вариантов ответа:
A. cout <<
B. post <<
C. cin>>
D. output <<
E. sout <<

14. Сұрақ
Вопрос
Что выполняется раньше:
A. написание исходного кода;
B. выбор языка программирования;
C. компиляция.
D. разработка алгоритма;
E. отладка

15. Сұрақ
Вопрос
Модульное программирование это:
A. структурирование;
B. разбиение программы на отдельные части;
C. программирование без участия программиста;
D. использование стандартных процедур и функций.
E. объединение задач в одну;

16. Сұрақ
Вопрос
Каков будет результат выполнения операторов:
a=(y=(x=5)*2+7)/4;
A. a = 4
B. error
C. a = 2,25
D. a = 2
E. a = 4,25

17. Сұрақ
Вопрос
Что такое структурная обработки исключительной ситуации?
A. Пошаговое выполнение программы с остановками на каждой команде или строке
B. Этап разработки компьютерной программы на котором обнаруживают, локализуют и устраняют синтаксические ошибки
C. Некоторая ситуация в программе, которая требует специальной обработки
D. механизм, предназначенный для описания реакции программы на ошибки выполнения и другие возможные проблемы, которые могут возникнуть при выполнении программы
E. Дополнительный модуль или расширение программы, добавляющее ей новые возможности

18. Сұрақ
Вопрос
Способы оценки надежности:
A. отладка
B. трассировка;
C. сравнение с аналогами;
D. тестирование;
E. оптимизация

19. Сұрақ
Вопрос
В каких единицах можно измерить быстродействие:
A. Кбит/сек
B. Кбайт/сек;
C. отказов/час;
D. км/час;
E. операций/сек.

20. Сұрақ
Вопрос
Переменные, принадлежащие классу или экземпляру класса – это
A. Методы
B. События
C. Функция
D. Поля
E. Свойства

21. Сұрақ
Вопрос
На языке программирования составляется:
A. алгоритм
B. исходный код
C. итоговый код
D. объектный код
E. Блок схема

22. Сұрақ
Вопрос
Доступ, при котором записи файла читаются в физической последовательности, называется:
A. основным.
B. прямым;
C. дополнительным
D. последовательным;
E. простым;

23. Сұрақ
Вопрос
Класс является
A. частью объекта
B. компонентом объекта
C. реализатором объекта
D. представителем объекта
E. описанием объекта

24. Сұрақ
Вопрос
Методы программирования (укажите НЕ верный ответ):
A. процедурное
B. типизированное;
C. модульное.
D. структурное;
E. объектно-ориентированный

25. Сұрақ
Вопрос
Что выведет следующая программа ?
```
#include <iostream>
using namespace std;
int main() {
int 2_i ;
for( 2_i = 0; 2_i < 9; 2_i++)
cout << 2_i +1;
return 1;
}
```

A. программа не будет построена из-за ошибок
B. цифры от 8 до 1
C. цифры от 1 до 9
D. цифры от 0 до 8
E. цифры от 2 до 9

26. Сұрақ
Вопрос
Как называется способ составления имен переменных, когда в начале имени сообщается тип переменной:
A. венгерской нотацией;
B. прямым указанием;
C. стиль программирования
D. поляризацией.
E. структурным программированием;

27. Сұрақ
Вопрос
На каком этапе производится выбор языка программирования:
A. проектирование;
B. сопровождение
C. отладка;
D. тестирование.
E. программирование;

28. Сұрақ
Вопрос
Защитное программирование это:
A. разделение доступа в программе;
B. создание задач защищенных от копирования
C. встраивание в программу отладочных средств;
D. использование паролей;
E. оформление авторских прав на программу.

29. Сұрақ
Вопрос
Какой тип возвращаемого значения должен иметь метод, возвращающий адрес учебного заведения?
A. void
B. числовой

C. логический
D. строковый
E. тип указывать не нужно

30. Сұрақ
Вопрос
C++Builder.Какая закладка позволяет выбрать цвета для отдельных синтаксических элементов?
A. Color
B. Display
C. General
D. Environment variables
E. Palette

9 Нұсқа
Вариант

1. Сұрақ
Вопрос
В следующем операторе Select * from rashod where kolvo>20 укажите имя таблицы?
A. rashod
B. from
C. select
D. kolvo
E. where

2. Сұрақ
Вопрос
Локализация ошибки:
A. определение места возникновения ошибки;
B. исправление ошибки.
C. анализ влияния ошибки на все приложение
D. определение причин ошибки;
E. обнаружение причин ошибки;

3. Сұрақ
Вопрос
Какой класс используется для вывода данных во внутреннюю область памяти?
A. strout
B. iostream
C. Cout
D. strstream
E. fstream

4. Сұрақ
Вопрос
Какое свойство компонента ADOTable содержит имя таблицы?
A. Name
B. Active
C. DataSource
D. DatabaseName
E. TableName

5. Сұрақ
Вопрос
Какой статус международного стандарта языка Си++?
A. принят ISO и тем самым автоматически принят во всех странах
B. принят только в США и ждет одобрения международной организации
C. стандарт не имеет статус международного
D. принят проект стандарта, дорабатывается
E. стандарт нигде не принят и не используется

6. Сұрақ
Вопрос
Когда программист может проследить последовательность выполнения команд программы:
A. при компиляции;
B. при трассировке;
C. при компоновке
D. при тестировании;
E. при выполнении программы

7. Сұрақ
Вопрос
Каков будет результат выполнения операторов
int a,b, m;
a = 4;
b=7;
m=(a>b)?a:b;
A. m = 11
B. m = 7
C. m = 4
D. m = 8
E. m = 3

8. Сұрақ
Вопрос
Что легко поддается автоматизации:
A. работа с файлами;
B. отладка
C. интерфейс;
D. сложные логические задачи;
E. алгоритмизация.

9. Сұрақ
Вопрос
Оптимизация программы это:
A. отладка
B. повышение сложности программы
C. модификация
D. уменьшение сложности программы
E. тестирование

10. Сұрақ
Вопрос
Фактический адрес в указателях - это:
A. Число
B. Слово
C. Указатель
D. Строка
E. Буква

11. Сұрақ
Вопрос
Логическое «и» обозначается:
A. ||
B. =
C. &&
D. &
E. @

12. Сұрақ
Вопрос
Когда можно обнаружить синтаксические ошибки:
A. при отладке;
B. при тестировании;
C. на этапе проектирования;
D. при эксплуатации.
E. при компиляции;

13. Сұрақ
Вопрос
Какая процедура устанавливает указатель в таблице на первую запись?
A. MoveBy
B. Prior
C. First
D. Next
E. Last

14. Сұрақ
Вопрос
Полиморфизм это:
A. изменение поведения потомков, имеющих общих предков;
B. изменение статуса экземпляров, имеющих общих предков;
C. применение стандартных процедур и функций.
D. изменение поведения потомков на разные события;
E. передача свойств по наследству;

15. Сұрақ
Вопрос
Какое утверждение верно:
A. потомки не могут иметь общих предков;
B. потомки наследуют свойства родителей.
C. родители наследуют свойства потомков;
D. потомки не наследуют свойства родителей;
E. предки наследуют свойства родителей;

16. Сұрақ
Вопрос
Что такое исключительная ситуация?
A. Этап разработки компьютерной программы на котором обнаруживают и синтаксические ошибки
B. Некоторая ситуация в программе, которая требует специальной обработки
C. Пошаговое выполнение программы с остановками на каждой команде или строке
D. Дополнительный модуль или расширение программы, добавляющее ей новые возможности

E. Подключаемый к основному приложению модуль, расширяющий (или наоборот, уменьшающий) его функциональность.

17. Сұрақ
Вопрос
Наследование...
A. экземпляр класса
B. защищает и то, и другое от внешнего вмешательства или неправильного использования
C. представляет собой механизм, который объединяет данные и методы, манипулирующие этими данными
D. позволяет использовать в дочерних классах функционал родительского класса и, в случае необходимости, дополнять его
E. представляет собой способность к изменению функционала, унаследованного от базового класса

18. Сұрақ
Вопрос
Какой тип данных отсутствует в C++ в отличие от большинства других языков:
A. Float
B. Char
C. Real
D. String
E. Int

19. Сұрақ
Вопрос
Какой компонент соответствует кнопке ?
A. Grid
B. DataSource
C. Table
D. DBGrid
E. Edit

20. Сұрақ
Вопрос
На какой закладке находится компонент DataSource?
A. Sistem
B. Standard
C. Data Access
D. Additional
E. Win 32

21. Сұрақ
Вопрос
В следующем операторе Select * from rashod where kolvo>20 укажите имя поля?
A. kolvo
B. select
C. from
D. rashod
E. where

22. Сұрақ
Вопрос
В результате выполнения программы
int x, y;
x = 1;
y = 1;
if (x) {y = 0; } переменная y получит значение
Выберите один из 5 вариантов ответа:
A. 1
B. -1
C. 2
D. 0
E. -2

23. Сұрақ
Вопрос
Тело любого цикла выполняется до тех пор, пока его условие ...
A. ложно
B. у цикла нет условия
C. !=0
D. истинно
E. =0

24. Сұрақ
Вопрос
Каким образом производится вывод на экран символов a=?
A. conio << "a=";
B. cin >> "a=";
C. cout << "a=";
D. cout >> "a=";
E. cin << "a=";

25. Сұрақ

Вопрос
Отладка – это:
A. процедура поиска ошибок, когда известно, что ошибка есть;
B. процесс установки приложения на ПК пользователя
C. определение списка параметров;
D. составление блок-схемы алгоритма
E. правило вызова процедур (функций);

26 Сұрақ
Вопрос
.Выберите из списка истинные утверждения, касающиеся диаграммы последовательностей
A. диаграммы последовательностей - это просто другая форма диаграмм прецедентов
B. диаграммы последовательностей используются для уточнения диаграмм прецедентов
C. диаграмма последовательностей не предназначена для отображения взаимодействия объектов
D. нет правильного ответа
E. диаграмма последовательностей описывает статические аспекты системы

27. Сұрақ
Вопрос
Что относится к этапу программирования:
A. работоспособность;
B. разработка интерфейса;
C. написание кода программы;
D. анализ требований.
E. формирование требований

28. Сұрақ
Вопрос
На чем акцентирует внимание диаграмма кооперации?
A. на ролях, которые объекты играют во взаимодействии
B. на отношениях между объектами, которые участвуют во взаимодействии
C. на объектах, которые участвуют во взаимодействии
D. на последовательности сообщений, которыми обмениваются объекты
E. на сообщениях, которыми обмениваются объекты в ходе взаимодействия

29. Сұрақ
Вопрос
Нахождение наилучшего варианта из множества возможных:
A. сопровождение.
B. оптимизация;
C. отладка;
D. тестирование;
E. автоматизация;

30. Сұрақ
Вопрос
Каким символом изображается прецедент?
A. A
B. E
C. D
D. C
E. F
6) B

10 Нұсқа
Вариант

1. Сұрақ
Вопрос
Какой компонент соответствует кнопке ?
A. DBGrid
B. DBComboBox
C. Table
D. Grid
E. ADOTable

2. Сұрақ
Вопрос
Какой язык используется для реляционного доступа к данным?
A. C
B. Fortran
C. SQL
D. Pascal
E. Basic

3. Сұрақ

Вопрос
Какую функцию должны содержать все программы на C++?
A. start()
B. main()
C. program()
D. unit
E. system()

4. Сұрақ
Вопрос
Что подразумевает хороший стиль программирования?
A. Качественные переменные
B. Использование программ
C. Использование комментариев
D. Использование переменных
E. Использование UI

5. Сұрақ
Вопрос
Компонент ADOConnection - ...
A. используется для подключения к БД (закладка палитры ADO, в некоторых версиях dbGO);
B. компонент, используется как связка данных из таблиц, с отображающими и управляющими компонентами (закладка Data Access);
C. кнопочная панель, способная управлять данными в привязанной к ней таблице (закладка Data Controls).
D. таблица, позволяющая вывести содержимое таблицы БД на пользовательскую форму (закладка Data Controls);
E. связывается с конкретной таблицей БД (закладка ADO);

6. Сұрақ
Вопрос
ГОСТ 19.101-77 отвечает за

A. Виды программ
B. Виды программной документации и общие положения
C. Виды программ и программной документации
D. Виды программ и программной обозначений
E. Виды программ и общие положения

7. Сұрақ
Вопрос
Какое свойство компонента DBEdit позволяет настроить параметры шрифта?
A. Font
B. DataField
C. Visible
D. DataSource
E. Alignment

8. Сұрақ
Вопрос
Какое свойство компонента DBImage осуществляет связь с базой данных?
A. Align
B. Alignment
C. Color
D. DataField
E. DataSource

9. Сұрақ
Вопрос
Каким образом объекты внутри системы взаимодействуют между собой?
A. путем обмена информацией через буфер обмена
B. путем прямого вызова операций друг друга
C. путем использования интерфейсов родительских классов
D. путем отправки и приема сообщений
E. путем прямой записи в память

10. Сұрақ
Вопрос
Что выполняется раньше:
A. программирование;
B. отладка;
C. сопровождение
D. тестирование.
E. проектирование;

11. Сұрақ
Вопрос
В стадии разработки программы не входит:

A. автоматизация программирования;
B. постановка задачи;
C. составление спецификаций;
D. эскизный проект;
E. тестирование.

12. Сұрақ
Вопрос
Способы оценки качества:
A. наличие документации;
B. оптимизация программы;
C. структурирование алгоритма
D. сравнение с аналогами;
E. отладка

13. Сұрақ
Вопрос
Операция ввода в C++ имеет вид
Выберите один из 5 вариантов ответа:
A. cout <<
B. post <<
C. cin>>
D. output <<
E. sout <<

14. Сұрақ
Вопрос
Что выполняется раньше:
A. написание исходного кода;
B. выбор языка программирования;
C. компиляция.
D. разработка алгоритма;
E. отладка

15. Сұрақ
Вопрос
Модульное программирование это:
A. структурирование;
B. разбиение программы на отдельные части;
C. программирование без участия программиста;
D. использование стандартных процедур и функций.
E. объединение задач в одну;

16. Сұрақ
Вопрос
Каков будет результат выполнения операторов:
a=(y=(x=5)*2+7)/4;
A. a = 4
B. error
C. a = 2,25
D. a = 2
E. a = 4,25

17. Сұрақ
Вопрос
Что такое структурная обработки исключительной ситуации?
A. Пошаговое выполнение программы с остановками на каждой команде или строке
B. Этап разработки компьютерной программы на котором обнаруживают, локализуют и устраняют синтаксические ошибки
C. Некоторая ситуация в программе, которая требует специальной обработки
D. механизм, предназначенный для описания реакции программы на ошибки выполнения и другие возможные проблемы, которые могут возникнуть при выполнении программы
E. Дополнительный модуль или расширение программы, добавляющее ей новые возможности

18. Сұрақ
Вопрос
Способы оценки надежности:
A. отладка
B. трассировка;
C. сравнение с аналогами;
D. тестирование;
E. оптимизация

19. Сұрақ
Вопрос
В каких единицах можно измерить быстродействие:
A. Кбит/сек
B. Кбайт/сек;
C. отказов/час;
D. км/час;
E. операций/сек.

20. Сұрақ
Вопрос

Переменные, принадлежащие классу или экземпляру класса – это
A. Методы
B. События
C. Функция
D. Поля
E. Свойства

21. Сұрақ
Вопрос
На языке программирования составляется:
A. алгоритм
B. исходный код
C. итоговый код
D. объектный код
E. Блок схема

22. Сұрақ
Вопрос
Доступ, при котором записи файла читаются в физической последовательности, называется:
A. основным.
B. прямым;
C. дополнительным
D. последовательным;
E. простым;

23. Сұрақ
Вопрос
Класс является
A. частью объекта
B. компонентом объекта
C. реализатором объекта
D. представителем объекта
E. описанием объекта

24. Сұрақ
Вопрос
Методы программирования (укажите НЕ верный ответ):
A. процедурное
B. типизированное;
C. модульное.
D. структурное;
E. объектно-ориентированный

25. Сұрақ
Вопрос
Что выведет следующая программа ?

```
#include <iostream>
using namespace std;
int main() {
int 2_i ;
for( 2_i = 0; 2_i < 9; 2_i++)
cout << 2_i +1;
return 1;
}
```

A. программа не будет построена из-за ошибок
B. цифры от 8 до 1
C. цифры от 1 до 9
D. цифры от 0 до 8
E. цифры от 2 до 9

26. Сұрақ
Вопрос
Как называется способ составления имен переменных, когда в начале имени сообщается тип переменной:
A. венгерской нотацией;
B. прямым указанием;
C. стиль программирования
D. поляризацией.
E. структурным программированием;

27. Сұрақ
Вопрос
На каком этапе производится выбор языка программирования:
A. проектирование;
B. сопровождение
C. отладка;
D. тестирование.
E. программирование;

28. Сұрақ
Вопрос
Защитное программирование это:
A. разделение доступа в программе;
B. создание задач защищенных от копирования
C. встраивание в программу отладочных средств;
D. использование паролей;
E. оформление авторских прав на программу.

29. Сұрақ
Вопрос

Какой тип возвращаемого значения должен иметь метод, возвращающий адрес учебного заведения?
A. void
B. числовой
C. логический
D. строковый
E. тип указывать не нужно

30. Сұрақ
Вопрос
C++Builder.Какая закладка позволяет выбрать цвета для отдельных синтаксических элементов?
A. Color
B. Display
C. General
D. Environment variables
E. Palette

9 Нұсқа
Вариант

1. Сұрақ
Вопрос
В следующем операторе Select * from rashod where kolvo>20 укажите имя таблицы?
A. rashod
B. from
C. select
D. kolvo
E. where

2. Сұрақ
Вопрос
Локализация ошибки:
A. определение места возникновения ошибки;
B. исправление ошибки.
C. анализ влияния ошибки на все приложение
D. определение причин ошибки;
E. обнаружение причин ошибки;

3. Сұрақ
Вопрос
Какой класс используется для вывода данных во внутреннюю область памяти?
A. strout
B. iostream
C. Cout
D. strstream
E. fstream

4. Сұрақ
Вопрос
Какое свойство компонента ADOTable содержит имя таблицы?
A. Name
B. Active
C. DataSource
D. DatabaseName
E. TableName

5. Сұрақ
Вопрос
Какой статус международного стандарта языка Си++?
A. принят ISO и тем самым автоматически принят во всех странах
B. принят только в США и ждет одобрения международной организации
C. стандарт не имеет статус международного
D. принят проект стандарта, дорабатывается
E. стандарт нигде не принят и не используется

6. Сұрақ
Вопрос
Когда программист может проследить последовательность выполнения команд программы:
A. при компиляции;
B. при трассировке;
C. при компоновке
D. при тестировании;
E. при выполнении программы

7. Сұрақ
Вопрос
Каков будет результат выполнения операторов
int a,b, m;
a = 4;
b=7;
m=(a>b)?a:b;
A. m = 11
B. m = 7
C. m = 4

D. m = 8
E. m = 3

8. Сұрақ
Вопрос
Что легко поддается автоматизации:
A. работа с файлами;
B. отладка
C. интерфейс;
D. сложные логические задачи;
E. алгоритмизация.

9. Сұрақ
Вопрос
Оптимизация программы это:
A. отладка
B. повышение сложности программы
C. модификация
D. уменьшение сложности программы
E. тестирование

10. Сұрақ
Вопрос
Фактический адрес в указателях - это:
A. Число
B. Слово
C. Указатель
D. Строка
E. Буква

11. Сұрақ
Вопрос
Логическое «и» обозначается:
A. ||
B. =
C. &&
D. &
E. @

12. Сұрақ
Вопрос
Когда можно обнаружить синтаксические ошибки:
A. при отладке;
B. при тестировании;
C. на этапе проектирования;
D. при эксплуатации.
E. при компиляции;

13. Сұрақ
Вопрос
Какая процедура устанавливает указатель в таблице на первую запись?
A. MoveBy
B. Prior
C. First
D. Next
E. Last

14. Сұрақ
Вопрос
Полиморфизм это:
A. изменение поведения потомков, имеющих общих предков;
B. изменение статуса экземпляров, имеющих общих предков;
C. применение стандартных процедур и функций.
D. изменение поведения потомков на разные события;
E. передача свойств по наследству;

15. Сұрақ
Вопрос
Какое утверждение верно:
A. потомки не могут иметь общих предков;
B. потомки наследуют свойства родителей.
C. родители наследуют свойства потомков;
D. потомки не наследуют свойства родителей;
E. предки наследуют свойства родителей;

16. Сұрақ
Вопрос
Что такое исключительная ситуация?

A. Этап разработки компьютерной программы на котором обнаруживают и синтаксические ошибки
B. Некоторая ситуация в программе, которая требует специальной обработки
C. Пошаговое выполнение программы с остановками на каждой команде или строке
D. Дополнительный модуль или расширение программы, добавляющее ей новые возможности

E. Подключаемый к основному приложению модуль, расширяющий (или наоборот, уменьшающий) его функциональность.

17. Сұрақ
Вопрос
Наследование...
A. экземпляр класса
B. защищает и то, и другое от внешнего вмешательства или неправильного использования
C. представляет собой механизм, который объединяет данные и методы, манипулирующие этими данными
D. позволяет использовать в дочерних классах функционал родительского класса и, в случае необходимости, дополнять его
E. представляет собой способность к изменению функционала, унаследованного от базового класса

18. Сұрақ
Вопрос
Какой тип данных отсутствует в C++ в отличие от большинства других языков:
A. Float
B. Char
C. Real
D. String
E. Int

19. Сұрақ
Вопрос
Какой компонент соответствует кнопке ?
A. Grid
B. DataSource
C. Table
D. DBGrid
E. Edit

20. Сұрақ
Вопрос
На какой закладке находится компонент DataSource?
A. Sistem
B. Standard
C. Data Access
D. Additional
E. Win 32

21. Сұрақ
Вопрос
В следующем операторе Select * from rashod where kolvo>20 укажите имя поля?
A. kolvo
B. select
C. from
D. rashod
E. where

22. Сұрақ
Вопрос
В результате выполнения программы
int x, y;
x = 1;
y = 1;
if (x) {y = 0; } переменная y получит значение
Выберите один из 5 вариантов ответа:
A. 1
B. -1
C. 2
D. 0
E. -2

23. Сұрақ
Вопрос
Тело любого цикла выполняется до тех пор, пока его условие ...
A. ложно
B. у цикла нет условия
C. !=0
D. истинно
E. =0

24. Сұрақ
Вопрос
Каким образом производится вывод на экран символов a=?
A. conio << "a=";
B. cin >> "a=";
C. cout << "a=";
D. cout >> "a=";
E. cin << "a=";

25. Сұрақ
Вопрос
Отладка – это:

A. процедура поиска ошибок, когда известно, что ошибка есть;
B. процесс установки приложения на ПК пользователя
C. определение списка параметров;
D. составление блок-схемы алгоритма
E. правило вызова процедур (функций);

26 Сұрақ
Вопрос
.Выберите из списка истинные утверждения, касающиеся диаграммы последовательностей
A. диаграммы последовательностей - это просто другая форма диаграмм прецедентов
B. диаграммы последовательностей используются для уточнения диаграмм прецедентов
C. диаграмма последовательностей не предназначена для отображения взаимодействия объектов
D. нет правильного ответа
E. диаграмма последовательностей описывает статические аспекты системы

27. Сұрақ
Вопрос
Что относится к этапу программирования:
A. работоспособность;
B. разработка интерфейса;
C. написание кода программы;
D. анализ требований.
E. формирование требований

28. Сұрақ
Вопрос
На чем акцентирует внимание диаграмма кооперации?
A. на ролях, которые объекты играют во взаимодействии
B. на отношениях между объектами, которые участвуют во взаимодействии
C. на объектах, которые участвуют во взаимодействии
D. на последовательности сообщений, которыми обмениваются объекты
E. на сообщениях, которыми обмениваются объекты в ходе взаимодействия

29. Сұрақ
Вопрос
Нахождение наилучшего варианта из множества возможных:
A. сопровождение.
B. оптимизация;
C. отладка;
D. тестирование;
E. автоматизация;

30. Сұрақ
Вопрос
Каким символом изображается прецедент?
A. A
B. E
C. D
D. C
E. F
6) B

Примерная тематика курсового проектирования

1. Проектирование и разработка базы данных салона сотовой связи
2. Проектирование и разработка базы данных фитнес-клуба
3. Проектирование и разработка базы данных мебельной фирмы
4. Проектирование и разработка базы данных букмекерской конторы
5. Проектирование и разработка базы данных магазина книг и канцтоваров
6. Проектирование и разработка базы данных детского сада
7. Проектирование и разработка базы данных клининговой службы
8. Проектирование и разработка базы данных отеля
9. Проектирование и разработка базы данных ломбарда
10. Проектирование и разработка электронного пособия «Создание web – страниц, сайтов с применением web – технологий»
11. Проектирование и разработка базы данных горнолыжной базы
12. Проектирование и разработка ПО справочного терминала для студентов колледжа
13. Проектирование и разработка базы данных студии полиграфии

Тематика рефератов и докладов

1. Методологии программирования.
2. Языки программирования, основанные на Си.
3. Языки, поддерживающие абстракцию данных.
4. Принципы объектно-ориентированного программированияЭволюционная модель разработки ПО.
5. Формальная разработка ПО.
6. Разработка ПО на основе ранее созданных компонентов
7. Модель пошаговой разработки ПО.
8. Проектирование и реализация ПО.
9. Метрическая оценка ПО. 14. Эволюция ПО.
10. Автоматизированные средства разработки ПО.
11. Классификация CASE-средств.
12. Процесс объектно-ориентированного проектирования.
13. Окружение системы и модели ее использования.
14. Минимизация ошибок и сбоев
15. Отказоустойчивые архитектуры
16. Проектирование безопасных систем.
17. Объектно-ориентированные языки программирования. Особенности описания классов.
18. Процессно-ориентированный подход к программированию.
19. Интерактивный объектно-ориентированный подход.
20. Объектно-ориентированное программирование в C++.
21. Высокоуровневые методы обработки информации и программирования.
22. Эволюция технологий программирования.
23. Классификация САПР.
24. Логическое программирование.
25. Разработка программного обеспечения для работы с базой данных с использованием технологии объектно-ориентированного программирования.
26. СУБД структуры «сервер-клиент».
27. Технология создания БД в C++Builder: BDE.
28. Работа с мультимедиа в среде C++Builder.
29. Средства C++Builder, используемые для работы с диаграммами.
30. Построение многодокументных приложений.
31. Консольное приложение C++Builder.
32. Компоненты для работы с файлами и каталогами в C++Builder.
33. Воспроизведение анимационных роликов на форме.
34. Компоненты работы с локальной сетью в среде C++Builder.

Список использованных источников

1. Павловская Т.А. С/С++. Программирование на языке высокого уровня / Т. А. Павловская. - СПб.: Питер, 2004. - 461 с.: ил. Павловская Т. А.,
2. Щупак Ю. А. С++. Объектно-ориентированное программирование: Практикум. - СПб.: Питер, 2006. - 265 с: ил. Кольцов Д.М. 100 примеров на Си. - СПб.: “Наука и техника”, 2017 – 256с.
3. Доусон М. Изучаем С++ через программирование игр. - СПб.: “Питер”, 2016. - 352.
4. Седжвик Р. Фундаментальные алгоритмы на С++. Анализ/Структуры данных/Сортировка/Поиск: Пер. с англ. Роберт Седжвик. - К.: Издательство “Диасофт”, 2001. - 688с.
5. Сиддкхартха Р. Освой самостоятельно С++ за 21 день. - М.: SAMS, 2013. 651 с.
6. Стивен, П. Язык программирования С++. Лекции и упражнения, 6-е изд. Пер. с англ. - М.: ООО "И.Д. Вильямс", 2012. - 1248 с.
7. Черносвитов, А. Visual C++: руководство по практическому изучению А. Черносвитов . - СПб. : Питер, 2002. - 528 с. : ил.
8. Страуструп Б. Дизайн и эволюция языка С++. - М.: ДМК, 2000. - 448 с.
9. Мейерс С. Эффективное использование С++. - М.: ДМК, 2000. - 240 с.
10. Бадд Т. Объектно-ориентированное программирование в действии. - СПб: Питер, 1997. - 464 с.
11. Лаптев В.В. С ++. Объектно-ориентированное программирование: Учебное пособие.- СПб.: Питер, 2008. - 464 с.: ил.
12. Страуструп Б. Язык программирования С++.
13. Керниган Б., Ритчи Д. Язык программирования Си. Режим доступа:
14. http://cpp.com.ru/kr_cbook/index.html.
15. Герберт Шилдт: С++ базовый курс. Режим доступа: https://www.bsuir.by/m/12_100229_1_98220.pdf,
16. Богуславский А.А., Соколов С.М. Основы программирования на языке Си++. Режим доступа: http://www.ict.edu.ru/ft/004246/cpp_p1.pdf.
17. Линский, Е. Основы С++. Режим доступа: https://www.lektorium.tv/lecture/13373.