ЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
ПАВЛОДАРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ С. ТОРАЙГЫРОВА

АЛГОРИТМИЗАЦИЯ И ОСНОВЫ ПРОГРАММИРОВАНИЯ

Кереку
Павлодар

Министерство образования и науки Республики Казахстан

Павлодарский государственный университет
им. С. Торайгырова

Факультет Физики, математики и информационных
технологий

Кафедра «Вычислительная техника и программирование»

АЛГОРИТМИЗАЦИЯ И ОСНОВЫ ПРОГРАММИРОВАНИЯ

Учебно-методическое пособие

Павлодар
Кереку
2016

УДК 004.43(075.8)
ББК 32.973 - 018я7
А45

Рекомендовано к изданию учебно-методическом советом Павлодарского государственного университета им. С. Торайгырова

Рецензенты:

С. И. Деревягин – кандидат технических наук, ассоц. профессор (доцент);

Д. И. Исмоилов – доктор физико-математических наук, профессор Инновационного Евразийского университета.

Составитель: Саринова А. Ж.

П30 Алгоритмизация и основы программирования : учебно-методическое пособие / состав. А. Ж. Саринова. – Павлодар : Кереку, 2016. – 157 с.

В учебно-методическом пособии изложены основы алгоритмизации и программирования на языке C++ в программной среде Visual Studio. Материал проиллюстрирован большим числом примеров и задач. Также составлены тестовые задания с ответами.

Учебно-методическое пособие рекомендуется студентам технических и инженерных специальностей.

УДК 004.43(075.8)
32.973 - 018я7

Введение

Целью данного учебного методического пособия является дать первое знакомство с языком программирования C++ и развить навыки разработки первых приложений в среде программирования Visual Studio C++.

Пособие предназначено для студентов первых и вторых курса технических и инженерных специальностей, делающих первые шаги в программировании. Оно может быть полезно всем, кто желает приобрести практические навыки программирования.

Первая часть пособия знакомит с основными понятиями этапа алгоритмизации вычислительного процесса, здесь же строится теоретическая база, необходимая для изложения материала.

Во второй части дается формальное изложение всех конструкций языка программирования C++. Материал проиллюстрирован примерами и задачами.

Приводится большое число задач на программирование, решения которых изложены достаточно подробно и завершены построением текста итоговой программы.

Все программы, приведенные в данном пособии, отлаживались в программной среде Microsoft Visual Studio C++ 2015.

1 Основы алгоритмизации и программирования

1.1 Этапы подготовки и решения задач программирования

На ЭВМ могут решаться задачи различного характера: научно-инженерные; разработки системного программного обеспечения; обучения; управления производственными процессами и т. д. В процессе подготовки и решения на ЭВМ научно - инженерных задач можно выделить следующие этапы:

- содержательная постановка задачи;
- математическая постановка (формализация) задачи;
- выбор и обоснование метода решения;
- алгоритмизация вычислительного процесса;
- составление программы;
- отладка программы;
- решение задачи на ЭВМ и анализ результатов.

В задачах разных классов некоторые этапы могут отсутствовать. Например, в задачах разработки системного программного обеспечения отсутствует математическая постановка задачи. Перечисленные этапы связаны друг с другом. Например, анализ результатов может показать необходимость внесения изменений в программу, алгоритм или даже в постановку задачи. Для уменьшения числа подобных изменений необходимо на каждом этапе по возможности учитывать требования, предъявляемые последующими этапами. В некоторых случаях связь между различными этапами, например, между постановкой задачи и выбором метода решения, между составлением алгоритма и программированием, может быть настолько тесной, что разделение их становится затруднительным.

Содержательная постановка задачи. На данном этапе формулируется цель решения задачи и подробно описывается ее содержание. Анализируются характер и сущность всех величин, используемых в задаче, и определяются условия, при которых она решается. Корректность постановки задачи является важным моментом, так как от нее в значительной степени зависят другие этапы.

Математическое описание задачи. Настоящий этап характеризуется математической формализацией задачи, при которой существующие соотношения между величинами, определяющими результат, выражаются посредством математических формул. Так формируется математическая модель процесса с определенной точностью, допущениями и ограничениями. При этом в зависимости

от специфики решаемой задачи используются различные разделы математики и других дисциплин.

Математическая модель должна удовлетворять, по крайней мере, двум требованиям: реалистичности и реализуемости. Под реалистичностью понимается правильное отражение моделью наиболее существенных черт исследуемого явления.

Реализуемость достигается разумной абстракцией, отвлечением от второстепенных деталей, чтобы свести задачу к проблеме с известным решением. Условием реализуемости является возможность практического выполнения необходимых вычислений за отведенное время при доступных затратах требуемых ресурсов.

Выбор и обоснование метода решения. Модель с учетом ее особенностей должна быть доведена до реализации при помощи конкретных методов и алгоритмов решения. Само по себе математическое описание задачи в большинстве случаев трудно перевести на язык машины. Выбор и использование метода решения задачи позволяет привести процесс решения задачи к конкретным машинным операциям. При обосновании выбора метода необходимо учитывать различные факторы и условия, в том числе точность вычислений, время решения задачи на ЭВМ, требуемый объем памяти и другие.

Одну и ту же задачу можно решить различными методами, при этом в рамках каждого метода можно составить различные алгоритмы.

Алгоритмизация вычислительного процесса. На данном этапе составляется алгоритм решения задачи согласно действиям, задаваемым выбранным методом решения. Процесс обработки данных разбивается на отдельные относительно самостоятельные блоки, и устанавливается последовательность их выполнения. Разрабатывается блок-схема алгоритма.

Составление программы. При составлении программы алгоритм решения задачи переводится на конкретный язык программирования. Для программирования обычно используются языки высокого уровня, поэтому составленная программа требует перевода ее на машинный язык. После такого перевода выполняется уже соответствующий машинный код.

Отладка и тестирование программы. Отладка заключается в поиске и устранении синтаксических и логических ошибок в программе.

В ходе синтаксического контроля программы транслятором выявляются конструкции и сочетания символов, недопустимые с

точки зрения правил их построения или написания, принятых в данном языке. Сообщения об ошибках ЭВМ выдает программисту, при этом вид и форма выдачи подобных сообщений зависят от вида языка и версии используемого транслятора.

После устранения синтаксических ошибок проверяется логика работы программы в процессе ее выполнения с конкретными исходными данными. Для этого используются специальные методы, например, в программе выбираются контрольные точки, для которых вручную рассчитываются промежуточные результаты. Эти результаты сверяются со значениями, получаемыми ЭВМ в данных точках при выполнении отлаживаемой программы. Кроме того, для поиска ошибок могут быть использованы отладчики, выполняющие специальные действия на этапе отладки, например, удаление, замена или вставка отдельных операторов или целых фрагментов программы, вывод или изменение значений заданных переменных.

Решение задачи на ЭВМ и анализ результатов. После отладки программы ее можно использовать для решения прикладной задачи. При этом обычно выполняется многократное решение задачи на ЭВМ для различных наборов исходных данных. Получаемые результаты интерпретируются и анализируются специалистом или пользователем, поставившим задачу.

Разработанная программа длительного использования устанавливается на ЭВМ, как правило, в виде готовой к выполнению машинной программы. К ней прилагается документация, включающая инструкцию для пользователя.

Чаще всего, при установке программы на диск для ее последующего использования помимо файлов с исполняемым кодом устанавливаются различные вспомогательные программы (утилиты, справочники, настройщики и т. д.), а также необходимые для работы программы разного рода файлы с текстовой, графической, звуковой и другой информацией.

Компиляция и интерпретация программ. ЭВМ непосредственно выполняет программы на машинном языке данной ЭВМ. При этом программа представляет собой последовательность отдельных команд компьютера. Эти команды являются достаточно «простыми», например, сложение, умножение, сравнение или пересылка отдельных данных. Каждая команда содержит в себе сведения о том, какая операция должна быть выполнена (код операции), с какими операндами выполняются вычисления (адреса данных или непосредственно сами данные) и куда (адрес) должен быть помещен результат.

Машинные языки были первыми языками программирования. Программирование на них затруднительно ввиду того, что, во-первых, эти языки различны для каждого типа ЭВМ, во-вторых, являются трудоемкими для большинства пользователей по причине необходимости знания особенностей конкретной ЭВМ и большого количества реализуемых ею операций (команд). Данные языки обычно используются для разработки системных программ, при этом чаще всего применяются специальные символические языки – Ассемблеры, близкие к соответствующим машинным языкам.

Человеку свойственно формулировать и решать задачи в выражениях более общего характера, чем команды ЭВМ. Поэтому с развитием программирования появились языки, ориентированные на более высокий уровень абстракции при описании решаемой на ЭВМ задачи. Эти языки получили название языков высокого уровня. Их теоретическую основу составляют алгоритмические языки, например, Паскаль, Си, Бейсик, Фортран, PL/1.

Для перевода программы, написанной на языке высокого уровня, в соответствующую машинную программу используются языковые процессоры. Различают два вида языковых процессоров: интерпретаторы и трансляторы.

Интерпретатор – это программа, которая получает исходную программу и по мере распознавания конструкций входного языка реализует действия, описываемые этими конструкциями.

Транслятор – это программа, которая принимает исходную программу и порождает на своем выходе программу уже на другом языке программирования (например, объектную программу). В частном случае, объектным может оказаться машинный язык, и в этом случае полученную на выходе транслятора программу можно сразу же выполнить на ЭВМ. В общем случае объектный язык необязательно должен быть машинным или близким к нему (автокодом). В качестве объектного языка может служить и некоторый промежуточный язык.

Для промежуточного языка может быть использован другой транслятор или интерпретатор – с промежуточного языка на машинный.

Транслятор, использующий в качестве входного язык близкий к машинному (автокод или язык Ассемблера), традиционно называют Ассемблером. Транслятор с языка высокого уровня называют компилятором.

1.2 Стили программирования

Одним из важнейших признаков классификации языков программирования является принадлежность их к одному из стилей, основными из которых являются следующие: процедурный, функциональный, логический и объектно-ориентированный.

Процедурное программирование. Процедурное (императивное) программирование является отражением архитектуры традиционных ЭВМ, которая была предложена фон Нейманом в 40-х годах. Теоретической моделью процедурного программирования служит алгоритмическая система под названием «машина Тьюринга».

Программа на процедурном языке программирования состоит из последовательности операторов (инструкций), задающих процедуру решения задачи. Основным является оператор присваивания, служащий для изменения содержимого областей памяти. Концепция памяти как хранилища значений, содержимое которого может обновляться операторами программы, является фундаментальной в императивном программировании.

Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты. Таким образом, с точки зрения программиста имеются программа и память, причем первая последовательно обновляет содержимое последней.

Процедурные языки характеризуются следующими особенностями:

- необходимостью явного управления памятью, в частности описанием переменных;
- малой пригодностью для символьных вычислений;
- отсутствием строгой математической основы;
- высокой эффективностью реализации на традиционных ЭВМ.

Одним из важнейших классификационных признаков процедурного языка является его уровень. Уровень языка программирования определяется семантической (смысловой) емкостью его конструкций и степенью его ориентации на программиста. Язык программирования частично ликвидирует разрыв между методами решения различного рода задач человеком и вычислительной машиной. Чем более язык ориентирован на человека, тем выше его уровень. Дадим краткую характеристику реализованным на ПЭВМ языкам программирования в порядке возрастания их уровня.

Двоичный язык является непосредственно машинным языком. В настоящее время такие языки программистами практически не применяются.

Язык Ассемблера – это язык, предназначенный для представления в удобочитаемой символической форме программ, записанных на машинном языке. Он позволяет программисту пользоваться мнемоническими кодами операций, присваивать удобные имена ячейкам и областям памяти, а также задавать наиболее удобные схемы адресации.

Язык Макроассемблера является расширением языка Ассемблера путем включения в него макросредств. С их помощью в программе можно описывать последовательности инструкций с параметрами – макроопределения. После этого программист может использовать снабженные аргументами макрокоманды, которые в процессе ассемблирования программы автоматически замещаются макрорасширениями. Макрорасширение представляет собой макроопределение с подставленными вместо параметров аргументами.

Языки Ассемблера и Макроассемблера применяются системными программистами-профессионалами с целью использования всех возможностей оборудования ЭВМ и получения эффективной по времени выполнения и по требуемому объему памяти программы. На этих языках обычно разрабатываются относительно небольшие программы, входящие в состав системного программного обеспечения: драйверы, утилиты и другие.

Язык программирования С (Си) первоначально был разработан для реализации операционной системы UNIX в начале 70-х годов. В последующем приобрел высокую популярность среди системных и прикладных программистов. В настоящее время этот язык реализован для большинства ЭВМ.

В С сочетаются достоинства современных высокоуровневых языков в части управляющих конструкций и структур данных с возможностями доступа к аппаратным средствам ЭВМ на уровне, который обычно ассоциируется с языком низкого уровня типа языка Ассемблера. Язык С имеет синтаксис, обеспечивающий краткость программы, а компиляторы способны генерировать эффективный объектный код.

Одна из наиболее существенных особенностей С состоит в нивелировании различий между выражениями и операторами, что приближает его к функциональным языкам. В частности, выражение может обладать побочным эффектом присваивания, а также может

использоваться в качестве оператора. Нет также четкой границы между процедурами и функциями, более того, понятие процедуры не вводится вообще.

Синтаксис языка затрудняет программирование и восприятие составленных программ. Отсутствует и строгая типизация данных, что предоставляет дополнительные возможности программисту, но не способствует написанию надежных программ.

Basic (Бэйсик) (Beginners All-purpose Symbolic Instruction Code) – многоцелевой язык символических инструкций для начинающих) представляет собой простой язык программирования, разработанный в 1964 году для использования новичками. Он был разработан как простейший язык для непосредственного общения человека с вычислительной машиной. Поэтому первоначально работа велась в интерактивном режиме с использованием интерпретаторов. В настоящее время для этого языка имеются также и компиляторы.

Согласно концепциям, заложенным в Basic, этот язык в смысле вольностей является антиподом языка Pascal. В частности, в нем широко распространены различные правила умолчания, что считается плохим тоном в большинстве языков программирования.

Basic широко распространен на ЭВМ различных типов и очень популярен в среде программистов, особенно начинающих. Существует множество диалектов этого языка, мало совместимых между собой. Basic активно поглощает многие концепции и новинки из других языков. Поэтому он достаточно динамичен, и нельзя однозначно определить его уровень.

Pascal (Паскаль) является одним из самых популярных среди прикладных программистов процедурным языком программирования. Разработанный в 1970 году швейцарским специалистом в области вычислительной техники профессором Н. Виртом на основе Алгола, язык Pascal назван в честь французского математика и по замыслу автора предназначался для обучения программированию. Однако язык получился настолько удачным, что стал одним из основных инструментов прикладных и системных программистов при решении задач вычислительного и информационно-логического характера. В 1979 году был подготовлен проект описания языка — Британский стандарт языка программирования Pascal BS6192, который стал также и международным стандартом ISO 7185.

В языке Pascal реализован ряд концепций, рассматриваемых как основа «дисциплинированного» программирования и заимствованных впоследствии разработчиками многих языков. Одним из существенных признаков языка Pascal является последовательная и

достаточно полная реализация концепции структурного программирования. Причем это осуществляется не только путем упорядочивания связей между фрагментами программы по управлению, но и за счет структуризации данных. Кроме того, в языке реализована концепция определения новых типов данных на основе уже имеющихся.

Функциональное программирование. Сущность функционального (аппликативного) программирования определена А. П. Ершовым как «... способ составления программ, в которых единственным действием является вызов функции, единственным способом расчленения программы на части является введение имени для функции, а единственным правилом композиции — оператор суперпозиции функции. Никаких ячеек памяти, ни операторов присваивания, ни циклов, ни, тем более, блок-схем, ни передачи управления».

Роль основной конструкции в функциональных языках играет выражение: К выражениям относятся скалярные константы, структурированные объекты, функции, тела функций и вызовы функций. Функция трактуется как однозначное отображение из X в X, где X – множество выражений.

Аппликативный язык программирования включает следующие элементы:

- классы констант, которыми могут манипулировать функции;
- набор базовых функций, которые программист может использовать без предварительного объявления и описания;
- правила построения новых функций из базовых;
- правила формирования выражений на основе вызовов функций.

Программа представляет собой совокупность описаний функций и выражений, которые необходимо вычислить. Данное выражение вычисляется посредством редукции, то есть серии упрощений, до тех пор, пока это возможно по следующим правилам: вызовы базовых функций заменяются соответствующими значениями; вызовы не базовых функций заменяются их телами, в которых параметры замещены аргументами.

Функциональное программирование не использует концепцию памяти как хранилища значений переменных. Операторы присваивания отсутствуют, вследствие чего переменные обозначают не области памяти, а объекты программы, что полностью соответствует понятию переменной в математике. В принципе, можно составлять программы и вообще без переменных. Кроме того, нет существенных различий между константами и функциями, то есть

между программами и данными. В результате этого функция может быть значением вызова другой функции и может быть элементом структурированного объекта. Число аргументов при вызове функции не обязательно должно совпадать с числом параметров, указанных при ее описании. Перечисленные свойства характеризуют аппликативные языки как языки программирования очень высокого уровня.

Первым таким языком был LISP (Лисп) (LISt Processing – обработка списков), созданный в 1959 году. Цель его создания состояла в организации обработки символьной информации. Существенная черта этого языка – унификация программных структур и структур данных: все выражения записываются в виде списков.

Логическое программирование. Новую область логическое, или реляционное программирование, – открыло появление языка PROLOG (Пролог) (PROgramming in LOGic – программирование в терминах логики). Этот язык был создан французским ученым А. Кольмероэ в 1973 году. В настоящее время известны и другие языки, однако наиболее развитым и распространенным языком логического программирования является именно Пролог. Имеется свыше 15 различных его реализаций на ПЭВМ. Языки логического программирования, в особенности Пролог, широко используются в системах искусственного интеллекта.

Центральным понятием в логическом программировании является отношение. Программа представляет собой совокупность определений отношений между объектами (в терминах условий или ограничений) и цели (запроса). Процесс выполнения программы трактуется как процесс общезначимости логической формулы, построенной из программы по правилам, установленным семантикой используемого языка. Результат вычисления является побочным продуктом этого процесса. В реляционном программировании нужно только специфицировать факты, на которых алгоритм основывается, а не определять последовательность шагов, которые требуется выполнить. Это свидетельствует о декларативности языка логического программирования. Она метко выражена в формуле Р. Ковальского: «алгоритм = логика + управление». Языки логического программирования характеризуются:

- высоким уровнем;
- строгой ориентацией на символьные вычисления;
- возможностью инверсных вычислений, то есть переменные в процедурах не делятся на входные и выходные;

- возможной логической неполнотой, поскольку зачастую невозможно выразить в программе определенные логические соотношения, а также невозможно получить из программы все правильные выводы.

1.3 Объектно-ориентированное программирование

Прототипом объектно-ориентированного программирования послужил ряд средств, входящих в состав языка SIMULA-67. Но в самостоятельный стиль оно оформилось с появлением языка SMALLTALK, разработанного А. Кеем в 1972 году и первоначально предназначенного для реализации функций машинной графики.

В основе объектно-ориентированного стиля программирования лежит понятие объекта, а суть его выражается формулой: «объект = данные + процедуры». Каждый объект интегрирует в себе некоторую структуру данных и доступные только ему процедуры обработки этих данных, называемые методами. Объединение данных и процедур в одном объекте называется инкапсуляцией и присуще объектно-ориентированному программированию. Для описания объектов служат классы. Класс определяет свойства и методы объекта, принадлежащего этому классу. Соответственно, любой объект можно определить как экземпляр класса. Программирование рассматриваемого стиля заключается в выборе имеющихся или создании новых объектов и организации взаимодействия между ними.

К наиболее современным объектно-ориентированным языкам программирования относятся C++ и Java.

Язык C++ был разработан в начале 80-х годов Б. Страуструпом, сотрудником лаборатории Bell корпорации AT&T. Им была создана компактная компилирующая система, в которой за основу был взят язык C, дополненный элементами языков BCPL, Simula-67 и Algol-68. К июлю 1983 года появился язык C с классами, а чуть позднее – C++. К 1990 году была выпущена третья версия языка C++, принятая комитетом ANSI в качестве исходного материала для его стандартизации.

В 1990 году сотрудник корпорации Sun Д. Гослинг на основе расширения C++ разработал объектно-ориентированный язык Oak, основным достоинством которого было обеспечение сетевого взаимодействия различных по типу устройств. Новая интегрируемая в Internet версия языка, получила название Java. Java является простым объектно-ориентированным и архитектурно-нейтральным языком интерпретирующего типа, обеспечивающим надежность, безопасность и переносимость, обладающим высокой

производительностью в сочетании с многопоточностью и динамичностью.

Принципиальной разницей между Java и C++ является то, что первый из них является интерпретируемым, а второй – компилируемым. Синтаксис языков практически полностью совпадает.

С точки зрения возможностей собственно объектно-ориентированных средств язык Java обладает рядом преимуществ перед языком C++. Так, язык Java демонстрирует более гибкую и мощную систему инкапсуляции информации. Механизм наследования, реализованный в Java, обязывает к более строгому подходу к программированию, что улучшает надежность и понимаемость кода. Язык же C++ обладает сложной, неадекватной и трудной для понимания системой наследования. Возможности динамического связывания объектов одинаково хорошо представлены в обоих языках, однако, синтаксическая избыточность C++ заставляет и здесь отдать предпочтение языку Java.

В силу своей конструктивности идеи объектно-ориентированного программирования используются во многих универсальных процедурных языках. Так, например, в состав интегрированной системы программирования на языке PASCAL (корпорации Borland International) версии 5.5 входит специальная библиотека объектно-ориентированного программирования Turbo Vision.

В последнее время многие программы, в особенности объектно-ориентированные, реализуются как системы визуального программирования. Отличительной особенностью таких систем является мощная среда разработки программ из готовых «строительных блоков», позволяющая создать интерфейсную часть программного продукта в диалоговом режиме, практически без кодирования программных операций. К числу объектно-ориентированных систем визуального программирования относятся; Visual Basic, Embarcadero Studio XE, C++Builder и MS Visual Studio C++ / C#.

В данном учебно-методическом пособии представлены основные теоретические и практические основы программирования на C++ в среде программирования MS Visual Studio C++.

2 Программирование на языке С++

2.1 Основы алгоритмического языка С++

Пример написания первой программы на С++. Пример простой программы на языке С++, которая запрашивает у пользователя два целочисленных значения переменных a и b, анализирует их и выводит наибольшее число.

```
#include "stdafx.h"
int main()
{
int a,b,max;
printf( "a="); //приглашение ввести значение a
scanf("%d",&a); //ввод значения переменной a
printf("b="); //приглашение ввести значение b
scanf("%d",&b); //ввод значения переменной b
if(a>b) max=a; //если a>b то max=a
else max=b; //иначе max=b
printf( "max =%d",max); //вывод максимального значения
return 0; //выход из функции main
}
```

Номера, которые проставлены в начале каждой строки программы, не являются принадлежностью программы. Они введены только для упрощения ссылок на описание действия тех или иных строк.

Строка 01 подключает (include – включить) к тексту программы, так называемые *заголовочные* (h от header – заголовок) файлы системы. В этих файлах описаны функции, их аргументы и типы этих аргументов. Используя эти описания, компилятор проверяет правильность вызова системных функций. В нашем случае программа использует функции ввода (scanf) и вывода (printf), описания которых находятся в заголовочном файле *stdio.h*.

Названия заголовочных файлов зачастую образовываются от каких-либо аббревиатур английских слов, их полезно научиться понимать, а не запоминать.

В нашем примере: используется модуль со стандартным именем StdAfx, состоящий из двух файлов: заголовка stdafx.h и реализации stdafx.cpp., создаваемый мастером AppWizard для каждого приложения. Этот модуль предназначен для создания так называемых прекомпилированных файлов (*.pch), которые повышают производительность компиляции.

Заголовочный файл stdafx.h включается в каждый исходный файл проекта первым и содержит директивы include для подключения библиотек C++. Таким образом, каждый модуль проекта имеет возможность использовать подключенные библиотеки, при этом во время его компиляции не требуется выполнять обработку файлов заголовков, поэтому, он может быть скомпилирован быстрее.

Строка 02 содержит заголовок функции main. Функция с таким названием обязана присутствовать в каждой программе на языке C, C++. Именно с нее начинается выполнение программы, она – главная (именно так переводится служебное слово main). Предшествующее ей служебное слово int (от integer – целый) сообщает, что результатом работы функции main должно быть целое число. По возвращаемому функцией значению операционная система, запустившая программу main, может "сообразить", правильно или неправильно завершилась работа программы. По общепринятому соглашению нулевое значение, возвращаемое функцией main (строка 12), свидетельствует о нормальном завершении работы программы. Пустота в круглых скобках, указывает, что функция main не использует аргументы.

Текст программы (тело функции) заключается в фигурные скобки (строки 03 и 13). В большинстве последующих строк присутствует пояснительный текст на русском языке, следующий после пары символов // – это комментарий, на содержание которого компилятор не обращает внимания.

В строке 04 объявлены три переменные с именами a, b и max, которые могут принимать только целочисленные значения (тип – int).

Строка 05 является первой строкой программы, которая производит некоторое действие – она выводит на дисплей сообщение, состоящее из двух символов (a=). Текст сообщения заключен в двойные кавычки. Строка 06 организует останов работы программы до тех пор, пока пользователь не наберет на клавиатуре какое-либо число и нажмет клавишу Enter. Поступившее значение будет интерпретировано, если оно целое, и помещено в переменную a. Точно таким же образом в строках 07 и 08 будет организован ввод значения числовой переменной b.

В строке 09 сравниваются (if – если) текущие значения переменных a и b. Если проверяемое условие выполнено, т.е. значение переменной a больше, то оно присваивается переменной max – выполняется действие, записанное после проверки условия. В противном случае (else – иначе) в переменную max заносится значение b.

Строка 11 выводит на дисплей два сообщения – текстовое (max=) и числовое (значение переменной max).

Последняя выполняемая строка с номером 12 возвращает управление операционной системе (return – вернуться) и выдает в качестве значения функции нулевой результат.

Обратите внимание на следующие детали. Если программа обращается к каким-либо системным функциям, то в первых ее строках обязательно должно стоять указание о подключении соответствующих заголовочных файлов. Программа может содержать более чем одну функцию, но среди них обязательно должна присутствовать функция с именем main. Каждая строка программы, содержащая какое-либо объявление или выполняемое действие, оканчивается точкой с запятой. Тело функции обязательно заключается в фигурные скобки.

Структура основной программы. Такая программа состоит всего из 12 символов, но заслуживает внимательного рассмотрения.

```
void main()
{
}
```

Основная программа в C++ всегда называется именем main (будьте внимательны – C++ различает большие и маленькие буквы, а все стандартные операторы C++ записываются маленькими буквами). Пустые скобки означают, что main не использует аргументов, а слово void (пустой) говорит о том, что она также и не возвращает никакого значения, то есть, является процедурой.

Фигурные скобки обозначают начало и конец процедуры main - поскольку внутри них ничего нет, наша программа ничего не делает, она просто соответствует правилам языка C++, ее можно скомпилировать и получить исполняемый файл. Составим теперь программу, которая делает что-нибудь полезное, например, выводит на экран слово «Привет».

#include "stdafx.h" /*подключение заголовочного файла stdio.h, в котором находится описание стандартных функций ввода и вывода, */

```
int main()
{
printf("Привет"); // вызов функции вывода на экран
return 0;
}
```

Перечислим новые элементы, использованные в этой программе.

Для вывода информации на экран используется функция printf. В простейшем случае она принимает единственный аргумент - строку в кавычках, которую надо вывести на экран.

Чтобы использовать стандартные функции, необходимо сказать компилятору, что есть функция с таким именем и перечислить тип ее аргументов - тогда он сможет определить, верно ли мы ее используем. Это значит, что надо подключить *описание* этой функции.

Описания стандартных функций C++ находятся в так называемых *заголовочных файлах* с расширением ***.h**

Для подключения заголовочных файлов используется директива (команда) *препроцессора* "*#include*", после которой в угловых скобках или кавычках ставится имя файла с заголовками. Все команды препроцессора начинаются символом "#" в начале строки. Для подключения каждого нового заголовочного файла надо использовать новую команду "*#include*".

В программу могут быть включены комментарии - тексты, поясняющие программу. Комментарии – не обязательная принадлежность программы. Описание нетривиального алгоритма, как правило, снабжается пояснениями, которые помогут разобраться в тексте программы персоналу сопровождения программных продуктов. В языке C++ предусматривается две разновидности комментариев – *многострочные* и *однострочные*.

Однострочный комментарий начинается вслед за парой символов "//" и продолжается до конца программной строки. Обычно с его помощью записывается комментарий к текущей строке или исключается фрагмент текущей строки из области обслуживания. Комментарии, заключенные в /* */ и записываются в любом месте программы и могут располагаться на нескольких строках.

2.2 Синтаксис языка C++

При написании программ на алгоритмическом языке можно пользоваться лишь символами, предусмотренными алфавитом этого языка.

Алфавит языка C++ включает в себя следующие символы.

Строчные и прописные буквы латинского алфавита

|a|b|c| ... |x|y|z|A|B|C| ... |X|Y|Z|

Арабские цифры

|0|1 |2| ...|9|

Специальные символы

|+|-|*|/| =|>|>=|<|<=|==|!|.|,|:|;|'|(|)|[|]|{| |}|&|@|$|#|

Язык С++ также включается набор зарезервированных слов, имеющих строго определенное назначение. Например, слова IF, ELSE используются в условном операторе, слово DEFAULT обозначает выбор по умолчанию. Назначение зарезервированных слов будет пояснятся в процессе изучения языка.

Константы и переменные. Константами являются программные элементы, имеющие определенный тип и не меняющие своего значения при выполнении программы. В качестве констант на языке С++ используются целые и вещественные числа, логические значения, символы и строки.

Целые числа записываются со знаком или без него по обычным арифметическим правилам.

Например: 15 +1000 -47 02

Вещественные числа могут записываться либо в форме десятичного числа, либо в экспоненциальной форме.

В записи десятичного числа целая часть отделяется от дробной части точкой.

Например: 2.5 -14.0 +0.33 0.0

Числа в экспоненциальной форме имеют вид: $a\ 10^p$, где a является мантиссой; p - десятичным порядком.

На языке С++ числа с порядком записываются с использованием буквы E, за которой следует порядок. Буква E читается как «умножить на 10 в степени».

Мантисса может быть целым или десятичным числом. Порядок всегда является целым числом. Следует помнить, что в написании вещественного числа с порядком должны обязательно присутствовать мантисса и порядок, табл.1.

Таблица 1 – Примеры записи чисел с порядком

Математическая запись	Запись на языке С++
$3{,}14\ 10^5$	3.14E5
$-17\ 10^{-2}$	-17E-2
25,625	2.5625E1
10^{-6}	1E-6
0,00048	4.8E-4

Символьная константа - это любой символ ПК, заключенный в апострофы, например: ‘7’ или ‘S’ или ‘*’.

Строковая константа (строка) - любая последовательность символов из набора символов ПК, заключенная в кавычки, например:

"X="

"Максимальное значение ="

Строки используются также при выводе текстов и оформлении комментариев.

Переменные. Программа оперирует информацией, представленной в виде различных объектов и величин. Переменная – это символическое обозначение величины в программе. Как ясно из названия, значение переменной (или величина, которую она обозначает) во время выполнения программы может изменяться.

С точки зрения архитектуры компьютера, переменная – это символическое обозначение ячейки оперативной памяти программы, в которой хранятся данные. Содержимое этой ячейки – это текущее значение переменной.

В языке С++ прежде чем использовать переменную, ее необходимо объявить.

В объявлении переменной первым стоит название типа переменной, а затем идентификатор или имя переменной. Тип переменной определяет, какие возможные значения эта переменная может принимать и какие операции можно выполнять над данной переменной. Тип переменной изменить нельзя, т.е. пока переменная существует, она всегда будет описанного типа.

Имя представляет собой последовательность латинских букв и цифр, которая начинается с буквы. В имени может присутствовать символ подчеркивания. Длина имени составляет от 1 до 63 символов. Следует помнить, что пробелы не должны входить в написание имени.

Таблица 2 – Примеры записи имен переменных

Математическая запись	x	a_p	y^1	α	$\sum$	*d-21*
Запись на языке С++	x	ap	y1	alpha	S	d_27

С точки зрения компьютера все данные в памяти – это числа (более точно – наборы нулей и единиц). Тем не менее, и вы (и компьютер) знаете, что с целыми и дробными числами работают по-разному. Поэтому в каждом языке программирования есть разные типы данных (переменных), для обработки которых используются разные методы. Основными данными в языке С++ являются - целые переменные (тип int - от английского *integer* - целый) занимают 2 байта в памяти; вещественные переменные, которые могут иметь

дробную часть (тип float – от английского *floating point* - плавающая точка), занимают 4 байта в памяти;
-символы (тип char - от английского *character* - символ) занимают 1 байт в памяти.
Типы задаются стандартными зарезервированными словами:

- int - целый тип;
- long – длинный целый тип;
- short – целый тип с меньшим диапазоном;
- float - вещественный тип;
- double - вещественный тип с двойной точностью;
- char - символьный тип;

Для использования все переменные необходимо объявлять - то есть сказать компьютеру, чтобы он выделил для них ячейку памяти нужного размера и присвоил ей нужное имя. Переменные обычно объявляются в начале программы. Для объявления надо написать название типа переменных (int, float или char и др.), а затем через запятую имена всех объявляемых этим типом переменных. При желании можно сразу записать в новую ячейку нужное число, как показано в примерах ниже. Если переменной не присваивается никакого значения, то в ней находится "мусор", то есть то, что было там раньше.

По описанию переменной в памяти компьютера резервируется ячейка для хранения ее значения. В зависимости от объявленного *типа* переменной ячейка может иметь разную внутреннюю структуру, т.е. содержать различное число байт.

Примеры:

int a; // выделить память под целую переменную ***a***
float b, c; // две вещественных переменных ***b*** *и* ***c***
int Tu104, Dl86=23, Yak42; // три целых переменных,
//причем в ***D186*** *сразу записывается число 23.*
float x=4.56, y, z; // три вещественных переменных,
// причем в ***x*** *сразу записывается число 4.56.*
char c, c2='A', m; // три символьных переменных,
//причем в ***c2*** *сразу записывается символ 'A'.*

Арифметические выражения. Арифметические выражения строятся из операндов, арифметических операций и круглых скобок. Операндами могут быть константы, переменные и функции.

В бесскобочных арифметических выражениях операции выполняются слева направо в соответствии с их приоритетом.

1. * (умножение); / (деление); % (остаток от деления целых чисел).

2. + (сложение); - (вычитание).

Изменить порядок выполнения операций можно с помощью *круглых* скобок. Выражение, заключенное в круглые скобки, выполняется в первую очередь. Например, выражению: a/b*c соответствует математическая запись: $\frac{a}{b}c$, а выражению a/(b*c) – запись $\frac{a}{bc}$.

Тип арифметического выражения определяется типом входящих и него операндов.

Арифметическое выражение является целым, если все входящие и него операнды целого типа.

Если в арифметическом выражении содержится хотя бы один вещественный операнд, то результат - вещественный. Целые операнды в вещественном арифметическом выражении всегда преобразуются к вещественному типу.

Операция выделения остатка или деление по модулю (%) применима только к целым числам. Результат ее выполнения имеет целый тип.

Исключение составляет операция деления с использованием символа '/' (косая черта). Результат выполнения этой операции всегда зависит от типа операндов.

Например, значением выражения 2/5 будет число 0.

Таблица 2 – Примеры вычисления арифметических выражений

Арифметические выражения	Результат	Тип результата
6 + 4 * (5 - 3)	14	Целый
6 + 4 * (5 - 3.0)	14.0	Вещественный
7 % 2	1	Целый
7/2.0	3.5	Вещественный

Стандартные функции. Часто используемые в арифметических выражениях математические функции оформлены в виде стандартных подпрограмм, которые хранятся в библиотеке системы C++ (файл с именем math.h). Для правильного обращения к стандартной функции необходимо записать имя функции, за которым в круглых скобках следует аргумент (параметр). Приоритет вычисления функций выше, чем приоритет арифметических операций.

В таблице ниже представлен набор стандартных функций с указанием типов функции и аргумента. В таблице приняты обозначения: I - целый тип, F - вещественный тип.

Таблица 3 – Примеры вычисления арифметических выражений

Математическая запись	Запись на языке C++	Тип аргумента	Тип функции
$\|x\|$	abs(x) fabs(x)	I F	I F
tg x	tan(x)	I или F	F
sin x	sin(x)	I или F	F
cos x	cos(x)	I или F	F
ln x	log(x)	I или F	F
$\sqrt{x}$	sqrt(x)	I или F	F
arctg(x) - арктангенс	atan(x)	I или F	F
arcctg(x)- арккотангенс $arcctg(x)=\frac{\pi}{2}-arctg(x)$	3.14/2.0-atan(x)	F	F
Округление до ближайшего целого > x	ccil(x)	F	I
Округление до ближайшего целого < x	floor(x)	F	I
Возведение в степень x^n	pow(x,n)	I F	I F
e^x	exp(x)	I или F	F
arcsin(x) - арксинус	asin(x)	I или F	F
arccos(x) - арккосинус	acos(x)	I или F	F
$\log_{10}(x)$ десятичный логарифм.	log10(x)	I или F	F
$sh(x)=\frac{e^x-e^{-x}}{2}$ гиперболический синус	sinh(x)	I или F	F
$ch(x)=\frac{e^x+e^{-x}}{2}$ гиперболический косинус	cosh(x)	I или F	F

Таблица 4 – Примеры программирования арифметических выражений

Математическая рапись	Запись на языке C++
$\frac{a+12b}{c_1-1.8\cdot10^3}$	(a+12*b)/(c1 - 1.8E3)
$e^{\frac{2\sin 4x+\cos^2 x}{3x}}$	exp((2*sin(4*x)+pow((cos(x* x),2)))/(3*x))
$\ln\|tg\alpha-\sin\alpha^2\|$	log(fabs(tan(al)-sin(pow(al,3))))

Задания для самостоятельного решения

1. Составить описания для заданных переменных

Таблица 5 – Примеры вариантов математических записей

Вариант	Математическая запись	Тип	Вариант	Математическая запись	Тип
1	$C_2, Lt, j,$ Z_1, T, rs	Целый Символьный	9	Sum, max, w T, Z_1, R	Вещественный Целый
2	$M, n_1, a_p,$ B, γ_0, η	Целый Вещественный	10	A C_3, k_b, β_1 X_1, Σ	Символьный Вещественный
3	$\Delta, h, E,$ st, w, v_1	Вещественный Символьный	11	$I, d_0, K_H Ch,$ Q, rez	Целый Символьный
4	T3, j, x, φ, w, Z_1	Целый Вещественный	12	N, Kc, a_2 Z, fd, η	Целый Символьный
5	N, x_1, b, Λ, η, min	Целый Вещественный	13	I, kd, l_1 ϑ, v, T_n	Символьный Вещественный
6	$J, I, K,$ F, S, R_2	Целый Символьный	14	$M_1, j, r,$ b_0, A, σ	Символьный ВещественныйI
7	$\beta, y_4, \Omega,$ n, a_0, k	Вещественный Символьный	15	$Mo, j, r_{cm} E,$ £, f_1	Целый Вещественный
8	$\lambda, \sigma, \delta,$ l, k_1, m	Вещественный Символьный	16	α, μ, c str, P_1, φ	Вещественный Символьный

2. Записать на языке С++ числовые константы
а)целые;
б)вещественные (в форме десятичных чисел и чисел с порядком).

Таблица 6 – Варианты заданий

Вариант	Математическая запись
1	а)+15; 72,93 10^3; -500 б) $\frac{1}{8}$; -16,7; 0.054 10^2; 62,7 10^{-5}
2	а) 02; 32 10^3; -8 10^2 б) -0.25; 2.89 10^2; 3,164; 23.7 10^{-4}
3	а) 600; -041; 52.625 10^4 б) $\frac{2}{3}$; 0.03 10^{-2}; -715.4; 2,1 10
4	а)735; -24 10^3; 13.14 10^2 б) 0.0265 10^3; -3,7; 54.6; 0.275 10^{-2}
5	а)014; 25 10^0; 27,3 10^3 б) 0.518; 2.546 10^2; -0.07 10^{-3}; 3,04
6	а)01; 25 10; 2,73 10^2 б) 15.37; 4.52 10^{-3}; 23; 0.0083
7	а)07; 94 10^2; 16.27 10^3 б) 8.4; -35.07 10^{-1}; 6.3 10^2; 0.004
8	а) 41; 286 10^0; 23.7 10^2 б) 0.817; 0.645 10^2; -0.718 10^3; 0.3556 10^0
9	а)377; -15 10^3; 25.4 10^3 б) -0.572 10^6; 6.28; 5886; 0.0695 10^2
10	а)012; 7,5 10^3; -0,05 10^2 б) -0.084 10^{-2}; 7.12; -52; 0,0085

Продолжение таблицы 6

1	2
11	а)0015; -9,4 10^3; 99,0 б)-5,14; 27,06 10^{-6}; $\frac{1}{4}$; 0,008 10^2
12	а) 276; 9,275 10^3; -02 б) -0,56 10^0; -0,07425; 6; -0,873
13	а) 03; 52 10^0; -1,0 10^3 б) $\frac{1}{5}$; 631,5 10^2; -8,25; 92,1 10^4
14	а)-016; 95,72 10^{-2}; -576 б)-2,75; 81,2 10^{-6}; -15,0 10^3; 0,033 10^{-3}
15	а) 05; -91,0 10^3; 14,5 10^2 б) 3,48; 1,5 10^2; -912 10^{-5}; 0,0385
16	а) 008; 7,2 10^4; -12 10^2 б)54; 17,8; 2,041 10^{-3}; 513 10^1

3. Записать на языке C++ арифметические выражения

Таблица 7 – Варианты математических записей

Вариант	Математическая запись	Вариант	Математическая запись
1	$\frac{3\cos^2(x-\frac{\pi}{6})}{0.5+\sin y^2}$	12	$\frac{2}{\sqrt{x}}\frac{x-\lambda}{\sqrt[3]{(x-\lambda)^2+\lambda^2}}$
2	$\frac{\lvert x-y\rvert}{(1+2x)^2}-e^{\sqrt{1+y}}$	13	$e^{x\cos\frac{\pi}{4}}\cos(x\sin\frac{\pi}{4})$
3	$\frac{5a^{10}}{b+c}-\sqrt{\lvert\cos x^3\rvert}$	14	$\sqrt{\lvert\ln\left(1-2x\cos\frac{\pi}{3}\right)+x^2\rvert}$
4	$\ln\lvert a^7\rvert+arctgx^2+\frac{\pi}{\sqrt{\lvert a+x\rvert}}$	15	$\sqrt{a_0+a_1x+a_2\sqrt[3]{\lvert\sin x\rvert}}$
5	$\sqrt[4]{\frac{(a+b)^2}{c+d}}+e^{\sqrt{x+1}}$	16	$\frac{tgx}{x}+\sqrt[3]{\sin x^3}$
6	$\frac{1}{2\pi}\sqrt{\frac{2mg}{m(a\sin\alpha+b\cos\alpha)}}$		
7	$\frac{1}{4}\left(\frac{1+x^2}{1-x}+\frac{1}{2}tgx\right)$		
8	$a+\sqrt{b}+\ln\lvert x^3\rvert+e^{\sqrt[3]{c}}$		
9	$\lvert\sin\alpha\rvert\cos\frac{\alpha}{2}+\sqrt{a^2+b^2}$		
10	$\sqrt[3]{x}(arctgz+\cos^2 y)$		
11	$\sqrt[4]{y+\sqrt[3]{x-y}}$		

4. Записать в виде алгебраической формулы

Таблица 8 – Варианты заданий

Вариант	Запись на языке С++
1	log(abs(x*x*x))+atan(x)+exp(omega)
2	x*sin(3.14/4.0)/(1-2*x*cos(3.14/4.0)+exp(1/3*log(x)))
3	sqrt(fabs(log(sin(x)+2)))
4	3*sin(x)+1/3*log(1+x*x)+sqrt(exp(x))
5	sqrt(abs(x-y))/(pow(sin(z),2)+1)
6	exp(abs(x+y))*exp(x*log(1 +sin(y)))
7	2*((x+y+z)*pow((x+y+z),2)/(2*x)-sin(x*x*x))
8	(sin(al)+atan(al))/(omega+pow(cos(al),2))
9	exp(cos(x))*pow(cos(sin(x)),2)+5.7E-1
10	exp(x)/(4E-1 +x*x)-sin(log(x))
11	4*a*a+x*(b*b*b+x*(pow(c,4)+x*(a-exp(x))))
12	18*a*x*x+exp(2/3*log(x))+sin(al/2)
13	2*b*b*b+log(a)-exp(2.5*log(x))
14	(x+y+z)/(pow(sin(a*x),2)+b*b)
15	exp(y/x*log(abs(x)))+exp(1/3*log(y/x))
16	atan(x*x*x)/(log(x)+sin(x)/cos(x))+1

5. Вычислить арифметическое выражение

Таблица 9 – Варианты заданий

Вариант	Запись на языке С++	Исходные данные
1	18.75 - 16.4E0+1/4	-
2	floor(sqrt(a+b))	a = 0,84; b = 0,16
3	3/x+x*x	x = 2
4	(x*x*t+2.1)*ceil(1/4)+5	x = 36; t =5,287
5	i+3-i*pow(i,2)	i = -3
6	4+exp(ceil(3/2)*log(6.25E-1))	-
7	1%3+1/3	-
8	2*10/4	. -
9	x/(y*x-y)+x/y+1.2E1	x = 3; y= 1,5
10	19/4+1%4	-
11	100*(99/100)	-
12	m+k*((m+k)/n)-n	M=2; k=7; n= 10
13	2*floor(4/10)+2E0	-
14	2.0*(1E5)	-
15	19/4+5%4	-
16	exp(1/3*log(27))+2	-

2.3 Линейные вычислительные процессы

Линейный вычислительный процесс представляет собой последовательность действий, выполняемых одно за другим. Основу программы линейного вычислительного процесса составляют оператор присваивания и операторы ввода-вывода данных.

Оператор присваивания. Оператор присваивания служит для вычисления выражения и записи результата в память компьютера.

Общий вид записи оператора:

переменная=выражение;

Знак = читается как «присвоить». Конец любого оператора на языке С++ фиксируется точкой с запятой.

При выполнении оператора присваивания происходит замена текущего значения переменной, стоящей слева от знака присваивания, новым значением, полученным в результате вычисления выражения, стоящего справа от знака равенства, например

c = a + b; // вычислит сумму **a** и **b** и запишет результат в **c.**

Тип результата выполнения оператора. При записи оператора присваивания необходимо строго следить за типом переменной, стоящей в левой части оператора, и типом выражения:

- если переменная слева - вещественного типа, то арифметическое выражение может быть как целым, так и вещественным.

В случае целого арифметического выражения происходит преобразование его результата к вещественному типу. Например, при выполнении оператора

m=20+15/4.0;

вещественной переменной **т** будет присвоено значение 23.0;

- если переменная слева - целого типа, то арифметическое выражение будет приведено к целому типу. Например, задано следующее описание типа переменных.

float a, b, c;

int n;

Тогда записать оператор присваивания в виде: n=a*b/c; приведет к усечению результата до целого типа.

В операторе присваивания конечный результат вычисления выражения в правой части приводится к типу переменной, которой должно быть присвоено это значение. Данный процесс может привести к понижению типа, как описано выше, или к повышению, при котором величина приводится к типу данных, имеющему более высокий приоритет. Последовательность имен типов, упорядоченных от высшего типа к низшему типу, выглядит так: double, float, long, int,

short, char. Применение ключевого слова unsigned повышает ранг соответствующего типа данных со знаком. Повышение типа обычно происходит гладко, в то время как понижение может привести к затруднениям (скрытым ошибкам). Запомним, что тип значения выражения в правой части оператора присваивания всегда преобразуется к типу переменной, которой присваивается это значение.

Примеры использования оператора присваивания:

1. Записать оператор присваивания, который позволяет вычислить расстояние между двумя точками на плоскости с координатами (x1 , y1) и (x2, y2).

Ответ: d=sqrt(pow(x1-x2,2)+pow(y1-y2,2));

2. Записать последовательность операторов присваивания, обеспечивающих обмен значениями переменных x и y в памяти компьютера.

Ответ: c=x;

x=y;

y=c;

Странные операторы присваивания. В программировании часто используются несколько странные операторы присваивания, например:

i = i + 1;

Если считать это уравнением, то оно бессмысленно с точки зрения математики. Однако с точки зрения информатики этот оператор служит для увеличения значения переменной **i** на единицу.

Буквально это означает: взять старое значение переменной **i**, прибавить к нему единицу и записать результат в ту же переменную **i**.

Инкремент и декремент. В языке C++ определены специальные операторы быстрого увеличения на единицу (**инкремента**)

i + +; (постфиксная форма) или *+ + i*; (префиксная форма)

что равносильно оператору присваивания

i = i + 1;

и быстрого уменьшения на единицу (**декремента**)

i--; или *--i;*

что равносильно оператору присваивания

i=i-1;

Между первой и второй формами этих операторов есть существенная разница, но только тогда, когда они входят в состав более сложных операторов или условий.

Операторы ввода-вывода. Операторы ввода-вывода данных служат для обмена информацией между внутренней памятью компьютера и внешними устройствами (монитор, клавиатура, принтер, накопители на жестких и гибких магнитных дисках и др.).

Информация, представленная на внешних устройствах, организуется в файлы. Файлы, предназначенные для хранения исходных данных, называются входными. Результаты вычислений (выходные данные) записываются в выходные файлы (они же могут стать потом входными).

Операторы ввода исходных данных с клавиатуры

Работа оператора ввода. Операторы ввода обеспечивают чтение данных с клавиатуры, преобразование их из внешнего представления во внутреннее и присваивание значений переменным, указанным в списке.

Потоковый ввод данных числового типа. Для начинающих программистов проще всего организовать в программе ввод числовой информации из буфера потока, связанного с так называемым стандартным устройством ввода (stdin). Данные в этом буфере оказываются в тот момент, когда программа обращается к пользователю и ждет окончания набора затребованных числовых значений с клавиатуры. Программа может запросить одно или несколько значений:

```
cin >> d;
.....
cin >> x1 >> x2 >> x3;
```

Первая строка соответствует запросу на ввод значения единственной переменной d. Следующая строка программы представляет запрос на ввод трех числовых значений, первое из которых будет присвоено переменной x1, второе значение будет предназначаться для переменной x2, третье значение - для переменной x3. В ответ на запрос программы пользователь должен набрать на клавиатуре затребованные числа, разделяя их, по крайней мере, одним пробелом. Набор данных завершается нажатием клавиши Enter. Количество числовых значений, набираемых пользователем в пределах строки, может оказаться как меньше, так и больше, чем требуется программе. В первом случае продолжение программы задержится до тех пор, пока пользователь не введет дополнительные строки с недостающими значениями. Если пользователь наберет слишком много значений во вводимой строке, то лишние числовые данные сохранятся в буфере ввода и будут переданы программе при выполнении следующего оператора cin.

Тип набираемого числового значения и его величина должны соответствовать типу переменной, в которую это значение должно быть введено. Забота об этом целиком возлагается на программиста.

Для обеспечения потокового ввода к программе следует подключить заголовочный файл iostream и применить директиву using namespace.

using namespace std позволяет использовать имена из пространства имен std.

std - это имя стандартной библиотеки C++.

```
#include <iostream>
using namespace std;
int main()
{ int i;
float f;
double d;
cin >> i >> f >> d;
return 0;
}
```

В переменные типа char или unsigned char из потока числовые значения ввести невозможно, т.к. они в данном случае воспринимаются системой не как числовые данные, а как символьные. В такие переменные попадет только первый символ набранного значения.

Если в последнем примере убрать строку подключения пространства имен using namespace std, то надо будет указывать наименование пространства имен явно:

```
std::cin >> i >> f >> d;
```

Тема «Пространство имен» не входит в круг вопросов, освещаемых в данном пособии. Для самостоятельного изучения данного вопроса рекомендуем воспользоваться источником [3].

Форматный ввод. Гораздо более сложным для начинающих программистов является ввод числовых данных, организуемый с помощью функции scanf, использующей так называемую форматную строку:

Общий вид записи оператора ввода

scanf(" форматная строка ", список адресов переменных);

форматы ввода

адреса вводимых переменных

Форматы ввода - это строка в кавычках, в которой перечислены один или несколько форматов (спецификаторов) ввода данных (% - признак спецификатора):

%d - ввод целого числа (для переменной типа int)

%f - ввод вещественного числа (для переменной типа float)

%c - ввод одного символа (для переменной типа char)

После форматов ввода через запятую перечисляются **адреса** ячеек памяти, в которые надо записать введенные значения. Почувствуйте разницу:

a - значение переменной **a**

&a - адрес переменной **a**

Например:

```
#include "stdafx.h"
 int main()
{ int p;
 float k;
 double m;
.....
 scanf("%d %f %lf",&p,&k,&m);
.....
}
```

Для обеспечения форматного ввода к программе следует подключить заголовочный файл *stdio.h*. Строка вводимых данных поступает со стандартного устройства ввода (stdin), которым по умолчанию считается клавиатура. Завершение набора строки ввода - нажатие клавиши Enter.

Первый аргумент функции *scanf* представляет форматную строку, управляющую процессом преобразования числовых данных, набранных пользователем в строке ввода, в машинный формат, соответствующий типам переменных, адреса которых указаны вслед за форматной строкой.

В приведенном примере переменной p (в списке ввода указан ее адрес - &p), объявленной с помощью спецификатора типа int, соответствует спецификатор %d. Это означает, что первым числовым значением, набранным на клавиатуре, может быть только целое десятичное число со знаком (d - от decimal, десятичный). Вещественной переменной k типа float в форматной строке соответствует формат %f. Это означает, что второе числовое значение в строке ввода должно принадлежать диапазону, предусмотренному для коротких вещественных данных. Для переменной m типа double использован формат %lf (l (это буква эль)- от long).

Количество форматов в строке должно быть равно количеству адресов в списке.

Кроме того, тип переменных должен совпадать с указанным форматом: например, если **a** и **b** - целые переменные, то следующие вызовы функций ошибочны

scanf ("%d %d", &a);

неясно, куда записывать второе введенное число;

scanf ("%d %d", &a, &b, &c);

переменная **c** не будет введена, так как для нее не задан спецификатор;

scanf ("%f %f", &a, &b);

нельзя вводить целые переменные по вещественному формату.

Если значения данных вводятся в одной строке, то они отделяются друг от друга одним или несколькими пробелами.

Пример ввода:

Пусть в задаче определены следующие исходные данные:

a = 4; x = 3,8; y =1,2*10^{4}; к=5; т = 74

Оператор ввода:

scanf("%d %f %f %d %d", &a, &x, &y, &k, &m);

вызывает чтение данных с клавиатуры, набранных следующим образом:

4 3.8 1.2E4 5 74 *[Enter]*

%d %f %f %d %d - такой список форматов будет соответствовать следующему описанию переменных в программе:

```
main()
{  int a,k,m;
float x,y;
```

Операторы вывода данных на экран. Потоковый вывод. Для начинающих программистов организация потокового вывода числовой информации на стандартное устройство вывода экран (stdout) представляется более простой:

```
#include <iostream>
using namespace std;
int main()
{ int i;
float f;
double d;
.....
cout << i + 1 << f << f*d;
.....
}
```

Здесь на первых порах не надо заботиться о форматах выводимых результатов. Для каждого типа данных существуют соответствующие

системные соглашения. Для вывода пробелов между числами на экране достаточно включить в поток символьные строчки, как это показано в следующем примере:

*cout << i+1 << " "<<f << " "<< f*d;*

Переход в начало следующей строки вывода на экране осуществляется путем включения в список вывода признака конца строки endl, например:

*cout << i+1 << endl << f << " "<<f*d;*

В таком случае значения переменных f и f*d будут выведены на новую строку экрана.

Форматный вывод. Форматный вывод числовых результатов на стандартное устройство вывода (*stdout*), которым по умолчанию является экран дисплея, осуществляется с помощью функции printf.

Общий вид записи оператора

printf("список форматов",список имен переменных и выражений);

Например:

printf ("Результат: %d + %d = %d \n", a, b, c);

Содержание скобок при вызове функции printf очень похоже на функцию scanf. Сначала идет символьная (управляющая) строка — форматы вывода, — в которой можно использовать специальные символы:

%d - вывод целого числа

%f - вывод вещественного числа

%c - вывод одного символа

%s - вывод символьной строки

\n - переход в начало новой строки экрана

все остальные символы (кроме некоторых других специальных команд) просто выводятся на экран.

В форматной строке мы сказали компьютеру, какие данные (целые, вещественные или символьные) надо вывести на экран, но не сказали откуда (из каких ячеек памяти) их брать. Поэтому через запятую после форматов вывода надо поставить список чисел или имен переменных, значения которых надо вывести. В список вывода могут входить арифметические выражения, значения которых предварительно будут подсчитаны, а потом выведены в соответствии с использованными форматными указателями. Выражения и имена переменных в списке вывода перечисляются через запятую. В качестве простейших выражений могут быть константы, переменные, символы, строки.

printf ("Результат: %d + %d = %d \n", a, 5, a+5);

Так же, как и для функции scanf, надо следить за совпадением типов форматов и типов переменных, а также количества форматов вывода и переменных.

Пример1:

```
#include "stdafx.h"
int main()
{ int i;
float f;
double d;
.....
printf("%d %f %lf", i+1, f, f*d);
.....
}
```

Пример 2:

Операторы

```
printf("Выходные данные:\n");
printf("K=%d  y=%f", 8+5,  7.0/10);
```

выводят на экран результаты в следующем виде:

Выходные данные:

K=13 y=0.7

Управление выводом данных.

При стандартной форме вывода вещественные числа отображаются на экране с шестью знаками после запятой. Мантисса выводится в нормализованном виде, т.е. с одной значащей цифрой в целой части и десятью цифрами в дробной части. На порядок отводится четыре позиции: первая позиция - под букву Е, вторая позиция - под знак порядка, третья и четвертая позиции - под цифры порядка.

Если программиста не устраивает стандартная форма вывода, он может использовать специальный форматированный вывод, в котором предусмотрены следующие параметры:

1. Ширина поля, определяющая число позиций на экране, отводимых для вывода всего числа, включая целую, дробную части, знак и десятичную точку.

2. Точность представления вещественного числа, определяющая число позиций в дробной части.

Вещественное число с указанием форматов всегда выводится в десятичной форме. Для целого числа используется только формат ширины поля.

В операторах вывода параметры форматов записываются после знака % и отделяются друг от друга точкой. Например, при записи оператора

printf("K=%3d y=%5.2f",8+3,7.0/10);

результаты отображаются на экране в следующем виде:

K= 11 y = 0.70

В приведенном примере в форматах %3d и %5.2f параметры 3 и 5 задают ширину поля, 2 - точность выводимого числа, т.е. значение выражения 7./10 будет выведено на экран с двумя значащими цифрами после запятой.

Пример линейной программы.

Составить программу вычисления площади треугольника по формуле: $S = \sqrt{p(p-a)(p-b)(p-c)}$

где $p = \frac{a+b+c}{2}$ - полупериметр; a, b, c - стороны треугольника.

Исходные данные: a = 1; b = 2; c = 0,5.

```
#include "stdafx.h"
#include<math.h>
int main()
{
float a, b, c, p, S;/* описание всех переменных, используемых в
программе */
printf("Введите исходные данные ");
scanf("%f %f %f",&a, &b,&c);/* ввод с клавиатуры значений для a,
                                b и c */
 p = (a + b + c) / 2;    // вычисление полупериметра
 S = sqrt(p *(p - a)*(p -b)*(p - c)); // вычисление площади
/*Вывод на экран*/
printf("Площадь треугольника S =%5.2f", S);
return 0;
}
```

При выполнении оператора *scanf* процессор переходит в режим ожидания набора исходных данных с клавиатуры. Для продолжения работы надо набрать на клавиатуре: 1 2 0.5 *[Enter]*

После нажатия клавиши [Enter] программа продолжит выполнение и на экран будет выведен результат в следующем виде:

Площадь треугольника S = 3.87.

Задания для самостоятельного решения

1. Записать на языке C++ операторы присваивания

Таблица 10 – Варианты заданий

Вариант	Математическая запись
1	а) $x=\dfrac{a-\sqrt{\sin(\varpi t+\varepsilon)}-e^{-m}}{\sqrt[3]{\ln(2k+d)+d^{1/k}}}$ б) $y(x)=\dfrac{\left(arctgx^3+\cos\sqrt{x}\right)^{2/3}}{e^{-x}+\ln\lvert 2.4x^{-3}\rvert}$
2	а) $m=(1+\sin^2 z)^{\sqrt{z^2+10}}$ б) $z(x,y)=y\dfrac{x^{y+1}+e^{y+1}}{1+x\sin(y+1)^2}$
3	а) $y=\left(\sqrt{\dfrac{ax+b}{c+dx}}+\sqrt{arctgx}\right)^{2.5}-e^{2x}$ б) $z(x,y)=\left(\dfrac{ay}{a+b}+\dfrac{c}{ax^2+bx}\right)^3$
4	а) $y=\pi x^2-a^3n+m\sin(\varpi t+\varepsilon)$ б) $z(x,y)=\dfrac{a+b}{e^x+\sin x}16e^x\ln x^2$
5	а) $y=\sqrt[3]{\left(\dfrac{ax}{b+cx}+tgx\right)^2}-e^{-x}$ б) $z(x,y)=\left(\dfrac{ax^2}{a+by}+\dfrac{\ln c}{b+cx^2}\right)^4+\sin^2 y$
6	а) $t=ae^{-\sqrt{\frac{\varpi}{2x}}}$; б) $f(x)=\dfrac{\sqrt{8+\lvert x-6\rvert^2+b}}{\ln x+2}+e^{\lvert x\rvert}(\ln x+2)$
7	а) $c=\sin(1+\sqrt[3]{\ln z}$; б) $f(x)=1-\dfrac{1+e^{rx}}{\dfrac{r}{x}(1+x^2)}$
8	а) $t=\dfrac{1}{2\pi}\sqrt{\dfrac{m}{c}}$ б) $y(x)=arctgx^2+\dfrac{\sin\beta x}{2}+e^{\sqrt{2x}}$
9	а) $y=\ln\lvert x^7\rvert+\lvert x^3-2\rvert-tgx^3$ б) $z(x)=e^x+a^3\left(\dfrac{b}{a+b}\right)-2\sin x^2$
10	а) $y=\dfrac{1}{2}\ln\left\lvert\dfrac{1+\sin x^2}{1-\cos^2 x}\right\rvert$ б) $z(x)=\ln\lvert\sin ax+\ln x^2\rvert$
11	а) $y=b_1\sin x+b_2\sin x^3$ б) $z(y)=ay^5+b\cos\lvert y\rvert+arctg\lvert y\rvert^5$
12	а) $y=\dfrac{\sin^3 x\cdot\cos^2 x}{5}+\dfrac{2}{5}\sin x^2$ б) $z(x)=x\cdot arctg\dfrac{x}{a}-\dfrac{a}{2}\ln\left(a^2+x^2\right)$
13	а) $y=\sqrt{2\pi}\cdot x^{a+1}\cdot e^{-x}$ б) $f(x)=\dfrac{\varpi x}{2}a^2\ln\lvert x+\varpi\rvert$
14	а) $b=1+\dfrac{\sin^2\lvert y\rvert}{\dfrac{3}{5}+\cos^2 z}$ б) $f(x)=\dfrac{1}{4}arctg\dfrac{\sqrt{3}}{x}+\dfrac{\sin 3x}{x+\cos^2 x}$
15	а) $y=\dfrac{3x^2+25e^{x^2}}{\lvert x^7\rvert+\sqrt{ax^2+2}}+\ln\lvert x\rvert$ б) $f(x)=\dfrac{e^a+\ln\lvert bx\rvert+1}{1+\ln\sqrt{\lvert dx\rvert}+e^x}$

2. Выполнить оператор присваивания

Таблица 11 – Варианты заданий

Вариант	Запись оператора на языке C++	Исходные данные	Тип переменных
1	y=3*7 /2 % 7/3;	-	*y* – веществ
2	a=exp(ceil(m+1/2)*log(x+2.5E-2));	x = 8.9 *m* = *1*	*a, x*- веществ. *m* - целая
3	k=n + round(m+b);	n = -6 b = 0.8 *m* = *2*	*к,n,m* - целые *b* - веществ
4	m=a*a*c + floor(b);	a = 45 b = 0.8 c = 2	*a,c* - целые *b,m* - веществ.
5	k=exp(ceil(x*b)*log(y));	x=1.2 b = 2 *y*= 4	*k,x* - веществ. *b,y* - целые
6	d=-a % b + a / b*c;	*a* = 5 b = 2 c = 3	*a,b,c* - целые *d* -веществ.
7	i=floor(k/n*n + a);	*к* = 99 *n* = 100 *a* = 5.87	*i,k,n* - целые *a* - веществ.
8	f=a + ceil(b/c)-64/pow(a,2)+1;	a = 4 b = 0,8 c = 2	*f,b* - веществ. *a,c* - целые
9	b=4*d*1E-2/cos(4*d-c)+floor(57.12);	c=12 d=3	*c,d*- целые b - веществ.
10	k=a*b/c + floor(12.7E-1);	*a* 6 b = 2 c = 4	a,b,c – целые *к* - веществ.
11	b=exp(ceil(n/m)*log(4+sqrt(4/m)));	m = 3 n = 2	*n,m* - целые b- веществ.
12	m=floor(a*(b / 4)+0.57E1);	a = 6 b=1	*m,a,b* - целые
13	a=exp(5 /2*log(4))-2;	-	*a* - веществ.
14	z=(4+sqrt(b+1)+24E-1 *c)*floor(b/4);	b = 1 c=10	*b,c* – целые z - веществ.
15	p=ceil(abs(b-a)) / 2-34 % 10;	a = 4.8 b = 0.8	*p.a.b* -веществ.
16	a=19 /4*4+floor(5.4) % 4;	-	*a* - целая

2.4 Разветвляющиеся вычислительные процессы

Вычислительный процесс называется разветвляющимся, если он реализуется по одному из нескольких направлений - ветвей. В программе должны быть учтены все возможные ветви вычислений. Выбор той или иной ветви осуществляется **по условию**, включенному в состав условного оператора. Для программной реализации условия используется **логическое выражение**. В сложных структурах с большим числом ветвей применяют оператор выбора.

Логические выражения. Логические выражения строятся из операндов, отношений, логических операций и круглых скобок.

Результатом вычисления логического выражения является одно из двух значений: (истина)(не ноль) или (ложь)(ноль).

В качестве операндов используются константы, переменные и функции.

Отношения. Отношение - это простейший вид логического выражения, состоящего из двух выражений арифметического, символьного или строкового типов, соединенных знаком операции отношения.

Операция отношения - это операция сравнения двух операндов:

< - меньше
<= - меньше либо равно
> - больше
>= - больше либо равно
= = - равно
!= - не равно.

Примеры записи отношений на языке С++

Отношение	Результат
5>3	(не ноль)
cos(x)>1	(ноль)
x*x+y*y<1	(не ноль) для всех точек, лежащих внутри круга с единичным радиусом и центром в начале координат
A!='Y'	(не ноль), если значение символьной переменной A не равно символу 'Y'

Следует помнить, что к операндам вещественного типа не применима операция = = из-за неточного представления чисел в памяти компьютера. Поэтому для вещественных переменных *a* и *b* отношение вида a= =b надо заменить отношением fabs(a-b)<E, где *E* - малая величина, определяющая допустимую погрешность.

Логические операции

Математическая запись	Запись на языке С++ аскаль	Название операции
¬	!	Отрицание
∧	&&	Операция «И» (логическое умножение)
∨	||	Операция «ИЛИ» (логическое сложение)

Действия логических операций удобно задать таблицами истинности, в которых приняты следующие обозначения: a, b - логические операнды; T - TRUE, F - FALSE.

a	b	a && b
T	T	T
T	F	F
F	T	F
F	F	F

a	b	a || b
T	T	T
T	F	T
F	T	T
F	F	F

Порядок выполнения операций в логических выражениях

В бесскобочных логических выражениях операции выполняются слева направо в соответствии с их приоритетом:

1. !
2. &&
3. ||
4. Отношения.

a	! a
T	F
F	T

Поскольку отношения имеют самый низкий приоритет, то их необходимо заключать в круглые скобки.

Пример.

Вычислить логическое выражение:

$$(-3 \geq 5) \vee \neg(7<9) \wedge (0 \leq 3)$$

Запись на языке С++ имеет вид:

```
(-3>= 5) || ! (7 <9) && (0 <= 3)
 1    6   4 2 5       3
```

Внизу под операциями указан порядок их выполнения.

Результаты:

1) -3>= 5=>F; 2) 7 <9 => T; 3) 0 <= 3 => T;
4) !(T) -> F; 5) F && T=>F; 6) F || F => F.

Ответ: FALSE (ноль).

Примеры записи логических выражений

Записать на языке С++ логические выражения, реализующие следующие условия:

а) переменная *x* принадлежит интервалу *[a, b]*.

Ответ: (x>= a) && (x <= b)

б) переменная *x* не принадлежит интервалу *[a, b]*.

Ответ: Данное условие можно записать в одном из двух вариантов:

1) (x <a) || (x> b);

2) или воспользоваться операцией отрицания:

! ((x>= a) && (x <= b))

Условные операторы

На языке С++ различают два вида условных операторов: короткий и полный.

Короткий условный оператор

Общий вид записи:

if (логическое выражение) P;

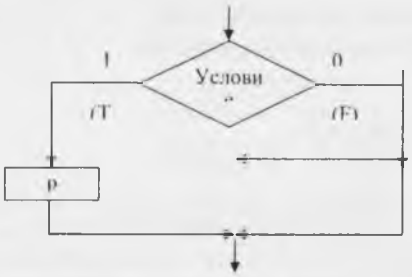
где P - любой оператор.

Работа оператора:

Сначала вычисляется логическое выражение (ЛВ), и если оно имеет значение TRUE, то выполняется оператор P, стоящий за логическим выражением. В противном случае оператор P игнорируется.

Графическая интерпретация оператора:

В схемах алгоритма короткому условному оператору соответствует структура **ЕСЛИ-ТО.**



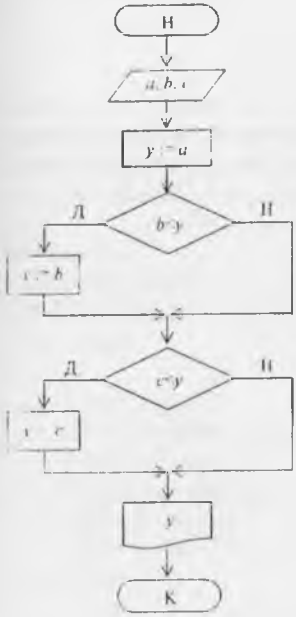
Замечание. По определению, конструкция короткого условного оператора включает единственный оператор P. Если в задаче по заданному условию требуется выполнить несколько операторов, то их необходимо заключить в операторные скобки { }, образуя тем самым составной оператор. Тогда запись условного оператора с использованием скобок имеет следующий вид:

```
if (логическое выражение)
{
   оператор  1;
   ..................
   оператор  N;
}
```

Составной оператор

Пример. Переменной *y* присвоить минимальное значение из трех различных чисел, т.е. *y* = min(a, *b, c).*

Схема алгоритма **Программа**




```
#include "stdafx.h"
#include<math.h>
int main()
{
float a, b, c,y;
printf("Введите числа a, b, c");
scanf("%f%f%f", &a,&b,&c);
y=a;
if (b<y)  y = b;
if (c<y)  y = c;
printf("y =%6.2f", y);
return 0;
}
```

Полный условный оператор

Общий вид записи:

if (логическое выражение) P1; else P2;

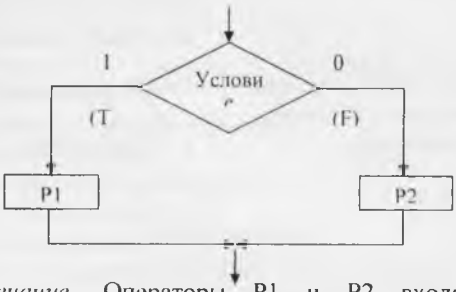
где P1, P2 - любые операторы или даже группы операторов.

Работа оператора:

Вычисляется логическое выражение, и если оно имеет значение TRUE(не ноль), то выполняется оператор P1, стоящий после логического выражения. В противном случае (FALSE (ноль)) оператор P1 пропускается, а выполняется оператор P2, стоящий после служебного слова else.

Графическая интерпретация оператора:

В схемах алгоритма полному условному оператору соответствует структура ЕСЛИ-ТО-ИНАЧЕ.



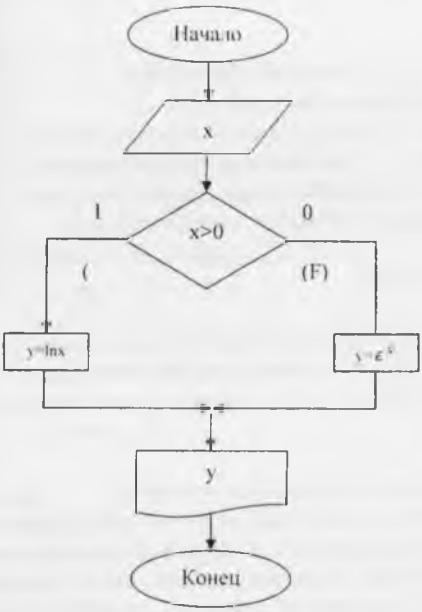
Замечание. Операторы P1 и P2 входят в стандартную конструкцию полного условного оператора как единственные. Если возникает необходимость выполнить в ветвях несколько операторов, то их заключают в операторные скобки { }. Вид записи условного оператора в этом случае следующий:

```
if (логическое выражение)
   {
оператор 1;
.....
   оператор n;
                  }
            else
                  {
оператор 1;
.....
оператор m;
                  }
```

Пример 1. Вычислить значение переменной y по одной из двух ветвей:

$$y=\begin{cases}\ln x, если\ \ x>0,\\ e^{x}, если\ \ x\le 0.\end{cases}$$

Схема алгоритма



Программа

```
#include "stdafx.h"
#include<math.h>
  int main()
  {
     float x,y;
     printf("Введите число x= ");
     scanf("%f",&x);
     if( x>0)  y = log(x) ;
      else    y= exp(x);
     printf("y =%6.2f", y);
  return 0;
  }
```

Пример 2. Вычислить корни полного квадратного уравнения $ax^2 + bx + c = 0$.

В программе предусмотреть проверку дискриминанта на знак. Если дискриминант окажется отрицательным, то вывести сообщение «Корни мнимые».

```
#include "stdafx.h"
#include<math.h>
int main()
{
float a, b, c, d, x1, x2;   // описание переменных
printf("Введите коэффициенты уравнения  ");
scanf("%f%f%f",&a, &b, &c); // ввод исходных данных
d = pow(b,2)-4*a*c;         //вычисление дискриминанта
if ( d<0 )                  // сравнение дискриминанта с нулем
printf ("Корни мнимые"); /*вывод «корни мнимые»,
                                          если   d  окажется < 0  */
else                        // иначе
   {
     x1=(-b+sqrt(d))/(2*a); // вычисление первого корня
     x2=(-b-sqrt(d))/(2*a); // вычисление второго корня
  printf("x1=%f  x2=%f",x1,x2);  // вывод корней на экран
          }
}
```

Вложенные структуры условных операторов

Структура называется вложенной, если после служебного слова else или при истинности логического выражения вновь используются условные операторы. Число вложений может быть произвольным. При этом справедливо следующее правило: служебное слово else всегда относится к ближайшему выше слову if.

Пример. Вычислить значение y по одной из трех ветвей:

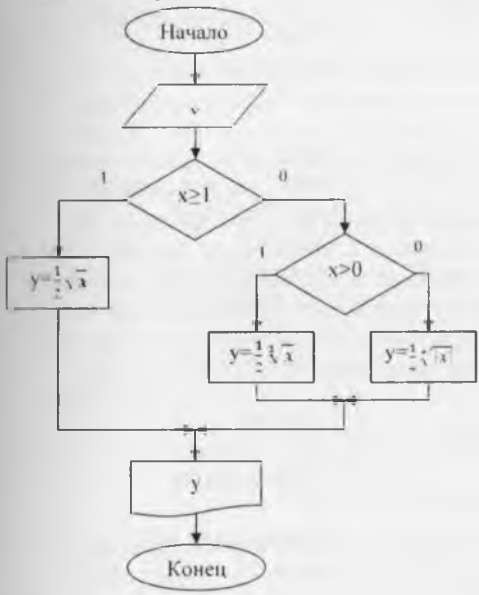
$$y = \begin{cases} \frac{1}{2}\sqrt{x}, & \text{если} \quad x \geq 1, \\ \frac{1}{3}\sqrt[3]{x}, & \text{если} \quad 0 < x < 1 \\ \frac{1}{4}\sqrt[4]{|x|}, & \text{если} \quad x \leq 0 \end{cases}$$

При решении данной задачи возможны два варианта программирования:

1) без вложенной структуры;
2) с вложенной структурой.

Ниже рассмотрены оба варианта решения задачи.

Вариант 1 (с использованием вложенной структуры)
Схема алгоритма

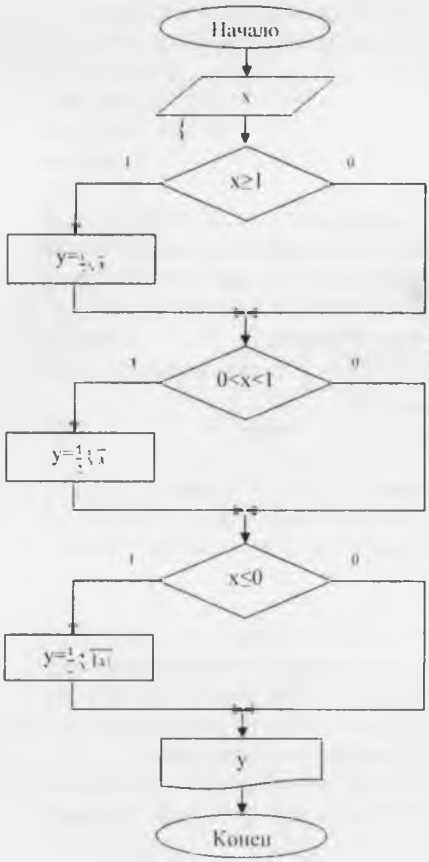


Программа

```
#include "stdafx.h"
#include<math.h>
 int main()
 {
  float a,x, y;
  printf("Введите число x= ");
  scanf("%f",&x);
  if (x>=1)
    y = sqrt(x)/2;
  else
         if(x>0){a=1.0/3; y=pow(x,a);}
         else {a=1.0/4; y=pow(x,a)/4;}
 printf("y=%6.2f",y);
 return 0;}
```

Вариант 2 (без использования вложенной структуры)

Схема алгоритма



Программа

```
#include "stdafx.h"
#include<math.h>
int  main()

 float a,x,y;
printf("Введите число x=" );
scanf("%f",&x);
 if (x>=1)
y = sqrt(x)/2;
if ((x>0)&&(x<1))
{ a= 1.0/3;
 y =pow(x,a)/2;
  }
  if(x<=0)
    {a= 1.0/4;
    y=pow(x,a)/4;
     }
  printf("y=%6.2f",y);
 return 0;
 }
```

Оператор выбора.

При многократном вложении условных операторов программная конструкция становится громоздкой и ее трудно понять. Считается, что число вложений не должно превышать двух-трех. При большем числе вложений рекомендуется использовать оператор выбора switch-case.

Общий вид записи оператора:

switch <селектор>

{

case константа выбора 1: оператор 1; break;
case константа выбора n: оператор n; break;
default: оператор n+1;
}

Селектор - это выражение целого или символьного типа.

Константы выбора - возможные значения селектора.

default – осуществляет обработку непредусмотренного значения селектора. Наличие этой метки в операторе switch необязательно.

Работа оператора:

По вычисленному значению селектора выбирается для исполнения case-оператор, содержащий константу выбора, равную значению селектора. После выполнения выбранного case-оператора управление передается на конец оператора case. Следующим в программе выполняется оператор, стоящий за оператором выбора switch.

Пример 1. Написать оператор выбора для вычисления величины y по формулам:

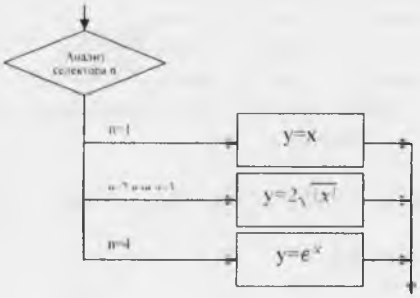
$$y = \begin{cases} x, & \text{если } n = 1, \\ 2\sqrt{|x|}, & \text{если } n = 2 \text{ или } n = 3, \\ e^x, & \text{если } n = 4 \end{cases}$$

Оператор выбора имеет следующую запись:

```
switch (n)
{
case 1:  y =x;  break;
case 2: case 3: y = 2 * sqrt(abs(x)); break;
case 4: y = exp(x); break;
default: printf("значение для n указано не верно\n");
}
```

Графическая интерпретация оператора:

В схемах алгоритма оператору switch соответствует структура ВЫБОР. Для приведенного выше примера 1 эта структура выглядит следующим образом:



Замечание. Если в строке выбора необходимо записать несколько операторов, то их заключают в операторные скобки {...}.

Пример 2. Вычислить значение y:

$$y = \begin{cases} \sin x, & \text{если } 1 \le x < 2 \\ e^{-x}, & \text{если } 2 \le x < 3 \\ \ln x, & \text{если } 3 \le x < 4 \\ tgx, & \text{если } 4 \le x < 5 \end{cases}$$

Если значение x не принадлежит рассматриваемым промежуткам, то вывести на экран соответствующее сообщение.

В задаче переменная x является вещественной и не может использоваться в качестве селектора оператора case. Введем новую переменную целого типа n, которой присваивается целая часть значения x. Тогда программа решения данной задачи с использованием оператора выбора может быть составлена следующим образом.

```
#include "stdafx.h"
#include<math.h>
  int main()
{
   float x, y;
   int n;
printf("Введите число x=   ");
scanf("%f",&x);
if( (x<1) || (x>=5))
printf("x не принадлежит рассматриваемой области\n");
 else
    { n = x;
 switch (n)
{
 case 1:  y =sin(x);   break;
 case 2:  y = exp(-x); break;
 case 3:  y =log(x);   break;
 case 4:  y = tan(x);  break;
   default:  printf(" такого решения нет \n");
        break;
}
if ((n==1)||(n==2)||(n==3)||(n==4))      printf("y=%6.2f", y);
          }
return 0; }
```

Задания для самостоятельного решения

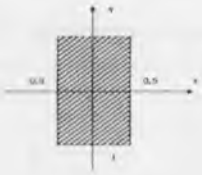
1)Записать на языке С++ логические выражения

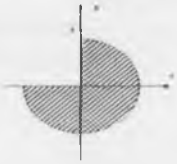
а) $\neg a \vee b$;

б) $-1 \le x \le 1$ *или* $2 \le x \le 4$;

в) переменная x находится вне интервала *[a, b]*;

г) все точки на плоскости находятся выше оси абсцисс;





д) все точки на плоскости находятся либо в первом, либо в третьем квадрантах;

е) все точки на плоскости лежат выше прямой y = 1 +x.

2) Начертить на плоскости область, в которой логическое выражение имеет значение TRUE

а) (x>= 0) && (y>= 0) && (y <= x) || (y <= -1);

б) (x*x +y*y <= 1) && (y>= 0) && !(y <x);

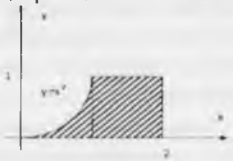
в) (y>= x) && (y>= -x);

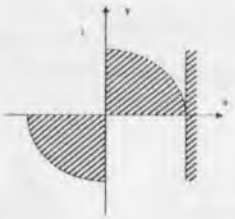
г) (y <= 2) && (x <- 2) || (x*y <0);

д) (x*y>= 0) && (y <= x) && (x <1) && (y>-1);

е) (x*x + y*y <= 4) &&(y>= x*x).

3) Записать на языке С++ логические выражения, принимающие значение TRUE для точек, принадлежащих заштрихованной области.





4) Записать на языке С++ логические выражения, расставить действия и вычислить при: i = 5, j = 2, *к=2, a* = TRUE, b = FALSE

а) $i \neq 1 \vee a \wedge \neg(b \wedge j > k)$;

б) $(i \leq 1 \vee a) \wedge (b \vee j = k)$;

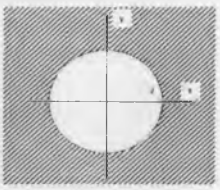
в) $\neg(i = j^2 + 1) \vee a \wedge b$;

г) $i > j + k \wedge a \wedge \neg b$;

д) $a \vee b \wedge (i \cdot j > k^3)$;

е) $(a \vee \neg b \wedge j^2 = k^2) \wedge \neg b$.

2.5 Циклические вычислительные процессы



Циклические вычислительные процессы характеризуются наличием многократно повторяющихся участков вычислений (циклов). Переменная, изменяющаяся в цикле и используемая для проверки продолжения или окончания цикла, называется *управляющей переменной*. Для программирования циклических алгоритмов используются операторы цикла с условием или с параметром.

Операторы цикла с условием

В языке С++ имеется два вида операторов цикла с условием:

1. while (пока) – цикл с предусловием;
2. do... while (повторять до тех пор, пока выполняется условие) – цикл с постусловием.

Оператор цикла *while* является наиболее универсальным, так как с его помощью можно запрограммировать практически любой циклический процесс.

Оператор цикла while

Общий вид записи:

while (логическое выражение) { <тело цикла>; }

<тело цикла> - единичный оператор или группа операторов, выполняемых в цикле.

Замечание. Если тело цикла состоит из нескольких операторов, то их обязательно заключают в операторные скобки {...}.

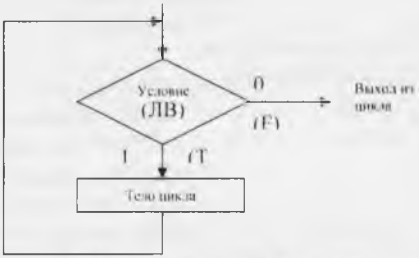
Работа оператора:

Тело цикла выполняется до тех пор, пока логическое выражение, определяющее условие выхода из цикла, имеет значение TRUE. В противном случае оператор цикла while завершает свою работу.

В состав логического выражения входит управляющая переменная, которая должна изменяться в теле цикла. Перед началом работы оператора определяется начальное значение управляющей переменной.

Графическая интерпретация оператора:

В схемах алгоритма оператору цикла while соответствует структура ЦИКЛ-ПОКА.



Оператор цикла do...while

Общий вид записи

do

<тело цикла>

while (логическое выражение);

Работа оператора

Выполняется тело цикла, после чего вычисляется логическое выражение, определяющее условие продолжения работы цикла. Если логическое выражение примет значение FALSE, то цикл do...while завершает свою работу, иначе тело цикла выполняется еще раз.

Управляющая переменная, как и в случае оператора цикла while должна включаться в состав логического выражения и изменяться в теле цикла. Перед началом работы оператора также производится начальная установка управляющей переменной.

Графическая интерпретация оператора do-while:

В схемах алгоритма оператору цикла do... while соответствует структура ЦИКЛ-ДО.

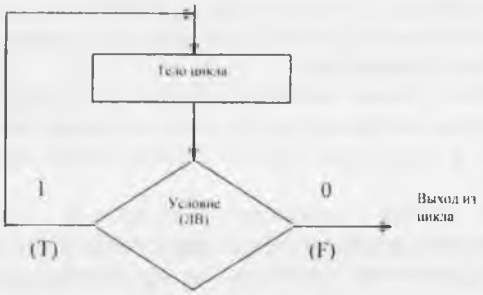
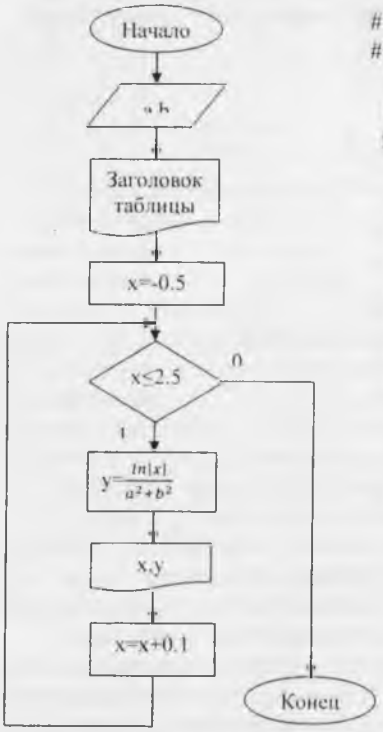


Схема алгоритма



Программа

```
#include "stdafx.h"
#include<math.h>
      int main()
 {
 float  a, b, x, y;
       printf("Введите a и b\n");
   scanf("%f%f",&a,&b);
    printf(" x     y(x)\n");
   x=-0.5; //нач. установка
   while(x <= 2.5)
    {
    y= log(fabs(x))/(a*a +b*b);
    printf("%8.1f   %8.1f",x,y);
     x=x + 0.1;
     }
        return 0;
   }
```

Пример 1. Алгоритм расчета значений функции с одной переменной. Вычислить таблицу значений функции:

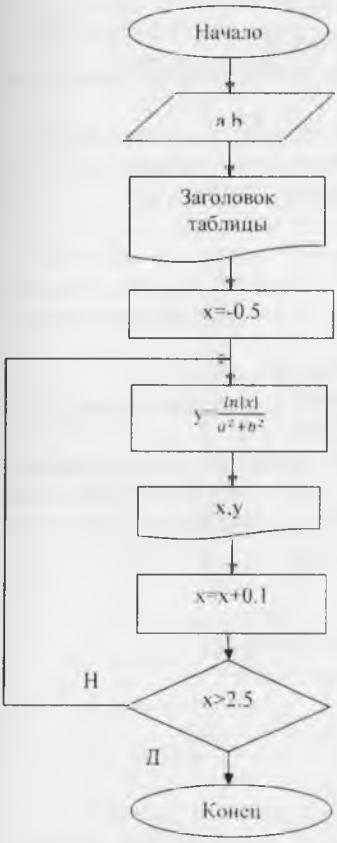
$$y(x) = \frac{\ln|x|}{a^2 + b^2}$$

для всех x, изменяющихся в интервале [-0.5 , 2.5] с шагом Δx = 0.1, a, b - заданные вещественные числа.

В данной задаче переменная x является управляющей переменной цикла.

Пример 2. Решить предыдущую задачу табулирования функции с использованием оператора цикла do…while.

Схема алгоритма **Программа**




```
#include "stdafx.h"
#include<math.h>
int main()
{
float a, b, x, y;
printf("Введите a и b ");
scanf("%f%f",&a,&b);
printf(" x      y(x)\n");
x=-0.5; //нач. установка
do
{
 y=log(fabs(x))/(a*a+b*b);
 printf("%8.1f   %8.1f\n",x,y);
 x= x+ 0.1;
} while( x<= 2.5);
return 0;
}
```

Основное отличие оператора цикла while от оператора цикла do...while

1. В операторе while тело цикла может не выполниться ни разу, если логическое выражение в начальный момент уже окажется ложным.

В операторе do...while логическое выражение записывается после тела цикла, поэтому тело цикла обязательно выполнится хотя бы один раз.

Оператор цикла с параметром

Общий вид записи:

for(i = m1; i<= m2; i=i+шаг)

{ <тело цикла>;}

i - параметр, управляющий работой цикла;

m1, m2 - выражения, определяющие соответственно начальное и конечное значения параметра цикла.

Замечание. Тело цикла состоит по стандарту из одного оператора. В случае выполнения в цикле нескольких операторов надо воспользоваться обязательно операторными скобками {...}.

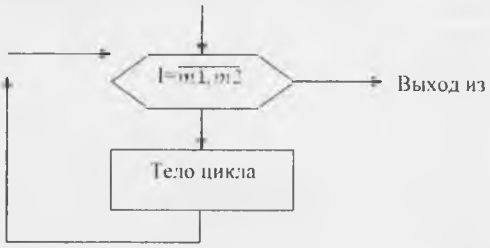
Работа оператора цикла for...

Тело цикла выполняется для каждого значения параметра i, начиная от m1 до конечного значения m2. После каждого выполнения тела цикла значение параметра i автоматически увеличивается на шаг (шаг – это любое, но заданное число).

Графическая интерпретация оператора цикла for...

В схемах алгоритма оператору цикла for..., как и в случае цикла while, соответствует структура ЦИКЛ-ПОКА.

Однако, из-за особенностей работы оператора и его широкого применения при программировании задач обработки массивов данных, для оператора for... имеется специальная структура следующего вида:



Замечание 1. Оператор цикла for...допускает применение любого шага для изменения своего параметра.

Правила использования оператора цикла с параметром

1. Параметр цикла i, а также его значения m1 и m2 могут быть любого типа.

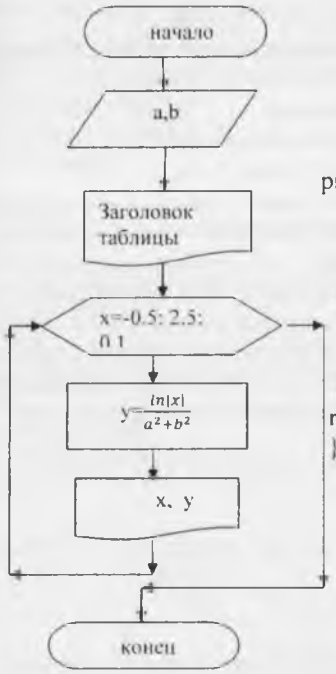
2. Параметр i, а также значения m1 и m2 не должны переопределяться (менять значения) в теле цикла.

3. При завершении работы оператора for параметр i становится неопределенным и переменную i можно использовать в других целях.

4. Тело цикла может не выполниться ни разу, если m1>m2 для цикла for... с положительным шагом, или m1<m2 для отрицательного шага.

Пример 3. Решить предыдущую задачу табулирования функции с использованием оператора цикла *for*.

Схема алгоритма



Программа

```
   #include "stdafx.h"
  #include<math.h>
  int  main()

    float a, b, x, y;
printf("введите a и b  ");
    scanf("%f%f",&a,&b);
  printf(" x      y(x)\n");
  for( x=-0.5; x<=2.5; x=x+0.1)
  {
   y=log(fabs(x))/(a*a+b*b);
   printf("%8.1f%8.1f\n",x,y);
  }
return 0;
}
```

Замечание 2. Оператор цикла while, как указывалось выше, наиболее универсальный из трех операторов цикла, используемых в языке С++. Однако конструкция оператора цикла for является

наиболее простой. Поэтому рекомендуется там, где возможно, использовать оператор for.

2.6 Базовые алгоритмы

Для реализации циклических вычислительных процессов часто используются следующие базовые алгоритмы:

- табулирование функций;
- организация счетчика;
- накопление суммы или произведения;
- поиск минимального или максимального члена последовательности.

Ниже приводятся примеры программирования задач на основе базовых алгоритмов.

Задача 1. **Алгоритм организации счетчика**

Дана последовательность:

$$\cos 1, \cos 3, \cos 5, \ldots, \cos 99.$$

Определить количество положительных членов последовательности.

Решение

Представим последовательность в общем виде:

$$a = \cos(2n - 1), \text{ где } n = \overline{1,50}.$$

Для организации счетчика в памяти компьютера выделяется ячейка, содержимое которой должно увеличиваться на 1 каждый раз, когда встречается положительный член последовательности. В программе ячейке (счетчику) соответствует переменная целого типа, например переменная L. Работа счетчика реализуется с помощью оператора присваивания L=L+1. В начальный момент содержимое ячейки должно быть равно нулю. С этой целью предварительно осуществляется очистка ячейки оператором присваивания L=0.

```
#include "stdafx.h"
#include<math.h>
int main()
{
float a;
int n,L;            // описание переменных
L=0;                // очистка счетчика
for(n=1;n<=50;n++)  // запуск цикла
{
a = cos(2*n – 1.0);       // тело цикла
if(a>0)   L = L + 1;   /*переменная-счетчик увеличивается на единицу, если a>0 */   }
        printf("L=%d", L);          // вывод значения счетчика
```

```
    return 0;
  }
```

Задача 2. Алгоритм накопления суммы

Дана последовательность:

sin 2x, sin 4x, sin 6x, ..., sin 16x

x - заданное вещественное число.

Вычислить сумму членов последовательности, которые по модулю больше 0.3.

Решение

Общий член последовательности имеет вид:

$a = \sin(2nx)$, где $n = \overline{1,8}$.

Для вычисления суммы в памяти компьютера выделяется ячейка S, к содержимому которой прибавляется член последовательности *a* каждый раз, когда выполняется условие $|a| > 0.3$. Накопление суммы реализуется оператором присваивания S=S+a;. В начальный момент ячейка для суммирования должна быть очищена оператором S=0;.

```
#include "stdafx.h"
#include<math.h>
int  main()
{
float a, x, S;   //описание переменных  задачи
int n;
printf("Введите значение  x= ");
scanf("%f",&x);
S=0;                          //очистка суммы
for(n=1;n<=8;n++)   // запуск цикла
{
a=sin(2*n*x);
  if ( abs(a)>0.3) S = S + a;   /* добавление числа   a   в сумму, если
|a|>0.3  */
  }
printf("S=%6.2f",S);   // вывод значения суммы на экран
return 0;
}
```

Задача 3. Алгоритм накопления произведения

Дана последовательность:

cos 0.1, cos 0.2, cos 0.3, ..., cos 10.

Вычислить значение: $P = |PO|$, где *PO* - произведение отрицательных членов последовательности.

Решение

Общий член последовательности имеет вид:

$y = \cos x$, где $0.1 \le x \le 10$; $\Delta x = 0.1$.

Для реализации алгоритма накопления произведения выделяется ячейка памяти PO, в которой осуществляется последовательное перемножение отрицательных членов последовательности с помощью оператора присваивания PO=PO*y; . В начальный момент в ячейку должна быть занесена единица оператором PO=1;.

```
#include "stdafx.h"
#include<math.h>
int  main()
{
float x, y, P, PO;
PO = 1;        // установка нач. значения произведения
for (x=0.1; x<=10; x=x+0.1)  //запуск цикла
{
        y = cos(x);
if ( y<0)  PO = PO*y;
      }
P = fabs(PO);
printf("P=%6.2f",P); //вывод на экран значения P
return 0;
}
```

Задача 4. Алгоритм поиска минимального члена последовательности

Дана последовательность:

$a_k = e^k \lg(2k + 1)$; $k = \overline{1,10}$.

Найти минимальный член последовательности.

Решение

Для реализации алгоритма выделяется ячейка памяти min, в которую сначала заносится первый член последовательности. Затем, начиная со второго, производится сравнение очередного вычисленного члена последовательности с содержимым ячейки min. Если текущий член последовательности меньше содержимого ячейки min, то он переписывается в эту ячейку. В противном случае содержимое ячейки min сохраняет прежнее значение. При завершении сравнения всех членов последовательности в ячейке min остается минимальное значение.

Замечание 1. Алгоритм поиска максимального члена последовательности отличается от поиска минимального члена лишь тем, что в ячейке (ей можно дать, например, имя max) запоминается больший из сравниваемых членов последовательности.

Замечание 2. В начальный момент в ячейку min можно занести не первый член последовательности, а достаточно большое число, которое превышало бы область определения сравниваемых чисел (например, min=+1E6;). Тогда при сравнении с содержимым ячейки min первый член последовательности обязательно окажется меньше и перепишется в ячейку min. При поиске максимального члена последовательности в ячейку max в начальный момент заносится, наоборот, достаточно малое число, которое должно быть меньше всех сравниваемых членов последовательности (например, max= -1E6;). В этом случае первый из сравниваемых членов последовательности окажется больше содержимого ячейки max и запишется в эту ячейку.

Программа поиска min:

```
#include "stdafx.h"
#include<math.h>
int  main()
{
float a, min;
int  k;
min = +1E6;  // нач. значение переменной min
 for( k=1; k<=10;k++)
   {
  a = exp(1.0*k)*tan(2*k + 1.0);
 if (a<min)  min = a; // условие для поиска min
 }
   printf("min=%6.2f", min);
return 0;
}
```

Задача 5. Табулирование функции (или кратные циклы)

Тело цикла может содержать любой оператор, в том числе и другой оператор цикла. Структура цикла, содержащая вложенный цикл, называется кратным циклом. Число вложений может быть произвольным. Если цикл содержит один вложенный цикл, то он называется двойным, если содержит два вложенных цикла, то является тройным и т.д. Цикл, который содержит вложенный цикл, называется внешним. Вложенный цикл называется внутренним.

Переменная внутреннего цикла всегда меняется быстрее, чем внешнего. Это означает, что для каждого значения внешней переменной цикла меняются все значения внутренней переменной.

Внешний и внутренний циклы могут использовать любой вид операторов цикла (while, do-while, for).

Пример. Алгоритм табулирования функции с двумя переменными.

Вычислить значение функции:

$z(x, y) = \sin x + \cos y$

при всех x, изменяющихся на интервале [-1, 1] с шагом $\Delta x = 0.2$, и y, изменяющихся на интервале [0, 1] с шагом $\Delta y = 0.1$.

Данный алгоритм реализуется с использованием двойного цикла, в котором x примем за внешнюю переменную цикла, y - за внутреннюю переменную цикла.

```
#include "stdafx.h"
#include<math.h>
  int  main()
   {
  float x, y, z;                    // описание переменных
     printf("x     y     z(x,y)\n");   // вывод заголовка
      x= -1;                 // начальное значение параметра  внешнего
цикла
      while (x<=1)    // запуск внешнего цикла, если x≤ 1
       {
  for( y=0; y<=1; y=y+0.1)   //запуск внутреннего цикла
     {
    z=sin(x) + cos(y);              // вычисление функции
     printf("%6.1f %6.1f  z=%6.1f",x, y, z); // вывод
       }
    x=x + 0.2;              // изменение параметра x на шаг
  }
  return 0;
 }
```

В результате выполнения программы вид таблицы на экране будет следующим:

X	y	z(x,y)
-1.0	0.0	z=…
-1.0	0.1	z=…
…	…	…
-1.0	1.0	z=…

-0.8	0.0	z=…
-0.8	1.0	z=…

Задача 6. Вычисление сумм последовательностей

Последовательности с заданным числом элементов

Пример 1. Найти сумму первых 20 элементов последовательности

S=1/2 – 2/4 + 3/8 – 4/16+…

Чтобы решить эту задачу, надо определить закономерность в изменении элементов. В данном случае можно заметить:

- каждый элемент представляет собой дробь.
- числитель дроби при переходе к следующему элементу возрастает на единицу.
- знаменатель дроби с каждым шагов увеличивается в 2 раза.
- знаки перед дробями чередуются (плюс, минус и т.д.).

Любой элемент последовательности можно представить в виде

S=z*c/d

У переменной ***z*** меняется знак (эту операцию можно записать в виде **z=-z**), значение переменной с увеличивается на единицу (***c++***), а переменная ***d*** умножается на 2 (***d=d*2***).

Алгоритм решения задачи можно записать в виде следующих шагов:

- записать в переменную ***S*** значение 0. В этой ячейке будет накапливаться сумма;
- записать в переменные **z**, ***c*** и ***d*** начальные значения (для первого элемента): z=*1*, *c=1*,*d=2*;

сделать 20 раз следующие две операции:

- добавить к сумме значение очередного элемента;
- изменить значения переменных **z**, ***c*** и ***d*** для вычисления следующего элемента.

```
#include "stdafx.h"
int main()
{
float S, z, c, d;
int i;
S = 0; z = 1; c = 1; d = 2;
for ( i = 1; i <= 20; i ++ )
{
 S = S + z*c/d;
 z = - z;
 c++;
 d = d * 2;
}
```

Начальные

добавить элемент к сумме

изменить переменные

```
printf("Сумма S = %f", S);
return 0;
}
```

Суммы с ограничивающим условием

Рассмотрим более сложную задачу, когда количество элементов заранее неизвестно.

Пример 2. Найти сумму всех элементов последовательности

$$S=1/2 - 2/4 + 3/8 - 4/16+\ldots$$

которые по модулю меньше, чем 0.001.

Эта задача имеет решение только тогда, когда элементы последовательности убывают по модулю и стремятся к нулю. Поскольку мы не знаем, сколько элементов войдет в сумму, надо использовать цикл while (или do - while). Один из вариантов решения показан ниже.

```
#include<math.h>
#include "stdafx.h"
int  main()
{
float S, z, c, d, a;
S = 0; z = 1; c = 1; d = 2;
a = 1;
while ( a >= 0.001 )
{
 a =fabs( c / d);
 S = S + z*a;
 z = - z;
 c ++;
 d = d * 2;
}
printf("Сумма S = %f", S);
return 0;
}
```

начальные значения

Запустить цикл, если a >0.001

добавить элемент к сумме

изменить переменные

Цикл закончится тогда, когда переменная ***a*** (она обозначает модуль очередного элемента последовательности) станет меньше 0.001. Чтобы программа вошла в цикл на первом шаге, в эту переменную надо записать любое значение, большее, чем 0.001.

Очевидно, что если переменная ***a*** не будет уменьшаться, то условие в заголовке цикла всегда будет истинно и программа «зациклится».

2.7 Указатели и массивы

Указатель - это переменная, содержащая адрес области памяти. Указатели широко применяются в языке C++. В некоторых случаях без них просто не обойтись, а в некоторых программы с использованием указателей становится короче и эффективнее.

Начнем с того, что поговорим о структуре памяти любого компьютера. Как известно, память компьютера представляет последовательность 8-битовых байтов. Каждый байт пронумерован, причем нумерация начинается с нуля. Номер байта называется *адресом*. Иногда говорят, что адрес указывает на определенный байт. Таким образом, *указатель* является просто адресом байта памяти.

Язык C++ позволяет определять переменные, которые могут хранить адреса памяти. Такие переменные и называются указателями. Значение указателя сообщает о том, где размещен объект, но ничего не говорит о значении самого объекта.

Присваивая указателю то или иное допустимое значение, можно обеспечить доступ к данным через этот указатель.

Для описания переменной типа указатель используется символ *.

Формат описания:

Тип *имя;

Указатель – адрес переменной какого-то определенного типа. Этот тип сообщается компилятору при объявлении указателя.

```
int *x;
char *y;
```

Пример следует понимать так: x – это указатель на ячейку, в которой хранится целое значение, а y – указатель на однобайтовую ячейку, предназначенную для хранения символа.

Двумя наиболее важными операциями, связанными с указателями, являются операция обращения по адресу * и определение адреса &.

Операция обращения по адресу предназначена для записи или считывания значения, размещенного по адресу, содержащемуся в переменной-указателе.

Например:

```
int *x;
. . .
*x=5;
```

Операция определения адреса & возвращает адрес памяти своего операнда. Операндом должна быть переменная.

Например:

```
int *x;
```

int a=5;
x=&a;

Кроме того, над указателями можно выполнять арифметические операции сложения и вычитания.

Если y – указатель на целое, то унарная операция y++ увеличивает значение адреса, хранящегося в переменной-указателе на число равное размеру ячейки целого типа, т.е. на 2 байта; теперь оно является адресом следующей ячейки целого типа. Соответственно, оператор y--; означает, что адрес уменьшается на 2 байта.

Указатели и целые числа можно складывать. Конструкция y + n (y - указатель, n - целое число) задает адрес n-го объекта, на который указывает y. Это справедливо для любых объектов (int, char, float и др.); транслятор будет масштабировать приращение адреса в соответствии с типом, указанным в определении объекта.

Любой адрес можно проверить на равенство (= =) или неравенство (!=), больше (>) или меньше (<) с любым другим адресом.

Рассмотрим следующий фрагмент программы:

```
#include "stdafx.h"
int main()
{
int *x, *w;
int y;
*x=16;
y=-15;
w=&y;

}
```

Этот текст можно понимать так:

Выделить память под три переменные x, w, y, где x и w – переменные типа указатель. Оператор *x=16; означает, что в ячейку, адрес которой записан в x, помещается значение 16. Затем переменной y присваивается значение -15. После чего в указатель w записывается адрес переменной y. Синтаксически текст записан правильно. Проблема заключается в том, что указатель x не инициализирован. Описание int *x; это лишь указание компилятору резервировать память, необходимую для хранения адреса целой ячейки. Но в этой памяти может оказаться адрес любой ячейки, в том числе и адрес, где хранится полезная информация, например, операционная система. Запись в такую ячейку может привести к сбою в работе компьютера. Поэтому при работе с указателями их надо правильно

инициализировать. Существует 4 способа правильного задания начального значения для указателя:

1) Описать указатель глобально, т.е. вне любой функции. При этом указатель будет инициализирован безопасным нулевым адресом. Кроме того любому указателю можно присвоить безопасный нулевой адрес, например:

```
int *x;
x=NULL;
```

Гарантируется, что этот адрес не совпадет ни с одним адресом, уже использованным в системе.

2) Присвоить указателю адрес переменной. Например: w=&y;

3) Присвоить указателю значение другого указателя, к этому моменту правильно инициализированного. Например: x=w;

4) Использовать функции выделения динамической памяти malloc() и calloc(). При использовании этих функция необходимо подключать библиотеку <malloc.h>. Рассмотрим пример использования функции malloc():

```
x=(int*)malloc(sizeof(int));
```

Приведенный пример означает, что функция выделит область памяти, размер которой определит функция sizeof(). Если вы знаете размер ячейки заданного типа, то можно написать проще: x=(int*)malloc(2);

По окончанию работы программы, память, выделенную функцией malloc() рекомендуется освободить функцией free(x); Вернемся к приведенному ранее фрагменту программы:

```
#include "stdafx.h"
#include <malloc.h>
 int main()
 {  int *x, *w;
    int y;
    x=(int*)malloc(sizeof(int));
    *x=16;
    y=-15;
    w=&y;
 }
```

Теперь никаких конфликтных ситуаций при работе с указателями не возникнет. В языке C++ существует еще одна пара операторов new и delete для динамического выделения и освобождения памяти. О них мы поговорим чуть позже.

Понятие массива

Массив представляет собой упорядоченное множество однотипных элементов. В языке C++ массив описывается переменной сложной структуры. При описании массива необходимо указать:

- способ объединения элементов в структуру (одномерный, двухмерный и т.д.);
- число элементов;
- тип элементов.

Общий вид описания массива

<тип элементов> имя [число элементов];

Доступ к каждому элементу массива осуществляется с помощью индексов. Индексы задают порядковый номер элемента, к которому осуществляется доступ. В языке C++ первый элемент массива имеет индекс ноль. Число индексов определяет структуру массива: если используется один индекс, то такой массив называется одномерным, если два индекса - двухмерным, и т.д. В общем случае размерность массива может быть произвольной.

Одномерные массивы

В математике одномерному массиву соответствует n-мерный вектор, например:

$\bar{x} = \{x_i\}; i = 1,\ldots, n$,

где x_i - компонента (координата) вектора;

i - номер компоненты;

n - число компонент.

Описание одномерного массива

На языке C++ описание одномерного массива задается следующим образом:

<тип элементов> <имя массива>[размер];

Компилятор отводит под массив память размером (sizeof(тип)*размер) байтов.

При описании массива можно задать начальные значения его элементов:

```
int dat[4]={5,8,-2,11};
float kom[]={3.5,6,-1.1};
```

Указатели могут обеспечить простой способ ссылок на массив. Имя массива фактически является константой-указателем, ссылающимся на начальный адрес данных (адрес первого элемента массива). Начальный адрес массива определяет компилятор в момент описания массива, и такой адрес не может быть переопределен. Первый элемент массива имеет индекс ноль.

Например:

```
int  Ar[5];
printf ("адрес Ar=%x\n",Ar);
printf ("адрес Ar=%x\n",&Ar[0]);
```

В приведенном фрагменте обе функции printf выводят адрес массива Ar, т.к. выражения Ar и &Ar[0] эквивалентны.

Индексированные переменные

Выбор отдельного элемента из массива осуществляется с помощью индексированной переменной, которая задается следующим образом:

x[i] - индексированная переменная (элемент массива).

Здесь x - имя массива;

i - индекс (номер элемента массива).

В качестве индекса используются:

- целые константы, например $x[2] \Rightarrow x_2$;
- целые переменные, например $x[k] \Rightarrow x_k$;
- индексные выражения, например $x[2*n+1] \Rightarrow x_{2n+1}$.

В языке C++ индексы элементов любого массива начинаются с нуля. Индексными выражениями являются арифметические выражения целого типа.

Переменная с индексом может стоять в левой части оператора присваивания, например:

```
x[3]=2.5;
```

Ввод-вывод одномерных массивов

Ввод-вывод массивов осуществляется поэлементно с помощью операторов scanf и printf соответственно и оператора цикла for..., в котором в качестве параметра используется индекс.

Пример 1. Организовать ввод с клавиатуры массива:

A = (1.2, 5, -6.8, 14).

Необходимо описать массив и индекс.

```
int main()
{  float A[4] ;
           int  i;
```

В программе ввод массива рекомендуется организовать в виде диалога, поместив перед оператором ввода оператор вывода (printf), которым выдается на экран поясняющее сообщение, например:

```
printf("Введите массив A\n");
for( i = 0;  i<4; i++)
   scanf("%f",&A[i]);
```

На клавиатуре через один или несколько пробелов набираются элементы массива и нажимается клавиша [Enter]:

```
1.2 5  -6.8  14 [Enter]
```

Замечание. Элементы массива можно вводить в «столбик», если после ввода каждого элемента нажимать клавишу [Enter].

Пример 2. Организовать вывод массива *A* на экран таким образом, чтобы все элементы располагались на одной строке экрана.

В программе надо записать следующие операторы:

```
for ( i = 0; i<4;  i++)
printf("%5.2f  ",A[i]);
printf("\n");
```

Вид выводимого массива на экране:

```
    1.2   5.0   -6.8   14.0
```

Оператор printf("\n"); без списка служит для перевода курсора к началу следующей строки экрана.

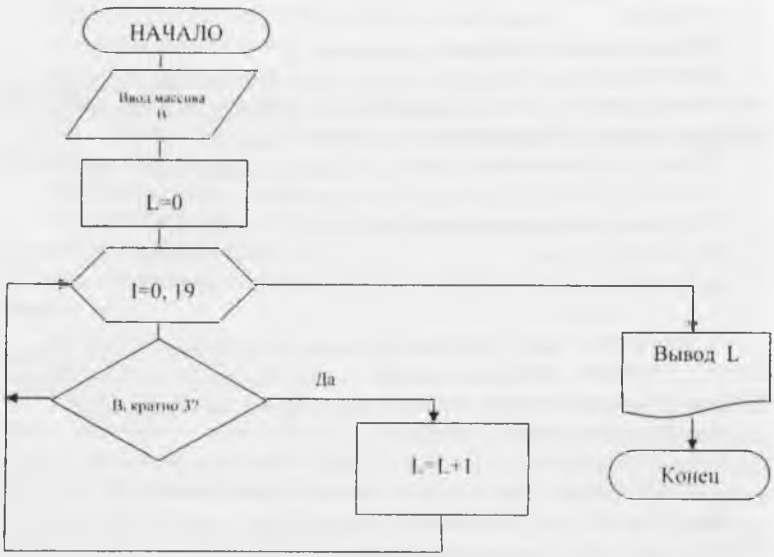
Обработка одномерных массивов

При решении задач обработки массивов используют базовые алгоритмы реализации циклических вычислительных процессов: организация счетчика, накопление сумм и произведений, поиск минимального и максимального элементов массива.

Задача 1. Организация счетчика

Дан целочисленный массив: $B = \{b_i\}; i=\overline{1,20}$. Определить количество элементов массива, которые делятся на 3 без остатка.

Схема алгоритма:



Текст программы:

```
#include "stdafx.h"
int main()
{
  int  B[20] ;                /* описание массива B*/
  int  i, L;
  printf("Введите массив B\n");
   for( i=0;  i<20;  i++)
    scanf("%d", &B[i]);    /* ввод данных в массив с клавиатуры*/
     L=0;
      for( i =0; i<20;  i++)
         if (B[i] % 3== 0) /*проверка элемента на кратность 3   */
          L++;
  printf("Кол-во=%d\n", L);
 return 0;
}
```

Задача 2. Накопление суммы и произведения

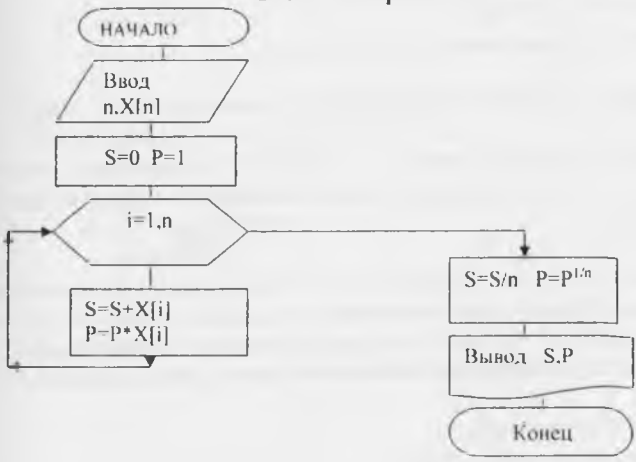
Дано целое число n и массив вещественных чисел:

$$X = \{x_i\}, \quad i = \overline{1,n}.$$

Вычислить среднее арифметическое и среднее геометрическое чисел массива, используя формулы:

$$S = \frac{1}{n}\sum_{i=1}^{n} x_i ; \qquad P = \sqrt[n]{\prod_{i=1}^{n} x_i} .$$

Схема алгоритма:



Текст программы:

```
#include "stdafx.h"
#include <math.h>
int main()
{
 float   X [100];          //описание массива X
 float z;
 int   n;
 int i;
 float   S=0, P=1; //начальные значения суммы и произведения
 printf("Введите размер массива n= ");
 scanf("%d",&n);
 printf("Введите массив X\n");
 for( i = 0; i<n; i++)
 scanf("%f",&X[i]);
 for( i=0;  i<n; i++)
 {
  S = S + X[i];   /* вычисление суммы элементов массива X */
  P =P*X[i];              /* вычисление произведения элементов X
*/
 }
 S = S/n;          /* вычисление среднего значения X */
 z=1.0/n;
 P=pow(P,z);       /*  вычисление среднего геометрического X */
 printf("S=%6.2f\n", S);
 printf("P=%10.6f\n",P);
return 0;
}
```

Задача 3. Поиск минимального и максимального элементов массива

Дан вещественный массив: $T = \{t_i\}$; $i=\overline{1,10}$. Поменять местами минимальный и максимальный элементы массива и вывести массив после обмена.

Решение

В этой задаче для осуществления обмена надо знать не только значения минимального и максимального элементов массива, но и их местоположение. Поэтому во время поиска минимального и максимального элементов необходимо фиксировать значения их индексов.

Введем обозначения:

min - минимальный элемент;

imin - индекс минимального элемента;
max - максимальный элемент;
imax - индекс максимального элемента.

Текст программы:

```
#include "stdafx.h"
#include<math.h>
 int main()
{
float T[10] ;        /* описание массива T */
 int  i, imin, imax;
float   min, max;
 printf("Введите массив T\n");
for( i = 0;i<10;i++)
scanf("%f",&T[i]);
min =+1E6;
max =-1E6;
for( i = 0; i<10; i++)
 {
  if (T[i]<min)
    {
       min = T[i];
       imin=i;          /* сохранение номера текущего min */
     }
    if( T[i]>max)
     {
       max = T[i];
       imax = i;       /* сохранение номера текущего max */
     }
 }
  T[imin]=max;
  T[imax]= min;
  for( i= 0; i<10;  i++)
      printf("%6.2f  ",T[i]);
          printf("\n");
return 0;
}
```

Двухмерные массивы

Двухмерные массивы в математике представляются матрицей:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

или сокращенно можно записать: $A = \{a_{ij}\}_{m \times n}$, где m – число строк; n - число столбцов; i,j - индексы (номера) строки и столбца, на пересечении которых находится элемент a_{ij}.

Описание двухмерного массива

Описание матрицы задается структурным типом вида:

<тип элементов><имя> [m][n] ;

где m – количество строк;

n –количество столбцов матрицы.

По описанию матрицы во внутренней памяти компьютера выделяется область из $m \times n$ последовательных ячеек, в которые при работе программы записываются значения элементов матрицы. Например, по описанию:

float A [3][5];

В памяти компьютера для элементов матрицы выделяется область, состоящая из 3× 5=15 последовательных ячеек вещественного типа. Из описания видно, что матрица состоит из трех строк и пяти столбцов.

Обращение к отдельным элементам матрицы осуществляется с помощью переменной с двумя индексами, причем индексы, как и для одномерного массива начинаются с нуля. Например:

A[i][j] $\Rightarrow a_{ij}$;

A[2][3] $\Rightarrow a_{23}$;

A[2*n][k+1] $\Rightarrow a_{2n,k+1}$.

Ввод-вывод двухмерного массива

Для поэлементного ввода и вывода матрицы используется двойной цикл for.... Если задать индекс i как параметр внешнего цикла, а индекс j как параметр внутреннего цикла, то ввод-вывод матрицы осуществляется построчно.

Пример 1. Организовать ввод целочисленной матрицы M по строкам.

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Описание матрицы вместе с текущими индексами имеет вид:

```
int main()
{
```

```
int M[2][3] ;
int  i, j;
```

Ввод в программе реализуется в форме диалога, т.е. сопровождается поясняющим сообщением:

```
printf("Введите матрицу M\n");
for( i = 0; i<2; i++)
   for( j = 0; j< 3; j++)
   scanf("%f",&m[i][ j]);
```

На клавиатуре желательно для наглядности восприятия набирать элементы матрицы по строкам, отделяя числа друг от друга одним или несколькими пробелами:

```
1    2  3  [Enter]
4    5  6  [Enter]
```

Пример 2. Организовать вывод матрицы M на экран.

Вывод матрицы необходимо реализовать в удобном для чтения виде, т.е. чтобы на одной строке экрана располагалась одна строка матрицы. С этой целью в тело внешнего цикла, помимо внутреннего цикла, включается оператор printf, который переводит курсор к началу следующей строки экрана после вывода текущей *строки* матрицы.

```
    for ( i = 0; i<2; i++)
  {
          for ( j = 0; j< 3; j++)
              printf("%3d  ",m[i][j]);
         printf("\n");
         }
```

Вид матрицы на экране будет следующим:

```
    1    2   3
    4    5   6
```

Обработка матриц

Базовыми алгоритмами обработки матриц являются те же алгоритмы, которые используются при обработке одномерных массивов. Однако реализацию этих алгоритмов можно условно рассматривать для двух типов задач.

1. Алгоритмы реализуются при просмотре всех элементов матрицы (просмотр может быть с условием). Начальная установка алгоритма выполняется перед двойным циклом. В этом случае запись операторов цикла для параметров i и j осуществляется последовательно друг за другом и имеет вид:

```
<начальная установка искомых параметров>
for (i = 0; i<n; i++)
   for (j = 0; j< n; j++)
```

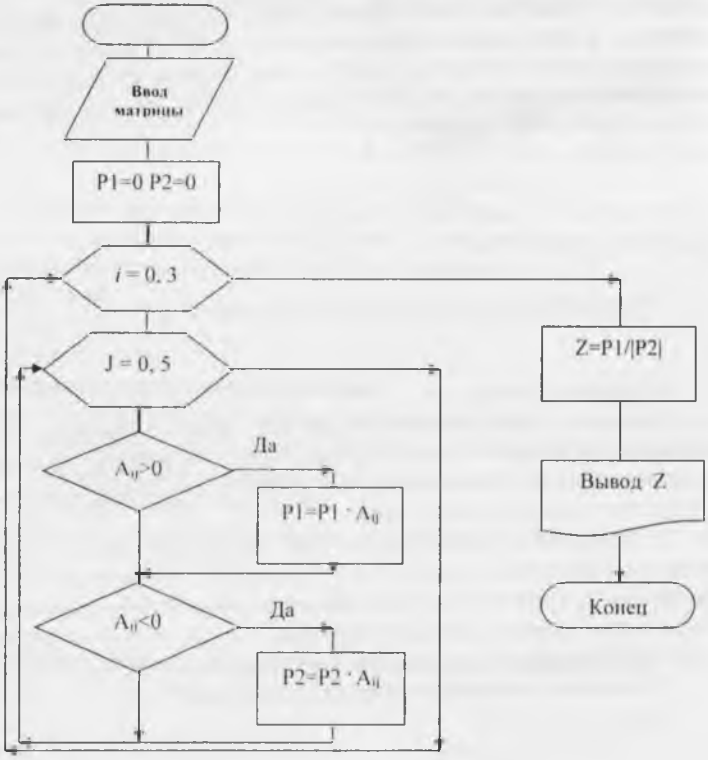
<тело цикла>;

2. Алгоритмы реализуются внутри каждой строки или каждого столбца матрицы. В этом случае начальная установка алгоритма выполняется между операторами цикла, записанными для параметров i и j. Например, если алгоритм реализуется для каждой строки, то запись в программе имеет следующий вид:

```
for ( i = 0; i< m ; i++)
{
        <начальная установка искомых параметров>
         for ( j = 0; j< n; j++)
       <тело цикла>;
}
```

Ниже рассмотрены примеры программирования задач каждого типа.

Реализация алгоритмов задач первого типа



Задача 1. Дана матрица вещественных чисел $A = \{a_{ij}\}_{4x6}$. Вычислить значение Z = P1/|P2|, где *P1* и *P2* – произведения положительных и отрицательных элементов матрицы соответственно.

Текст программы:

```
#include "stdafx.h"
  #include<math.h>
   void main()
   {
    float A [4][6] ;       // описание матрицы A
    int i;
    int j;
    float P1;
    float P2;
    float Z;
    printf("Введите матрицу A\n");
    for ( i =0; i<4; i++)
       for ( j = 0;j<6; j++)
          scanf("%f",&A[i][j]);
    P1=1;                  // установка начальных значений
    P2=1;                   // произведений
    for ( i = 0; i<4; i++)
      for ( j =0; j<6 ;j++)
       {
        if (A[i][j]>0) P1= P1*A[i][j];  // произведение   +
        if ( A[i][j]<0) P2 = P2*A[i][j];// произведение   -
       }
   Z=P1/fabs(P2);
   printf("Z=%10.2f  \n",Z);
   }
```

Задача 2. В квадратной целочисленной матрице $B = \{b_{ij}\}_{5x5}$ вычислить модуль разности между числом нулевых элементов, стоящих ниже главной диагонали, и числом нулевых элементов, стоящих выше главной диагонали.

Введем обозначения:

L1 – счетчик нулевых элементов ниже главной диагонали;

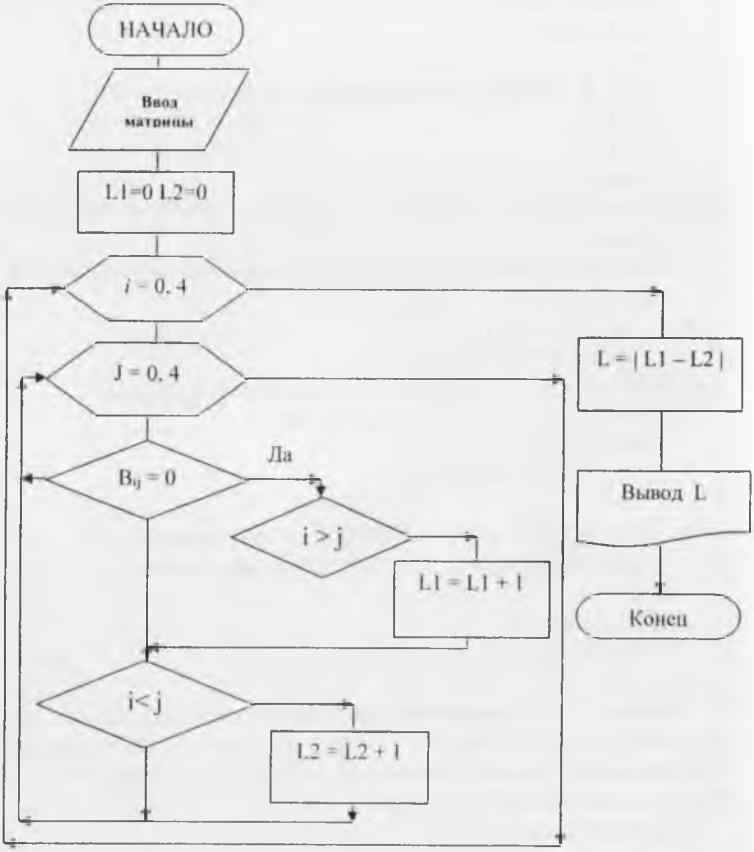
L2 – счетчик нулевых элементов выше главной диагонали;

L = |L1 -L2|.

Для заполнения матрицы воспользуемся датчиком случайных чисел.

Текст программы:

```
#include "stdafx.h"
#include<math.h>
  void main()
 {
```

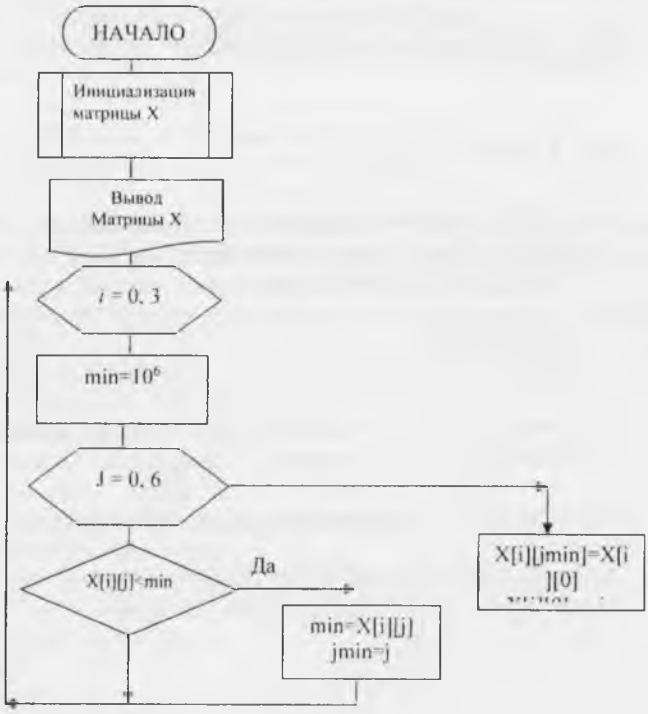



```
   int B[5][5] ;        /* описание матрицы  B   */
   int i, j, L1, L2, L;
   for ( i = 0; i< 5; i++)
     for (j = 0;j<5; j++)
        B[i][j]= floor(rand()/1000.0);
   L1 = 0;              /* инициализация счетчиков L1 и L2  */
```

```
    L2 = 0;
    for ( i = 0; i< 5; i++)
     for (j = 0; j< 5; j++)
       if (B[i][j]==0 )  /* поиск элементов, равных нулю   */
         {
           if ( i>j)   L1=L1+1;  /* выше  главной диагонали */
           if ( i<j)   L2=L2+1;  /* ниже главной диагонали  */
              }
    L = abs(L1 - L2);
    printf("L=%d", L);
}
```

Реализация алгоритмов задач второго типа

Задача 1. В матрице $X = \{x_{ij}\}_{3\times6}$ вещественных чисел первый элемент каждой строки поменять местами с минимальным элементом этой строки. Вывести матрицу X после обмена. (Для заполнения матрицы воспользуемся датчиком случайных чисел).



Текст программы:

```
#include "stdafx.h"
#include<math.h>
   void  main()
{
   float X[3][6];       // описание матрицы X
   int  i,j,jmin;
   float  min;

    for(i=0; i<3;  i++) /* заполнение  матрицы  случайными
числами*/
     for (j= 0; j<6;  j++)
       X[i][j] =rand()/100;

  printf(" матрица X\n");
  for(i=0; i<3; i++) //вывод матрицы до перестановки
    {
     for (j=0; j<6; j++) printf("%8.2f",X[i][j]);
     printf("\n");
    }

    for(i=0; i<3; i++) //цикл по  строкам
    {
     min=+1E6; // установка начального значения min
      for (j=0; j<6; j++)  //цикл по  столбцам
        if (X[i][j]<min)    // поиск минимума
        {
           min=X[i][j];
           jmin=j;
        }
      X[i][jmin]=X[i][0];      // перестановка  первого элемента
      X[i][0]=min;             // матрицы с наименьшим
    }
    for(i=0; i<3; i++) //вывод матрицы после перестановки
    {
     for (j=0; j<6; j++) printf("%8.2f",X[i][j]);
     printf("\n");
    }
}
```

Задача 2. Дана матрица вещественных чисел $C = \{c_{ij}\}_{8\times4}$. Вычислить среднее арифметическое каждого столбца. Результат оформить в виде одномерного массива $S = \{s_j\}; j = \overline{1,4}$.

```
#include "stdafx.h"
 void main()
{ float  C[8][4];
  float S[4];
  int  i, j;
  printf("Введите матрицу C:\n");
  for(i=0; i<8;  i++)
      for (j= 0; j<4;  j++)
        scanf("%f",&C[i][j]);
  for (j= 0; j<4;  j++)
    {
     S[j]=0; //начальная установка  элемента  массива для сумм
     for(i=0; i<8;  i++)
        S[j]= S[j] + C[i][j]; //накопление суммы j-го столбца
     S[j]=S[j]/8;        //вычисление среднего значения суммы j
столбца
     }
for (j= 0; j<4;  j++) printf("%8.2f",S[j]); // вывод всех сумм
 printf("\n");
}
```

В приведенной выше программе для вычисления каждого элемента S[j] организован двойной цикл, в котором индекс j является внешним параметром цикла, а индекс i - внутренним.

Приведем вариант программы без использования одномерного массива S.

```
#include "stdafx.h"
 void main()
{ float  C[8][4];
  float S;
  int  i, j;
  printf("Введите матрицу C:\n");
  for(i=0; i<8;  i++)
    for (j= 0; j<4;  j++)
            scanf("%f",&C[i][j]);
  for (j= 0; j<4;  j++)
    {
     S=0;
     for(i=0; i<8;  i++)
```

```
            S= S + C[i][j];
        S = S/8;
    printf("Среднее арифметическое %d-го столбца=%8.2f\n", j, S);
        }
    }
```

2.8 Подпрограммы

Структура сложной программы.

Любая программа на языке высокого уровня может быть разбита на ряд логически завершенных программных единиц - подпрограмм. Такое разделение вызвано двумя причинами.

1. *Экономия памяти.* Каждая подпрограмма записывается в программе один раз, в то время как обращаться к ней можно многократно из разных точек программы.

2. *Структурирование программы.* Алгоритм решения задачи может быть достаточно сложным, поэтому целесообразно выделить самостоятельные смысловые части алгоритма и оформить их в виде подпрограмм.

В языке Си существует один вид подпрограмм, который называется функция. Каждая программа должна иметь главную функцию (main), которая служит точкой входа в программу. Кроме главной функции в программе может быть оформлено произвольное число функций.

Описание подпрограммы само по себе никаких действий не вызывает. При запуске программы выполнение начинается с операторов главной функции main(). Чтобы выполнить подпрограмму, в нужной точке главной функции необходимо записать обращение к подпрограмме.

Ниже схематично приведена структура программы, в которой описана подпрограмма-функция.

```
    #include "stdafx.h"
   //Раздел описаний функций
тип имя_функции(тип имя_параметра_1, тип
имя_параметра_2,...)
    {
      тело функции
    }
.....
main() //начало главной функции
  {
    Обращение к подпрограмме;
```

```
……
} //конец главной функции
```

Функции

Функция – это автономная часть программы, реализующая определенный алгоритм и допускающая обращение к ней из различных частей программы.

Каждая функция по отношению к другим является внешней.

Общий вид описания функции

Описание функции содержит заголовок со списком формальных параметров и тело функции.

```
Тип   Имя(список формальных параметров)
{
    Описание локальных переменных;
    Операторы тела функции;
    return  результат;
}
```

Тип указываемый в заголовке функции определяет тип результата ее работы, который будет возвращаться в точку вызова. Если тип не указан, то по умолчанию подразумевается *int* (целый). Для возврата значения в теле функции должен быть оператор *return*. В дальнейшем будем называть такую функцию *типизированной*.

Если функция не должна возвращать результат, то она считается *не- типизированной*, что задается ключевым словом *void*, стоящим на месте типа. В этом случае оператор *return* в функции не требуется.

```
void   Имя(список формальных параметров)
{
    Описание локальных переменных;
    Операторы тела функции;
}
```

Список формальных параметров обеспечивает передачу исходных данных в функцию.

Параметры, указанные в заголовке функции, называются формальными, а параметры, указываемые при ее вызове – фактическими.

Рассмотрим пример оформления функции для вычисления максимального значения из двух заданных.

```
#include "stdafx.h"
int max(int a, int b)
 {
   int c;
   if (a>b)
```

```
        c=a;
    else
        c=b;
    return c;
}
void main()
{   int x,y,z;
    printf("Введите x  и y:");
    scanf("%d%d",&x,&y);
    z=max(x,y);
    printf("max=%d\n",z);
}
```

Обращение к функции

Обращение к *типизированной* функции не является специальным оператором, а включается в состав выражения. Результат выполнения функции возвращается в основную программу через имя функции. Обращение к функции записывается аналогично записи стандартной функции (например, sin(x), exp(x) и т.п.) в виде операнда:

<имя функции>(<список фактических параметров>);

При вычислении выражения операнд обращения к функции заменяется значением функции.

Пример программы с функцией

Вычислить значение: $Z = \frac{a^5 + a^{-5}}{2a^2}$,

где *a* - заданное вещественное число.

В этой задаче требуется многократно использовать алгоритм возведения числа в целую степень. Оформим функцию, в которой данный алгоритм можно формально описать как алгоритм накопления произведения.

$$P = x^n = \underbrace{x \cdot x \cdot x \cdot \ldots \cdot x}_{n \text{ раз}} = \prod_{i=1}^{n} x,$$

где i – номер шага вычисления (умножения);

n – число шагов.

При описании функции надо с помощью списков параметров (формальных и фактических) связать формальный параметр *x* с основанием степени, параметр *n* — с показателем. Поскольку в задаче требуется вычислить три раза операцию возведения в степень, то в главной функции main() будет организован вызов этой функции из выражения.

```
#include "stdafx.h"
```

```
float ST(float x, int  n)    // начало функции ST
   {
    int   i;
    float   P;
    P=1;
    for( i=1; i<= n; i++)
    P = P*x;
    return  P;
   }            // конец функции ST
  void main()
  {
   float a,Z;
   printf("Введите число a:");
   scanf("%f",&a);
   Z = (ST(a, 5) + ST(1/a, 5))/(2* ST(a, 7));
   printf("Z=%f\n", Z);
  }
```

В процессе выполнения программы после ввода заданного числа *a* вычисляется значение Z по формуле. В данной формуле обращение к функции ST осуществляется с помощью трех операндов.

При каждом вызове функции происходит соответствующая замена формальных параметров (x, n) на фактические. Вычисленный результат возвращается в выражение. Далее вычисляется значение Z и выводится на экран.

Для того чтобы функция могла быть вызвана, т.е. была доступна, необходимо, чтобы до ее вызова о ней было известно компилятору. Это значит, что либо мы текст функции должны поместить до функции, из которой она вызывается (например, из main()), либо перед main() записывается прототип функции.

Прототип функции по форме аналогичен заголовку функции, в конце которого ставится ";".

Например, рассмотрим предыдущую программу:

```
#include "stdafx.h"
              //    прототип функции ST
  float ST(float , int  );
            // функция main
  void  main()
  {  float a,Z;
     printf("Введите число a:");
     scanf("%f",&a);
     Z = (ST(a, 5) + ST(1/a, 5))/(2* ST(a, 7));
```

```
    printf("Z=%f\n", Z);
  }
// функция ST
 float ST(float x, int  n)
 {
  int   i;
  float   P;
  P=1;
  for( i=1; i<= n; i++)
        P = P*x;
  return  P;
 }
```

Согласование параметров

Формальные и фактические параметры должны быть согласованы друг с другом по количеству, типу и порядку следования. Это означает, что количество формальных параметров должно быть равно количеству фактических параметров, и каждый формальный параметр должен иметь тот же тип и занимать в списке то же место, что и соответствующий ему фактический параметр.

Механизм замены параметров

В языке С++ существует два механизма передачи параметров в функции: по значению и по адресу.

Как правило, параметры функции (кроме массивов и указателей) передаются по значению. Параметры, передаваемые по значению, играют роль входных параметров. Фактическим параметром, передаваемым по значению, может быть константа, переменная или выражение. Для них при вызове функции в памяти компьютера временно выделяются ячейки, в которые передаются копии значений фактических параметров. При выполнении функции значения в этих ячейках могут измениться, однако соответствующие им фактические параметры останутся без изменения.

Результат работы функции возвращается в точку вызова с помощью оператора return. Таким образом, функция может вернуть только одно скалярное значение. Если функция должна вернуть несколько результатов, то этот возврат реализуется с помощью указателей, т.е. параметров, передаваемых по адресу.

При передаче параметров по адресу все действия в процедуре выполняются непосредственно над фактическим параметром, а не его копией. Поэтому любое изменение формального параметра приводит к изменению соответствующего ему фактического параметра.

Рассмотрим два примера, иллюстрирующих механизмы передачи параметров

Пример 1

```
#include "stdafx.h"
void  Z (int  y)
 {
y=1;
 }
void main()
{ int x;
 x=0;
 Z(x);
 printf("x=%d", x);
}
```

Пример 2

```
#include "stdafx.h"
void  Z (int  *y)
{
     *y=1;
 }
  void main()
{ int x;
   x=0;
   Z(&x);
   printf("x=%d", x);
}
```

В примере 1 процедура Z содержит формальный параметр y, который передается по значению, поэтому его изменение в процедуре (y=1;) не влияет на значение фактического параметра x. После выполнения программы на экран будет выведено: x=0.

В примере 2 формальный параметр y – это указатель. Это означает, что в функции изменяется значение в той ячейки памяти, адрес которой передавался в функцию Z, т.е. по адресу фактического параметра X. На экран будет выведено: x = 1.

Параметры-массивы в функциях

Если мы хотим передать в подпрограмму отдельный элемент массива, то в качестве соответствующего ему формального параметра указывается простая переменная того же типа.

Массивы, так же как и простые переменные, можно передавать в функции в качестве параметров. Так как имя массива – это адрес, то передача массива происходит всегда по адресу.

Рассмотрим, например, функцию, вычисляющую среднее значение элементов массива. Желательно сделать ее так, чтобы в нее можно было передавать массивы любого размера и она всегда правильно вычисляла результат. В языке C++ функции не могут самостоятельно определять размер массива, поэтому он должен быть обязательно одним из параметров.

```
#include "stdafx.h"
int Sum ( int A[], int N )
{
int i, sum;
sum = 0;
for ( i = 0; i < N; i ++ )
```

```
sum += A[i];
return sum/N;
}
void main()
{
    int x[5]={1,2,3,4,5},y[3]={11,22,33};
    printf("\nsr x=%d   sr  y=%d\n",
    Sum(x,5),Sum(y,3));
}
```

Обратите внимание, что в заголовке функции размер массива указан отдельно. Нельзя объявлять массив-параметр как *A[N]*, а только как *A[]* или **A*.

Если в функцию передаётся двумерный массив, то описание соответствующего аргумента функции должно содержать количество столбцов; количество строк - несущественно, поскольку фактически передаётся указатель. Например: int X[ ][5], или X[5][5].

Рассмотрим пример функции, перемножающей матрицы A и B; результат - матрица C.

```
const nmax = 10;
void product(int A[][nmax], int B[][nmax],int C[][nmax], int m, int n,
int k)
{
/* m - число строк в матрице A;
  n - число строк в матрице B (должно быть равно числу
столбцов в матрице A);
  k - число столбцов в матрице B.
 */
   for (int i=0; i< m; i++)
   for (int j=0; j< k; j++){
   C[i][j]=0;
   for (int l=0; l< n; l++)
      C[i][j] += A[i][l]*B[l][j];
 }
 }
```

В приведённом примере есть недостаток - здесь заранее фиксируется максимальная размерность матриц. Но использоваться может только часть памяти.

Рекурсия

В теле функции известны все объекты, описанные во внешнем блоке, т.е. все глобальные переменные и имя самой функции.

Таким образом, внутри любой функции можно вызывать любую доступную функцию, в том числе и саму себя. Ситуация, когда функция вызывает саму себя, называется рекурсия.

Рекурсия возможна благодаря тому, что при вызове функции создаются новые экземпляры локальных переменных, которые сохраняются во внутреннем стеке машины. Стек функционирует по принципу LIFO - Last In – First Out (последний вошел – первый вышел).

Переменные помещаются в стек одна за другой и выбираются из стека в обратном порядке.

Обязательным элементом всякого рекурсивного процесса является утверждение, определяющее условие завершения рекурсии. Оно называется опорным условием рекурсии.

Если опорное условие выполняется, то может быть задано некоторое фиксированное значение, заведомо достижимое в ходе вычисления. Это позволит организовать своевременную остановку рекурсивного процесса.

Рассмотрим пример вычисления факториала 5.

$5! = 1 \times 2 \times 3 \times 4 \times 5$, где $1 \times 2 \times 3 \times 4$ - это 4!

T.e $5! = 4! \times 5$

Факториал нуля равен 1. Отсюда формула вычисления N-факториала:

$$N! = \begin{cases} 1, & если\ N = 0 \\ (N-1)! N, & если\ N > 0 \end{cases}$$

Реализуем вычисление факториала в виде функции:

```
#include "stdafx.h"
float fact(int n)
{
   if (n==0)
        return 1;
    else
       return (fact(n-1)*n);
}
void main()
{
    printf("факториал 5=%f\n",fact(5));
}
```

Примеры программирования задач с использованием подпрограмм

При программировании задач возможны несколько вариантов. Выбор правильного (более эффективного и надежного) варианта зависит от опыта программиста

Задача 1

Даны два вектора: $\bar{x} = \{x_i\}$; $i = \overline{1,8}$ и $\bar{y} = \{y_i\}; i = \overline{1,10}$. Вычислить значение: $D = \frac{\Delta x}{\Delta y}$, где $\Delta x = |mx - sx|$; $\Delta y = |my - sy|$;

mx, my - максимальные компоненты векторов $\bar{x}$ и $\bar{y}$ соответственно;

sx, sy - средние значения компонент векторов $\bar{x}$ и $\bar{y}$ соответственно.

Решение:

```
#include "stdafx.h"
#include <math.h>
  float Mod_Otkl (float *a, int n)  /* функция для нахождения максимального компонента и среднего значения  в любом массиве */
    {
      float ma, sa, Da;
       int i;
       ma =-10000;
        sa=0;
        for (i =0;i<n;i++)
          {
                  if (a[i]>ma) ma= a[i];
                  sa += a[i];
           }
          sa = sa/n;
          Da= fabs(ma - sa);
          return Da;
       }
void main()
{ float X[10],Y[10];
  int   i;
    float Dx, Dy, D;
     printf("Введите массив X:\n");
     for (i =0;i<8;i++)
      scanf("%f",&X[i]);
     printf("Введите массив Y:\n");
```

```
    for (i =0;i<10;i++)
       scanf("%f",&Y[i]);
    Dx = Mod_Otkl(X, 8); //вызов функции Mod_Otkl для массива X
    Dy = Mod_Otkl(Y,10); //вызов функции Mod_Otkl для массива Y
    D = Dx/Dy;
   printf("D=%f\n",D);
   }
```

Задача 2

Даны две матрицы: $A = \{a_{ij}\}_{5x6}$ и $B = \{b_{ij}\}_{4x7}$.

Вычислить разность: C = *KA* - *KB*, где *KA* и *KB* - количество положительных элементов в матрицах *A* и *B* соответственно.

```
#include "stdafx.h"
int CP(float D[7][7], int m, int n) /* функция для подсчета
количества положительных элементов в матрице */
 { int   i, j, KD;
KD=0;
for (i=0;i<m;i++)
   for (j=0;j<n;j++)
         if (D[i][j]>0) KD ++;
return KD;
}
int main()
{
   float     A[7][7], B[7][7];
   int  i,j, C;
   printf("Введите матрицу A\n");
   for (i=0;i<5;i++)
    for (j=0;j<6;j++)
       scanf("%f",&A[i][j]);
  printf("Введите матрицу B\n");
  for (i=0;i<4;i++)
   for (j=0;j<7;j++)
     scanf("%f",&B[i][j]);

  C= CP(A, 5,6)- CP(B, 4, 7);
  printf("C=%d\n", C);
  return 0;
}
```

Задача 3

На плоскости декартовыми координатами заданы 10 точек:

$$\{x_1,y_1\},\{x_2,y_2\}, \ldots, \{x_{10},y_{10}\}.$$

Вывести полярные координаты точки, имеющей наибольший полярный радиус. Вычисление полярных координат одной точки оформить подпрограммой. Расчетные формулы для вычисления полярных координат следующие:

$\rho = \sqrt{a^2 + b^2}$; $\varphi = arctg\frac{b}{a}$;, где a и b - координаты точки.

Решение:

```
#include "stdafx.h"
#include<math.h>
void PK(float a, float b, float *ro, float *fi)  /* функция для расчета полярных координат точки   */
 {
        *ro = sqrt(a*a + b*b);
         *fi = atan(b/a);
 }
int main()
{
   float X[10], Y[10], R[10], F[10];
   int i, n;
   float max;
   printf("Введите абсциссы 10 точек\n");
   for (i=0;i<10;i++) scanf("%f",&X[i]);
   printf(" Введите ординаты 10 точек\n");
   for (i=0;i<10;i++) scanf("%f",&Y[i]);
   max = 0;
   for (i= 0;i< 10;i+ +)
     {
       PK(X[i],Y[i],&R[i],&F[i]);
       if (R[i]>max)     // поиск максимального  радиуса
                 {
                       max =R[i];
                       n=i;
                 }
      }
     printf("romax=%f fimax=%f\n", R[n],F[n]);
   return 0;
 }
```

Задача 4

Для заданных квадратных матриц: $A = \{a_{i,j}\}_{3x3}$ и $B = \{b_{i,j}\}_{4x4}$ вычислить симметричные матрицы по правилу:

$$x_{ij} = x_{ji} = \frac{y_{ij} + y_{ji}}{2}$$

Решение:

```
#include "stdafx.h"
 void SM(float Y[4][4], int n, float X[4][4])
   {
   int i,j;
   for (i=0;i<n;i++)
   for (j=i;j<n;j++)
      {
              X[i][j]=(Y[i][j] + Y[j][i])/2;
              X[j][i] =X[i][j];
         }
   }
void main()
{
     float  A[4][4], B[4][4], C[4][4], D[4][4];
     int i,j;
     printf("Введите матрицу A\n");
     for (i=0;i<3;i++)
     for (j=0;j<3;j++) scanf("%f",&A[i][j]);
     printf("Введите матрицу B\n");
     for (i=0;i<4;i++)
     for (j=0;j<4;j++) scanf("%f",&B[i][j]);
     SM(A, 3,C);
     SM(B, 4, D);
     printf("Симметричная матрица C\n");
     for (i=0;i<3;i++)
     {for (j=0;j<3;j++) printf("%8.2f",C[i][j]);
      printf("\n");
     }
     printf("Симметричная матрица D\n");
     for (i=0;i<4;i++)
     {for (j=0;j<4;j++) printf("%8.2f",D[i][j]);
      printf("\n");
     }
}
```

Текстовые данные

В языке С++ текстовая информация представляется двумя типами данных: с помощью символов и строк - массивов символов.

Символьный тип данных

Значением данных символьного типа является любой символ из набора всех символов компьютера или его код. Каждому символу соответствует порядковый номер (код) в диапазоне 0..255. Для кодировки символов первой половины диапазона используется код ASCII (американский стандартный код для обмена информацией), или более современные стандарты в последних версиях языка Си.

При написании программ символьные данные могут быть представлены либо константами, либо переменными.

Символьная константа представляет собой одиночный символ, заключенный в апострофы, например:

'Y' '!' '_' 'Д'

Символьная переменная объявляется с помощью ключевого слова char, например:

char cr;

Во внутренней памяти компьютера каждый символ занимаем 1 байт.

Ввод-вывод символьных данных

Для ввода символьных данных используются функции: scanf() – форматированный ввод, getchar() или getch() – специальные функции для ввода символа. Для форматного ввода и вывода символьных констант используется спецификатор (формат) %c. Необходимо помнить, что нажатие любой небуквенной клавиши при вводе ([пробел], [Enter] и др.) будет значимым и восприниматься как символ.

Пример 1. Организовать ввод символьных переменных:

```
a='i'   b='j'   c='k'
main()
{   char a,b,c;
        printf("Введите исходные данные");
        scanf("%c%c%c",&a,&b,&sc);
    }
```

При вводе символы набираются без апострофов и пробелов:

ijk [Enter]

Символ клавиши [Enter] выходит за пределы списка ввода, поэтому он игнорируется.

При вводе символьной информации с помощью функции getchar() надо помнить, что функция переводит программу в состояние ожидания, но при нажатии клавиши символ выводится на

экран. А, например, при выполнении следующего фрагмента программы

```
printf("Введите исходные данные");
a=getch();b=getch();c=getch();
```

переменные будут введены, но на экране их значения не отразятся.

Для вывода символьных данных используются функции printf() и putchar().

Пример 2. Организовать вывод указанных выше переменных на экран в одну строку. Запись оператора вывода будет следующей:

```
printf("%c%c%c\n",a,b,c);
```

На экране будет отображено:

ijk

Если использовать для вывода функцию putchar():

```
putchar(a); putchar(b); putchar(c);
```

на экране будет отображен тот же результат.

Обработка символьных данных

Поскольку символы в языке С++ упорядочены, к ним можно применять операции отношения (>, >=, <, <=, = =, !=). Это дает возможность записывать логические выражения с символьными данными в условных операторах. Например, оператор

```
if (ch =='!' ) ch ='.';
```

сравнивает значение переменной ch с символом '!' и в случае их равенства следующая команда заменяет в символьной переменной ch восклицательный знак точкой.

Символьные данные могут использоваться и в операторах цикла for. Так, при выполнении операторов:

```
for( ch='a'; ch>='d'; ch++)  printf("%c",ch);
```

в строку экрана выводится последовательность:

a b c d

Если значение символьной переменной вывести с помощью спецификатора для целых чисел %d, то на экране отобразится код символа. Например:

```
for(ch='a'; ch>='d'; ch++)  printf("%d ",ch);
```

на экран будет выведено:

97 98 99 100

Над символьными данными можно выполнять арифметические операции сложения и вычитания. Так, например, операция ch++; из предыдущего примера увеличивает код символа, хранящегося в переменной ch на 1. Или, выполняя операцию ch='a'-'A'; будет получена разница кода большой (А) и маленькой буквы (а) латинского

алфавита. Так, например, если в символьной переменной ch1 хранится маленькая буква алфавита, то, выполнив действия:

```
char ch,ch1,ch2;
ch='a'-'A';
ch1='k';
ch2=ch1-ch;
printf("%c-%d %c-%d\n",ch1,ch1,ch2,ch2);
```

в переменную ch2 запишется та же буква, только большая, а на экран будет выведено

k-107 K-75

Строки

Значением строки является любая последовательность символов. Причем для компьютера – это набор байтов.

Строковая константа - это строка, заключенная в кавычки, например:

"Язык программирования"

Строковая переменная или строка представляет собой массив символов, поэтому и объявляется она именно так:

```
char st[30];
```

В квадратных скобках указывается максимальное число символов в строке st.

Под значение строковой переменной в памяти компьютера отводится MAX байт, пронумерованных от 0 до MAX-1, где MAX - объявленный размер строки.

Строка отличается от несимвольного массива тем, что она заканчивается кодом 0 - признаком окончания строки. По местоположению этого специального символа определяется фактическая длина строки.

Начальное значение строки можно задать при ее объявлении следующим образом:

```
char s[80] = "Язык программирования Си";
```

Символы в кавычках будут записаны в начало массива *s*, а затем - признак окончания строки *'\0'*.

При описании строки можно также написать так:

```
char s[] = "Язык программирования Си";
```

В этом случае компилятор подсчитает символы в кавычках, выделит памяти на 1 байт больше и занесет в эту область саму строку и завершающий ноль.

Ввод-вывод строковых данных

При вводе строк, как и символов, используется функция scanf(). При этом для форматного ввода и вывода строк используется

спецификатор %s. Однако нажатие клавиши [Enter] или клавиши [пробел] не является значимым символом. При вводе строки с помощью функции scanf() нажатие одной из этих клавиш формирует символ конца строки. Таким образом надо помнить, что функция scanf() позволяет записать в строку только одно слово.

Пример. Организовать ввод ФИО студента.

```
char fam[20];
printf ("Введите фамилию и инициалы студента");
scanf("%s", fam);
```

На клавиатуре строка набирается без кавычек, например:

Программирование [Enter]

Одновременно с вводом строки в байт с индексом восемь запишется символ с кодом 0. Инициалы студента в эту строку записаны не будут, так как пробел после фамилии будет воспринят командой scanf как конец строки.

Для ввода текста содержащего пробелы следует использовать специальную функцию gets(). При вводе строки с помощью этой функции только нажатие клавиши [Enter] сформирует символ конца строки.

Так, например, в предыдущей задаче:

```
char fam[20];
printf("Введите фамилию и инициалы студента");
        gets(fam);
```

использование функции gets() позволит записать в строку fam не только фамилию, но и инициалы.

Вывод строк осуществляется с помощью функции printf() и специальной функции puts(). Например, оператор

```
printf("|%20s|",fam);
```

выведет на экран в правую часть поля из 20 позиций строку fam:

```
|      Иванов С.В.|
```

Специальная функция puts() позволяет вывести содержимое строки и переводит курсор на следующую строку. Например:

```
putchar('|'); puts(fam); putchar('|');
```

приведет к получению следующего результата:

```
|Андреева С.В.
|
```

Последний символ будет выводиться в следующей строке экрана.

Обработка строковых данных

К любому символу строки можно обратиться как к элементу одномерного массива, например, запись st[i] определяет i-ый символ в строке st. Поэтому при решении некоторых задач обработку

строковых данных можно проводить посимвольно, организуя циклы для просмотра строки.

Например: Дано предложение. Определите количество слов в нем.

Решение:

Слова в предложении разделяются пробелами. Подсчитав количество пробелов, можно определить количество слов, учитывая, что между словами введен только один пробел.

```
#include "stdafx.h"
#include<string.h>
int main()
{ char slova[120];
  int i, n, k=1;
  printf("Введите предложение\n");
  gets(slova);
  n= strlen(slova); // функция strlen() возвращает длину строки
  for(i=0;i<n; i++)
      if(slova[i]==' ')k++; //поиск и подсчет пробела
  printf("k=%d\n",k);
 return 0;
}
```

Стандартные функции обработки строк

Большинство действий над строками реализуется с помощью стандартных функций. Библиотека языка Си содержит большое количество таких функций, прототипы которых определяются в заголовочном файле string.h. Рассмотрим некоторые из них.

Сравнение строк:

strcmp(str1,str2) – сравнивает две строки str1 и str2 и возвращает 0, если они одинаковы; результат отрицателен, если str1<str2 и положителен, если str1>str2.

strncmp(str1, str2, kol) – сравниваются части строк str1 и str2 из kol символов. Результат равен 0, если они одинаковы.

Сравнение двух строк выполняется последовательно слева неправо с учетом кодировки символов. Например, сравнивая стоки st1 и st2

```
char st1[10]="Пример";
char st2[10]="ПРимер";
int a;
 if (strcmp(st1,st2)>0)
     a=1;
else
```

a=2;

переменной a будет присвоено значение 1, так как код символа 'p' больше кода символа 'P'.

Сцепление строк

strcat(str1,str2) - сцепление строк в порядке их перечисления.

strncat(str1,str2,kol) – приписывает kol символов строки str2 к строке str1.

Функция служит для объединения двух строк в одну. Например, в результате выполнения операторов:

```
char fam[] = "Андреева С.В. ";
char pr[7]= "      "; //7 пробелов
strcat(fam ,pr);
 printf("|%20s|", fam);
```

на экран выведется строка:

|Андреева С.В. |

Заметим, что строка вывода занимает поле в 20 позиций, а переменная fam располагается в левой части поля.

Определение длины строки

strlen(str) – определяет длину строки str.

Пример. Определить длину строки

```
char fam[] = "Андреева С.В.";
printf("%d",strlen(fam));
```

функция strlen() вернёт значение равное 13 (символов).

Копирование строк

strcpy(str1,str2) – копирует строку str2 в строку str1.

strncpy(str1, str2, kol) – копирует kol символов строки str2 в строку str1.

Пример. Скопировать фамилию сотрудника в переменную fam и вывести на экран.

```
#include "stdafx.h"
#include<string.h>
int main()
 { char fam[15];
   char *str = " Андреева С.В.";
   strcpy(fam. str);
   printf("|%s|\n", fam);
   return 0;
 }
```

В результате выполнения данных операторов на экран будет выведена строка:

|Андреева С.В.|

Поиск символа в стоке

strchr(st, ch) - функция поиска адреса символа ch в строке st. Результатом выполнения поиска является адрес найденного символа в строке st, иначе возвращается нулевой адрес. Чтобы вычислить порядковый номер символа ch в строке, можно из адреса P вычесть адрес начала строки.

Пример. В заданной фамилии определить порядковый номер символа 'n'.

```
#include "stdafx.h"
#include<string.h>
int main()
{ char fam[] = "Ivanov";
  char fam[20];
  char  a='n';
  char *p;
   p=strchr(fam,a);
   if(p)
       printf("|%s|%d\n", fam, p-fam);
   else
    printf("нет такого символа в фамилии!\n");
 return 0;
}
```

Пример программы для задачи с текстовыми данными

Исходным текстом является предложение, заканчивающееся точкой. Слова в предложении отделяются друг от друга одним пробелом. Определить самое длинное слово в предложении.

```
#include "stdafx.h"
#include<string.h>
int main()
{ char slovo[12],x[120];    // описание строк
int i,m=0,n,k=0;
gets(x);                    // ввод строки x
for(i=0; i<strlen(x); i++) //цикл до конца строки x
if(x[i]!=' ')  k++;         // считаем символы до пробела
else
   { if (k>m){ m=k;n=i;}  //поиск max значения счетчика k
    k=0;
   }
    k=0;
    for(i=n-m;i<n;i++)   //выбор из строки x самого длинного слова
slovo[k++]=x[i];
```

```
slovo[k]=0;
   printf("%s   \n%s\n",slovo,x); /*вывод найденного слова и всей
                                строки x  */
printf("%d  %d\n",strlen(slovo),strlen(x));//вывод их длин
return 0;
    }
```

Задания для самостоятельного решения

Подготовить текст исходного предложения в соответствии с вариантом задачи, указанным в таблице (предложение должно заканчиваться точкой, разделитель слов в предложении - пробел). Составить программу и выполнить ее на ПК.

Таблица 12 – Варианты заданий

Вариант	Условие задачи
1	Из заданного предложения, начиная с первой встретившейся буквы 'а', скопировать в подстроку все символы до первой встретившейся буквы ' к'.
2	В исходном предложении все символы пробела заменить символами подчеркивания.
3	Вывести символы, которые встречаются в исходном предложении по одному разу.
4	В исходном предложении удалить все символы пробела. Вывести преобразованный текст и количество удаленных пробелов.
5	Определить, сколько раз в заданном предложении встречается сочетание двух первых букв.
6	В качестве исходного предложения задать арифметическое выражение, записанное на языке С: $e^{\frac{\sqrt[3]{\ln\lvert\sin x\rvert}}{a+b}} + 1$. Проверить, соблюдается ли баланс открывающихся и закрывающихся скобок. Если равенство соблюдается, то вывести соответствующее сообщение, в противном случае вывести количества открывающихся и закрывающихся скобок.
7	В исходном предложении поставить между словами произвольное число пробелов. Отредактировать текст, удалив лишние пробелы и оставив только по одному пробелу.

Продолжение таблицы 12

1	2
8	В заданном предложении найти слово, в котором доля буквы 'a' максимальна. Вывести найденное слово и посчитать, сколько раз буква 'a' встречается в этом слове.
9	В заданном предложении символы самого длинного слова заменить символами 'x'.
10	Сформировать числовой массив *N*, элементы которого указывают длину каждого слова в исходном предложении.
11	В исходном предложении перед каждым словом поставить знак ?.
12	Составить строку, содержащую первые буквы из каждого слова заданного предложения.
13	В каждом слове заданного предложения поменять местами первую букву и последнюю.
14	Указать, сколько раз каждый символ встречается в заданном предложении.
15	В заданном предложении найти самое короткое и самое длинное слово.
16	Исходное предложение задать в виде арифметического выражения, записанного на языке С. Последовательность символов 'x[i]' в заданном предложении заменить последовательностью 'a[j]'. Вывести преобразованный текст, а также число произведенных замен.
17	В заданном предложении указать слово, в котором доля букв 'т' и 'р' максимальна. Вывести найденное слово и количество букв 'т' и 'р', встретившихся в этом слове.
18	В заданном предложении удалить все запятые. Вывести преобразованный текст, а также число удаленных запятых.
19	В заданном предложении заменить все строчные латинские буквы на прописные. Определить, сколько раз в преобразованном предложении встречается сочетание 'AB'
20	В заданном предложении в конце каждого слова поставить многоточие. Вывести преобразованное предложение, а также длину полученного текста.

2.9 Структуры данных

Понятие структуры. Очень часто при обработке информации приходится работать с блоками данных, в которых присутствуют разные типы данных. Например, информация о книге в каталоге библиотеки включает в себя автора, название книги, год издания, количество страниц и т.п.

Для хранения этой информации в памяти не подходит обычный массив, так как в массиве все элементы должны быть одного типа. Конечно, можно использовать несколько массивов разных типов, но это не совсем удобно.

В современных языках программирования существует особый тип данных, который может включать в себя несколько элементов более простых (причем разных!) типов.

Структура - это тип данных, который может включать в себя несколько полей – элементов разных типов (в том числе и другие структуры).

В общем случае при работе со структурами следует выделить четыре момента:

- объявление и определение типа структуры,
- объявление структурной переменной,
- инициализация структурной переменной,
- использование структурной переменной.

Данные типа структура, как и массивы, относятся к сложным структурам данных. Структура состоит из фиксированного числа элементов, называемых полями. Однако, в отличие от массива, поля могут быть различного типа. Например, структурой можно считать строку экзаменационной ведомости:

Иванов С.В. 4 5 5

Данная структура состоит из четырех полей: одно поле - строка (ФИО студента) и три числовых поля (оценки студента по предметам).

Поскольку структура - это новый тип данных, его надо предварительно объявить в начале программы. Определение типа структуры делается так:

```
struct Имя
{
<тип> <имя 1-го поля>;
<тип> <имя 2-го поля>;
……….
<тип> <имя последнего поля>;
};
```

Например, задание типа записи строки экзаменационной ведомости выглядит следующим образом:

```
struct student
{
char  fam[20];
int mathematics, informatics, history;
};
```

Тогда при описании переменных можно использовать этот тип:

```
struct student  X;
```

Здесь X - переменная типа структура; struct student - тип; fam, mathematics, informatics, history - поля структуры.

Отметим, что определение типа структуры может быть задано в программе на внешнем уровне, при этом имя пользовательского типа имеет глобальную видимость (при этом память не выделяется). Определение типа структуры также может быть сделано внутри функции, тогда имя типа структуры имеет локальную видимость.

Чтобы упростить обращение к структурному типу, можно воспользоваться директивой #define.

Например, для предыдущей структуры:

```
#define stud struct student
stud
{
   char  fam[20];
   int mathematics, informatics, history;
  };
```

Теперь идентификатор stud заменит в любом месте программы громоздкий тип struct student.

Теперь описании переменной типа структура будет выглядеть так:

```
stud  X;
```

В более поздних версиях языка С ключевое слово typedef позволяет в программе создать синоним типа, который удобно использовать для объявления переменных структурного типа. Например:

```
typedef struct student
{
char  fam[20];
   int mathematics, informatics, history;
} STUD;
```

Идентификатор STUD представляет собой синоним типа struct student. С помощью синонима STUD можно объявить переменную:

STUD X;

Для обращения к отдельным полям переменной типа структура используется составное имя:

<имя переменной>.<имя поля>

Например, для переменной X обращения к полям записываются следующим образом:

X.fam, X. mathematics, X. informatics, X. history.

При размещении в памяти структурной переменной можно выполнить ее инициализацию. Неявная инициализация производится для глобальных переменных, переменных класса static. Структурную переменную можно инициализировать явно при объявлении, формируя список инициализации в виде константных выражений.

Формат: struct Имя переменная ={значение1, ... значениеN};

Внутри фигурных скобок указываются значения полей структуры, например:

struct student X={"Андреева С.В.", 4, 5, 5};

при этом первое значение записывается в первое поле, второе значение – во второе поле и т. д., а сами значения должны иметь тип, совместимый с типом поля.

Обработка структур

Над структурами возможны следующие операции:

- присваивание значений структурной переменной;
- получение адреса переменной с помощью операции &;
- ввод и вывод значений переменных структурного типа;
- сравнение полей переменных структурного типа.

Операция присваивания применима, как к отдельным полям переменной структурного типа, так и к переменным в целом.

При присваивании полям структуры значений, необходимо учитывать типы полей. Например:

```
#include "stdafx.h"
#include <string.h>
 typedef struct student  // описание структуры
{
char  fam[20];
   int mathematics, informatics, history;
} STUD;
  main()
  { STUD X;          //описание переменной структурного типа
```

```
    strcpy(X.fam, "Андреева С.В. "); /*копирование фамилии в поле fam переменной X */
        X. mathematics=4;
        X. informatics=5;
        X. history=5;
        printf("\n %s %d %d %d", X.fam, X.mathematics, X.informatics,X.history); /*вывод информации из полей переменной X
    }
```

Для структурного типа возможно присваивание значений одной структурной переменной другой структурной переменной, при этом обе переменные должны иметь один и тот же тип.

Присваивание значения одной переменной другой выполняется путем копирования значений соответствующих полей, например:

```
main()
{ STUD X, Y;
   strcpy(X.fam, "Андреева С.В. ");
   X. mathematics=4;
   X. informatics=5;
   X. history=5;
   Y=X;      // копирование информации из X в Y
   printf("\n %s  %d  %d  %d",
   Y.fam, Y.mathematics, Y.informatics, Y.history);
}
```

В результате выполнения этого копирования в Y.fam будет записано значение "Андреева С.В.", а в Y. mathematics – оценка 4, в. Y.informatics – 5 и в Y.history – тоже 5.

Работа со структурной переменной обычно сводится к работе с отдельными полями структуры. Такие операции, как ввод с клавиатуры, сравнение полей и вывод на экран применимы только к отдельным полям. Например, в выше приведенном примере вывод информации о студенте осуществляется выводом значений отдельных полей с помощью функции printf().

С помощью структурного типа можно формировать массивы записей. Так, например информацию о 20 студентах можно хранить в массиве из 20 элементов структурного типа:

```
typedef struct student
{
char fam[20];
```

```
    int mathematics, informatics, history;
} STUD;
main()
{ STUD Spis[20];
  . . .
}
```

Пример задачи с использованием структурированных данных

Рассмотрим пример программы, в которой вводится информация об абонентах сети: ФИО, телефон и возраст. В программе выбираются абоненты моложе 25 лет и их список выводится в алфавитном порядке.

```
#include "stdafx.h"
#include<conio.h>
#include<stdlib.h>
typedef struct abon  //описание структуры
{ char f[10],i[10],o[10];
  long tel;
  int voz;
}ABON;
const int n=5;
 int i,k,j;
 int main()
 { ABON z[n],y[n];    //описание массивов структур
   ABON x;
for (i=0; i<n; i++)//ввод в цикле исходной информации о пяти
абонентах
{printf("Введите ФИО абонента:");
 scanf("%s%s%s",z[i].f, z[i].i, z[i].o);
 printf("введите его телефон и возраст:");
 scanf("%ld%d",&z[i].tel,&z[i].voz);
}
printf("-------------------------------------------------\n");
printf("| Фамилия |  Имя   | Отчество| Телефон |  Возраст |\n");
printf("-------------------------------------------------\n");
for (i=0;i<n;i++) //вывод в цикле информации о пяти абонентах
 printf("|%9s|%8s|%9s|%7ld  |  %5d   |\n", z[i].f,z[i].i,z[i].o,
z[i].tel,z[i].voz);
}
printf("-------------------------------------------------\n");
for (i=0;i<n;i++)
```

```
    {if(z[i].voz<25) // поиск абонента моложе 25 лет
     y[k++]=z[i];
     }
   for(i=1;i<k;i++)  //сортировка списка абонентов моложе 25 лет
      for(j=k-1;j>=i;j--)
       if(y[j].f[0]<y[j-1].f[0])
        {x=y[j];
        y[j]=y[j-1];
        y[j-1]=x;}
      printf("mologe 25\n");
   printf("-------------------------------------------------\n");
   printf("| Фамилия |  Имя   | Отчество| Телефон |  Возраст |\n");
   printf("-------------------------------------------------n");
   for (i=0;i<k;i++) // вывод отсортированного списка
    {printf("|%9s|%8s|%9s|%7ld  |  %5d   |\n", y[i].f,y[i].i, y[i].o,
y[i].tel,y[i].voz);
    }
   printf("-------------------------------------------------n");
   return 0;
    }
```

Полем структурной переменной может быть переменная любого типа, в том числе другая структурная переменная. Поле, представляющее собой структуру, называется вложенной структурой.

Файлы данных. Понятие файла

В большинстве своем *файлы* представляют собой именованные области внешней (дисковой) памяти, с которыми программы могут обмениваться информацией. Необходимость в таких обменах, во-первых, возникает, когда объем оперативной памяти недостаточен для хранения нужной информации. Во-вторых, программа может воспользоваться данными, полученными ранее другой программой и предусмотрительно записанными на диск. Наконец, в программах, требующих во время своей работы ввод исходных данных достаточно большого объема, целесообразно считывать эти данные из файла – данные в файле можно подготовить заблаговременно и тщательно выверить.

Файл – это информация, размещенная на каком-либо носителе (диске) или в буфере ввода/вывода устройства (клавиатура). Файлы предназначены только для хранения информации, а обработка этой информации осуществляется программами.

Для обмена данными файл должен быть открыт, по завершении этого процесса – закрыт.

Поток – это логический канал, предназначенный для выполнения операций ввода/вывода. Каждому файлу при его открытии ставится в соответствие поток.

В языке Си существуют стандартные потоки:

stdin – стандартный консольный ввод (клавиатура по умолчанию);

stdout – стандартный консольный вывод (монитор по умолчанию);

Стандартные потоки открываются при каждом запуске программы.

Работа с файлами

Для работы с файлами в программах на С++ 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.).

Поля структуры типа FILE доступны с помощью специальных С-функций. Для организации работы с файлом используется определенная последовательность действий.

Объявление переменной-указателя на структуру типа FILE, в которой будут храниться атрибуты файла

FILE *fl;

где *fl – указатель на файл.

Открытие файла

fl=fopen("путь к файлу","режим работы файла");

Параметр "путь к файлу" указывает размещение файла на диске. Он обязательно содержит имя файла и может содержать имя логического диска и путь к нему по папкам.

Параметр "режим работы файла" показывает, как будет использоваться файл:

"w" – для записи данных (вывод);

"r" – для чтения данных (ввод);

"a" – для добавления данных к существующим записям.

Примеры открытия файлов:

```
FILE *fin,*out;
        fin= fopen("My_file1","r");
        out=fopen("My_file2","w");
```

Функция fopen() возвращает значение указателя на структуру типа файл. Если файл по каким-либо причинам не открывается, функция fopen() возвращает значение NULL.

Рассмотрим особенности режимов открытия файлов. Если файл открывается в режиме записи данных "w", то указатель текущей позиции устанавливается на начало файла. Если указанный в функции fopen() файл не существует, то он создается. Необходимо помнить, что открытие существующего файла в режиме "w" приводит к уничтожению его старого содержания.

Открытие файла для чтения в режиме "r" возможно только для созданного ранее файла, при этом указатель текущей позиции устанавливается на начало файла. Если открываемый на чтение файл не существует, функция fopen() возвращает пустой указатель со значением NULL.

Если файл открывается в режиме добавления данных "a", то указатель текущей позиции устанавливается на конец файла. Данные, ранее помещенные в файл, остаются без изменений. Если указывается несуществующий файл, то он создается заново.

В C++ файл можно открыть для чтения и/или записи в текстовом или бинарном (двоичном) режиме. Поэтому можно указать дополнительные условия режима открытия файла:

"b" – двоичный поток;

"t" – текстовый поток;

"+" – обновление файла.

Пример:

"r+" – чтение файла с обновлением, т. е. возможна перезапись данных с усечением;

"w+" – запись в файл и одновременно чтение;

"a+" – добавление данных и чтение.

Для поочередного выполнения чтения и записи в режиме "+" необходимо ручное позиционирование курсора.

Обработка открытого файла

Каким образом можно прочитать уже открытый файл или записать в него информацию? Для этого в языке С существуют специальные функции:

Чтение (ввод)	Запись (вывод)
fgetc()	fputc()
fscanf()	fprintf()
fgets()	fputs()
fread()	fwrite()

При каждой операции ввода/вывода указатель текущей позиции файла смещается на одну позицию в сторону конца файла.

Проверка признака конца файла

Так как при каждой операции ввода/вывода происходит перемещение указателя текущей позиции в файле, в какой-то момент указатель достигает конца файла. Структура типа FILE имеет поле – индикатор конца файла. Функция feof() проверяет состояние индикатора конца файла и возвращает значение 0, если конец файла не был достигнут, или значение, отличное от нуля, если был достигнут конец файла. Функция имеет единственный аргумент – указатель на FILE. Вызов функции в команде if:

if (! feof(fin))…

проверяет, что конец файла еще не достигнут.

Закрытие файла

После завершения обработки файла его следует закрыть с помощью функции fclose(). При этом разрывается связь указателя на файл с внешним набором данных. Освободившийся указатель можно использовать для другого файла. Формат вызова функции:

fclose(fin);

При нормальном завершении программы в большинстве операционных систем все открытые файлы закрываются автоматически, но рекомендуется закрывать все файлы, дальнейшая обработка которых в программе не предполагается, при помощи функции fclose().

Функции ввода/вывода

Простейший способ выполнить чтение из файла или запись в файл – использовать функции fgetc() или fputc().

Функция getc() выбирает из файла очередной символ; ей нужно только знать указатель на файл, например,

char Symb=fgetc(fin);

Если при обработке достигается конец файла, то функция getc() возвращает значение EOF(end of file).

Функция putc() заносит значение символа Symb в файл, на который указывает out. Формат вызова функции:

fputc(Symb,out);

Пример 1. Текст из файла my_char.txt выводится на экран. Если файл не найден, на экран выводится сообщение "File not found!":

```
#include "stdafx.h"
int main()
{
FILE *ptr;    //описание указателя на файл
unsigned char ch;
if ((ptr=fopen("my_char.txt","r"))!=NULL)//открытие файла для
чтения
```

```
{
ch=fgetc(ptr);  //чтение первого  символа из файла
while (!feof(ptr))  //цикл до конца файла
{
printf("%c",ch);    //вывод символа, взятого из файла
ch=fgetc(ptr);      //чтение  следующего символа  из файла
}
fclose(ptr);      //закрытие файла
}
else printf("\nFile not found!");
return 0;
}
```

В этом примере для чтения файла используется указатель ptr. При открытии файла производится проверка. Если переменной ptr присвоено значение NULL, то файл не найден; на экран выводится соответствующее сообщение, и программа завершается. Если ptr получил ненулевое значение, то файл открыт. Далее выполняется чтение символов из файла до тех пор, пока не будет достигнут конец файла (!feof(ptr)). Прочитанный символ помещается в переменную ch, а затем выводится на экран.

Пример 2. Записать в файл буквы, вводимые с клавиатуры. Ввод продолжается до нажатия клавиши F6 или CTRL/z (ввод символа EOF – конца файла):

```
#include "stdafx.h"
int main(void)
{
char c;
FILE *out;  // описание указателя на файл
out=fopen("Liter.txt","w");  //открытие файла для записи
while (  (c=getchar( ))!=EOF)  /*пока не будет введен символ
конца
fputc(c,out);  //  запись введенного символа в файл
fclose(out);   //закрытие файла
return 0;
}
```

Функции fscanf() и fprintf() выполняют форматированный ввод/вывод.

Чтение из файла выполняет функция fscanf():

fscanf(fin,["строка формата"],[список адресов переменных]);

Функция возвращает количество введенных из файла значений или EOF.

Запись в файл осуществляет функция fprintf():

fprintf(out,["строка формата"],[список переменных, констант]);

Возвращает количество выведенных в файл байт (символов) или EOF.

Строка формата функций fscanf() и fprintf() формируется так же, как было описано ранее в главе, посвященной консольному вводу/выводу и функциям printf() и scanf().

Следует заметить, что вызов функции

fscanf(stdin,[строка формата],[список адресов переменных]);

эквивалентен вызову

scanf([строка формата],[список адресов переменных]);

Аналогично,

fprintf(stdout, [строка формата],[список переменных, констант]);

эквивалентно

printf([строка формата],[список переменных, констант]);

Рассмотрим примеры программ, использующих эти функции.

Пример 3. В программе создается массив, состоящий из четырех целых чисел. Вывести массив в файл:

```
#include "stdafx.h"
#define n 4
int main()
{
int i=0;
int array[n]={4,44,446,4466}; //описание и инициализация массива
FILE *out;                    //описание указателя на файл
out=fopen("num_arr.txt","w"); //открытие файла для записи
for(;i<n;i++)
fprintf(out,"%6d",array[i]);  //запись в файл элемента массива
fclose(out);                  //закрытие файла
return 0;
}
```

В результате выполнения программы в файл num_arr.txt будет помещена следующая информация:

Пример 4. Имеется файл данных, содержащий целые числа, разделенные пробелами. Количество чисел в файле неизвестно. Требуется найти среднее арифметическое значение этих чисел:

```
#include "stdafx.h"
int main()
```

```
{
int S=0, count=0, numb;  //описание переменных
FILE *in;                //описание укателя на файл
if((in=fopen("num_arr.txt","r"))!=NULL)/*открытие файла для
чтения*/
{
while (!feof(in))        //пока не конец файла
{
fscanf(in,"%d",&numb);  //читать из файла число в переменную
numb
S+=numb;                 // добавить numb  в сумму
count++;                 //увеличиваем счетчик на 1 количество
printf("%d\n", numb);   //выводим значение numb на экран
}.
double aver=(double)S/count;  //считаем среднее значение
printf("Average=%lf\n", aver); //вывод среднего значения
fclose(in);                  //закрыть файл
}
else
printf("\nФайл не найден!");
return 0;
}
```

Чтение чисел из файла выполняется в переменную numb до тех пор, пока не будет достигнут конец файла. Одновременно ведется подсчет количества прочитанных чисел в переменной count и накопление суммы прочитанных чисел в переменной S. Переменные S и count целые, поэтому для правильного вычисления среднего арифметического, необходимо выполнить преобразование одной из этих переменных в формат double.

Рассмотрим другие библиотечные функции, используемые для работы с файлами (все они описаны в файле stdio.h):

1. Функция fputs( ) записывает строку символов в файл. Она отличается от функции puts( ) тем, что в качестве второго параметра должен быть записан указатель на переменную файлового типа.

Например:

```
fputs("Example", fp);
```

При возникновении ошибки возвращается значение EOF.

2. Функция fgets( ) читает строку символов из файла. Она отличается от функции gets( ) тем, что в качестве второго параметра должно быть указано максимальное число вводимых символов плюс единица, а в качестве третьего - указатель на переменную файлового

типа. Строка считывается целиком, если ее длина не превышает указанного числа символов, в противном случае функция возвращает только заданное число символов.

Рассмотрим пример:

fgets(string, n, fp);

Функция возвращает указатель на строку string при успешном завершении и константу NULL в случае ошибки либо достижения конца файла.

3. Функция fread( ) предназначена для чтения блоков данных из потока. Имеет прототип:

unsigned fread(void *ptr,unsigned size,unsigned n,FILE *fp);

Она читает n элементов данных, длиной size байт каждый, из заданного входного потока fp в блок, на который указывает указатель ptr. Общее число прочитанных байтов равно произведению n*size. При успешном завершении функция fread( ) возвращает число прочитанных элементов данных, при ошибке - 0.

4. Функция fwrite( ) предназначена для записи в файл блоков данных. Имеет прототип:

unsigned fwrite(void *ptr,unsigned size,unsigned n, FILE *fp);

Она добавляет n элементов данных, длиной size байт каждый, в заданный выходной файл fp. Данные записываются с позиции, на которую указывает указатель ptr. При успешном завершении операции функция fwrite( ) возвращает число записанных элементов данных, при ошибке - неверное число элементов данных.

5. Функция fseek( ) позволяет выполнять чтение и запись с произвольным доступом и имеет следующий прототип:

*int fseek(FILE *fp, long count, int access);*

Здесь fp - указатель на файл, возвращенный функцией fopen( ), count - номер байта относительно заданной начальной позиции, начиная с которого будет выполняться операция, access - способ задания начальной позиции.

Переменная access может принимать следующие значения:

0 - начальная позиция задана в начале файла;

1 - начальная позиция считается текущей;

2 - начальная позиция задана в конце файла.

При успешном завершении возвращается нуль, при ошибке - ненулевое значение.

6. Функция ferror( ) позволяет проверить правильность выполнения последней операции при работе с файлами. Имеет следующий прототип:

*int ferror(FILE *fp);*

В случае ошибки возвращается ненулевое значение, в противном случае возвращается нуль.

7. Функция remove( ) удаляет файл и имеет следующий прототип:

*int remove(char *file_name);*

Здесь file_name - указатель на строку со спецификацией файла. При успешном завершении возвращается нуль, в противном случае возвращается ненулевое значение.

8. Функция rewind( ) устанавливает указатель текущей позиции в начало файла и имеет следующий прототип:

*void rewind(FILE *fp);*

Работа с текстовыми файлами

Файлы бывают текстовые (в которых можно записывать только буквы, цифры, скобки и т.п.) и двоичные (в которых могут храниться любые символы из таблицы). В текстовых файлах не употребляются первые 31 символ кодовой таблицы ASCII (управляющие), а символы конца строки 0x13 (возврат каретки, CR) и 0x10 (перевод строки LF) преобразуются при вводе в одиночный символ перевода строки \n (при выводе выполняется обратное преобразование). Эти символы добавляются в конце каждой строки, записываемой в текстовый файл. При обнаружении в текстовом файле символа с кодом 26 (0x26), т.е. признака конца файла, чтение файла в текстовом режиме заканчивается, хотя файл может иметь продолжение.

Создать текстовый файл можно с помощью текстового редактора и с помощью программы. Рассмотрим пример создания текстового файла. Следующая программа записывает в файл строку из 65 символов, а затем переписывает в другой файл только английские буквы.

```
#include "stdafx.h"
#include <string.h>
int main()
{ FILE *f, *r;        // Указатели на файлы
  char ch, pr[65];
  char text[]="1,2,3,4,5 i caught a fish alive, 6,7,8,9,10 i let it go
again!";

  f=fopen("FIL1.txt","w");   // Создание нового файла FIL1.txt
  fputs(text,f);             // Запись в файл строки text
  fclose(f);                 // Закрытие файла f
  f=fopen("FIL1.txt","r");   // Открытие файла f для чтения
  r=fopen("FIL2.txt","w");   // Создание нового файла FIL2.txt
  while (!feof(f))           // Пока не конец файла
```

```
{ ch=fgetc(f);            // Чтение символа ch из файла f
  if (ch >='a'&&ch<='z')  // Прочитанный символ - буква?
 fputc(ch,r);           // Запись в файл r символа ch
 }
 fclose(r);              // Закрытие файла r
 rewind(f);              // Возврат указателя на начало файла f
 fgets(pr,65,f);    // Чтение из файла f строки в переменную pr
 printf("%s\n",pr);         // Вывод строки pr на дисплей
 r=fopen("FIL2.txt","r"); // Открытие файла r для чтения
 while (!feof(r))         // Пока не конец файла pr
 {  ch=fgetc(r);           // Чтение символа из файла r
    putchar(ch);         // Вывод символа ch на дисплей
 }
 printf("\n");
 fclose(f);              // Закрытие файлов
 fclose(r);
 return 0;
}
```

Обработка бинарных файлов

Если файл открыт в бинарном режиме, его можно записывать или считывать побайтно. Функция fseek() позволяет обращаться с бинарным файлом как с массивом и переходить к любой позиции в файле, обеспечивая возможность произвольного доступа. Если текстовые файлы являются файлами с последовательным доступом, то к бинарным файлам может применяться произвольный доступ.

Составим программу создания нового файла с информацией о городах: код, название, численность жителей.

```
#include "stdafx.h"
#include <conio.h>
typedef struct city
   { int kod;
     char name[10];
     long  c; } town;
town t;
int main()
{ char c;
  FILE *f;
char ch;
 f=fopen("file1.dat","wb"); //открытие бинарного файла для
записи
  printf("\n Ввод информации о городе ");
```

```
 do
{ printf("\nКод: "); scanf("%d", &t.kod);
 printf("\nназвание: "); scanf("%s", t.name);
 printf("\nколичество жителей: "); scanf("%ld", &t.c);
 fwrite(&t, sizeof(t), 1, f); //запись в файл одной структуры t
  printf("\n END Закончить?  y/n  ");
  ch=getch();
}
 while (ch != 'y');
 fclose(f);
}
```

Выполнение этой программы приведет к созданию бинарного файла с информацией о городах.

Рассмотрим еще одну программу, которая будет читать из файла информацию о городах и выводить на экран список городов, количество жителей в которых превышает миллион.

```
#include "stdafx.h"
#include <conio.h>
typedef struct city
   { int kod;
     char name[10];
     long  c; } town;
town t;
int main()
{
 FILE *f;
f=fopen("file1.dat","rb"); //открытие бинарного файла для чтения
fread(&t, sizeof(t), 1, f); //чтение из файла одной структуры t
while (!feof(f))
{ if(t.c>1000000)
 printf("\n%3d название: %10s  количество жителей: %ld",
 t.kod, t.name, t.c);
 fread(&t, sizeof(t), 1, f);
}
 fclose(f);}
```

Задания для самостоятельного решения

Составить программу решения задачи с использованием файла данных и выполнить ее на ПК.

Таблица 13 – Варианты заданий

Вариант	Условие задачи
1	Создать файл, содержащий массив целых чисел. Найти наименьшее из модулей отклонения чисел от их среднего значения.
2	Создать файл, содержащий массив вещественных чисел. Определить количество чисел, меньших среднего арифметического значения всех чисел массива.
3	Создать файл, содержащий фамилии студентов и их возраст. Вывести фамилии студентов, имеющих наименьший возраст.
4	Создать файл, содержащий названия всех месяцев года. Вывести сначала летние месяцы, а затем - зимние.
5	Создать файл, содержащий фамилии студентов. Вывести список студентов, имеющих фамилии наибольшей длины.
6	Создать файл, содержащий произвольные текстовые строки. Подсчитать количество строк, начинающихся с буквы 'A'.
7	Создать файл, содержащий текстовые строки произвольной длины. Дополнить все строки символом 'X' до самой длинной строки.
8	Создать текстовый файл с произвольным числом строк. Подсчитать общее количество символов в каждой строке.
9	Переписать в файл выходных данных таблицу значений функций: y = sinx и z = tgx, рассчитанных для всех x на отрезке [0, 3] с шагом 0.1. Значения x записать в файле с одной цифрой в дробной части, значения y - с двумя цифрами в дробной части, значения z - в форме числа с порядком.
10	Создать файл, содержащий текстовые строки произвольной длины. Вывести строки с наименьшей длиной.
11	Создать файл, содержащий ФИО студентов и их экзаменационные оценки по трем дисциплинам. Вывести ФИО студентов, успешно сдавших все экзамены.

Тесты

1. Пример программы на языке С:

A) #include <stdio.h>main() {int x;printf(«Введите число «);}
B) Option Base IConst Pi as Double=3.14159 x=2disc = Pi*x^2
C) class Def {public static void main (String[] args) { short k=100;System.out.println(«k=» };}
D) class hello {public static void main (String[] args) {System.out.println(«Hello, world»);}}
E) #include <stdio.h> main() {int x=10; x++;printf(«x=%d «, x); return 0;}
F) <?phpprint («Hello, world»);?>
G) #include <stdio.h> main(){ printf(«Hello, world \ n»); return 0;}

Верный ответ: A, E, G

2. Пример шестнадцатеричной константы в языке С:

A) 123Fcd
B) 0x100
C) 16ABCD
D) 0x1D2A
E) 0xABCD

Верный ответ: B, D, E

3. Поразрядная (битовыми) операция в языке С:

A) ?^
B) ==
C) %
D) ^XOR
E) ++
F) --
G) & AND

Верный ответ: D, G

4. Элемент массива p[6] в языке С:

A) p[0]
B) p[7]
C) p[20]
D) p[3]
E) p[2]
F) p[9]
G) p[6]

Верный ответ: A, D, E

5. Пример указателя в языке С:
A) char p;
B) struct p;
C) int toy;
D) int *toy;
E) float poisk;
F) int year;
Верный ответ: D

6. Функция языка С для для работы со строками:
A) strcatQ
B) getcharQ
C) char()
D) mainQ
E) strlen()
F) strcpy()
Верный ответ: A, E, F

7. Функция языка С для работы с файлами:
A) fclose()
B) strlen()
C) strcpy()
D) fopen()
E) fgets()
Верный ответ: A, D, E

8. Функция для работы с файлами в языке С:
A) fclose()
B) strupr()
C) fread()
D) setcolor()
E) rand()
F) arc()
Верный ответ: A, C

9. Графические возможности языка С:
A) Для работы в графическом режиме должен быть подключен файл conio.h
B) Управление экраном в графическом режиме производится с помощью набора функций, прототипы которых находятся в файле graphics.h

C) Для работы в графическом режиме должен быть подключен файл stdio.h
D) Графические функции подключаются автоматически
E) Для работы в графическом режиме должен быть подключен файл signal.h
F) Прежде чем использовать графические функции, необходимо установить видеоадаптер в графический режим
Верный ответ: B, F

10. Оператор в языке С, использующий поразрядное представление данных:
A) >=
B) <
C) >
D) <<
E) <=
F) ~
Верный ответ: D, F

11. Запись формулы y=sin2x2 на языке программирования С:
A) y=sin(x)*sin(x);
B) y=2*sin(2*x);
C) y=pow(sin(pow(2,x)),2);
D) y=pow(sin(pow(x,2)),2);
E) y=sqr(sin(x*x));
F) y=pow(2,sin(x*x));
G) y=sin(x*x)*sin(x*x);
Верный ответ: D, G

12. Директива препроцессора, позволяющая проводить выборочную (условную) компиляцию программы:
A) #else
B) #elif
C) #pragma
D) #define
E) #include
Верный ответ: A, B

13. По направлению перебора вектор-строка обрабатывается:
A) От обоих концов к середине
B) Справа налево

C) Слева направо
D) По главной диагонали
E) По диагонали сверху вниз
F) Снизу вверх
G) По диагонали снизу вверх
Верный ответ: A, B, C

14. Верное утверждение о функциях в языке С:
A) Функция может возвращать в качестве результата массив любого типа
B) Формальные параметры функции могут не соответствовать фактическим параметрам функции
C) Функция является основным строительным элементом языка С
D) Определение одной функции может содержать в себе определение другой функции
E) Перед использованием или реализацией новой функции необходимо описать ее прототип
F) Переменные, определенные в функции, являются глобальными
G) Прототип функции сообщает информацию, содержащую имя функции, тип возвращаемого значения, количество и типы ее аргументов
Верный ответ: C, E, G

15. Пример заголовка определения функции в языке С:
A) int gov(int x, int y);
B) double Square;
C) double fun[5];
D) int gov;
E) double mask;
Верный ответ: A

16. Пояснение к фрагменту кода: char *ch; int *temp, i, k, *j; float *pif, mas; float arr;:
A) temp - имя функции без аргумента
B) Объявлены указатели mas, arr и к типа float
C) Объявлены переменные mas, arr типа float
D) символ * определяет комментарий
E) Объявлены переменные i, к типа int
Верный ответ: C, E

17. Пример объявления структуры в языке С:
A) struct DPoint {int x; double y; double z;}
B) double str(double a); int x; double y; double z
C) int arr(int x, int y); double x; double y; double z
D) struct DataB { char fam[15]; char NameS[12]; long Tel;)
E) double Summa[20]; char NameS[12]; double x
F) struct R3Vector { double x; double y; double z;}
Верный ответ: A, D, F

18. Динамические структуры данных:
A) стеки
B) адреса ячеек
C) спецификаторы формата
D) буфер данных
E) линейные списки
F) очереди
Верный ответ: A, E, F

19. Пример объявления структуры в языке С:
A) double str(double a); int x; double y
B) struct arr { double x; double y;}
C) double arr (double x, double y)
D) struct dateV {int day; int month ; double z;)
E) double Sum(double x, int day)
F) double str; double y; double person
G) int book[100]; char autor[15]; char title
Верный ответ: B, D

20. Описание кода программы на языке С: int main () { int t=3, a=5, k=8 ;
if (t> { if (a<k) k=a ; else k=t; } return (0); }

A) В результате выполнения программы k станет равным 5
B) Переменные t, a, k объявлены типа int и инициализированы
C) В результате выполнения программы k станет равным 3
D) Переменные t, a, k являются числами с плавающей точкой
E) Выражение t>a принимает значение ложь, поэтому операторы в фигурных скобках не выполняются
Верный ответ: B, E

21. Описание кода программы на языке С: int main() {
int i=3, sum; sum=0;
while (i<=5) {
sum = sum + i;
i = i + 1; }

A) Вычисляются квадраты чисел от 1 до 5
B) Как только переменная цикла i примет значение 4, цикл прекратится
C) Как только переменная цикла i примет значение 6, цикл прекратится
D) Значение переменной sum равно 9
E) Вычисляется сумма чисел от 3 до 5
F) Вычисляется сумма чисел от 1 до 5
G) Используется оператор цикла for
Верный ответ: C, E

22. Описание кода программы на языке С: int main() {
int i;
for (i=l; i<100; i++) {
if (i%7) continue;
printf(«%8d «, i);} }

A) Программа выводит на экран натуральные числа кратные семи
B) Программа выводит на экран четные числа от 2 до 100
C) Как только переменная цикла i примет значение 98, цикл прекратится
D) Программа выводит на экран числа от 1 до 100
E) На каждом шаге цикла параметр цикла i будет уменьшен на единицу
F) Программа выводит на экран нечетные числа от 1 до 99
Верный ответ: A

23. Описание кода программы на языке С:int main() {
int x=1; int y; int i;
for (i=l; i<=3; i++) {
y = x * x;
x += 2;)
A) Вычисляются квадраты четных чисел от 2 до 6
B) Как только переменная цикла i примет значение 4, цикл прекратится

C) Как только переменная цикла i примет значение 2, цикл прекратится
D) Используется оператор цикла while
E) Вычисляется произведение чисел от 1 до 5
F) В конце цикла значение переменной x будет равно 12
Верный ответ: B

24. Описание кода программы на языке C:#include <stdio.h>

```
#include <conio.h>
int MySum(int M, int N);
int main(void) {
int i, N, Sum ;
i=12; N=800;
Sum=MySum(i, N);
Sum=MySum(5,100);
 printf(«Sum= %d «, Sum);
return 0; }
/* определение функции MySum */
int MySum(int M, int N) {
int i, Sum;
i=M; Sum=0;
while (i <= N) {
Sum= Sum +i;
i ++; }
return Sum;}
```

A) Переменная i в функции MySum() передает свое значение функции main()
B) Строка Sum=MySum(i, N); - пример вызова функции MySum
C) В строке заголовка функции MySum() в круглых скобках располагаются выходные (фактические) параметры
D) Переменная Sum в функции MySum() связана с переменной Sum в main()
E) Переменные, заданные внутри тела функции MySum(), являются глобальными
Верный ответ: B

25. Функция scanf () в языке C:
A) основная функция ввода с консоли
B) вызывает тело другой функции
C) осуществляет форматированный вывод данных

D) резервирует место для строки
E) копирует содержимое экрана в файл
F) подключает к программе специальную библиотеку
Верный ответ: A

26. Заголовочный файл языка С:
A) time.h
B) const.h
C) windows.h
D) mac.h
E) doc.h
Верный ответ: A

27. Верное утверждение в языке С:
A) Директивы препроцессора языка С начинаются знаком \ \
B) Директивы препроцессора языка С начинаются знаком /*
C) Препроцессор языка С включает директиву #include
D) Препроцессор языка С содержит справочную информацию
E) Директивы препроцессора языка С начинаются знаком #
F) Препроцессор языка С выполняет отладку программы
G) Препроцессор языка С обеспечивает распределение памяти компьютера
Верный ответ: C, E

28. Верное утверждение о массивах в языке С:
A) Каждый элемент массива определяется именем массива и порядковым номером (индексом) элемента
B) В языке С индекс первого элемента всегда начинается с единицы
C) В языке С многомерные массивы нельзя инициализировать
D) В языке С не допускается использование многомерных массивов
E) Каждый элемент массива может определяется только именем массива
F) Для объявления массива достаточно указать только его имя
Верный ответ: A

29. Верное утверждение в языке С:
A) Для записи имен переменных можно использовать буквы кириллицы
B) Выполнение программы на языке С начинается с вызова функции main()
C) Выполнение программы на языке С начинается с функции get()

D) В файле stdio.h находится информация о стандартной функции вывода printf()
E) Каждая программа на языке С должна содержать функцию main()
Верный ответ: B, D, E

30. Пример указателя в языке С:
A) int year;
B) char *m;
C) struct m;
D) struct stud;
E) int *year;
Верный ответ: B, E

31. Верное утверждение об указателях в языке С:
A) Указатели не могут обеспечивать ссылку на заранее определенные объекты
B) Указателю нельзя присвоить адрес некоторой переменной
C) Указатель предназначен для размещения в памяти нескольких переменных разного типа
D) Указатель предназначен для выполнения операций ввода/вывода на консоль
E) Указатель содержит адрес некоторого объекта
F) Указатель - это переменная, содержащая адрес другой переменной
Верный ответ: E, F

32. Пример записи строковых констант в языке С:
A) 'A'
B) "Java"
C) '1'
D) 12345
E) '+'
Верный ответ: B

33. Режим открытия файлов в языке С:
A) a (a+)
B) c (+)
C) k (k--)
D) s (++)
E) l (l+)
F) v(v+)
Верный ответ: A

34. Графические возможности языка С:
A) Графический экран представляет собой векторное поле
B) Чтобы выйти из графического режима и вернуться в текстовый режим можно использовать указатель на объединение
C) Чтобы выйти из графического режима и вернуться в текстовый режим используют функцию closegraph()
D) Графический экран представляет собой массив пикселей
E) Для установки (инициализации) видеоадаптера используют функцию fopen()
F) Графический экран представляет собой массив векторов
G) Чтобы выйти из графического режима и вернуться в текстовый режим достаточно набрать команду close
Верный ответ: C, D

35. По количеству участвующих операндов в языке С различают операции:
A) унарные
B) базовые
C) двумерные
D) одномерные
E) тернарные
F) унифицированные
Верный ответ: A, E

36. Функция printf () в языке С:
A) вызывает тело другой функции и передает отдельные значения
B) подключает к программе специальную библиотеку
C) резервирует место для обработки строк
D) осуществляет форматированный вывод данных
E) спецификатор формата %f указывает на печать числа с плавающей запятой
F) выводит форматированные данные в стандартный поток
Верный ответ: D, E, F

37. Простой метод сортировки массива:
A) Сортировка выделением (выбором)
B) Сортировка вставкой (включением)
C) Сортировка заменой
D) Сортировка обменом
E) Сортировка удалением
Верный ответ: A, B, D

38. Элемент массива b[2][2] в языке С:
A) b[1][2]
B) b[0][2]
C) b[2][1]
D) b[0][1]
E) b[1][0]
F) b[2][2]
G) b[0][0]
Верный ответ: D, E, G

39. Утверждение в языке С о функции rand():
A) Сортирует элементы массива в в порядке убывания
B) Генерирует вещественные случайные числа
C) Прототип функции расположен в заголовочном файле <stdlib.h>
D) Генерирует псевдо-случайные числа
E) Определяет самый минимальный элемент массива
Верный ответ: C, D

40. Пример заголовка определения функции в языке С:
A) double summa;
B) double Sq [5] [3];
C) int Myfun;
D) int Myfun(int x, int y, double h);
E) double Sq (double x, double y, int k);
F) double Sq;
G) double summa(double x1, double x2, double x3);
Верный ответ: D, E, G

41. Пример объявления структуры в языке С:
A) struct person { char fam[20]; char N [15];}
B) double Sum(double x, int day)
C) struct dateV {int day; int month ; double z;}
D) double str(double a); int x; double y
E) int book[100]; char autor[15]; char title
Верный ответ: A, C

42. Пример объявления структуры в языке С:
A) struct DataB { char fam[15]; char NameS[12]; long Tel;}
B) double sum(double x, double y); long Tel
C) struct DPoint {int x; double y; double z;}
D) struct R3Vector { double x; double y; double z;)

E) double str(double a); int x; double y; double z
F) int arr(int x, int y); double x; double y; double z
G) int Dpoint[100]; char fam[15]; long Tel
Верный ответ: A, C, D

43. Описание фрагмента программы в языке С:#include <stdio.h>
void main() {
FILE *pf;
pf = fopen(«FileName», «rb»); }

A) Файл всегда успешно открывается с помощью функции fopen()
B) Функция fopen() открывает текстовый файл для записи
C) Функция fopen() открывает текстовый файл для чтения
D) pf - указатель на файл, который содержит служебную информацию о файле
E) Функция fopen() открывает двоичный файл для чтения
F) Строка "FileName" задает имя файла и путь к нему
Верный ответ: D, E, F

44. Верное утверждение при работе с файлами в языке С:
A) Все необходимые действия для работы с файлами выполняются только с помощью функций, созданных программистом
B) При работе с файловой системой используется понятие поток данных
C) Все функции для работы с файлами включены в стандартную библиотеку string.h
D) Все необходимые действия для работы с файлами выполняются с помощью функций, включенных в стандартную библиотеку stdio.h
E) Файловая система Windows всегда гарантирует успешность открытия файла
F) При работе с файлами для выполнения операции чтения/записи пользователю достаточно указать имя файла
G) При работе с файловой системой используется понятие инкапсуляция
Верный ответ: C, D

45. В состав графического пакета языка С входит:
A) Заголовочный файл locale.h
B) Заголовочный файл graphics.h
C) Драйверы графических устройств (*.BGI)
D) Заголовочный файл time.h

E) Заголовочный файл float.h
F) Драйверы принтеров
G) Заголовочный файл stdio.h
Верный ответ: B, C

46. Описание кода программы на языке С: int main () { int t=7, a=2, k=5 ;
if (t> { if (a<k) k=a ; else k=t;} return (0); }

A) Переменные t, a, k объявлены типа int и инициализированы
B) В результате выполнения программы k станет равным 5
C) Переменные t, a, k являются числами с плавающей точкой
D) В результате выполнения программы k станет равным 3
E) Выражение t>a принимает значение истина, поэтому выполняются операторы в фигурных скобках
F) В результате выполнения программы k станет равным 2
Верный ответ: A, E, F

47. Описание кода программы на языке С: int main() {int i=3, sum;
sum=3;
while (i<=5) {
sum = sum + i;
i = i + 1; }

A) Значение переменной sum равно 15
B) Вычисляется сумма чисел от 1 до 5
C) Вычисляются квадраты чисел от 1 до 5
D) Вычисляется сумма чисел от 3 до 5
E) Используется оператор цикла for
F) Как только переменная цикла i примет значение 3, цикл прекратится
G) Как только переменная цикла i примет значение 6, цикл прекратится
Верный ответ: A, D, G

48. Описание кода программы на языке С: int main() {int i;
for (i=l; i<=50; i ++) {
if (i%3) continue;
printf(«%8d «, i); } }

A) На каждом шаге цикла параметр цикла i будет уменьшен на единицу
B) В начале каждого шага цикла проверяется условие i<=50
C) Сначала параметр цикла i устанавливается равным 1
D) Программа выводит на экран все числа от 1 до 50
E) Программа выводит на экран натуральные числа кратные трем
Верный ответ: B, C, E

49. Библиотечная функция для работы с динамической памятью в языке С:
A) void *free(*int)
B) void*calloc(unsigned n)
C) void*malloc(unsigned n, unsigned m)
D) void*malloc(s)
E) void*realloc(void *p,unsigned s)
F) void*calloc(unsigned n, unsigned m)
Верный ответ: E, F

50. Потоковая функция ввода/вывода символа в языке С:
A) putc()
B) printf()
C) getchar()
D) gets()
E) fopen
Верный ответ: A, C, D

51. Верное утверждение в языке С:
A) Переменные, заданные внутри тела функции, являются локальными
B) Переменные, заданные внутри тела функции, являются глобальными
C) перед функцией main не нужно помещать строку: #include <stdio.h>
D) Выполнение любой программы начинается с функции main.
E) любая программа состоит из одной или более функций, задающих действия, которые нужно выполнить.
Верный ответ: A, D, E

52. Верное утверждение о алфавите языка С:
A) включает только строчные буквы
B) включает только прописные буквы
C) включает буквы русского алфавита

D) Идентификатор может содержать до 32 символов и состоит из букв и цифр, но начинается обязательно с буквы.
E) включает латинские прописные и строчные буквы, цифры и специальные знаки.
Верный ответ: D, E

53. Типы данных определяется одним из следующих ключевых слов:
A) double
B) floats
C) floatr
D) unsigned
E) shorts
F) int
Верный ответ: A, D, F

54. Ввод и вывод информации в языке С:
A) Каждая спецификация преобразования начинается со знака $ и заканчивается некоторым символом, задающим преобразования.
B) Управляющая строка содержит объекты двух типов: обычные символы, которые просто выводятся на экран дисплея, спецификации преобразования, каждая из которых вызывает вывод на экран значения очередного аргумента из последующего списка.
C) Каждая спецификация преобразования начинается со знака # и заканчивается некоторым символом, задающим преобразования.
D) Каждая спецификация преобразования начинается со знака % и заканчивается некоторым символом, задающим преобразования.
E) Управляющая строка содержит объекты трех типов: обычные символы, которые просто выводятся на экран дисплея, спецификации преобразования, каждая из которых вызывает вывод на экран значения очередного аргумента из последующего списка и управляющие символы-константы
F) printf("управляющая строка", аргумент1, аргумент2, ...);
Верный ответ: D, E, F

55. Символ преобразования в языке С:
A) g - значением аргумента является указатель (адрес).
B) u - значением аргумента является беззнаковое целое число;
C) f - значением аргумента является целое число;
D) f - значением аргумента является вещественное десятичное число с плавающей точкой;

E) d - значением аргумента является вещественное число;
F) d - значением аргумента является десятичное целое число;
Верный ответ: B, D, F

56. Символ преобразования в языке С:
A) g - значением аргумента является указатель (адрес).
B) u - значением аргумента является беззнаковое целое число;
C) c - значением аргумента является целое число;
D) f - значением аргумента является експоненциальное число;
E) d - значением аргумента является целое число;
F) o - значением аргумента является восьмеричное целое число;
Верный ответ: B, F

57. Правильно написанное выражение ввода и вывода в языке С:
A) printf("%", S);
B) printf("%-6d");
C) printf("i=% j=%",i,j);
D) printf("%2d", S);
E) printf("i=%d j=%d",i,j);
Верный ответ: D, E

58. Правильное написанное выражение управляющих символьных констант в языке С:
A) \ v - для горизонтальной табуляции.
B) \ r - для перевода курсора в конец текущей строки;
C) \ t - для вертикальной табуляции;
D) \ b - для перевода курсора направо на одну позицию;
E) \ n - для перехода на новую строку;
F) \ a - для кратковременной подачи звукового сигнала;
Верный ответ: E, F

59. Правильное написанное выражение управляющих символьных констант в языке С:
A) \ v - для горизонтальной табуляции.
B) \ t - для вертикальной табуляции;
C) \ n - для перехода на первую строку;
D) \ r - для перевода курсора в конец строки;
E) \ a - для кратковременной подачи звукового сигнала;
F) \ b - для перевода курсора влево на одну позицию;
Верный ответ: E, F

60. Ввод данных в языке С:
A) scanf(«управляющая строка»; аргумент1, аргумент2,...);
B) scanf(«управляющая строка», аргумент1, аргумент2,...);
C) scanf(параметры функции, аргумент1, аргумент2,...);
D) scanf(«управляющая строка»);
E) scanf(аргумент1, аргумент2,...);
F) scanf(«управляющая строка», аргумент1, аргумент2,...);
Верный ответ: F

61. Функции ввода и вывода строк в языке С:
A) ons
B) get
C) off
D) puts
E) put
F) gets
Верный ответ: D, F

62. Верное утверждение в языке С:
A) Результатом деления по модулю является целое число
B) Модификатор const, используемый отдельно, не эквивалентен const int.
C) В языке СИ не может быть использован модификатор const, запрещающий какие бы то ни было переопределения константы: ее уменьшение, увеличение и т.п.
D) Порядок выполнения при вычислении значения выражения определяется их приоритетами и может регулироваться с помощью круглых скобок.
E) Выражения состоят из операндов (переменные, константы и др.), соединенных знаками операций (сложение, вычитание, умножение и др.).
F) Выражения широко используются в программах на языке СИ и представляют собой формулы для вычисления переменных.
Верный ответ: D, E, F

63. функции, применяемые при программировании в языке С:
A) double log2(double X)
B) double log(double X)
C) double log10(double X)
D) double log16(double X)
E) double acos(double X)

F) double acos(double (X))
Верный ответ: B, C, E

64. В каких выражениях используются бинарные арифметические операции?
A) --xx
B) sizeof(xx)
C) delete
D) new
E) c % d + 2
F) xx * Y
G) xx++
H) X + Y
Верный ответ: E, F, H

65. Какие слова не относятся к ключевым словам в языке программирования С?
A) else;
B) float;
C) asm;
D) program;
E) made;
F) using;
G) main;
H) const;
Верный ответ: D, E, G

66. Из приведенных ниже ключевых слов выберите те из них, которые не принадлежат к основным типам данных:
A) double;
B) wchar_t;
C) int;
D) char;
E) short;
F) bool;
G) long;
H) signed
Верный ответ: E, G, H

67. Какие управляющие символьные константы описаны неверно?

A) горизонтальная табуляция ' \ r'
B) возврат каретки ' \ f'
C) возврат каретки ' \ r'
D) вертикальная табуляция ' \ v'
E) нулевой символ ' \ t'
F) новая строка ' \ n'
G) нулевой символ ' \ 0'
H) перевод формата ' \ f'
Верный ответ: A, B, E

68. В каких случаях используется составной оператор?
A) для удаления несколько логических связанных операторов;
B) для обозначения коментариев;
C) при совместном использовании функций scanf( ) и printf( );
D) для ограничения видимости определенной части программы;
E) в качестве тела функции;
F) для определения группы констант;
G) чтобы сгруппировать несколько логических связанных операторов в один оператор;
H) чтобы сгруппировать несколько функций в одну функцию;
Верный ответ: D, E, G

69. Укажите те структуры, с помощью которых можно составить программу для решения задачи любой сложности.
A) объединение;
B) цикл;
C) разбиение;
D) прерывание;
E) передача управления;
F) блокирование;
G) ветвление;
H) следование;
Верный ответ: B, G, H

70. Укажите типы данных, которые предназначены для хранения вещественных значений:
A) integer;
B) bool;
C) float;
D) int;
E) double;

F) long int;
G) long double;
H) char ;
Верный ответ: C, E

71. Какие из перечисленных языков являются языками низкого уровня?
A) Java ;
B) Ассемблер;
C) CIL;
D) Fortran;
E) Forth;
F) C# ;
G) Си ;
H) C++ ;
Верный ответ: C, E

72. Укажите на не выполнимые в неявном виде арифметические преобразования типов:
A) операнды типа float преобразуется к типу long double;
B) если один операнд - unsigned int, то второй преобразуется к типу unsigned short;
C) операнды типа float преобразуется к типу unsigned char;
D) любые операнды типа unsigned char и unsigned short преобразуется к типу unsigned int;
E) любые операнды типа char и short преобразуется к типу int;
F) любые операнды типа float и double преобразуется к типу int;
G) если один операнд - long double, то второй преобразуется к этому же типу;
H) операнды типа float преобразуется к типу double;
Верный ответ: B, C, F

73. Какие из перечисленных языков программирования относятся к процедурным?
A) Delphii;
B) Паскаль;
C) Кобол;
D) PHP;
E) Си;
F) Java;
G) C#;

H) C++;
Верный ответ: B, C, E

74. Что из перечисленных ниже ключевых слов относятся к директивам препроцессора?
A) #define
B) #begin
C) #string
D) #include
E) #array
F) #end
G) #undef
H) #unsigned
Верный ответ: A, D, G

75. Текст программы па С или С++ , называющееся исходным кодом - записывают в файл с расширением
A) .asm ;
B) .bmp;
C) .xml;
D) .java;
E) .doc;
F) .cpp;
G) .c;
H) .cc;
Верный ответ: F, G, H

76. Операции в языке программирования Си подразделяются на:
A) нет правильного ответа;
B) мультиарные;
C) мультарные;
D) полиадические;
E) кватернарные;
F) тернарные;
G) бинарные;
H) унарные;
Верный ответ: E, F, G, H

77. Нелогическими операциями являются:
A) %;
B) !=;

C) !;
D) =;
E) &;
F) &&;
G) ==;
H) ||;
Верный ответ: A, D, E

78. В каких выражениях используются унарные арифметические операции?
A) c-d+2;
B) !c;
C) c!=0;
D) --b;
E) d++;
F) c+++b;
G) c1 + d2;
H) s2 % d % 2;
Верный ответ: D, E, F

79. Отметьте фрагменты кода, которые можно назвать выражениями:
A) x = 3;
B) int x = 0;
C) 3 = 5;
D) double c;
E) cond = x < 2;
F) f + r*12 - 14;
G) int z;
H) x = y = 13;
Верный ответ: E, F, H

80. Битовыми операциями являются:
A) &&
B) ~
C) ^
D) &
E) +
F) ||
G) !=
H) =

Верный ответ: B, C, D

81. В каких выражениях возвращаемое значение будет логическим?
A) (c~3);
B) (c=1);
C) (c&b);
D) (c^b);
E) (!c);
F) (c == b);
G) (i > 3);
H) int x = 2, y = 3, z; z = x + y;
Верный ответ: E, F, G

82. На сколько разрядов необходимо сделать сдвиг влево или вправо переменной V2al, чтобы значение temp принимало значение 1,32,16 ?
(int val = 4;
int temp = val << ? или int temp = val >>?)

A) 7
B) 2
C) 1
D) 5
E) 4
F) 3
G) 6
H) 8
Верный ответ: B, C, F

83. Функции. Отметьте ошибочные утверждения:
A) Достигнув конца функции, управление вернется в ту точку, откуда функция была вызвана, подставив вместо нее вычисленный результат;
B) Встретив имя функции в выражении, программа передаст управление на ее начало и начнет выполнять операторы;
C) Программа на языке Си++ может состоять без единой функции;
D) Не может быть объявлена как void;
E) Выполняет необходимые действия и возвращает результат;
F) При обращении к функции фактические параметры заменяются формальными;

G) Строгое согласование по типам между формальными и фактическими параметрами требует, чтобы в модуле до первого обращения к функции было помещено либо ее описание, либо определение;
H) Си++ обеспечивает строгий контроль типов;
Верный ответ: C, D, F

84. Какие из следующих утверждений об операторе return являются верным?
A) Необходим для немедленного перхода в другую функцию;
B) Нельзя использовать в любом месте внутри функции;
C) Возвращает управление в вызывающую функцию, в точку, непосредственно следующую за вызовом;
D) Оператор return завершает выполнение функции;
E) в теле функции должен присутствовать только один оператор return;
F) Оператор return должен стоять последним в теле функции;
G) С помощью return можно прекратить работу программы;
H) Можно использовать в любом из операторов цикла;
Верный ответ: C, D, F

85. Укажите несколько типов, которые существует для представления целых чисел в языке Си:
A) long;
B) double;
C) short;
D) wchar_t;
E) bool;
F) float;
G) char;
H) long double;
Верный ответ: A, C, G

86. Укажите в каких выражениях правильно определены целочисленные переменные?
A) short x = 23;
B) bool m;
C) double pi_d = 3.1415;
D) long double pi_l = 3.1415L;
E) const float pi_f = 3.14f;
F) const char c;

G) unsigned short g = 0x2;
H) unsigned int a = 2, b = 3;
Верный ответ: A, F, H

87. Свойства, которыми должен обладать алгоритм:
A) внедряемость;
B) правильность;
C) результативность;
D) представительность;
E) выполняемость;
F) вариантность;
G) однозначность;
H) производительность;
Верный ответ: A, C, G

88. Типы алгоритмов:
A) циклический;
B) пакетный;
C) комплексный;
D) разветвляющийся;
E) периодический;
F) линейный;
G) ступенчатый;
H) прямой;
Верный ответ: A, B, F

89. Этапы решения задач на компьютере:
A) алгоритмизация задачи;
B) корректировка входных данных;
C) планирование результатов;
D) выбор модели представления данных;
E) модификация;
F) декларирование типов и переменных;
G) постановка задачи;
H) составление требований заказчиком;
Верный ответ: A, E, G

90. Способы записи алгоритмов:
A) графические слайды;
B) на языке эвм;
C) на языке программирования;

D) покадровая анимация;
E) на естественном языке;
F) в виде схемы (блок-схемы);
G) на языке диаграмм;
H) в реляционных таблицах;
Верный ответ: C, E, F

91. Контроль структуры программы:
A) подвижный;
B) согласованный;
C) сквозной;
D) непрерывный;
E) открытый;
F) абсолютный;
G) смежный;
H) статический;
Верный ответ: C, G, H

92. Математическая модель - это:
A) представление алгоритма в форме, понятной человеку;
B) запись алгоритма задачи при помощи одного из языков программирования;
C) представление задачи в виде одного из языков, понятных эвм;
D) набор операторов, который используется для управления поведением системы;
E) упрощенное описание реальности с помощью математических понятий;
F) следование конкретных действий в определенном порядке;
G) математическое представление реальности;
H) система уравнений и концепций, используемых для описания поведения объекта;
Верный ответ: E, G, H

93. Документация, определяющая строение программ и структур данных программы:
A) Планирование системного анализа.
B) Описание модульной системы, включая внешнюю спецификацию каждого включенного модуля.
C) Описание внедрения программы для регулярной эксплуатации.
D) Описание архитектуры программы, включая внешнюю спецификацию.

E) Руководство по сопровождению программы.
F) Ознакомление с поставленной задачей.
G) Внешнее описание.
H) Описание этапов повышения эксплуатационных характеристик.
Верный ответ: C, D, G

94. Характеристики программного модуля:
A) легкость в использовании;
B) новизна;
C) быстродействие;
D) общедоступность;
E) сцепление с другими модулями;
F) прочность модуля;
G) размер модуля;
H) жесткость;
Верный ответ: E, F, G

95. Правильное обращение к атрибуту "a" класса "A" в языке C++:class A
{ public:
int a;
};
...
A* obj;
int* obj2;

A) obj.a
B) obj2=&obj->a
C) obj[a]
D) obj2=&obj->a
E) obj::a
F) (*obj).a
G) obj->a
H) (*obj)->a
Верный ответ: D, F, G

96. В языке C++ допустимые варианты в названиях идентификаторов:
A) знаки "$", "@", "#" и латинские буквы;
B) знаки "$", "@" и латинские буквы;
C) знак "$" и латинские буквы;

D) любые символы, кроме используемых в арифметических и логических операциях;
E) любые символы с ascii-кодом старше 128;
F) знак "@" и цифры;
G) знак "_", цифры;
H) заглавные и строчные латинские буквы;
Верный ответ: C, G, H

97. Правильно оформленные условия в операторе ветвления языка C++:
A) if(a%b)
B) if(a++b)
C) if(a~b)
D) if(a<b && c>d)
E) if(a<b And c>d)
F) if(!!a)
G) if(a&b)
H) if(a<b or c>d)
Верный ответ: A, D, G

98. Правильно оформленная инициирующая часть оператора цикла for в языке C++:
A) for(int lo = 0, hi = max, mid = max/2; ...)
B) for(i==true; ...)
C) for(1..2; ...)
D) for(int i = 0; ...)
E) for(i==0; ...)
F) for(i=i; ...)
G) for(1; ...)
H) for(/* пустой инициирующий оператор */; ...)
Верный ответ: A, D, H

99. Правильно оформленное условие повторения оператора цикла for в языке C++:
A) for(...; index < arraysize; ...)
B) for(...;1; ...)
C) for(...;i<>0; ...)
D) for(...; ; ...)
E) for(...; ch = fun(); ...)
F) for(...;i=true; ...)
G) for(...;i=i; ...)

H) for(...;1..2; ...)
Верный ответ: A, D, E

100. Правильно оформленная часть изменения значений оператора цикла for в языке C++:
A) for(...; ...;i==i)
B) for(...; ...;i==true)
C) for(...; ...; ++i)
D) for(...; ...;)
E) for(...; ...; 1..2)
F) for(...; ...; i /= 2)
G) for(...; ...;i<>0)
H) for(...;...; 1)
Верный ответ: C, D, F

101. Операторы перехода в языках Паскаль, C++:
A) break
B) abort
C) leave
D) cancel
E) return
F) continue
G) goto
H) jmp
Верный ответ: A, F, G

102. Обозначениями логических операций на языке C++ являются:
A) ||
B) |
C) &&
D) =
E) !
F) <>
G) ~
H) &
Верный ответ: A, C, E

103. Число 5 будет получено в результате следующих битовых операций в языке C++:
A) 12 ^ 9

B) 1 & 9
C) 15 | 3
D) 1 | 4
E) 11 >> 1
F) 10 << 1
G) 11 >> 2
H) ~11
Верный ответ: A, D, E

104. Операции в языке C++
A) <<
B) &&
C) <<
D) ==
E) ()
F) +
G) *
H) =
Верный ответ: A, B, E

105. Допустимыми объявлениями простых типов на языке C++ являются:
A) long bool
B) long float
C) unsigned long
D) unsigned float
E) unsigned int
F) long char
G) unsigned char
H) unsigned bool
Верный ответ: C, E, G

106. Два байта оперативной памяти в языке C++ занимают:
A) double char
B) short int
C) boolean
D) small int
E) short float
F) int
G) word
H) unsigned int

Верный ответ: B, F, H

107. Вещественные типы в языке C++ объявляются:
A) real
B) set
C) extended
D) double
E) single
F) long double
G) enum
H) float
Верный ответ: D, F, H

108. Допустимыми именами переменных на языке C++ являются:
A) asm
B) Asm
C) __@__
D) $m312
E) _V2ar
F) %Asm
G) 1str
H) _asm
Верный ответ: B, D, E

109. Выражения с правильно выполненным присваиванием в языке C++:
A) int a='123';
B) char a="123";
C) int a="123";
D) char a[]="123";
E) int a[]="123";
F) int a[4]='123';
G) char a[4]='123';
H) char a='123';
Верный ответ: A, D, H

110. Допустимые операторы на языке C++:
A) <<=
B) .=
C) $=

D) +=
E) ++=
F) &=
G) <=>
H) ===
Верный ответ: A, D, F

111. Ключевые слова на языке C++:
A) not
B) sizeof
C) end
D) constructor
E) of
F) new
G) or
H) this
Верный ответ: B, F, H

112. Используемые escape-последовательности в языке C++ при выводе неотображаемых символов:
A) \ n
B) \ &
C) \ "
D) \ /
E) \ +
F) \ !
G) \ c
H) \ t
Верный ответ: A, C, H

113. Для начала вывода в консоли с новой строки на языке C++ используется:
A) cout<<newline;
B) cout<<" \ r"<<"a";
C) cout<<" \ 010";
D) cout<<" \ 012";
E) cout<<" \ x0D";
F) cout<<" \ 013";
G) cout<<" \ n";
H) cout<<endl;
Верный ответ: D, G, H

114. Для табуляционного отступа перед словом "hello" в консоли на языке С++ используется:
A) cout<< 009<< "hello";
B) cout<< 9<< "hello";
C) cout<< " \ x9"<< "hello";
D) cout<< " \ t"<< "hello";
E) cout<< " \ 9"<< "hello";
F) cout<< " \ tab"<< "hello";
G) cout<< " \ 11"<< "hello";
H) cout<< " \ oct9"<< "hello";
Верный ответ: C, D, G

115. Для вывода вещественного числа "r" в консоли на языке С++, в формате 2 знака после запятой:
A) write(r:6:2);
B) printf("%>>6.2f",r);
C) printf("%+6.2f",r);
D) printf("%:6.2f",r);
E) printf("% 6.2f",r);
F) write("%s6.2f",r);
G) printf("%-6.2f",r);
H) write(r:"%6:2");
Верный ответ: C, E, G

116. Правильно оформленные выражения в языке ObjectPascal: var fs: file; i,j:byte; c: char; s:string;
A) rewrite(fs
B) blockread(fs
C) assignfile(fs
D) readln(fs
E) read(fs
F) write(fs
G) blockwrite(fs
H) blockwrite(fs
Верный ответ: A, C, H

117. Функции в языке С++, выполняющие запись в файл:
A) Tofile
B) fprint
C) fwrite
D) writefile

E) fprintf
F) Fputs
G) sendfile
H) Puts
Верный ответ: C, E, F

118. Функции по работе с файлами в языке C++:
A) fstate
B) fcopy
C) fgetline
D) fflush
E) Feof
F) Flock
G) fsetpos
H) fputline
Верный ответ: D, E, G

119. Компонентные файлы в языке Pascal могут работать с компонентами следующих видов:
A) Записи
B) Списки
C) компоненты пользователя
D) объекты
E) Файлы
F) массивы
G) Строки
H) Классы
Верный ответ: A, F, G

120. Функции по работе с файловой системой в языке Pascal:
A) FileList
B) FindLast
C) FileExpand
D) FindNext
E) FindFirst
F) FilledFile
G) Ffruct
H) EraseFile
Верный ответ: C, D, E

121. Правильное обращение к элементам массива в языке С++:
int a[2][8]; char i,j; double k;
A) a[i/j][7]
B) a[i/k][7]
C) a[k/i][7]
D) a[i%j][1]
E) a[i%k][1]
F) a[k*i][7]
G) a[i*j][7]
H) a[2][8]
Верный ответ: A, D, G

122. Правильная инициализация массива в языке С++:
A) int a[2][3]={ {1,2,3}, {21,22,23} };
B) int a[2][3]={1,2,3{}};
C) int a[2][3]={ ,,, {21,22,23} };
D) int a[2][3]={1,2,3, 21};
E) int a[2][3]={ {,,}, {21,22,23} };
F) int a[2][3]={ {0,0,0} {21,22,23} };
G) int a[2][3]={{1,2,3, 21,22,23 }};
H) int a[2][3]={1,2,3, 21,22,23 };
Верный ответ: A, D, H

123. Функции по работе со строками в языке С++:
A) strcmp
B) strcpy
C) strln
D) strlod
E) strsub
F) stradd
G) strcmps
H) strncmp
Верный ответ: A, B, H

124. Допустимая инициализация строки в языке С++:
A) char a(5)="123";
B) char a[5]="123";
C) char(5) a="123";
D) char[5] a="123";
E) char* a="123";
F) char a="123";

G) char:5 a="123";
H) char a[]="123";
Верный ответ: B, E, H

125. Правильно оформленная функция на языке С++:
A) int main() {return true;}
B) int main() {return ;}
C) void main() {}
D) void main() {return 1;}
E) int = //{return 1}
F) int main (){Return 1}
G) int main() {return true}
H) int main() {return 1;}
Верный ответ: A, C, H

126. Прототип функции описывает:
A) Количество передаваемых параметров.
B) Тип передаваемых параметров.
C) Порядок следования передаваемых параметров.
D) Соответствие возвращаемых параметров возможностям функции.
E) Тип возвращаемых параметров.
F) Соответствие передаваемых параметров возможностям функции.
G) Порядок следования возвращаемых параметров.
H) Количество возвращаемых параметров.
Верный ответ: A, B, C

127. Область видимости переменных в Языке С++:
A) Глобальные, видимые в пределах модуля.
B) Локальные, видимые в пределах функции.
C) Глобальные-межмодульные, видимые в пределах проекта.
D) Глобальные-межклассовые, видимые для дочерних классов, имеющих определенные привилегии.
E) Локальные, видимые в пределах блока, заключенного в фигурные скобки.
F) Скрытно-локальные, видимые в пределах одного выражения текущего оператора.
G) Особо-глобальные, видимые в пределах кода на расстоянии 64 кБт.
H) Глобальные-межклассовые, видимые для всех дочерних классов.
Верный ответ: A, B, E

128. Свойства функций:
A) В теле функции есть описания локальных переменных и исполняемые операторы.
B) Хорошая криптопригодность.
C) Легкая сопровождаемость.
D) У функции могут быть параметры.
E) Возможность статичной и динамической загрузки функций.
F) Возможность параллельного выполнения кода.
G) Если вызов функции встречается ранее ее определения, то в начале программы должно содержаться описание функции.
H) Экономия ресурсов памяти.
Верный ответ: A, D, G

129. Передача параметров в функцию по значению допустима для переменных типа:
A) Строковый.
B) Запись (структура).
C) Логический.
D) Символьный.
E) Список.
F) Числовой.
G) Массив.
H) Объект.
Верный ответ: C, D, F

130. Передача параметров в функцию по ссылке обязательна для переменных типа:
A) Простой.
B) Целочисленный.
C) Символьный.
D) Запись (структура).
E) Массив.
F) Вещественный.
G) Логический.
H) Строковый.
Верный ответ: D, E, H

131. Работа функции будет завершена на языке C++ в следующих случаях:
A) по достижении закрывающей тело функции круглой скобки.
B) при встрече оператора "RETURNs".

C) при встрече оператора "return".
D) при вызове функции "exit(0)".
E) по достижении закрывающей тело функции фигурной скобки.
F) при вызове функции "exit()".
G) при встрече оператора "Ret".
H) при встрече оператора "Return".
Верный ответ: C, D, E

132. К базовым принципам объектно-ориентированного стиля программирования относятся:
A) Формализация.
B) Виртуальные методы.
C) Модификация.
D) Полиморфизм.
E) Инкапсуляция.
F) Визуальность компонентов.
G) Наследование.
H) Инициализация конструктора.
Верный ответ: D, E, G

133. Спецификатор доступа к членам класса обозначается:
A) interface;
B) set..of;
C) implementation;
D) wit;
E) private;
F) public;
G) do;
H) class;
Верный ответ: E, F, G

134. Допустимые операторы циклов в языке C++:
A) while
B) switc
C) foreach
D) for
E) repeat...until
F) label...continue
G) label...loop
H) do...while
Верный ответ: A, D, H

Литература

1. ISO/IEC 14882:2003. Information Technology – Programming Languages – C++. – 2-nd edition. – ISO/IEC, 2003. – 757 pp.

2. ISO/IEC 9899:1999. Programming Languages – C. – ISO/IEC, 1999. – 538 pp.

3. STL – стандартная библиотека шаблонов C++ : пер. с англ. / П. Плаугер, А. Степанов, М. Ли, Д. Массер. – СПб. : БХВ-Петербург, 2008. – 656 с.

4. Аммерааль Л. STL для программистов на C++ : пер. с англ. – М. : ДМК, 2007. – 240 с.

5. Голицына О. Л. Основы алгоритмизации и программирования. М. : Форум. 2008. – 431с.

6. Джосьютис Н. М. C++. Стандартная библиотека. Для профессионалов : пер. с англ. – СПб. : Питер, 2004. – 730 с.

7. Керниган Б. В., Ритчи Д. М. Язык программирования Си : пер. с англ. – 3-е изд. – СПб. : Невский Диалект, 2011. – 352 с.

8. Липпман С. Б. Основы программирования на C++ : пер. с англ. – М. :Вильямс, 2002. – 256 с.

9. Липпман С. Б., Лажойе Ж. Язык программирования C++. Вводный курс : пер. с англ. – 3-е изд. – М. : ДМК, 2007. – 1104 с.

10. Лишнер Р. STL. Карманный справочник : пер. с англ. – СПб. : Питер, 2009. – 187 с.

11. Мейерс С. Эффективное использование STL : пер. с англ. – СПб.: Питер, 2009. – 224 с.

12. Страуструп Б. Язык программирования C++ : пер. с англ. – 3-е спец. изд. – М. : Бином, 2003. – 1104 с.

13. Страуструп Б. Дизайн и эволюция языка C++. Объектно-ориентированный язык программирования : пер. с англ. – М.: ДМК пресс, Питер, 2006. – 448 с.

14. Эккель Б. Философия C++. Введение в стандартный C++ : пер. с англ. – 2-е изд. – СПб. : Питер, 2004. – 572 с.

15. Эккель Б., Эллисон Ч. Философия C++. Практическое программирование : пер. с англ. – СПб. : Питер, 2004. – 608 с.

16. Эпштейн М. С. Практикум по программированию на языке С. М. : Академия, 2007, – 102с.

Содержание

А. Ж. Саринова

**АЛГОРИТМИЗАЦИЯ
И ОСНОВЫ ПРОГРАММИРОВАНИЯ**

Учебно-методическое пособие

Технический редактор З. Ж. Шокубаева
Ответственный секретарь З. С. Искакова

Подписано в печать 13.07.2016 г.
Гарнитура Times.
Формат 60x90/16. Бумага офсетная.
Усл.печ. л. 9,04 Тираж 300 экз.
Заказ № 2861

Издательство «КЕРЕКУ»
Павлодарского государственного университета
им. С.Торайгырова
140008, г. Павлодар, ул. Ломова, 64

Утверждаю
Проректор по АР
ПГУ им. С. Торайгырова
_____ Г. Г. Ахметова
11 07 2016 г.

Составитель А. Ж. Саринова

Кафедра «Вычислительная техника и программирование»

Алгоритмизация и основы программирования
Учебно-методическое пособие

Одобрено на заседании кафедры 15 04 2016 г.
Протокол № 9

Заведующий кафедрой _____ О. Г. Потапенко

Одобрено советом факультета ФМиИТ 18 04 2016 г.
Протокол № 9

Председатель УМС _____ Н. Ж. Жуспекова

Рекомендовано учебно-методическом советом Павлодарского государственного университета им. С. Торайгырова 20 04 2016 г.
Протокол № 9

СОГЛАСОВАНО
Декан ФМиИТ _____ Н. А. Испулов 18 04 2016 г.

ОАиМК н/к _____ Г. С. Баяхметова 8 07 2016 г.

ОДОБРЕНО
Начальник УМО _____ А. Б.Темиргалиева 20 04 2016 г.