Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

М. В. ШИШКИНА

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Практикум

Владимир 2021

УДК 004.42
ББК 32.973-018
Ш55

Рецензенты:
Доктор физико-математических наук, профессор
зав. кафедрой физики и прикладной математики
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
С. М. Аракелян

Кандидат технических наук
генеральный директор ООО «ФС Сервис»
Д. С. Квасов

Издается по решению редакционно-издательского совета ВлГУ

Шишкина, М. В. Основы программирования : практикум /
Ш55 М. В. Шишкина ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. –
Владимир : Изд-во ВлГУ, 2021. – 112 с. – ISBN 978-5-9984-1408-4.

Изложены теоретические основы программирования, представлено достаточное количество примеров, приведены задания для лабораторных и самостоятельных работ, а также вопросы для самоконтроля и закрепления материала.

Предназначен для студентов бакалавриата направлений подготовки 01.03.02 «Прикладная математика и информатика», 02.03.02 «Математическое обеспечение и администрирование информационных систем», 02.03.01 «Математика и компьютерные науки».

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Ил. 36. Табл. 3. Библиогр.: 5 назв.

УДК 004.42
ББК 32.973-018

ISBN 978-5-9984-1408-4

ПРЕДИСЛОВИЕ

Стремительное развитие информационных технологий, проникновение их во все сферы современной жизни влечёт за собой повышенный интерес студентов к получению образования и последующему построению карьеры в области информационных технологий.

Полноценное освоение профессии невозможно без тщательного и вдумчивого изучения основ программирования. Именно знание основ программирования открывает перед студентами возможности дальнейшего понимания всех нюансов современных информационных технологий. Освоение основ программирования позволяет студентам не только приобрести навыки самостоятельной разработки алгоритмических решений, написания и отладки кода, но в дальнейшем самостоятельно углублять свои знания по изучаемому языку программирования, изучать другие языки программирования, принимать самостоятельное, обдуманное решение при выборе средств разработки программного продукта. Не менее важно приобретение навыков не только разработки, но и понимания уже написанного кода, поиска и исправления ошибок.

Практикум разработан в соответствии с рабочими программами дисциплины «Основы программирования» и может быть рекомендован в качестве основного пособия для освоения дисциплины «Основы программирования» студентам направлений подготовки 01.03.02 «Прикладная математика и информатика», 02.03.02 «Математическое обеспечение и администрирование информационных систем», 02.03.01 «Математика и компьютерные науки».

ВВЕДЕНИЕ

Практикум позволяет закрепить и углубить знания, полученные в ходе освоения школьного курса информатики и ИКТ. Далее, опираясь на эти знания, студент сможет плавно перейти к изучению основ программирования на примере языка *C++*.

Практикум содержит девять разделов, позволяющих подготовиться к выполнению лабораторных работ курса.

Каждый раздел включает: подробную теоретическую часть с достаточным количеством примеров по изучаемой теме, цель и задачи лабораторной работы, контрольные вопросы и задания для самостоятельной работы.

Контрольные вопросы составлены таким образом, что позволяют студенту повторить и закрепить материал изучаемой темы, самостоятельно оценить свой уровень знаний и подготовиться к защите лабораторной работы.

Задания для самостоятельной работы направлены на закрепление знаний и навыков, полученных студентами во время аудиторных занятий.

Структура практикума организована таким образом, что каждая следующая работа основана на знаниях и навыках, полученных при выполнении предыдущей работы.

В приложении практикума содержатся требования к отчёту по лабораторной работе, рекомендации по форматированию отчёта, некоторые алгоритмы для самостоятельного изучения и справочные данные по языку программирования *C++*.

1. ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ

Слово «алгоритм» происходит от имени арабского ученого Аль-Хорезми, жившего в IX веке. Среди широкого круга его интересов особое место занимали исследования в области математики. В одной его, к сожалению, не сохранившейся до наших дней работе была впервые описана позиционная десятичная система счисления и сформулированы правила вычисления в этой системе.

Алгоритм – чёткое предписание исполнителю (человеку или автомату) выполнить точно определённую последовательность действий, направленных на достижение заданной цели.

Исполнителем алгоритма может быть человек или автоматическое устройство, способное воспринимать запись алгоритма.

Выделяют следующие ***свойства алгоритма***, отличающие его от любых других действий.

1. *Дискретность* – разбиение алгоритма на ряд отдельных законченных действий – шагов.

Каждое действие должно быть закончено исполнителем алгоритма прежде, чем он приступит к исполнению следующего действия.

2. *Точность* – однозначность указаний.

На каждом шаге однозначно определено преобразование объектов среды исполнителя, полученной на предыдущих шагах алгоритма. Таким образом, при многократном применении алгоритма к одному и тому же набору исходных данных на выходе каждый раз должен быть получен один и тот же результат. На каждом шаге алгоритма однозначно определено, какую команду надо выполнять следующей.

3. *Понятность* – однозначное понимание исполнителем каждого шага алгоритма.

Алгоритм должен быть записан на понятном для исполнителя языке, то есть состоять из команд, входящих в систему команд исполнителя (СКИ).

4. *Результативность* (конечность) – обязательное получение результата за конечное число шагов.

Каждый шаг (и алгоритм в целом) после своего завершения даёт среду, в которой все объекты однозначно определены. Если это по каким-либо причинам невозможно, то алгоритм должен сообщить пользователю, что решения задачи не существует.

В любом случае работа алгоритма должна быть завершена за конечное число шагов.

Вопрос о бесконечных алгоритмах в этом курсе не рассматривается.

5. *Массовость* – применение алгоритма к решению целого класса однотипных задач.

Способы записи алгоритма

Выделяют следующие формы записи алгоритмов:

1) словесную запись (псевдокоды);

2) графическую запись (блок-схемы);

3) язык программирования;

4) математическую формулу.

Словесная форма записи алгоритма – это описание на естественном языке последовательных этапов обработки данных. Этот способ широко распространён в быту, но крайне редко используется в профессиональной деятельности, так как такие описания строго не формализованы, поэтому может возникнуть неоднозначность в понимании инструкций.

Алгоритм, записанный с помощью псевдокода, представляет собой полуформализованное описание на условном алгоритмическом языке, включающее как основные элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и т. п.

Графическая форма записи, называемая также *блок-схемой* алгоритма, представляет собой изображение алгоритма в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий. Графическая запись является более компактной и наглядной по сравнению со словесной.

Требования к представлению алгоритма в виде блок-схемы содержатся в ГОСТ 19.701-90 «ЕСПД. Схемы алгоритмов, программ данных и систем. Обозначения условные и правила выполнения».

Требования к форме и размерам блоков изложены в ГОСТ 19.003-80 «Схемы алгоритмов и программ. Обозначения условные и графические».

При отображении блоков размер *a* выбирается из ряда 15, 10, 20 мм, допускается увеличение значения параметра *a* на число, кратное пяти. Сторона *b* соотносится со стороной *a* следующим образом: $b = 1.5a$.

Блок-схема, представляющая алгоритмическое решение задачи, должна содержать блок, представленный на рис. 1. Этот блок обозначает начало и конец алгоритма, вход во внешнюю среду и выход из среды. Обычно этот блок имеет следующее содержание: «начало»/«конец», «запуск»/«останов», «перезапуск». В рамках курса «Основы программирования» блок-схемы всех учебных задач следует оформлять таким образом, чтобы первый блок схемы содержал слово «Начало», а последний – «Конец».

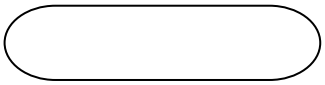


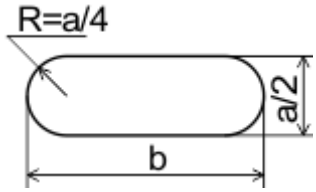
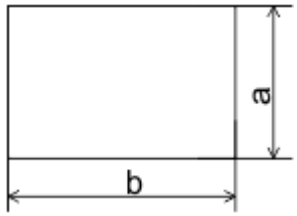
Рис. 1. Блок, обозначающий начало и конец алгоритма

Между этими блоками располагается последовательность блоков, соединённых линиями; линии могут содержать стрелки, показывающие направление потока.

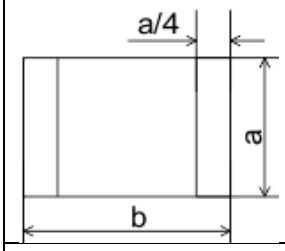
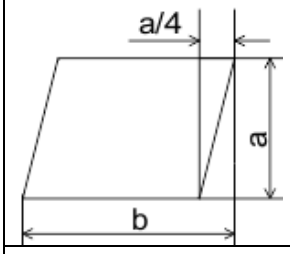
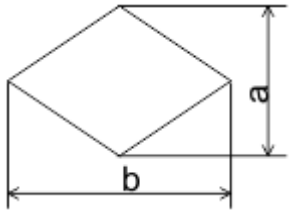
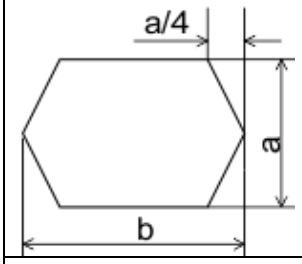
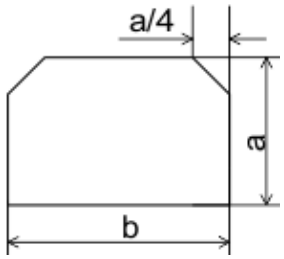
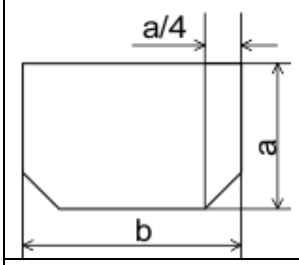
Основные блоки, используемые при представлении алгоритмического решения в виде блок-схемы, представлены в табл. 1. В этой же таблице приведено назначение блоков и указано соотношение их сторон. О соотношении параметров *a* и *b* было упомянуто выше.

Таблица 1

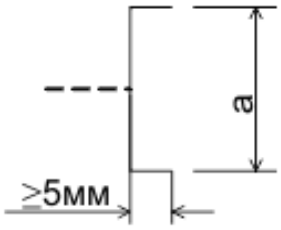
Основные блоки

Форма блока	Назначение блока
	**Терминатор** Обозначает вход и выход во внешнюю среду. «Начало», «Конец» либо прерывание обработки данных
	**Процесс** Обозначает одно или несколько действий обработки данных. Выполнение этих действий ведёт к изменению данных, их значения или формы хранения

Продолжение табл. 1

Форма блока	Назначение блока
	**Предопределённый процесс** (функция) Использование ранее созданных и отдельно описанных алгоритмов или программ
	**Данные** (ввод – вывод) Используют при вводе или выводе данных. Преобразование данных в форму, подходящую для обработки (ввод) или отображения результатов обработки (вывод)
	**Решение** (условие), или переключатель Выбор направления выполнения алгоритма. Блок содержит некоторое условие, имеет один вход и несколько альтернативных выходов. По какой ветви будет произведена дальнейшая работа – определяется значением этого условия. Значения могут быть подписаны рядом с соответствующими линиями
	**Подготовка** (цикл с параметром) Может содержать установку переключателя, модификацию индекса
Начало цикла  Конец цикла 	**Граница цикла** Состоит из двух частей, отображающих начало и конец цикла, имеющих один и тот же идентификатор; условия инициализации, приращения, завершения цикла помещают в начале или конце цикла в зависимости от того, какой это цикл: с предусловием или с постусловием

Окончание табл. 1

Форма блока	Назначение блока
	**Соединитель** Используют при необходимости обрыва линии и продолжении её в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение
	**Комментарий** Используют для описания и/или пояснения каких-то действий
	Пропуск Три точки используют для отображения пропущенного символа или группы символов

На рис. 2 представлен пример использования комментариев. Если комментарий относится к совокупности блоков, то эти блоки необходимо заключить в пунктирную рамку.

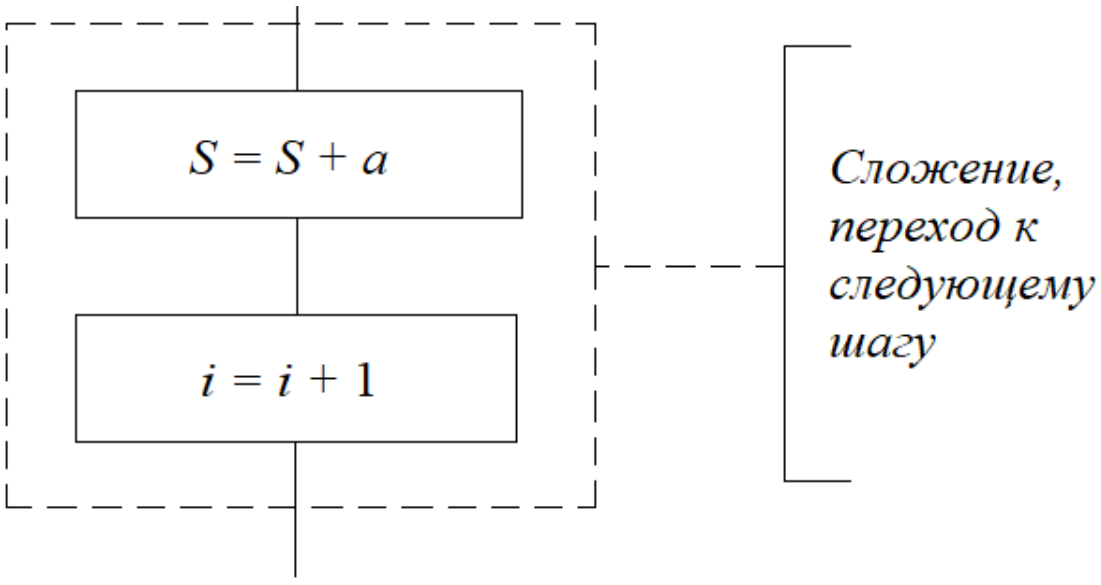


Рис. 2. Использование комментариев в блок-схеме

Если алгоритмическое решение задачи достаточно сложное и объёмное, то запись его в виде блок-схемы может получиться довольно громоздкой. Этого можно избежать, заменив некоторую последовательность блоков, решающих определённую подзадачу, блоком «предопределённый процесс», содержащим обращение к ранее разработанному и описанному алгоритму, например вызову функции. Заменяемый блок должен быть не элементарен (то есть должен быть не один блок, а совокупность блоков), количество входящих в него шагов выбирается в зависимости от конкретной задачи, если подзадача правильно определена и решена, то заменяемый блок (в данном случае под блоком понимают совокупность шагов алгоритма, а не конкретный блок блок-схемы) имеет все признаки самостоятельного алгоритма, то есть представляет собой законченный алгоритм решения соответствующей задачи.

Блоки в графическом представлении алгоритма связаны друг с другом только через точки входа и выхода, поэтому если блок верно решает свою задачу, то его внутренняя структура несущественна для остальной части алгоритма.

Разбиение задачи на подзадачи удобно на первых этапах разработки алгоритмического решения, детальная разработка блоков производится позднее и, возможно, другими разработчиками.

Блоки в схеме соединены друг с другом соединительными линиями, которые могут содержать стрелки, задающие направление. Если стрелки отсутствуют, то направление по вертикали считается сверху вниз, а по горизонтали – слева направо. Линии должны быть направлены к центру символа.

При отображении блок-схемы блоки в ней должны быть размещены равномерно, нужно стремиться к небольшой длине соединений и минимальному числу длинных линий в схеме. Следует избегать пересечения линий, в точках пересечения не допускается изменения направления линий. Во избежание лишних пересечений и излишней длины линий их следует разрывать, используя соединители.

В случае необходимости слияния линий потока место слияния должно быть обозначено точкой. Примеры слияния линий представлены на рис. 3.

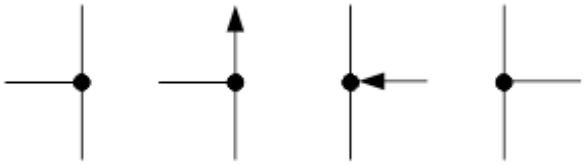


Рис. 3. Слияние линий потока

Текст внутри блока должен быть минимально необходимым для понимания назначения этого блока. Текст записывается слева направо и сверху вниз. Если нужный текст не помещается в блок, его необходимо вынести в комментарии.

Выделяют следующие три основные алгоритмические конструкции (или виды алгоритмов): *следование* (линейный), *ветвление* (разветвляющийся) и *цикл*, или *повторение* (циклический алгоритм). Алгоритмы, содержащие в себе несколько видов структур, называют комбинированными.

Следование – это алгоритмическая структура, в которой все команды выполняются последовательно, одна за другой. Блок-схема такой конструкции, показанная на рис. 4, представляет собой последовательность блоков «процесс». Алгоритмы, в которых используется только алгоритмическая конструкция «следование», называются *линейными алгоритмами*.

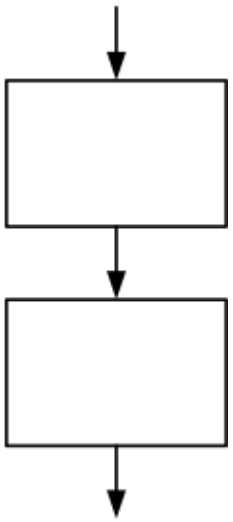
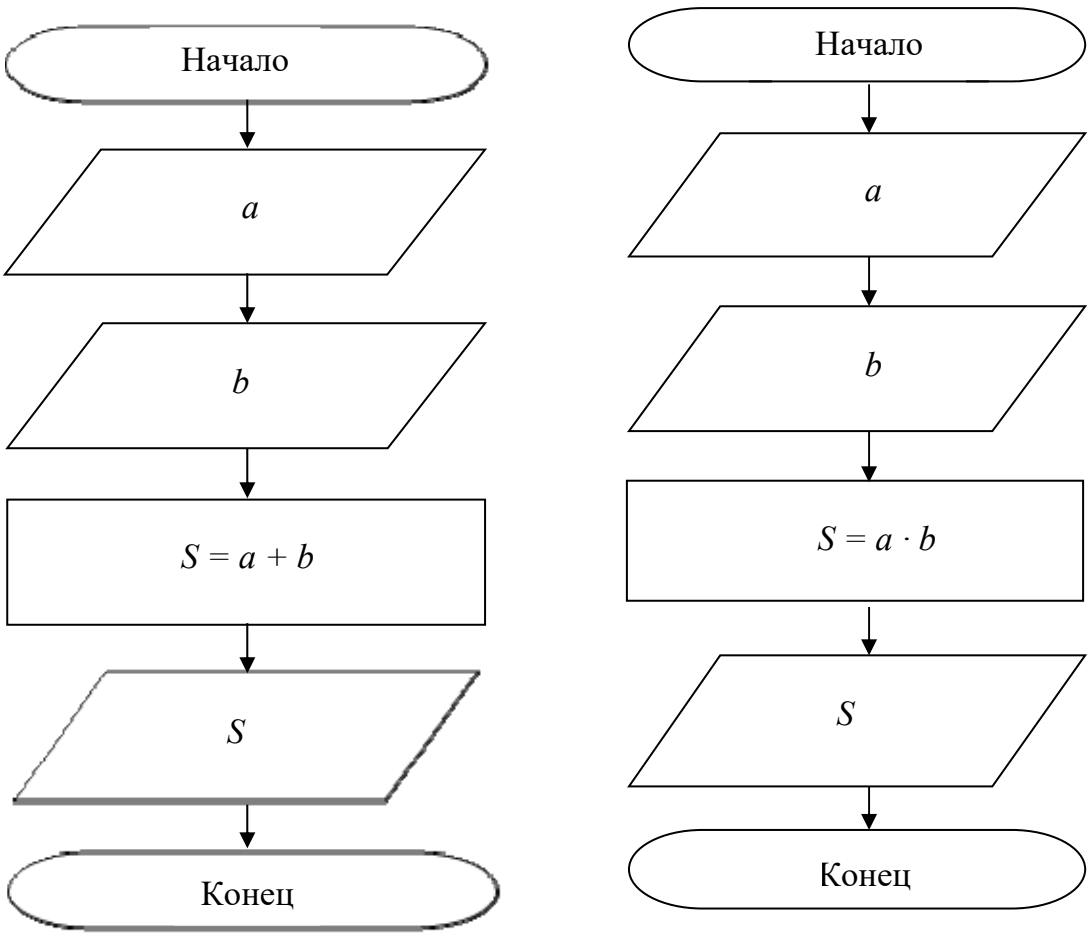


Рис. 4. Схематичное представление алгоритмической структуры «следование»

Например, при помощи алгоритмической структуры «следование» можно вычислить площадь прямоугольника со сторонами a и b или сумму двух слагаемых. Примеры блок-схем линейных алгоритмов, реализованных при помощи алгоритмической структуры «следование», представлены на рис. 5.



а) *б)*

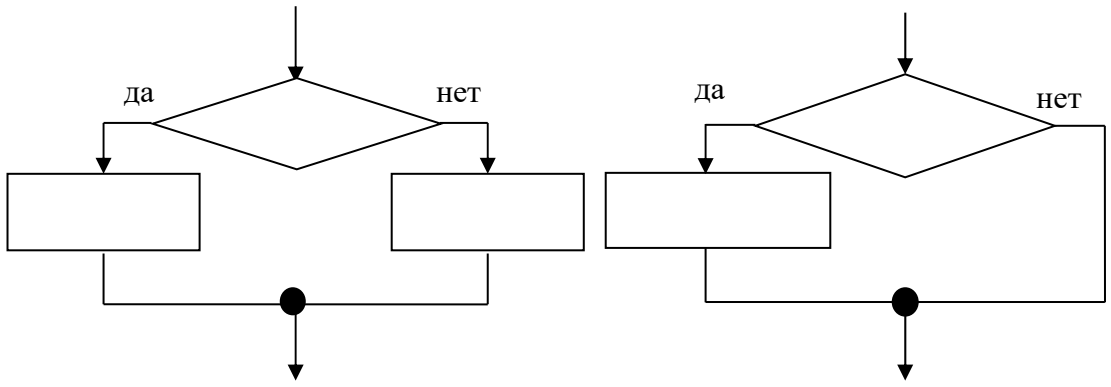
Рис. 5. Блок-схемы линейных алгоритмов: а – вычисление суммы двух чисел; б – вычисление площади прямоугольника

Ветвление – это алгоритмическая структура, в которой в зависимости от значения условия будет выполняться либо одна, либо другая последовательность действий.

В качестве условия может быть использовано любое понятное исполнителю утверждение, истинное или ложное. Таким образом, структура «ветвление» включает в себя условие и одну или две последовательности команд.

Алгоритмическая структура «ветвление» может быть реализована в двух формах: полной и неполной. *Полное ветвление* содержит два действия (последовательности команд), одно из которых будет выполнено при истинности условия, а второе – когда условие ложно.

Неполное ветвление содержит только одну последовательность команд, выполняющуюся при истинности условия. На рис. 6 приведены схемы полной и неполной форм ветвления.



а) *б)*

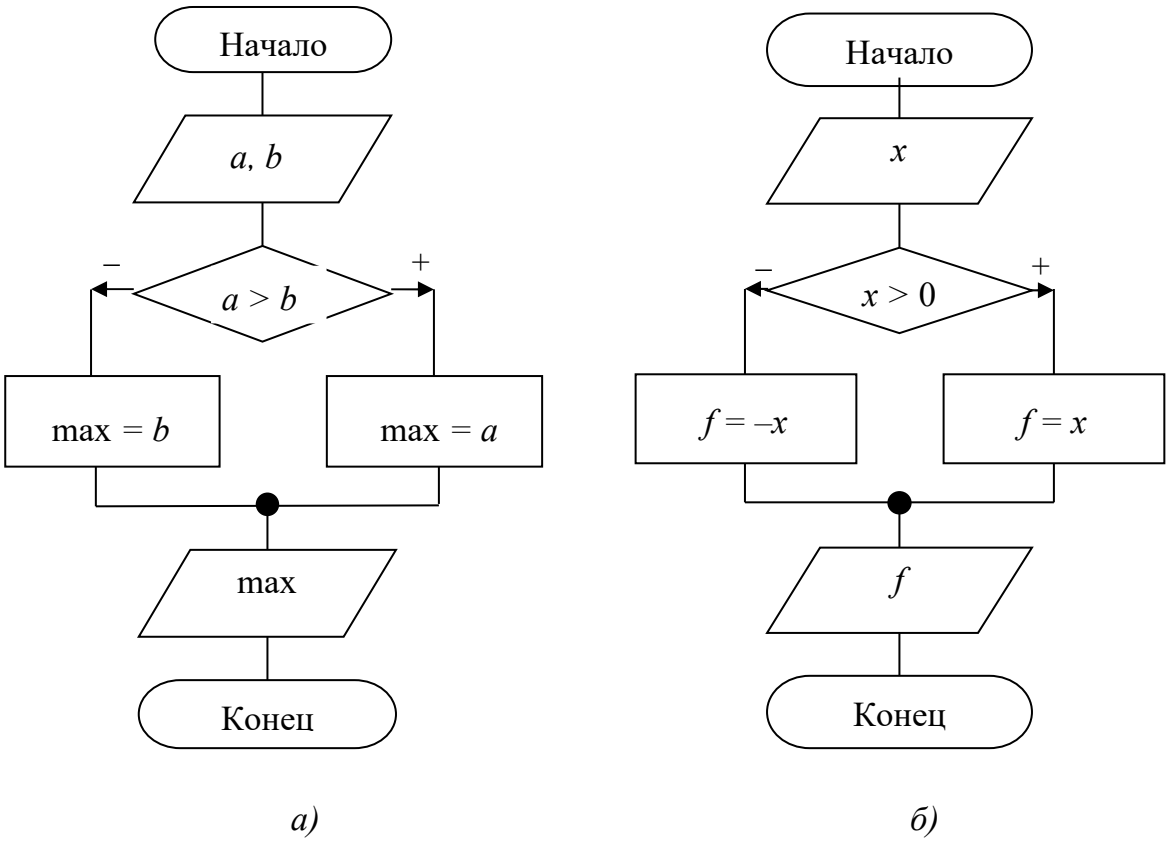
Рис. 6. Алгоритмическая структура «ветвление»:
а – полное ветвление; б – неполное ветвление

Алгоритмы, в основе которых лежит алгоритмическая конструкция «ветвление», называют *разветвляющимися*.

На рис. 7 приведены примеры реализации в виде блок-схем разветвляющихся алгоритмов: поиска максимального из двух введённых значений и вычисления функции $f(x) = |x|$.

В алгоритме, представленном в виде блок-схемы на рис. 7, *а*, первым действием осуществляется ввод переменных a и b, далее происходит проверка условия $a < b$; если условие истинно, то переменная max принимает значение a, иначе – значение b. Последним действием производится вывод значения переменной max.

В алгоритме, представленном в виде блок-схемы на рис. 7, *б*, в блоке ввода данных указана переменная x. В следующем блоке происходит анализ введённой переменной. Если условие $x > 0$ истинно, переменная f получает значение x, иначе переменная f получает значение $–x$. Далее производится вывод на экран значения переменной f. На этом работа алгоритма завершается.



а) *б)*

Рис. 7. Разветвляющие алгоритмы:
а – поиск максимального значения; б – модуль числа

Условие в конструкции «ветвление» может быть простым и состоять из одной операции или составным, в этом случае условие составлено с использованием логических операций «и», «или», «отрицание» (например, при определении принадлежности точки к отрезку [*a, b*]). Алгоритм представлен на рис. 8.

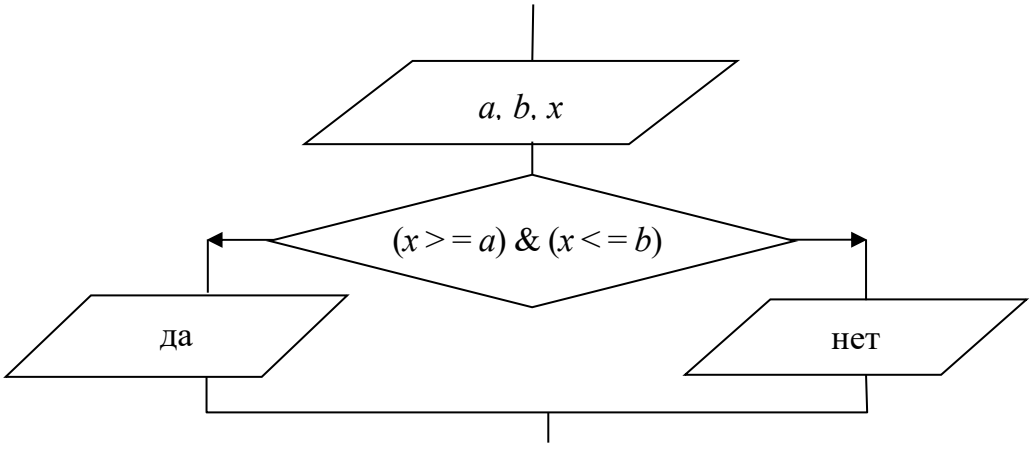


Рис. 8. Алгоритм определения принадлежности точки к отрезку

Одна или обе ветви условной конструкции также могут содержать условие, в этом случае говорят о вложенном ветвлении. Структурная схема такой конструкции приведена на рис. 9.

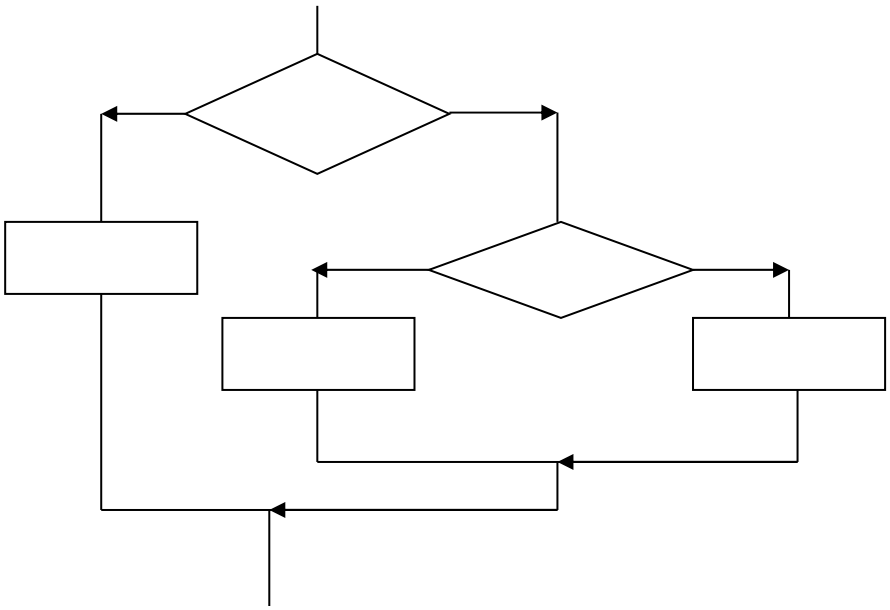


Рис. 9. Структурная схема конструкции «ветвление»

Повторение (***цикл***) – алгоритмическая конструкция, содержащая последовательность многократно повторяющихся действий. Алгоритмы, в основе которых лежит конструкция «повторение», называют *циклическими*, или *циклом*.

Последовательность действий, многократно повторяющуюся в процессе выполнения цикла, называют *телом цикла*.

В зависимости от способа организации повторений выделяют три вида циклов:

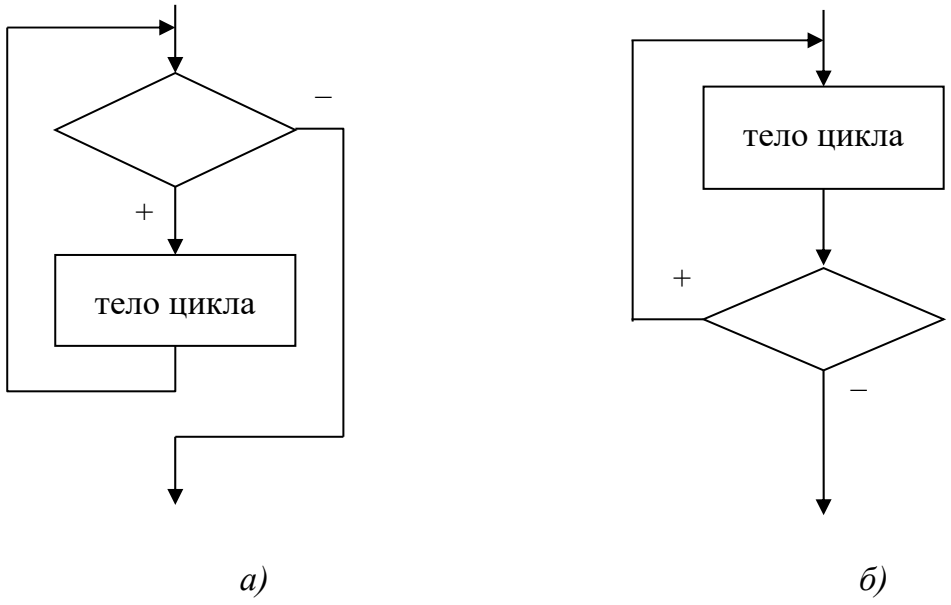
1) цикл с условием *продолжения* работы;
2) цикл с условием *окончания* работы;
3) цикл с *заданным числом повторений*.

В зависимости от взаимного расположения тела цикла и условия продолжения или завершения цикла выделяют два типа циклов:

1) цикл с *постусловием*;
2) цикл с *предусловием*.

В конструкции цикла с предусловием сначала происходит проверка условия; если условие истинно, то выполняется тело цикла или выход из цикла. В случае цикла с постусловием тело цикла

выполняется минимум один раз, после чего осуществляется проверка условия, и в зависимости от результата происходит повторение тела цикла или выход из цикла. Схематично конструкции цикла с постусловием и цикла с предусловием представлены на рис. 10.

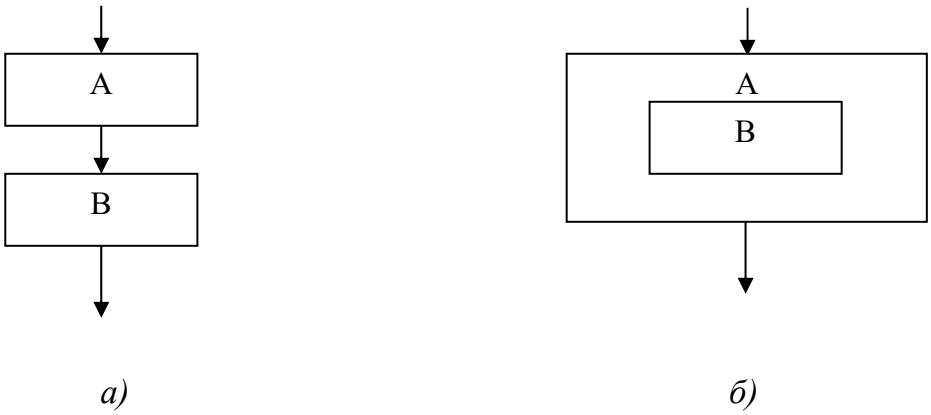


а) *б)*

Рис. 10. Алгоритмическая конструкция «повторение»:
а – цикл с предусловием; б – цикл с постусловием

Для организации конструкции «повторение» до начала цикла необходимо задать начальные значения переменных, используемых в теле цикла. В теле цикла необходимо предусмотреть изменение значения переменных, анализируемых в условии продолжения, или условие выхода из цикла либо изменение параметра, отвечающего за номер шага, если это цикл с заранее известным количеством повторений.

Если в одном алгоритме используется несколько конструкций повторения, то они могут либо следовать друг за другом, в этом случае это *последовательные* циклы, либо один из циклов может быть частью тела другого цикла, в таком случае первый цикл называют *вложенным*, а второй – *внешним*. На рис. 11 схематично показаны способы группировки конструкций «повторение».



а) *б)*

Рис. 11. Алгоритмическая конструкция «повторение»:
а – последовательные циклы; б – вложенный цикл

Каждая из рассмотренных алгоритмических конструкций может быть реализована на любом языке программирования. Перед реализацией задачи на языке программирования необходимо грамотно представить алгоритм решения задачи с помощью блок-схемы.

Рассмотрим пример решения задачи с использованием комбинации различных алгоритмических конструкций.

Задача: вычислить произведение десяти введённых с клавиатуры целых чисел.

На рис. 12 представлен алгоритм решения этой задачи в виде блок-схемы.

В первом блоке «действие» производится задание начальных данных – параметров цикла. Количество повторений – *n*, начальное значение произведения – *P*, номер текущей итерации – *i*. В теле цикла производятся следующие действия: ввод значения переменной *x,* вычисление текущего значения произведения, которое хранится в переменной *P*, увеличение на единицу значения переменной счётчика. Условие продолжения цикла будет проверено после выполнения тела цикла. В задаче использован цикл с постусловием. Если значение параметра цикла *i* не больше значения *n,* то выполнение цикла продолжается. Как только значение параметра *i* превысит значение *n,* выполнение цикла будет завершено. После завершения выполнения цикла осуществляется вывод произведения, хранящегося в переменной *P,* на экран.

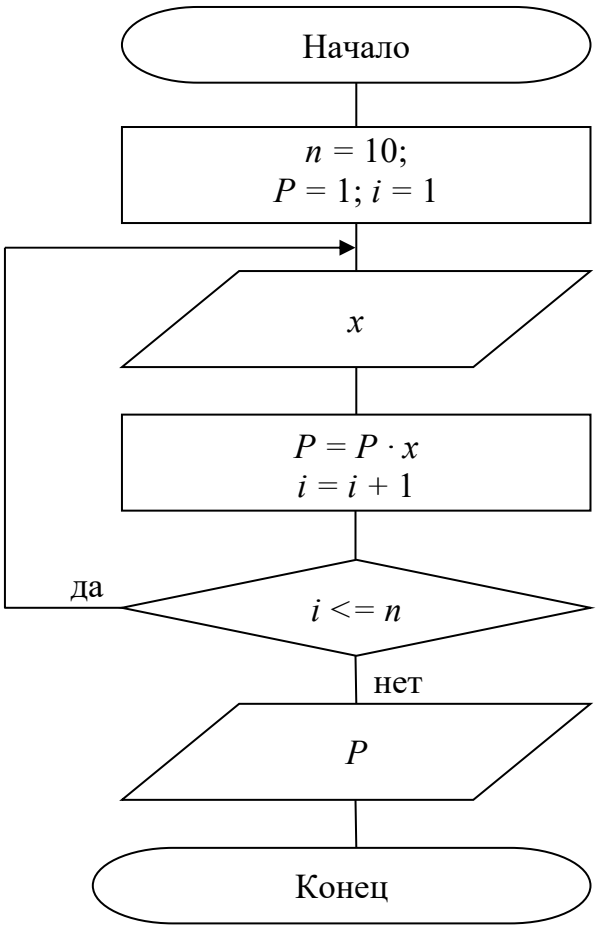


Рис. 12. Блок-схема алгоритма нахождения произведения десяти введённых с клавиатуры чисел

Лабораторная работа № 1
ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ

Цель работы. Повторить известные из школьного курса информатики и ИКТ основные алгоритмические структуры, понятие алгоритма, свойства алгоритмов, способы их представления. Получить и закрепить на практике навыки разработки алгоритмических решений и представления их на естественном языке в виде блок-схемы и на языке программирования высокого уровня.

Постановка задачи

1. Вычислить сумму двух введённых с клавиатуры вещественных чисел

$S = a + b$,

где a и b – вещественные числа.

2. Вычислить сумму пяти введённых с клавиатуры чисел

$$S = \sum_{i=1}^{n=5} a_i.$$

3. Вычислить значение переменной *y(x)*, где *x* – введённое с клавиатуры значение целочисленной переменной

$$y = \begin{cases} x^2, & x < 0 \\ -x, & x > 0 \\ 7, & x = 0 \end{cases}.$$

4. Вычислить сумму пяти введённых с клавиатуры чисел, больших нуля. Отрицательные числа в сумме не учитываются, число попыток ввода не ограничено, то есть ввод значений a_i осуществляется до достижения необходимого количества введённых чисел, больших нуля

$$S = \sum_{i=1}^{n=5} a_i, \qquad a_i > 0.$$

5. Вычислить сумму введённых с клавиатуры чисел. В сумме учитывать только числа, большие нуля, число попыток ввода ограничить пятью

$$S = \sum_{i=1}^{n} a_j, \qquad a_j > 0.$$

6. Вычислить значение переменной *y* при всех значениях переменных *x* и *a* из указанного интервала

$$y = \frac{x^2 + a}{x + a}.$$

Значения следующие: $x_0 = 1$; $x_k = 5$; $\Delta x = 0.2$; $a_0 = 1.5$; $a_k = 10$; $\Delta a = 0.5$.

Вопросы для самоконтроля и подготовки к защите лабораторной работы

1. Что такое алгоритм?
2. Какие существуют способы представления алгоритмических решений?
3. Перечислите достоинства и недостатки способов из предыдущего вопроса.

4. Назовите и представьте структурную схему основных алгоритмических конструкций.

5. Какие алгоритмы называют линейными?

6. Какие алгоритмы называют разветвляющимися?

7. Что такое цикл, тело цикла, итерация цикла?

8. Составьте алгоритм определения количества одинаковых цифр в трёхзначном числе.

9. Какой цикл называют циклом с предусловием, а какой – циклом с постусловием?

10. Какие действия необходимо выполнить до входа в цикл для организации цикла?

11. Какие действия необходимо выполнить в теле цикла для организации своевременного выхода из цикла?

12. В каком случае предпочтительнее использовать цикл с постусловием?

13. Что такое вложенный цикл? Поясните на схеме.

14. Какое действие необходимо совершить в теле цикла, чтобы цикл не был вечным?

15. Представьте блок-схему цикла с предусловием и возможностью досрочного выхода из цикла.

Задания для самостоятельной работы

1. Дано натуральное число n. Найти сумму первой и последней цифры этого числа.

2. Дано натуральное число n. Переставить местами первую и последнюю цифры этого числа.

3. Даны два натуральных числа m и n ($m \leq 9999$, $n \leq 9999$). Проверить, есть ли в записи числа m цифры, совпадающие с цифрами в записи числа n.

4. Дано натуральное число n, $n \leq 9999$. Проверить, есть ли в записи числа три одинаковые цифры.

5. Дано натуральное число n, $n \leq 99$. Дописать к нему цифру k в конец и начало.

6. Даны натуральные числа n, k. Проверить, есть ли в записи чисел n и k цифра m.

7. Вычислить $y = asin(x)$ для всех a и x, заданных значениями a_0 и x_0 и шагом по a и x соответственно.

8. Вычислить $y = cos2(x) + ac$ для всех a и x, заданных значениями a_0 и x_0 и шагом по a и x соответственно.

9. Вычислить $z = 2\prod_{i=1}^{k}\left(a_i - \frac{1}{a_i}\right)$, где k задано, a_i задаёт пользователь вводом с клавиатуры.

10. Вычислить $z = 2\sum_{i=1}^{k}(a_i - \frac{1}{a_i})^2$, где k задано, a_i задаёт пользователь вводом с клавиатуры.

11. Вычислить max $\left\{k^3 \sin\left(n + \frac{k}{n}\right), k = \overline{1,n}\right\}$, где n задано.

12. Вычислить функцию f = max $(x + y + z, xyz)$.

13. Вычислить f = min (x, y, z).

14. Даны три целых числа x, y, z. Вывести эти числа на экран в порядке убывания.

15. Вводится символ, вывести сообщение о том, что это за символ: цифра, буква или знак?

16. Задано любое целочисленное x, определить, кратно ли оно трём.

17. Спортсмен в первый день пробегает S км, где $S < 25$, каждый следующий день он увеличивает дистанцию на 10 процентов от нормы предыдущего дня, при этом по достижении 25 км дистанция не увеличивается и остаётся равной 25 км. Начиная с какого дня спортсмен будет пробегать 25 км в день?

18. Составить алгоритм, с помощью которого можно определить, является ли треугольник со сторонами a, b, c равносторонним.

19. Составить алгоритм возведения входного числа в квадрат, если оно чётное, и в куб, если нечётное.

20. Одноклеточная амёба каждые три часа делится на две клетки. Составить алгоритм вычисления времени, через которое количество амёб достигнет n штук, где n – задаваемое пользователем.

2. БАЗОВЫЕ ТИПЫ ЯЗЫКА *C++*

Тип данных определяет количество выделяемой памяти под переменную или константу соответствующего типа, определяет правила хранения данных этого типа, допустимые операции для данных этого типа.

Диапазон допустимых значений данных конкретного типа определяется количеством памяти, выделенным под переменную, и правилами хранения данных.

Типы данных языка *C++* можно разделить на две большие группы: *базовые* (основные, встроенные, фундаментальные) и *пользовательские* (составные, производные).

К базовым типам языка относят: *void, int, char, float double, bool.*

Спецификаторы типов *signed, unsigned, short, long* влияют на внутреннее представление целых чисел (знаковое или беззнаковое представление), количество выделяемой памяти (*short, long*) и, следовательно, диапазон значений соответствующей переменной.

Спецификаторы *signed* (знаковый), *unsigned* (беззнаковый) могут быть применены к типам *int, short, long, char*.

По умолчанию представление этих данных знаковое. Наличие спецификатора *unsigned* говорит о том, что в переменной соответствующего типа могут храниться значения из диапазона от нуля и до максимального значения, определяемого количеством памяти, выделяемой под переменную данного типа.

Спецификаторы *short* и *long* могут употребляться с типом *int*, увеличивая или уменьшая количество выделяемой памяти вдвое. При этом тип *int* может быть не указан, в таком случае он подразумевается.

Для целых чисел в памяти компьютера существует два способа представления: *беззнаковое* (только для неотрицательных целых чисел) и *знаковое* (в диапазон этих чисел входят положительные, ноль, отрицательные числа).

Для беззнакового представления целого все ячейки отводятся под представление самого числа. Например, в 1 байте (8 бит) можно представить беззнаковые числа от 0 до 255. Поэтому если известно,

что числовая величина является неотрицательной, то рациональнее представлять её как беззнаковую.

0	0	0	0	0	0	0	0	0 – минимальное значение
1	1	1	1	1	1	1	1	255 – максимальное значение

Для представления чисел со знаком старший (крайний слева) бит отводится под знак числа, остальные разряды – под само число. Если число положительное, то в знаковый разряд помещается 0, если отрицательное – 1.

Положительные числа в памяти компьютера хранятся в прямом коде, отрицательные – в дополнительном.

Незначащие разряды, следующие за знаковым, заполняются нулями. Например, если под переменную отведено восемь разрядов, то двоичные числа 1001 и –1001 в памяти будут представлены так:

0	0	0	0	1	0	0	1
1	0	0	0	1	0	0	1

Прямой код отрицательного числа отличается от прямого кода соответствующего положительного числа лишь содержимым знакового разряда. Отрицательные целые числа в памяти компьютера хранятся не в прямом, а в дополнительном коде. Такой подход позволяет сократить количество операций, производимых при вычислениях. Например, операция сложения при таком подходе сводится к их поразрядному сложению.

Дополнительный код положительного числа равен прямому коду этого числа. Дополнительный код отрицательного числа m равен $2^k - |m|$, где k – количество разрядов в ячейке.

Для получения дополнительного кода отрицательного числа необходимо:

– модуль отрицательного числа представить в прямом коде;

– инвертировать значение всех бит числа, то есть все нули заменить на единицы, а единицы – на нули;

– к полученному обратному коду прибавить единицу.

Получим 8-разрядный дополнительный код числа –57:

00111001 – |–57| = 57 в прямом коде
11000110 – –57 в обратном коде
11000111 – –57 в дополнительном коде

Для перевода из дополнительного кода в прямой нужно проделать эти действия в обратном порядке, заменив сложение вычитанием. При этом важно помнить правила вычитания двоичных чисел. В случаях, когда занимается единица старшего разряда, она даёт две единицы младшего разряда. Если занимается единица через несколько разрядов, то она даёт по одной единице во всех промежуточных нулевых разрядах и две единицы в том разряде, для которого занималась.

Пример вычитания в двоичной системе:

	1	0	1	1	0	0	1	89
–		1	0	0	1	1	0	38
		1	1	0	0	1	1	51

Количество памяти, выделяемое под данные типа *int,* зависит от процессора и компилятора. Независимо от разрядности процессора под данные *short* выделяется 2 байта, под *long* – 4 байта.

Диапазон представления целочисленных данных определяется следующим образом.

Если это беззнаковое представление числа, минимальное значение – ноль, максимальное определяется следующим образом: $2^n - 1$, где n – число разрядов, отводимых под число. Например, для одного байта диапазон записи целых чисел будет таким: $0 – 2^8 – 1$.

При знаковом представлении числа старший бит отводится под хранение знака, таким образом, число разрядов, отводимых под хранение самого числа, уменьшается на единицу. Знаковое целое число будет лежать в диапазоне от $–2^{n-1}$ до $2^{n-1} – 1$.

Для хранения вещественных типов данных в *C++* определены три типа: *float, double, long double*.

При представлении чисел с плавающей запятой часть разрядов выделенной памяти отводится для записи порядка числа, остальные разряды – для записи мантиссы. По одному разряду в каждой группе

отводится для изображения знака порядка и знака мантиссы. Для того чтобы не хранить знак порядка, используют смещённый порядок

$$a = \pm m \cdot q^p,$$

где m – целое число, называемое мантиссой числа, q – основание системы счисления, p – порядок числа. Схематично представление хранения вещественных чисел изображено на рис. 13.

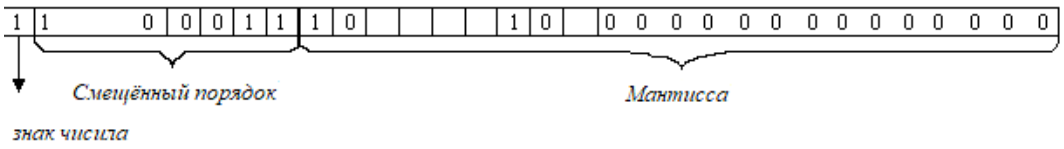


Рис. 13. Схематичное представление хранения вещественных чисел

Для хранения логических значений в языке программирования *C++* служит тип `bool`, имеющий два значения: *true* и *false*. При этом целочисленное значение 0 будет трактоваться как ложь, любое другое значение – как истина. Однако при написании кода подобных приведений типов стоит избегать.

Множество значений типа `void` пусто. Этот тип используют при описании функций, не возвращающих значение, при объявлении указателей и в операциях приведения типов.

Для работы с символами используют типы `char` и `wchar_t`.

Для хранения символов из 256 набора *ASCII* существует тип `char`. Этот тип можно использовать для хранения целых чисел, не превышающих границы диапазона, определяемого количеством выделяемой памяти, и знакового, и беззнакового представления.

Для работы с символами, для кодировки которых недостаточно одного байта, используют тип `wchar_t`.

В стандарте языка *C++* строго не указано количество байт, отводимое под определённый тип. Есть только указания о соотношении количества выделяемой памяти:

$$1 \equiv sizeof(char) \le sizeof(short) \le sizeof(int) \le sizeof(long)$$
$$1 \le sizeof(bool) \le sizeof(long)$$
$$sizeof(float) \le sizeof(double) \le sizeof(long\ double)$$
$$sizeof(N) \equiv sizeof(unsigned\ N) \equiv sizeof(signed\ N)$$

Таким образом, если алгоритм зависит от количества памяти, выделяемой под переменную, то необходимо предварительно этот размер определить. Сделать это можно при помощь оператора `sizeof()`, указав в качестве операнда тип или переменную соответствующего типа. Например, так: `printf("int =%u\n", sizeof(int));`

Граничные значения диапазона для целых чисел хранятся в файле `<limits.h>`, для вещественных – в файле `<float.h>`:

```
int M = INT_MAX;
```

Переменная – это именованная область памяти, хранящая значения определённого типа. Перед использованием переменной или именованной константы в коде их необходимо объявить. Для этого нужно указать название типа и через пробел – имя переменной. Заканчивается объявление точкой с запятой:

<имя типа> *<имя переменной>*`;`

```
int a;
```

Объявление переменной может быть совмещено с её инициализацией (заданием начального значения):

<имя типа имя переменной> `=` *<значение>*`;`

```
int a = 25;
```

Если необходимо объявить несколько однотипных имён, то их разделяют запятой:

```
int a,b;
```

Значение переменной может быть изменено во время работы программы, значение константы в соответствии с её названием остаётся постоянным. Константы должны быть проинициализированы при объявлении:

<имя типа> `const` *<имя константы>* `=` *<значение>*`;`

```
int const N=7;
```

Правила именования в языке C++

Инициализатор (имя) может включать в себя латинские буквы, цифры и символ подчёркивания. Имя не должно начинаться с цифры. Символы в верхнем и нижнем регистре считаются разными, поэтому

`int a;` и `int A;` – это объявление двух разных переменных. При выборе имени следует руководствоваться назначением переменной.

Описание идентификатора задаёт его область действия. Под *областью действия* понимают часть кода, в которой можно обращаться через указанный идентификатор к связанной с ним области памяти.

Область действия может быть локальной или глобальной. *Локальной* будет переменная, описанная внутри блока. Под блоком подразумевают часть кода, ограниченную фигурными скобками, например тело функции или тело цикла. *Глобальной* переменной называют переменную, объявленную вне какого-либо блока. Обратиться к такой переменной можно в файле, в котором она объявлена от точки объявления и ниже. При этом локальные переменные перекрывают глобальные. Следующий пример это наглядно демонстрирует.

```
    int f() //описание функции
    {
        int x = 5; //объявление локальной перемен-
ной x
        printf("x=%i\n", x); /*вывод значения ло-
кальной переменной*/
        return ++x; /*в точку вызова функции воз-
вращается значение локальной переменной, увеличен-
ное на единицу*/
    }

    int x = 17; //объявление глобальной переменной

    int main()
    {
    printf("x=%i\n", x); /*вывод на экран значения
глобальной переменной*/
        int x = 1;
        printf("x=%i\n", f()); /*вывод на экран
значения, возвращаемого функцией
```

```
        printf("x=%i\n", x); /*вывод значения ло-
кальной переменной*/
        for (int x = 0; x < 3; x++) {
printf("x=%i\n", x); /*вывод в цикле значения ло-
кальной переменной */}}
```

Результат работы будет таким:

x = 17 //вывод значения глобальной переменной
x = 5 //вывод значения локальной для функции переменной
x = 6 //вывод значения, возвращаемого функцией
x = 1 //вывод значения локальной для *main* переменной
//вывод в цикле значения локальной переменной
x = 0 //значение локальной переменной на первой итерации
x = 1 //значение локальной переменной на второй итерации
x = 2 //значение локальной переменной на третьей итерации

Если в выражениях участвуют разнотипные данные, прибегают к приведению типов. Приведение типа может быть *явным* и *неявным* (по умолчанию).

Преобразование по умолчанию осуществляется по правилу «от меньшего к большему», то есть если в вычислении задействованы переменные разных типов, то результат будет того типа, под который отводится больше памяти. При выполнении операции присваивания тип правого операнда приводится к типу левого.

При этом приведение вещественного типа к целому сводится к отбрасыванию дробной части, целого к вещественному – к появлению нулевой дробной части:

```
float a = 5.25;
int b = 3;
int r1 = a*b;
float r2 = a*b;

printf("r1=%i\n", r1);
printf("r2=%f\n", r2);
r2 = r1; printf("r2=%f\n", r2);
```

Результат работы представленного фрагмента кода будет таким:

*r*1 = 15
*r*2 = 15.750000
*r*2 = 15.000000

Явное преобразование типа можно осуществить несколькими способами.

В стиле языка *C*. Для этого необходимо перед идентификатором, тип которого приводится, указать нужный тип в круглых скобках. Например,

```
float b = 7.5;
int a = (int)b; /*значение переменной a будет равно 7*/
b = a; //значение переменной b станет равно 7.000
```

При таком подходе к конвертации типов не происходит проверки на переполнение, поэтому она может быть неправильно использована, например, при конвертации типов `const` или изменении типов данных без учёта их диапазонов. Следовательно, это не самый лучший способ.

В случае, когда есть уверенность, что переполнения не произойдёт, используют операцию оператор `static_cast:`

`static_cast` *<тип> (<выражение>)*

Такое преобразование может выполняться между целыми типами, целыми и вещественными, целыми и перечисляемыми типами, указателями и ссылками на объекты одной иерархии:

```
int a = 54;
char ch = i; //неявное преобразование
```

Такой подход приведёт к предупреждающему сообщению во время компиляции; чтобы избежать этого, лучше сделать так:

```
int a = 54;
char ch = static_cast<char>(a);
```

При подобных преобразованиях внутреннее представление данных может быть изменено, хотя значение останется неизменным.

```
float f = 10.25;
int d = static_cast<int>(f);
```

При этом внутренне представление данных изменится, так как представление в памяти целых и вещественных чисел различное.

Лабораторная работа № 2

БАЗОВЫЕ ТИПЫ ЯЗЫКА *C++*

Цель работы. Изучить базовые типы данных языка *C++*, особенности выполнения операций на этих типах данных, приоритет операций.

Постановка задачи

1. Определить количество памяти, отводимое под переменные базовых типов:

– объявить переменные всех базовых типов, в том числе переменную типа `void`, убедиться в том, что такое объявление невозможно, после чего закомментировать соответствующую строку, проинициализировать объявленные переменные допустимыми значениями, вывести на экран с соответствующими спецификаторами;

– определить количество памяти, отводимое под переменные базовых типов, используя `sizeof()`;

– основываясь на результатах, рассчитать диапазоны всех целых типов данных.

2. Изучить способы приведения типов в языке *C++*. Набрать и отладить следующий фрагмент кода:

```
int a = 25;
float b = 123.54;
char c = 'c';
a = b; printf("a =%i\n", a);
a = c; printf("a =%i\n", a);
```

```
a = 25; printf("a =%i\n", a);
b = a; printf("b =%f\n", b);
b = c; printf("b =%f\n", b);
b = 123,54; printf("b =%f\n", b);
c = a; printf("c =%c\n", c);
c = b; printf("c =%c\n", c);
```

Пояснить полученные результаты.

3. Объявленные выше переменные целого, вещественного и символьного типов вывести на экран со спецификаторами не соответствующих им типов; то есть целочисленную переменную со спецификатором – для вещественного и символьного типов, вещественную – для целого и символьного, символьную – для целого и вещественного типов. Прокомментировать результаты с точки зрения хранения данных в памяти компьютера.

4. Рассмотреть вопросы переполнения типов:

– объявить две символьные переменные, проинициализировать их значениями 65 и 274, вывести значения каждой переменной со спецификаторами для целого и символьного типов;

– объявить две переменные следующих типов: `signed char` и `unsigned char`, проинициализировать их значением 0, вывести значения переменных на экран со спецификаторами для целого знакового и символьного типов. Задать значение объявленных выше переменных равным 255, вывести на экран с указанными выше спецификаторами, увеличить значение каждой переменной на два, вывести на экран изменённые значения со спецификаторами для целого знакового и символьного типов, проанализировать результаты;

– объявить две целочисленные переменные: знаковую и беззнаковую, проинициализировать их максимальным значением для соответствующего типа, увеличить значения на три, вывести значение каждой переменной на экран дважды со спецификаторами для знакового и беззнакового типов, объяснить полученные результаты.

5. Приоритет операций.

В выражениях указать порядок действий. Если выражение синтаксически некорректно, исправить ошибки, вывести на экран значения как искомых переменных, так и всех переменных, значения которых изменялись в выражении.

```
int d, i=1;
d=++i+++2++;
int d, i=1;
d=++i++;
int d = 1;
d+=d++;
int a,b,c,d,k;
b=2; d=3;
k=(a=b)+(c=d);
int i,l,j,k; i=l=j=k=0;
int a = i++ && ++j || k || l++;
int a,b,k;
a=2; b=1;
k=(a!=b)?(a-b++):(++a-b);
int a = 2; int b = 3;
float y1, y2, c=3.5;
y1=c*a/b; y2=c*(a/b)
```

Вопросы для самоконтроля и подготовки к защите лабораторной работы

1. На какие две группы можно разделить все типы данных в языке *C++*?
2. Перечислите известные вам целочисленные типы данных.
3. В чём отличие способов хранения знаковых и беззнаковых целочисленных данных?
4. Каким образом будет определено значение переменной при попытке присвоить ей значение, выходящее за границу диапазона?
5. Какие типы данных вы знаете для работы с вещественными переменными в языке *C++*?
6. Перечислите все известные вам базовые типы языка *C++*.
7. Какой подход к приведению типов называют явным, какой – неявным?
8. Каким образом осуществляется приведение типов операндов в операторе присваивания?

9. По какому правилу происходит работа с данными при использовании их в выражениях разнотипных данных?

10. Какой результат будет получен при попытке увеличения на единицу максимально возможного значения знаковой целочисленной переменной?

Задания для самостоятельной работы

1. Объявить вещественные переменные x, y, z и S. Записать фрагмент кода, присваивающий переменным x, y и z значения, вводимые с клавиатуры. Вычислить квадрат суммы трёх введённых вещественных чисел x, y, z. Вывести результат на экран.

2. Ввести значение угла в градусах, предварительно объявив соответствующую переменную. Вычислить и вывести на экран значения cos, sin и tg этого угла. Вывести эти значения на экран.

3. Объявить вещественную переменную x. Ввести значение переменной с клавиатуры такое, что $x > 0$. Вычислить десятичный и натуральный логарифмы введённого значения. Вывести результат на экран.

4. Объявить константу x, задать ей значение 25.5, ввести с клавиатуры значения переменной y, вычислить значение переменной f по формуле $f = 2x + x^y$. Вывести результат на экран.

5. Объявить вещественные переменные x, y, z и S. Написать программный код, реализующий следующие действия: присвоение переменным x, y и z значений, вводимых с клавиатуры. Вычислить квадрат разности трёх введённых вещественных чисел $S = (x - y - z)^2$. Вывести результат на экран.

6. Ввести значение углов a и b в градусах. Вычислить и вывести значения cos и sin этих углов.

7. Объявить вещественные переменные a, b, c и d. Записать программный код, реализующий присвоение переменной a значения 87.3, переменной b – значения 111, ввести значения переменной c с клавиатуры. Вычислить значение $f = d^c + b - a$. Вывести результат на экран.

8. Объявить вещественные переменные y и x. Ввести с клавиатуры значения переменных x и y такие, что $(y > 0)$ и $(x > 0)$. Вычислить десятичный и натуральный логарифмы введённых переменных. Вывести результат на экран.

9. Ввести значение угла в градусах такое, что $b \in [-1,1]$. Вычислить и вывести на экран значения arccos, arcsin и arctg этого угла. Результат решения задачи вывести на экран.

10. Объявить переменные *x*, *y*, *z* и *S*. Присвоить переменной *x* значение –25.6, переменной *y* – значение 128, значение переменной *z* ввести с клавиатуры. Вычислить сумму значений трёх величин по формуле $S = x + y + z$. Вывести результат на экран.

11. Идёт *k*-я секунда суток. Написать программу, рассчитывающую, сколько целых часов *h* и целых минут *m* прошло с начала суток.

12. Записать выражение, зависящее от координат точки *X*, *Y* и принимающее значение *true*, если точка принадлежит выделенной области, представленной на рис. 14, и *false*, если не принадлежит. Результаты вычислений вывести на экран. При выполнении задания использовать переменную логического типа, а не условный оператор.

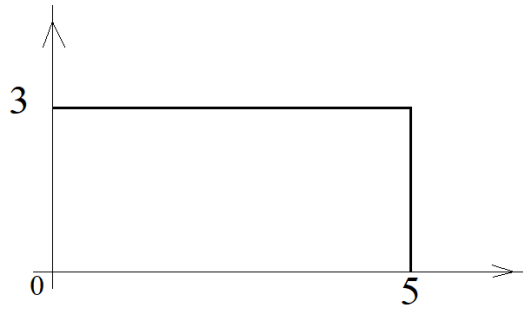


Рис. 14. Область для решения логической задачи 12

13. Определить тип заданных выражений, найти значения этих выражений для введённых с клавиатуры

a) $k=--n--m++$;
b) $m++<--n$;
c) $k=++n \cdot ++m$;
d) $m++<n$;
e) $k=n+(++m)$;
f) $m---->n$;
g) $k=++m+--n$;
h) $m++<n$;
i) $k=n++ \cdot m$;
j) $n++<m$;
k) $k=--m--n++$;
l) $m \cdot m<n++$;
m) $k=m++--n++$;
n) $m++>--n$;
o) $k=m+--n$;
p) $m++<--n$;

q) $k=(n++)+m--$;

r) $n \cdot m<n++$;

s) $k=n++ \cdot m$;

t) $++m>n$.

14. Вычислить и вывести на экран сумму цифр введённого с клавиатуры целого трёхзначного числа.

15. Дано неотрицательное целое число. Найти число десятков в его десятичной записи (то есть вторую справа цифру его десятичной записи).

16. Пусть дано n клеток и m зайцев, которых рассадили по этим клеткам. Рассчитать максимальное количество зайцев, которое гарантированно окажется в одной клетке.

17. Идёт k-я секунда суток. Определить, сколько целых часов h и целых минут m прошло с начала суток. На вход подается целое число k ($0 \le k \le 86\,399$).

18. Часовая стрелка повернулась с начала суток на d градусов. Определите, сколько сейчас целых часов h и целых минут m. На вход программе подаётся целое число d ($0 \le d < 360$).

19. В книге на одной странице помещается k строк. Таким образом, на первой странице печатаются строки с 1-й по k, на второй – с $k + 1$ по $2k$ и т. д. Написать программу, определяющую по номеру строки в тексте номер страницы, на которой будет напечатана эта строка, и порядковый номер этой строки на странице. На вход программе подаются число k – количество строк на странице и число n – номер строки в тексте ($1 \le k \le 200$, $1 \le n \le 20\,000$).

20. Даны значения двух моментов времени, принадлежащих одним и тем же суткам: часы, минуты секунды для каждого из моментов времени. Известно, что второй момент времени наступил не раньше первого. Определить, сколько секунд прошло между двумя моментами времени. В первой строке входных данных находятся три целых числа – часы, минуты и секунды первого момента времени. Во второй строке – три числа, характеризующие второй момент времени. Число часов лежит в диапазоне от 0 до 23, число минут и секунд – от 0 до 59.

21. Вычислить и вывести на экран сумму цифр введённого с клавиатуры целого трёхзначного числа.

22. Дано неотрицательное целое число. Найти число десятков в его десятичной записи (то есть вторую справа цифру его десятичной записи).

3. ОПЕРАТОРЫ ВЕТВЛЕНИЯ *C++*

Условный оператор if

Для реализации алгоритмической конструкции «ветвление» в языке *C++* используют оператор `if`, имеющий следующий вид:

`if` *(выражение) оператор1*`;` *[*`else` *оператор2;]*

В качестве условия может быть использовано любое арифметическое, логическое или сложное выражение, результат вычисления этого выражения будет интерпретирован логически. При этом ноль трактуется как ложь, любое отличное от нуля значение трактуется как истина.

Операторы 1 и 2 – это любые операторы языка *C++*, которые могут быть и составными.

На первом этапе выполнения оператора `if` вычисляется выражение, если результат вычисления – «истина», выполняется *оператор1*, «ложь» – выполняется *оператор2*. Если используется неполное ветвление, то ключевое слово `else` и *оператор2* отсутствуют; в этом случае если значение выражения – «ложь», управление передаётся оператору, следующему за оператором `if`. Ниже приведены примеры использования оператора ветвления в полной и неполной форме.

```
    if (a>b) max = a; else max = b; /*полная форма
оператора if*/
    if (a>b) max = a; /*неполная форма оператора if*/

    if (a<b &&(a>d || a==0)) b++; else {b* = a; a++;}

    if (a<b) {if (a<c) m=a; else m=c;}
    else {if (b<c) m=b; else m=c;}
```

Оператор множественного ветвления switch

`switch` *(<целочисленное выражение>)*
`{`
`case` *константное выражение1: оператор1;*

`case` *константное выражение2: оператор2;*
`case` *константное выражение3: оператор3;*
...
`case` *константное выражениеN – 1: операторN – 1;*

[`default`*: <операторN>]*
`}`

Выполнение оператора начинается с вычисления выражения, далее происходит последовательное сравнение полученного значения со значениями целочисленных констант до первого совпадения (все константы должны быть уникальными). После совпадения выполняется оператор соответствующей константы и далее последовательно без сравнения с константами выполняются все операторы до закрывающей фигурной скобки оператора `switch`.

В случае если не произошло совпадения ни с одной константой и используется полная форма оператора `switch`, то есть присутствует ключевое слово `default,` то выполняется оператор, указанный после него. Если не произошло ни одного совпадения и используется сокращённая форма, ключевое слово `default` отсутствует, управление передаётся оператору, следующему за оператором `switch`.

Если алгоритм решения задачи не требует выполнения всех операторов, указанных в строках после совпадения с константой, необходимо досрочно выйти из оператора `switch`, использовав оператор `break`.

Ниже приведены примеры использования оператора `switch` в программном коде.

```
int i = 2;
switch (i)
{
    case 1:printf("1\n");
    case 2:printf("2\n");
    case 3:printf("3\n");
    case 4:printf("4\n");
    default:printf("No!\n");
    ;
    }
```

Результат выполнения этого фрагмента кода будет таким:
2
3
4
No!

```
switch (i)
{
    case 1: {printf("1\n"); break; }
    case 2:{printf("2\n"); break; }
    case 3:{printf("3\n"); break; }
    case 4:{printf("4\n"); break; }
    default:printf("No!\n");
}
```

Результат:
2
Результат при *i* = 12:
No!

При *i* = 12 и отсутствии ключевого слова `default` на экран не будет выведено ничего и управление передастся оператору, следующему за оператором `switch`.

Тернарная операция

Тернарная операция позволяет реализовать ветвление и вернуть значение выполненного оператора в точку вызова:

(<операнд1>)`?`*<операнд2>*`:`*<операнд3>*`;`

Операнд1 представляет собой выражение, результат которого интерпретируется логически. Если результат этого выражения – «истина», выполняется *операнд2*, в противном случае выполняется *операнд3*:

```
int a = 7;
int b = 8;
int c = (a < b) ? a++: b++;
```

После выполнения этих строк кода переменные *a*, *b, c* примут следующие значения: *a* = 8, *b* = 8, *c* = 7.

Если начальное значение переменной *a* задать не меньше *b*, например *a* = 17, то выходные значения будут соответственно *a* = 17, *b* = 9, *c* = 8.

Второй и третий операнды могут представлять собой сложные выражения. Например,

```
int a, b, k; a = 10; b = 30;
k = a<b ? a++ : ++b ? c=a+b, printf("a=%i\n", a) : a;
```

Результат операции на представленных входных данных будет таким:

a = 11
b = 30
k = 10

Изменим начальное значение переменной *a*, например, так: *a* = 100; результат работы тернарной операции будет такой:

a = 100
b = 31
k = 6
c = 131

Разберём этот пример подробнее. На первом этапе будет выполнен первый операнд – это логическое выражение `a < b`, результат этого выражения – «ложь», 100 не меньше 30. Следовательно, будет выполнен не второй, а третий операнд.

Третий операнд начинается после символа «двоеточие». В данном случае в качестве третьего операнда выступает тернарная операция

```
++b ? c=a+b, printf("a=%i\n", a) : a;
```

Рассмотрим внимательнее эту операцию:

`++b` – первый операнд;

`c=a+b, printf("a=%i\n", a)` – второй операнд;
`a` – третий операнд.

Результат выполнения выражения, использованного в качестве первого операнда, будет равен 31 и проинтерпретирован как логическое значение, 31 – это отличное от нуля значение, следовательно, результат – «истина», а значит, будет выполнен первый операнд.

Переменной *c* будет присвоено значение суммы переменной *a* и переменной *b*, здесь важно вспомнить, что значение переменной *b* увеличено на единицу во время выполнения первого операнда. Таким образом, $c = 100 + 31$. Далее на экран будет выведено значение переменной *a*. В точку вызова будет возвращено целое число, равное количеству символов, выведенных на экран, в данном случае шесть символов.

Лабораторная работа № 3
ОПЕРАТОРЫ ВЕТВЛЕНИЯ *C++*

Цель работы. Изучение синтаксиса и правил работы с условными операторами, унарными префиксными и постфиксными операторами. Формирование навыков реализации разветвлённых алгоритмов.

Постановка задачи

1. Вычислить значение переменной *y* в зависимости от значения введённого с клавиатуры параметра *x*:

$$y = \begin{cases} x - 12, x > 0 \\ 5, x = 0 \\ x^2, x < 0. \end{cases}$$

2. Ввести с клавиатуры целое число. Интерпретируя это число как возраст мужчины, заполняющего анкету, вывести на экран одно из четырёх сообщений:

– если указан возраст от 18 и до 27 лет, сообщить, что человек подлежит призыву на срочную службу или может служить по контракту;

– если указан возраст от 28 до 59 лет, сообщить, что заполняющий анкету может служить по контракту;

– если указан возраст менее 18 или более 59 лет, сообщить о том, что заполняющий находится в непризывном возрасте;

– если указан неположительный возраст, сообщить об ошибке.

3. Составить расписание на неделю. Пользователь вводит порядковый номер дня недели, после чего на экране отображается расписание на этот день.

Реализовать два варианта задачи.

3.1. Вводимое значение должно быть целым числом в диапазоне от одного до семи (по количеству дней недели), в противном случае выдать сообщение об ошибке.

3.2. Вводимое значение может быть любым целым числом.

Выдать сообщение об ошибке, если число меньше одного.

Выдать сообщение о выходе за диапазон значений, если число больше количества дней в семестре.

Во всех остальных случаях выдать название дня недели и расписание на этот день. Учесть, что учебный год может начинаться не с понедельника. День недели, с которого начинается семестр, задать входным параметром.

4. Написать программу вычисления суммы цифр введённого с клавиатуры натурального числа.

5. Дан год. Вывести на экран название животного, символизирующего этот год по восточному календарю.

Вопросы для самоконтроля и подготовки к защите лабораторной работы

1. Запишите синтаксис и пример условного оператора `if`.

2. Какие две формы условного оператора существуют?

3. Поясните назначение форм оператора `if`, упомянутых в предыдущем вопросе.

4. Запишите синтаксис и пример использования конструкции `switch case`.

5. В каком случае предпочтительнее использовать оператор `switch`, а в каком – оператор `if`?

6. В каком случае будут выполнены инструкции, указанные после ключевого слова `default` в операторе множественного выбора?

7. Каким образом можно осуществить досрочный выход из оператора `switch`?

8. Запишите синтаксис и пример использования тернарного оператора.

9. Что будет возвращено в точку вызова тернарного оператора?

10. Назовите отличия в работе префиксного и постфиксного инкремента/декремента.

Задания для самостоятельной работы

1. Даны два целых числа, определить меньшее из них.

2. Определить, является ли данный год високосным. (Год является високосным, если его номер кратен 4, но не кратен 100, а также если он кратен 400.)

3. Ввести целое число x, вывести значение sign(x).

Функция sign(x) (знак числа) определена так:

sign$(x) = 1$, если $x > 0$,

sign$(x) = -1$, если $x < 0$,

sign$(x) = 0$, если $x = 0$.

4. Даны два целых числа, каждое записано в отдельной строке. Вывести число 1, если первое число больше второго, число 2, если второе больше первого, число 0, если они равны.

5. Даны три целых числа. Вывести наибольшее из данных чисел.

6. Определить, бьёт ли ладья, стоящая на клетке с указанными координатами (номер строки и номер столбца), фигуру, стоящую на другой указанной клетке. Даны четыре целых числа в интервале от 1 до 8: координаты ладьи (два числа) и координаты другой фигуры (два числа). Вывести слово *YES*, если ладья сможет побить фигуру за один ход, и *NO* – в противном случае. (Шахматная ладья может ходить на любое количество клеток по горизонтали или вертикали.)

7. Определить, бьёт ли слон, стоящий на клетке с указанными координатами (номер строки и номер столбца), фигуру, стоящую на другой указанной клетке. Даны четыре целых числа в интервале от 1 до 8: координаты слона (два числа) и координаты другой фигуры (два

числа). Вывести слово *YES*, если ладья сможет побить фигуру за один ход и *NO* – в противном случае. (Шахматная фигура слон может ходить на любое количество клеток по горизонтали.)

8. Определить, бьёт ли ферзь, стоящий на клетке с указанными координатами (номер строки и номер столбца), фигуру, стоящую на другой указанной клетке. Ввести четыре числа: координаты ферзя и координаты другой фигуры. Координаты – целые числа в интервале от 1 до 8. Вывести слово *YES*, если ферзь может побить фигуру за один ход, в противном случае вывести слово *NO*. (Шахматная фигура ферзь может ходить в любом направлении на любое количество клеток.)

9. Поле шахматной доски определяется парой чисел (*a, b*), каждое от 1 до 8, первое число задаёт номер столбца, второе – номер строки. Заданы две клетки. Определить, может ли шахматный король попасть с первой клетки на вторую за один ход. (Шахматный король может перемещаться за один ход на одну клетку в любом направлении.)

10. Определить, бьёт ли конь, стоящий на клетке с указанными координатами (номер строки и номер столбца), фигуру, стоящую на другой указанной клетке. Ввести четыре числа: координаты коня и координаты другой фигуры. Все координаты – целые числа в интервале от 1 до 8. Вывести слово *YES,* если конь может побить фигуру за один ход, в противном случае вывести слово *NO*.

11. Определить, можно ли от шоколадки размером $n \times m$ долек отломить k долек, если разрешается сделать один разлом по прямой между дольками (то есть разломить шоколадку на два прямоугольника).

12. В каждую крайнюю клетку квадратной доски $n \times n$ поставили по фишке. Могло ли оказаться, что выставлено ровно k фишек? (Например, если доска 2×2 клетки, то выставлено 4 фишки, а если 6×6 клеток, то выставлено 20 фишек.)

13. Решить в целых числах уравнение $ax + b = 0$.

Ввести 2 целых числа: a и b. Вывести значение x, если решение есть, *NO* – если решений нет, и *INF* – если решений бесконечно много.

14. Товар стоит a рублей и b копеек. За него заплатили c рублей d копеек. Сколько сдачи требуется получить? Ввести четыре числа: a, b, c и d. Вывести два числа: число рублей и копеек сдачи.

15. Ввести целое число, обозначающее код символа по таблице *ASCII*. Определить: введённое число – это код английской буквы или цифры или какой-либо другой символ?

16. Ввести два целых числа. Проверить, делится ли первое на второе. Вывести на экран соответствующее сообщение, а также остаток (если он есть) и частное (в любом случае).

17. Квадратное уравнение задаётся коэффициентами a, b, c, вводимыми с клавиатуры. Найти корни этого квадратного уравнения и вывести их на экран, если они есть. Если корней нет, то вывести соответствующее сообщение. Коэффициенты вводит пользователь.

18. Определить четверть координатной плоскости, которой принадлежит точка. Координаты точки ввести с клавиатуры.

19. Вводятся координаты $(x; y)$ точки и радиус круга r. Определить, принадлежит ли данная точка кругу, если его центр находится в начале координат.

20. По длинам трёх отрезков, введённых пользователем, определить возможность существования треугольника, составленного из этих отрезков. Если такой треугольник существует, то определить, является ли он разносторонним, равнобедренным или равносторонним. (Треугольник существует, если сумма длин двух его любых сторон больше третьей.)

4. ОПЕРАТОРЫ ОРГАНИЗАЦИИ ЦИКЛОВ *C++*

Алгоритмическая конструкция «повторение» реализуется при помощи операторов цикла.

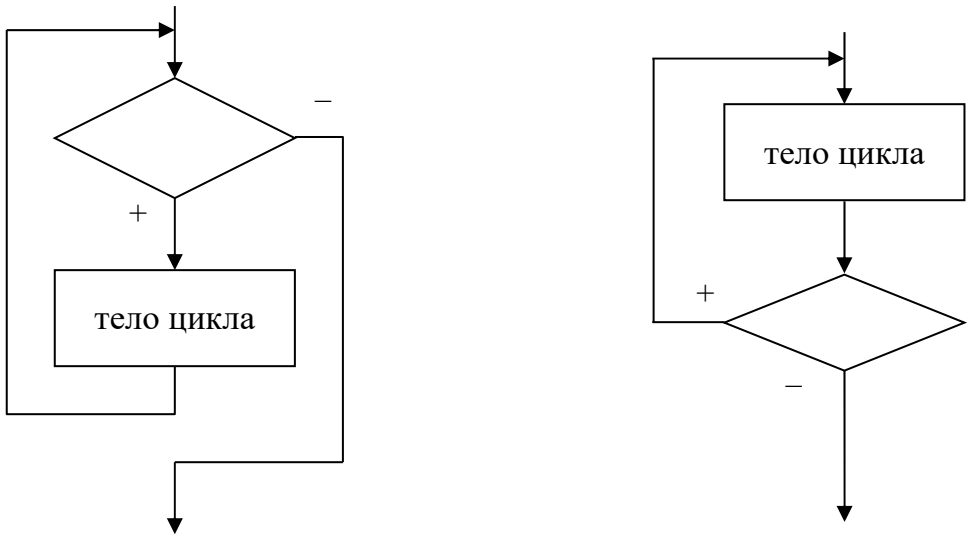
В любом цикле можно выделить *тело цикла* – многократно повторяющиеся действия и *проверку условия продолжения цикла*.

Однократное выполнение тела цикла называют *итерацией.*

По взаимному расположению тела цикла и условия продолжения цикла выделяют *цикл с предусловием* (условие расположено перед телом цикла) и *цикл с постусловием* (условие расположено после тела цикла).

Структурные схемы циклов представлены на рис. 15.

В случае, когда из условий задачи ясно, что тело цикла необходимо выполнить как минимум один раз, используют цикл с постусловием, если возможна ситуация, в которой тело цикла не выполнится ни одного раза, используют цикл с предусловием.



а) *б)*

Рис. 15. Структурная схема организации цикла:
а – цикл с предусловием; б – цикл с постусловием

Переменные, используемые в условии продолжения цикла, называют *параметрами цикла*. Целочисленные параметры цикла, изменяющиеся с определённым шагом, называют счётчиками цикла.

В теле цикла обязательно должна содержаться модификация параметра цикла. Задание начальных значений, в том числе и параметра цикла, должно быть обязательно осуществлено до тела цикла.

Цикл с предусловием while

Синтаксис оператора `while` определён следующим образом:

`while` *(выражение) оператор*`;`

Результат вычисления выражения интерпретируется логически. Если результат – «истина», выполняется оператор *тело цикла*. Если в теле цикла необходимо выполнить более одного оператора, используют составной оператор {}.

Ниже представлен фрагмент программы вычисления суммы $n = 5$ введённых с клавиатуры чисел с использованием оператора `while`.

```
int s = 0;
int const n = 5;
int a = 0;
int i = 1;
while (i<=n)
{   printf("a=");
    scanf_s("%i", &a);
    s = s + a; i++;
}
printf("s=%i\n", s);
```

Результат работы программы представлен на рис. 16.

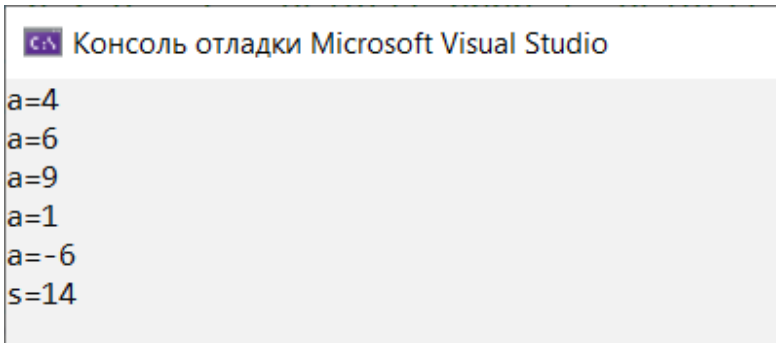


Рис. 16. Результат работы фрагмента программы вычисления суммы элементов с использованием цикла while

Цикл с постусловием do while

Синтаксис оператора `while` определён следующим образом:

`do` <*оператор*> `while` *(выражение)*`;`

После выполнения оператора вычисляется выражение. Если результат выражения – «истина», то снова выполняется тело цикла. Выход из цикла будет осуществлён, когда значение выражения примет результат «ложь».

Далее представлен фрагмент программы вычисления суммы введённых с клавиатуры чисел с использованием оператора `do while.`

```
do
    {
    printf("a=");
    scanf_s("%i", &a);
    s = s + a; i++;
    } while (i < n);
printf("s=%i\n", s);
```

На рис. 17 представлен результат работы этого фрагмента кода.

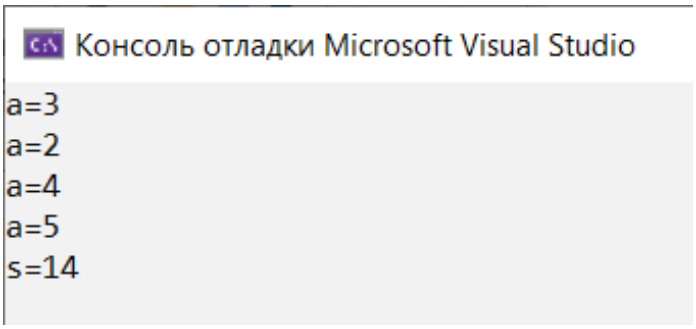


Рис. 17. Результат работы фрагмента программы вычисления суммы элементов с использованием цикла do while

Цикл с параметром for

Этот оператор позволяет реализовать цикл с предусловием и условием продолжения.

Правило записи оператора следующее:

`for (`*выражение1*`;` *выражение2*`;` *выражение3)* `оператор;`

Выражение1 – инициализатор, обычно содержит объявления и инициализацию переменных, используемых в цикле. Может содержать несколько операторов, разделённых запятой. Выполняется один раз, перед выполнением цикла.

Выражение2 интерпретируется логически, если результат – «истина», выполняется тело цикла, иначе управление передаётся оператору, следующему за оператором `for`.

Выражение3 – итератор, вычисляется после выполнения тела цикла, далее вновь выполняется *выражение2*.

Рассмотрим фрагмент программы вычисления суммы введённых с клавиатуры чисел с использованием оператора `for`. Результат работы программного кода представлен на рис. 18.

```
for (int i = 0; i < n; i++)
{
    printf("a=");
    scanf_s("%i", &a);
    s = s + a;
}
printf("s=%i\n", s);
```

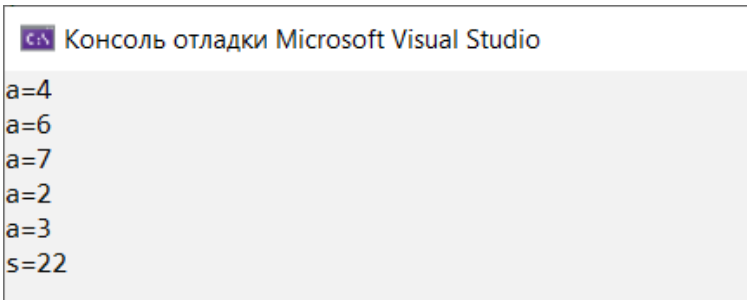


Рис. 18. Результат работы фрагмента программы вычисления суммы элементов с использованием цикла for

Любое из выражений или даже все три выражения в организации цикла `for` могут отсутствовать. Наличие точки с запятой при этом обязательно. Инициализация параметра цикла при таком подходе должна быть выполнена до тела цикла, а проверка условия и изменения должна быть обязательно выполнена в теле цикла.

Разберём подробнее на примерах способы организации цикла `for`, опуская одно или несколько выражений.

```
int s = 0; int a = 0;
int const n = 5;
int i = 0; //инициализация параметра цикла i
    for (; i < n; i++)
    {
    scanf_s("%i", &a);
    s = s + a;
    }
```

В примере, приведённом выше, отсутствует *выражение1*. Инициализация параметра цикла произведена на строчку выше до оператора `for`.

В следующем примере отсутствует *выражение3*.

```
int s = 0;
int const n = 5;
int a = 0;
for (int i = 0; i < n; )
{
    printf("a=");
```

```
    scanf_s("%i", &a);
    s = s + a;
    i++;
}
printf("s=%i\n", s);
```

Изменение параметра цикла выполнено в теле цикла, так как *выражение3* отсутствует.

В следующем примере *выражение3* также отсутствует, но изменение параметра цикла произведено во втором выражении. Результат работы представленного фрагмента кода будет таким же, как и в предыдущем примере.

```
int s = 0;
int const n = 5;
int a = 0;
for (int i = 0; i++ < n; )
{
    printf("a=");
    scanf_s("%i", &a);
    s = s + a;
}
printf("s=%i\n", s);
```

В данном примере изменение параметра во втором выражении должно быть произведено именно в постфиксной форме. Если использовать префиксную форму инкремента, количество итераций сократится, так как `for` – цикл с предусловием, и если использовать префиксную форму инкремента `++i`, то при проверке условия `++i < n` будет использовано уже изменённое значение.

Рассмотрим пример организации цикла `for` с пропущенным *выражением2*. Выход из цикла осуществлён при помощи оператора досрочного выхода `break`:

```
int s = 0;
int const n = 5;
int a = 0;
for (int i = 0; ;i++)
```

```
{
    if (i>=n)break;
    printf("a=");
    scanf_s("%i", &a);
    s = s + a;
}
printf("s=%i\n", s);
```

Осталось рассмотреть пример использования цикла `for`, исключив все три выражения. Пример такого подхода представлен ниже.

```
int s = 0;
int const n = 5;
int a = 0;
int i = 1;
for(;;)
{if (i>n) break;
    printf("a=");
    scanf_s("%i", &a);
    s = s + a; i++;
}
    printf("s=%i\n", s);
```

В представленном фрагменте кода все три выражения в цикле `for` отсутствуют. Инициализация параметра цикла осуществлена до цикла. Проверка условия продолжения цикла и изменения параметра осуществлены в теле цикла.

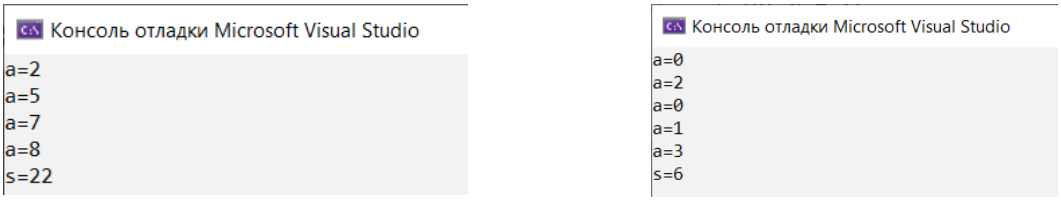
В случае необходимости досрочного прерывания выполнения цикла используют следующие операторы: `break, goto, return.`

Оператор `break` используют для досрочного выхода из операторов цикла и оператора `switch`. После того как оператор `break` будет встречен в коде, все последующие операторы тела цикла игнорируются и управление передаётся оператору, следующему за оператором, в теле которого встретился оператор выхода `break`.

Далее рассмотрен пример вычисления суммы чисел, вводимых с клавиатуры. Вычисления ведутся до тех пор, пока не будет введено нужное количество слагаемых либо сумма не превысит определённого значения *N*, в этом случае будет выполнено досрочное прерывание цикла.

```
int s = 0;
int const n = 5;
int const N = 15;
int a = 0;
int i = 1;
while (i<=n)
{
    printf("a=");
    scanf_s("%i",&a);
    s = s + a; i++;
    if (s>N) break;
}
printf("s=%i\n", s);
```

На рис. 19 представлены результаты работы приведенного выше фрагмента кода с различными входными значениями.



а) *б)*

Рис. 19. Результат работы программы вычисления суммы с использованием оператора досрочного прерывания: а) условие досрочного прерывания выполнилось; б) досрочного прерывания не произошло, выполнились все итерации цикла

Использование оператора `return` приводит к досрочному выходу из функции, в теле которой он был использован.

Оператор безусловного перехода `goto` имеет следующий вид: `goto` <метка>; Использование этого оператора позволяет выйти из тела цикла и перейти на оператор, указанный после метки. Под меткой понимается идентификатор, область действия которого – функция, в теле которой он задан. Однако использование этого оператора нарушает принципы структурного и модульного программирования, и использования оператора `goto` стоит избегать при написании программ.

Все задачи в изучаемом курсе «Основы программирования» могут и должны быть решены без использования оператора `goto`.

К использованию оператора `goto` прибегают при необходимости выйти из нескольких вложенных циклов. Пример использования оператора `goto` приведён ниже.

```
int s = 0;
int const n = 5;
int const N = 15;
int a = 0;
int i = 1;
while (i <= n)
{
    printf("a=");
    scanf_s("%i", &a);
    s = s + a; i++;
    if (s > N) goto LLL;
}
LLL: printf("s=%i\n", s);
```

Оператор `continue` позволяет перейти на следующую итерацию цикла, минуя указанные за ним операторы тела цикла:

```
for (int count = 0; count <= 30; count++)
     {if ((count % 5) != 0) continue;
        std::cout << count << std::endl; /*Это
действие будет пропущено, если число не делится
нацело на 5*/
    }
```

В результате работы представленного выше фрагмента кода будут выведены все числа от нуля до тридцати, делящиеся на пять без остатка.

Лабораторная работа № 4

ОПЕРАТОРЫ ОРГАНИЗАЦИИ ЦИКЛОВ *C++*

Цель работы. Изучение синтаксиса и правил работы с операторами организации циклов языка *C++*.

Постановка задачи

1. Начав готовиться к соревнованиям, спортсмен в первый день пробежал 10 км. Каждый день он увеличивал дневную норму на 10 процентов от нормы предыдущего дня. Написать программу, определяющую, какой суммарный путь пробежит спортсмен за 7 дней.

2. Одноклеточная амеба каждые три часа делится на две клетки. Написать программу, определяющую, сколько амёб будет через 3, 6, 9, 12, 15, 18, 21, 24 часа.

3. Около стены наклонно стоит палка длиной x метров. Один её конец находится на расстоянии y метров от стены. Написать программу, определяющую значение угла α между палкой и полом для значений $x = k$ метров и y, изменяющегося от 2 до 3 метров с шагом h метров.

4. У гусей и кроликов вместе 64 лапы. Сколько может быть кроликов и гусей? Вывести на экран все возможные сочетания.

При решении задач лабораторной работы должно быть продемонстрировано умение использовать операторы `while, do while, for`.

Вопросы для самоконтроля и подготовки к защите лабораторной работы

1. Какую алгоритмическую конструкцию называют циклом?
2. Что такое тело цикла?
3. Что такое итерация цикла?
4. При помощи каких операторов в языке программирования *C++* организовывают цикл?
5. Какие операторы можно использовать при написании программы на языке программирования *C++* для досрочного выхода из цикла?
6. Напишите синтаксис и приведите пример использования оператора `for`.
7. Напишите синтаксис и приведите пример использования оператора `while`.
8. Напишите синтаксис и приведите пример использования оператора `do while`.

9. Какой цикл называют циклом с предусловием, какой – циклом с постусловием?

10. При использовании какой алгоритмической конструкции тело цикла выполнится как минимум один раз?

11. При помощи какого оператора языка программирования *C++* можно реализовать эту конструкцию?

Задания для самостоятельной работы

1. Составить программу, выводящую на экран квадраты чисел от 10 до 20 включительно.

2. Даны натуральные числа от 35 до 87. Вывести на консоль те из них, которые при делении на 7 дают остаток 1, 2 или 5.

3. Найти сумму $1 + 2 + 3 + \ldots + n$, где число n вводится пользователем с клавиатуры.

4. Найти произведение цифр трёхзначного числа.

5. Найти количество чётных цифр данного натурального числа.

6. Найти наибольшую цифру данного натурального числа.

7. Найти все четырёхзначные числа, сумма цифр каждого из которых равна 15.

8. Составить таблицу значений функции $y = 5 - x^2/2$ на отрезке $[-5; 5]$ с шагом $h = 0.5$ изменения x.

9. Вводится натуральное число. Найти сумму чётных цифр, входящих в его состав.

10. Вводится целое число A. Преобразовать его в число B, цифры которого будут следовать в обратном порядке по сравнению с введенным числом.

11. Вывести на экран кубы чисел от A до B. Числа A и B вводит пользователь.

12. Вычислить факториал числа. Факториал числа – это произведение всех натуральных чисел от 1 до этого числа включительно. Например, факториал числа 7 выглядит так: $1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7$.

13. Найти сумму n-го количества элементов ряда 1, –0.5, 0.25, –0.125 и т. д.

14. Определить, из каких цифр состоит число. Вывести эти цифры на экран.

15. Вывести все квадраты натуральных чисел, не превосходящие данного числа *N*. Значение *N* вводит пользователь.

16. Вывести на экран ряд чисел Фибоначчи, состоящий из *n* элементов. Числа Фибоначчи – элементы числовой последовательности 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, в которой каждое последующее число равно сумме двух предыдущих.

17. Возвести число в степень. Число и показатель степени ввести с клавиатуры.

18. Вывести на экран таблицу умножения.

19. Из натурального числа удалить заданную цифру. Число и цифру вводить с клавиатуры. Например, задано число 5683. Требуется удалить из него цифру 8. Результат – число 563.

20. Найти сумму первой и последней цифр любого целого положительного числа.

21. Написать программу, находящую все комбинации из трёх чисел до определённого предела, которые в сумме дают заданное число. Предел трёх чисел и заданное число-сумму вводить с клавиатуры.

22. Дано целое число, состоящее из разных цифр. Определить, какая из цифр заданного числа больше, то есть найти наибольшую цифру числа.

23. Вычислить сумму ряда чисел $1/1^2 + 1/2^2 + 1/3^2 + \ldots + 1/n^2$, где *n* пользователь вводит с клавиатуры.

5.1. УКАЗАТЕЛИ И ССЫЛКИ

Под ***указателем*** в *C++* понимают производный (пользовательский) тип данных, предназначенный для хранения адресов какой-либо ячейки памяти. Ячейка может быть именованной или неименованной. Указатель обязательно должен быть связан с каким-то другим базовым или производным типом данных.

Для объявления указателя необходимо указать тип данных указуемого и символ * перед именем указателя:

```
<тип данных>*<имя указателя>;
int *p;
```

где `p` – указатель на целое число.

Для инициализации указателя необходимо в правой части оператора присваивания указать адрес конкретной области данных.

В случае, если речь идёт о переменной или константе, перед их именем необходимо указать символ амперсанд &:

```
int a = 7;
int* p;
p=&a;
```

Инициализация указателя может быть совмещена с его объявлением. В этом случае последний фрагмент кода можно переписать так:

```
int a = 7;
int* p=&a;
```

Схематично результат написанного выше фрагмента кода показан на рис. 20.

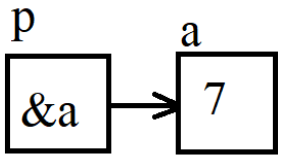


Рис. 20. Схематичное представление связи указателя с указуемым

Стрелочка на рис. 20 означает, что указатель `p` содержит адрес переменной `a`, то есть указывает на переменную `a`.

При работе с указателем возможна работа как с указателем, так и с указуемым.

Для работы с указуемым, то есть для перехода по адресу, необходима операция

разадресации (разыменовывания). Для этого нужно использовать символ * перед уже объявленным и проинициализированным указателем.

К символу * при работе с указателями нужно относиться очень внимательно!

Употребление символа * перед идентификатором при объявлении означает, что объявлен указатель.

Употребление символа * перед идентификатором объявленного ранее указателя означает разыменовывание указателя, то есть обращение к указуемому.

Например, строка `*p=5;` означает запись в переменную `a` значения целочисленной неименованной константы 5.

Если значение указателя должно быть постоянным во время работы программы, то указатель объявляют константным.

Для этого необходимо указать `const` перед именем указателя, то есть после символа * при объявлении:

```
int a = 7;
int * const p = &a;
```

Если `const` указать перед звёздочкой, то получим указатель на константу, при этом значение указателя можно изменить, то есть можно перенастроить указатель на другую целочисленную константу:

```
const int b = 10;
int const *p = &b;
const int c = 30;
p = &c;
printf("*p=%d", *p); //будет выведено *p = 30
```

Возможно объявление указателя на указатель, правила объявления те же, что и для указателя:

```
int a = 7;
int* p = &a;
int** pp = &p;
**pp = 9;
```

Тип переменной `pp` – указатель на указатель на целое.

После выполнения последней строки кода значение переменной `a` будет равно 9.

Схематично строка объявления и инициализации переменной `pp` показана на рис. 21.

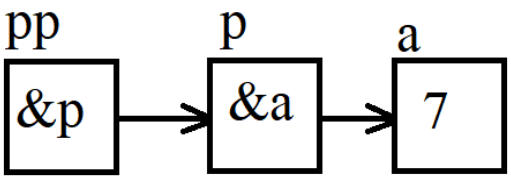


Рис. 21. Схематичное представление инициализации цепочки указателей

Кроме инициализации и разыменовывания с указателями можно выполнять следующие операции: сравнение, вычитание, сложение с константой, инкремент и декремент.

Эти операции можно рассматривать нагляднее, если указатели настроены на последовательность однотипных элементов – *массив*. Подробнее массивы будут рассмотрены в следующем разделе. В примере продемонстрировано объявление массива и обращение к его элементам.

```
int mas[5] = {1,2,3,4,5}; //объявление массива
int *p1 = &mas[0]; //указатель настроен на нулевой элемент
int* p2 = &mas[1]; //указатель настроен на первый элемент
```

Если учесть, что имя массива можно рассматривать как константный указатель на его первый (нулевой) элемент, то следующие две строки можно переписать так:

```
int *p1 = mas;
int* p2 = mas+1;
```

В приведённом выше фрагменте кода объявлен и проинициализирован массив целых чисел из пяти элементов. Так как индексация элементов массива начинается с нуля, номер последнего элемента – четыре, то есть на единицу меньше общего количества элементов.

Указатели `p1` и `p2` настроены на нулевой и первый элементы соответственно. Схематично это показано на рис. 22.

Сравнение указателей приводит к сравнению адресов указуемых. В приведённом ниже примере результат сравнения вернёт *true*, так как адрес, на который настроен указатель `p2`, старше.

```
if (p1 < p2)printf("p1\n"); else printf("p2\n");
```

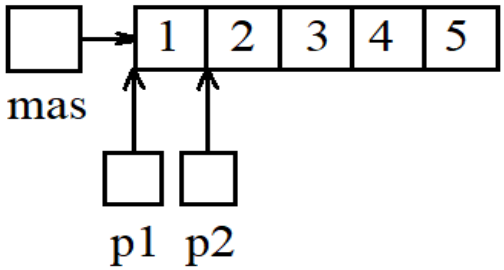


Рис. 22. Схематичное представление настройки указателей на элементы массива

Результатом работы этого оператора будет вывод на экран строки *p1*.

Приведём примеры возможных операций с указателями.

`int* p3 = p2;` /*объявление указателя *p3* и запись в него значения указателя *p2**/

В операторе присваивания возможно использования разнотипных указателей, при этом должно быть использовано приведение типа указателя.

`p3++;` /*увеличение значения указателя *p3* на единицу, один размер типа указуемого*/

`(*p3)++;` /*разыменовывание указателя *p3* и изменение значения указуемого на единицу*/

На рис. 23 представлены разобранные выше примеры допустимых операций на указателях: инициализация, инкремент, работа с указуемым.

Стрелочка в схематичном изображении показывает настройку указателя, то есть в ячейке-указателе лежит адрес указуемого – ячейки, к которой проведена стрелка.

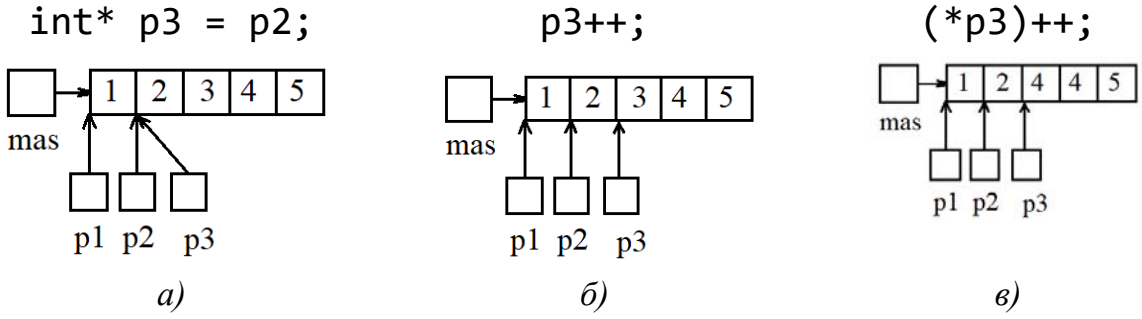


Рис. 23. Схематичное представление операций на указателях: а – настройка указателя p3 на ячейку mas[2]; б – сдвиг p3 на одну ячейку размера указуемого; в – изменение значения указуемого

Операции постфиксного и префиксного инкремента и декремента изменяют значение указателя на единицу. За единицу принимается размер типа указуемого. Таким образом, осуществляется сдвиг указателя в большую или меньшую сторону на количество байт, равное количеству байт, отводимому под переменные указуемого типа. В данном случае это размер типа `int`:

```
p3++; //p3=p3+sizeof(*p3);
```

Разность указателей – это целое число, равное количеству ячеек указуемого типа между значениями указателей:

```
int r = p3-p1;
```

В данном примере переменная `r` примет значение 2, наглядно это было показано на рис. 23.

Под ***ссылкой*** понимают альтернативное имя объекта. Таким образом, обратиться к переменной можно по имени или через ссылку на эту переменную. Ссылку можно рассматривать как константный разыменованный указатель, то есть указатель, который нельзя перенастроить, но через него можно работать с указуемым.

Чаще всего ссылки используют при передаче параметров в функции.

Ссылка должна быть проинициализирована при объявлении, ссылку нельзя перенастроить. Все операции, производимые с ссылкой, кроме инициализации, производятся с переменной, с которой связана ссылка.

Объявление и инициализация ссылки происходят по следующему правилу:

<тип данных> `&` *<имя ссылки>* `=` *<переменная>*`;`

```
int a = 10;
int b = 5;
int &ref = a; //объявление ссылки ref и связь её с переменной a.
ref = b; /*в переменную a будет записано значение переменной b, ссылка ref по прежнему связана с переменной a*/
ref++; //изменение значения переменной, связанной с ссылкой ref, значение переменной a равно шести.
```

Вопросы для самоконтроля и подготовки к защите лабораторной работы

1. Что такое указатель?
2. Для чего в программировании используют указатели?
3. Что такое ссылка, для чего используют ссылки?
4. Каким образом можно изменить значение переменной, обращаясь к ней через указатель?
5. Каким образом можно изменить значение переменной, обращаясь к ней через ссылку?
6. Возможно ли изменение значения ячейки через константный указатель?
7. Возможно ли перенастроить константный указатель?
8. Возможно ли перенастроить указатель на константу на другую константу?
9. Продемонстрируйте обращение к элементам массива через указатель на первый элемент.
10. Чему равна разность указателей, настроенных на два соседних элемента одного массива?

Задания для самостоятельной работы

1. Объявить указатель на вещественное число, настроить его на переменную соответствующего типа и увеличить значение этой переменной вдвое, обращаясь к ней через указатель.

2. Объявить константный указатель на вещественную константу. Настроить его на соответствующую константу, предварительно объявив и задав её значение равным значению ускорения свободного падения. Объявить и проинициализировать вещественные переменные m и h, объявить два вещественных указателя и настроить их на переменные m и h. Вычислить значение $E_p = mgh$, обращаясь к переменным m, h и константе g через соответствующие указатели.

3. Объявить вещественную переменную *f*, указатель на вещественную переменную *pf*, константный указатель на указатель на вещественную переменную *cp*, проинициализировать объявленные пе-

ременные и константу *cp*, связав их в единую цепочку. Изменить значение переменной *f,* обращаясь к ней через указатель *cp*.

4. Объявить и проинициализировать следующую цепочку данных: указатель на указатель на переменную типа `char`. Вывести на экран значение символа, предварительно изменив его значение, обе операции нужно выполнить, обращаясь к переменной типа `char` через указатель на указатель.

5. Объявить и проинициализировать следующую цепочку данных: константный указатель на константный указатель на вещественную константу. Вывести значение вещественной константы на экран, обращаясь к ней через указатель на указатель.

6. Объявить и проинициализировать две целочисленные переменные. Объявить указатель и ссылку на целое, настроить указатель на одну из объявленных переменных, ссылку связать со второй переменной. Объявить третью целочисленную переменную, сохранить в неё сумму первых двух объявленных переменных. К слагаемым обращаться через указатель и ссылку соответственно.

7. Объявить три целочисленные переменные. Объявить ссылки на эти переменные. Проинициализировать две переменные значениями, ведёнными с клавиатуры. В третью переменную записать удвоенную разность квадратов первых двух переменных. Все операции выполнять, обращаясь к переменным через ссылки.

8. Объявить и проинициализировать две целочисленные переменные. Объявить указатель и ссылку на целое, настроить на одну из объявленных переменных, ссылку связать со второй переменной. Увеличить значение переменных вдвое, обращаясь к ним через указатель и ссылку соответственно.

9. Объявить и проинициализировать две целочисленные переменные. Объявить два указателя на целочисленные переменные и настроить их на объявленные переменные. Поменять значения переменных местами, обращаясь к ним через указатели.

10. Объявить и проинициализировать массив из десяти целых чисел. Объявить указатель на целочисленную переменную и настроить его на минимальный элемент массива.

5.2. СТАТИЧЕСКИЕ МАССИВЫ

Под ***массивом*** в программировании понимают последовательность однотипных элементов, расположенных в памяти подряд и объединённых под одним именем.

Правило объявления массива в языке программирования *C++* следующее:

<тип элементов массива> <имя массива>`[`*<количество элементов>*`];`

Индексация элементов массива начинается с нуля, поэтому номер последнего элемента массива на единицу меньше общего количества элементов массива.

Например, массив можно объявить так:

```
int mas [5];
```

В приведённом примере объявлен массив из пяти целочисленных элементов.

Количество элементов массива при объявлении должно быть *обязательно задано константой*. Так как выделение памяти под массив происходит на шаге компиляции, в этот момент должна быть однозначная точная информация о необходимом количестве выделяемой памяти.

Константа может быть неименованной, как в предыдущем примере, или именованной, как в следующем примере:

```
int const N=5;
int mas [N];
```

Предпочтительнее использование именованной константы.

Массив можно проинициализировать при объявлении, в цикле или поэлементно.

При инициализации массива при объявлении после квадратных скобок указывают знак «=», далее в фигурных скобках перечисляют элементы массива через запятую. Например,

```
int mas [N]={1,2,3,4,5};
```

Если количество элементов, перечисленных в фигурных скобках, меньше количества элементов, указанного в квадратных скобках, то перечисленными значениями будут проинициализированы первые элементы массива, оставшиеся будут проинициализированы значениями по умолчанию для соответствующего типа, как показано на рис. 24.

```
int mas [N]={1,2};
```

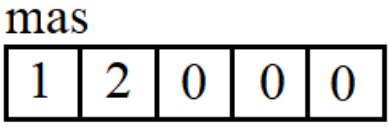


Рис. 24. Схематичное представление одномерного массива в памяти и с точки зрения языка C++, пример инициализации

Если же, наоборот, количество значений в фигурных скобках превосходит заданное количество элементов массива, возникает ошибка во время компиляции:

```
int mas [3]={1,2,3,4,5,6,7,8,9}; //ошибка
```

Для обращения к элементам массива существует два подхода: через индексное выражение и через использование имени массива как указателя на его нулевой элемент.

В первом способе необходимо указать после имени массива в квадратных скобках номер элемента, не забывая, что индексация начинается с нуля:

```
mas[3]=45;
```

В этом примере третьему элементу массива *mas* присвоено значение 45, что проиллюстрировано на рис. 25.

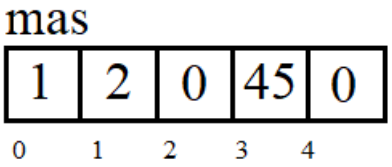
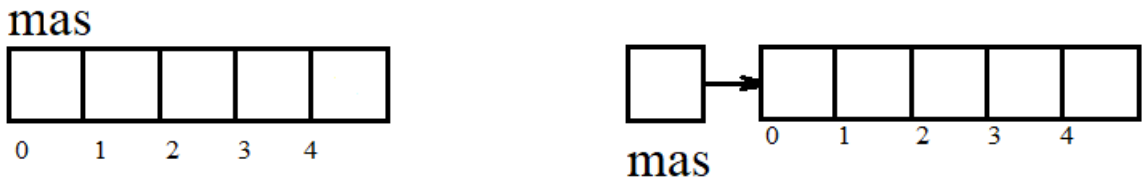


Рис. 25. Схематичное представление одномерного массива в памяти и с точки зрения языка C++, запись значения в ячейку с номером 3

Для реализации второго способа (использования имени массива в качестве константного указателя на нулевой элемент) необходимо вспомнить, что элементы массива в памяти расположены подряд, и обратиться к рис. 26.

В языке *C++* имя массива рассматривается как константный виртуальный указатель на нулевой элемент массива. То, что указатель константный, означает, что его нельзя перенастроить. То, что указатель виртуальный, означает, что его не существует в памяти, но с точки зрения языка работа с именем массива ведётся по правилам работы с указателями.

На рис. 26 схематично представлено расположение массива в памяти и с точки зрения языка программирования *C++*.



а) *б)*

Рис. 26. Схематичное представление одномерного массива в памяти и с точки зрения языка C++: а – расположение одномерного массива в памяти компьютера; б – одномерный массив с точки зрения языка C++

Если рассматривать имя массива как указатель на нулевой элемент, то для обращения к этому элементу достаточно разыменовать этот указатель:

```
*mas = 7;
```

Для обращения к следующему элементу необходимо сначала вычислить адрес этого элемента. И только после этого разыменовать.

При этом важно помнить, что имя массива – это константный указатель. Следовательно, его нельзя перенастроить, адрес нужного элемента можно только вычислить, используя этот константный указатель в выражении. Это показано в приведенном ниже примере, результат выполнения которого продемонстрирован на рис. 27.

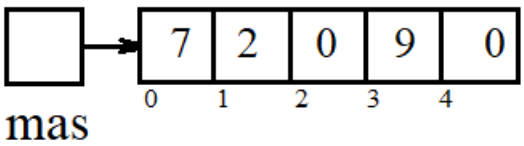


Рис. 27. Схематичное представление одномерного массива с точки зрения языка C++, запись значения в ячейку с номером 3

```
*(mas+3)=9;
```

Предыдущие два примера демонстрировали инициализацию массива поэлементно.

Объявим и проинициализируем статический одномерный массив в цикле, обращаясь к элементам двумя описанными ранее способами:

```
int const N = 5;
int mas[5];
for (int i = 0; i < N; i++)
{
*(mas+i)=i*i;
printf("%i\n",mas[i]);
}
```

Полученный результат схематично представлен на рис. 28.

При переборе элементов массива в цикле необходимо внимательно отслеживать выход за его границы. При выходе за границу массива синтаксической ошибки не случится, но будет ошибка в расчётах.

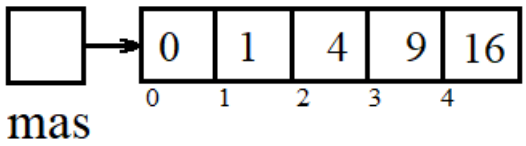


Рис. 28. Схематичное представление одномерного массива с точки зрения языка C++ после инициализации

Лабораторная работа № 5
УКАЗАТЕЛИ И ССЫЛКИ

Цель работы. Освоить принципы работы с указателями и ссылками в языке *C++*. Изучить операции, выполнение которых допустимо на указателях и ссылках. Получить навыки работы с массивами, в том числе навыки организации циклического перебора элементов, приёмов накопления.

Постановка задачи

1. Объявить и проинициализировать переменные следующих типов: `int, float, char`. Объявить переменную типа `void`.

Объявить указатели и ссылки на все перечисленные выше типы, настроить указатели и ссылки на объявленные переменные базовых типов. Вывести значения переменных на экран, обращаясь к ним через указатели.

2. Выполнить перенастройку указателей на переменные других типов, используя явное и неявное приведение типов указателей.

3. Привести примеры допустимых операций над указателями и ссылками. Пояснить смысл данных операций.

4. Объявить и проинициализировать следующие переменные и константы:

– константный указатель на `int`;

– указатель на целочисленную константу;

– указатель на указатель на `int`;

– константный указатель на указатель на `int`;

– указатель на константный указатель на целое;

– указатель на указатель на целочисленную константу;

– константный указатель на указатель на целочисленную константу;

– константный указатель на константный указатель на `int`;

– константный указатель на константный указатель на целочисленную константу.

Пояснить схематично организованные цепочки данных.

5. Объявить и проинициализировать одномерный целочисленный массив. Вычислить и вывести на экран сумму элементов этого массива, больших нуля.

6. Объявить и проинициализировать вводом с клавиатуры одномерный массив вещественных чисел. Вычислить и вывести на экран разность квадратов максимального и минимального значения элементов этого массива.

7. Объявить одномерный целочисленный массив. Вычислить среднее арифметическое элементов, обращаясь к ним через имя массива, используя его как указатель. Объявить два указателя на целое, настроить один из них на максимальный элемент массива, другой – на минимальный. Вывести значения максимального и минимального элемента на экран, обращаясь к ним через эти указатели.

Вопросы для самоконтроля и подготовки к защите лабораторной работы

1. Что такое массив?
2. С какого значения начинается индексация элементов массива в языке программирования *C++*?
3. Каким образом необходимо задать количество элементов статического массива?
4. Запишите синтаксис и приведите пример создания одномерного статического массива на языке *C++*.
5. Перечислите способы инициализации элементов массива в языке *C++*.
6. Перечислите способы обращения к элементам массива в языке *C++*.
7. Объявите одномерный целочисленный массив из десяти элементов, проинициализировав его при объявлении.
8. Объявите одномерный целочисленный массив из десяти элементов, проинициализировав его элементы в цикле значениями их удвоенных индексов.
9. Объявите одномерный целочисленный массив из десяти элементов, проинициализируйте его в цикле вводом с клавиатуры, замените на ноль максимальное значение в массиве.
10. Объявите одномерный целочисленный массив из десяти элементов, проинициализируйте его в цикле вводом с клавиатуры. Объявите указатель на целое и настройте его на максимальный элемент массива.

Задания для самостоятельной работы

1. Объявить одномерный целочисленный массив из десяти элементов, проинициализировать его в цикле вводом с клавиатуры. Объявить указатель на целое и настроить его на минимальный элемент массива.
2. Объявить одномерный целочисленный массив из десяти элементов, проинициализировать его в цикле вводом с клавиатуры. Объявить указатель на целое и настроить его на максимальный элемент массива.

3. Объявить одномерный целочисленный массив из десяти элементов, проинициализировать его в цикле вводом с клавиатуры. Вычислить квадрат разности максимального и минимального значений элементов массива.

4. Объявить одномерный целочисленный массив из *N* элементов, проинициализировать его любым способом. Заменить на ноль все элементы массива, значение которых повторяется более одного раза.

5. Объявить одномерный целочисленный массив из *N* элементов, проинициализировать его любым способом. Найти минимальный по модулю элемент этого массива.

6. Объявить одномерный целочисленный массив из *N* элементов, проинициализировать его любым способом. Найти и вывести на экран сумму индексов максимального и минимального элементов.

7. Объявить одномерный целочисленный массив из *N* элементов, проинициализировать его любым способом. Найти и вывести на экран наименьший чётный элемент массива. Если такого нет, вывести первый элемент.

8. Объявить одномерный целочисленный массив из *N* элементов, проинициализировать его любым способом. Найти сумму элементов массива между двумя первыми нулями. Если двух нулей в массиве нет, то вывести ноль.

9. Объявить одномерный целочисленный массив из *N* элементов, проинициализировать его любым способом. Вычислить произведение элементов массива с нечётными номерами.

10. Объявить одномерный целочисленный массив из *N* элементов, проинициализировать его любым способом. Вычислить сумму отрицательных элементов массива.

11. В одномерном целочисленном массиве найти количество элементов, больших некоторого параметра *k*; если таких элементов нет, выдать соответствующее сообщение.

12. В одномерном целочисленном массиве найти количество элементов, не превосходящих значение параметра *A*, и при этом меньших параметра *B*. Если таких элементов нет, выдать соответствующее сообщение.

13. В одномерном целочисленном массиве найти количество элементов, лежащих между параметрами *N1* и *N2* и при этом не превосходящих значение параметра *A*. Если таких элементов нет, выдать соответствующее сообщение.

14. В одномерном целочисленном массиве найти количество элементов, меньших параметра *N1,* при этом больших *N2* и не принадлежащих отрезку [*a*; 2*c*]. Если таких элементов нет, выдать соответствующее сообщение.

15. Определить, есть ли в целочисленном массиве из *N* элементов положительные элементы, не превосходящие параметр *A*. Если таких элементов нет, выдать соответствующее сообщение.

16. В одномерном целочисленном массиве из *N* элементов найти первый элемент, не превосходящий по абсолютному значению параметр *A*. Если такой элемент есть, вывести на экран его и его номер, иначе – выдать соответствующее сообщение.

17. В одномерном целочисленном массиве из *M* элементов найти первый положительный элемент, не превосходящий абсолютное значение параметра *A*. Если такой элемент есть, вывести на экран его и его номер.

18. Задан массив из *N* целых чисел. Найти произведение чётных элементов массива.

19. Задан массив *A* из *N* целых чисел. Создайте массив *B* из *N* целых чисел, состоящий из элементов *A*, значение которых удваивается. Вычислить *K* – количество отрицательных элементов массива *B* и *T* – наименьший элемент массива *B*.

20. Задан массив *A* из *N* целых чисел. Найти количество элементов, лежащих в диапазоне от *a* до *b.*

6.1. МНОГОМЕРНЫЕ МАССИВЫ

Для объявления многомерного массива необходимо указать каждое их измерение в квадратных скобках после имени массива:

<тип элементов имя массива> [*<целочисленная константа1>*][*<целочисленная константа2>*];

Например, так объявлен двумерный массив:

```
int const M = 5;
int const N = 3;
int mas2[N][M];
```

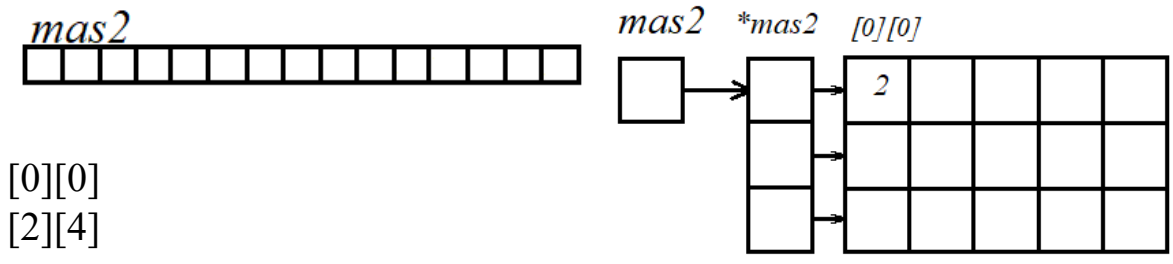
А так объявлен трёхмерный массив:

```
int mas3[N][M][N];
```

Многомерные массивы можно рассматривать как массив массивов. В памяти такие массивы располагаются подряд построчно. При этом при переходе к следующему элементу в первую очередь изменяется последний индекс.

Для обращения к элементам массива необходимо указать оба индекса в квадратных скобках, например `mas[0][0]=2;` запись целочисленной константы 2 в ячейку с индексами ноль, ноль.

На рис. 29 схематично представлено расположение двумерного массива в памяти и с точки зрения синтаксиса языка *C++*.



а) *б)*

Рис. 29. Схематичное представление двумерного массива:
а – расположение двумерного массива в памяти; б – представление массива с точки зрения языка C++

Для инициализации элементов многомерного массива при объявлении необходимо указать элементы в фигурных скобках списком. Если размерность указана в квадратных скобках, элементы необходимо разделить только запятыми:

```
int mas2[3][2] = {1,2,3,4,5,6};
```

Если вторая размерность не указана, то каждую строку необходимо заключить в дополнительные фигурные скобки:

```
int mas2[3][] = { {1,2},{3,4},{5,6} };
```

Следующий фрагмент кода реализует форматированный вывод элементов двумерного массива:

```
for (int i = 0; i < 3; i++)
{
for (int j = 0; j < 2; j++)printf("%i",
mas2[i][j]);
printf("\n");
}
```

Запишем в ячейки массива значения, обращаясь к элементам массива через индексное выражение и используя имя массива как указатель:

```
mas2[1][1] = 2;
*(*(mas2+1)+2) = 3;
```

Если двумерный массив имеет одинаковое количество столбцов и строк, его называют *квадратной матрицей*.

При этом если i – номер строки, j – номер столбца, то элемент массива расположен:

- на главной диагонали, если $i = j$;
- ниже главной диагонали, если $i > j$;
- выше главной диагонали, если $i < j$;
- на побочной диагонали, если $i = n - j - 1$;
- ниже побочной диагонали, если $i > n - j - 1$;
- выше побочной диагонали, если $i < n - j - 1$.

6.2. ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ

В предыдущих разделах курса память под переменные была выделена на этапе компиляции программы. Выделенную таким образом оперативную память называют *статической*, а хранимые таким образом переменные называют *статическими*. Переменные, память под которые выделена во время исполнения программы, называют *динамическими*. Такой способ выделения памяти называют *динамическим*, распределяемую таким способом память – *динамической памятью*, или *кучей*.

Динамически распределяемая память (***куча***) – это свободная область оперативной памяти, которая может быть выделена под переменные программы во время исполнения программы. Использование динамической памяти позволяет изменять объём обрабатываемых данных, динамические переменные можно удалить во время исполнения программы, тем самым освободив динамически распределяемую память. Работа с такими переменными ведётся через указатели.

Динамически выделить память под переменную или массив данных можно при помощи операции `new`:

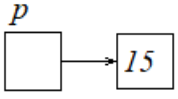
`new` *<тип данных>* [*<количество элементов>*]`;`
или
`new` *<тип данных> (<инициализатор>)*`;`

Оператор `new` возвращает в точку вызова типизированный указатель на начало захваченной области памяти, дальнейшая работа с захваченной памятью ведётся через это значение.

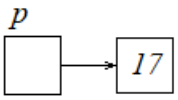
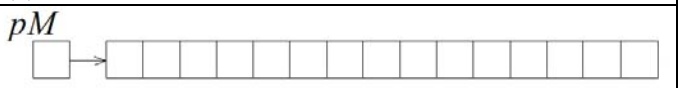
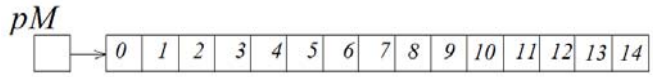
Ниже в табл. 2 приведены фрагменты работы с динамическими денными и даны схематичные пояснения к строкам кода.

Таблица 2

Работа с указателями

Пример кода	Схематическое изображение
`int* p = new int(15);`	
Объявление указателя на целое, выделение динамической памяти, запись в выделенную ячейку значения 15, запись адреса выделенной переменной в указатель	

Окончание табл. 2

Пример кода	Схематическое изображение
`*p += 2;`	
Изменение значение указуемого на два	
`int* pM = new int[15];`	
Создание динамического массива из 15 элементов. Созданный таким образом массив нельзя проинициализировать при объявлении	
`for (int j = 0; j < 15; j++) {pM[j] = j;}`	
Элементы динамического массива проинициализированы значениями своих индексов	

Динамическая память, захваченная при помощи `new`, должна быть освобождена при помощи `delete`:

```
delete[] <указатель>;
delete p;
delete []pM;
```

Самостоятельно схематично поясните следующие строки программного кода:

```
int i = 10;
int *pi = &i;
int *pi2 = int(10);
int *pi3 = new int[10];
```

Для захвата последовательности блоков можно использовать функции `malloc()` и `calloc()`, эти функции унаследованы из языка *C,* их прототипы можно найти в `<stdlib>`:

```
void*malloc(unsigned int n) //захватывает n байт
void*calloc(unsigned int n, unsigned int m)
```
//захватывает `n` раз по `m` байт.

Функции возвращают адрес начала захваченной области памяти. Если памяти недостаточно, чтобы удовлетворить запрос, вернётся ну-

левой указатель. Необходимо проверять возвращаемое значение на его равенство NULL, прежде чем пытаться использовать этот указатель.

Для освобождения памяти, захваченной при помощи функций malloc() и calloc(), используют функцию void free (void*).

Для работы с динамической памятью предпочтительнее использовать new и delete.

Рассмотрим вариант создания двумерного массива в динамической памяти:

```
int** d = new int* [3];
for (int i = 0; i < 3; i++) d[i] = new int[4];
```

В первой строке примера выделена память под переменную *d*. Тип этой переменной – указатель на указатель на целое. Далее происходит захват памяти под массив из трёх указателей на целое и инициализация указателя *d* адресом начала захваченной области.

Во второй строке – в цикле for – происходит инициализация элементов массива. Так как тип элементов массива – указатель на целое, то эти элементы могут быть проинициализированы только адресами целочисленных переменных. После захвата в динамической памяти массива из четырёх элементов адрес начала захваченной области возвращается в точку вызова и записывается в текущий элемент массива указателей.

Для обращения к элементам созданного таким образом массива по-прежнему возможно использование индексного выражения или операции вычисления адреса и разадресации.

В следующем примере для инициализации элементов массива и вывода значений элементов массива на экран использованы оба вышеупомянутых подхода:

```
for (int i = 0; i < 3; i++)
{for (int j = 0; j < 4; j++)
{*(*(d+i)+j) = i + j;
printf("%i ", d[i][j]);
}
printf("\n");
}
```

Лабораторная работа № 6
МНОГОМЕРНЫЕ МАССИВЫ. ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

Цель работы. Освоить работу с многомерными статическими и динамическими массивами, изучить способы создания, инициализации, перебора элементов в многомерных массивах.

Постановка задачи

1. Объявить двумерный статический массив вещественных чисел. Проинициализировать массив любым способом.

Объявить массив указателей на вещественный тип данных.

Настроить элементы последнего массива на минимальные элементы в каждой строке двумерного массива (*i*-й элемент массива указателей на минимальный элемент *i*-й строки массива вещественных чисел).

Организовать форматированный вывод элементов двумерного массива и минимальных элементов.

При выводе минимальных элементов на экран обращаться к ним через элементы массива указателей.

2. Создать и проинициализировать двумерный целочисленный динамический массив.

В созданном массиве найти:

– максимальное значение среди сумм элементов в строках;

– минимальное значение среди произведений элементов в столбцах;

– сумму всех элементов массива;

– найти и вывести на экран индексы минимального элемента массива.

3. Объявить указатель на указатель на целое, настроить его на последовательность из пяти указателей на целое, каждый из которых настроить на последовательность из пяти целых чисел.

Вычислить квадрат разности максимального и минимального элементов организованной таким образом динамической структуры целочисленных значений.

Вопросы для самоконтроля и подготовки к защите лабораторной работы

1. Какие массивы называют многомерными?
2. Напишите синтаксис объявления двумерного статического массива.
3. Объявите и проинициализируйте трёхмерный статический массив.
4. Какие переменные называют динамическими?
5. Объявите и проинициализируйте целочисленную динамическую переменную.
6. Умножьте на два и выведите на экран значение переменной, объявленной в предыдущем вопросе.
7. Объявите и проинициализируйте одномерный динамический массив. Выведите значения элементов массива на экран. Значения выведите в строку, разделяя их пробелом.
8. Объявите и проинициализируйте двумерный динамический массив. Выведите значения элементов массива на экран в виде таблицы, разделяя значения пробелом.
9. Объявите и проинициализируйте трёхмерный динамический массив. Выведите значения элементов массива на экран в виде таблицы, разделяя значения пробелом, а таблицы отделяя друг от друга пустой строкой.
10. Каким образом необходимо освободить динамическую память, захваченную при помощи оператора `new`?

Задания для самостоятельной работы

В задачах с 1 по 21 дана прямоугольная целочисленная матрица, то есть двумерный массив размером $n \times m$, где n и m – заданное количество элементов: n – число строк, m – число столбцов.

В задачах с 22 по 35 дана квадратная целочисленная матрица, то есть двумерный массив размером $n \times n$, где n – количество строк и столбцов.

1. Вычислить сумму элементов двумерного массива.

2. Найти максимальный элемент двумерного массива и его номер.

3. Найти минимальный элемент двумерного массива и его номер.

4. Заменить в двумерном целочисленном массиве все отрицательные элементы на ноль.

5. Заменить в двумерном целочисленном массиве все элементы, превосходящие значение некоторого параметра k, на вводимое с клавиатуры значение A.

6. В целочисленной прямоугольной матрице определить количество строк, не содержащих ни одного нулевого элемента.

7. В целочисленной прямоугольной матрице определить максимальное из чисел, встречающихся в заданной матрице более одного раза.

8. В двумерном целочисленном массиве найти количество столбцов, не содержащих ни одного отрицательного элемента.

9. Дан двумерный числовой массив. Переставляя строки этого массива местами, расположить их в соответствии с возрастанием суммы элементов в строке.

10. Дан двумерный числовой массив. Переставляя строки этого массива местами, расположить их в соответствии с возрастанием суммы положительных элементов в строке.

11. Дан двумерный числовой массив. Переставляя строки этого массива местами, расположить их в соответствии с возрастанием суммы положительных чётных элементов в строке.

12. Дана целочисленная прямоугольная матрица. Найти и вывести на экран количество строк, содержащих хотя бы один нулевой элемент.

13. Дана целочисленная прямоугольная матрица. Найти и вывести на экран номер строки, в которой находится самая длинная серия одинаковых элементов.

14. Переставляя столбцы заданной матрицы, расположить их в соответствии с ростом суммы модулей их отрицательных нечётных элементов.

15. Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

16. Коэффициенты системы линейных уравнений заданы в виде прямоугольной матрицы. С помощью допустимых преобразований привести систему к треугольному виду.

17. Найти количество строк прямоугольной матрицы, среднее арифметическое элементов которых меньше заданной величины.

18. Найти номер первой из строк, содержащих хотя бы один положительный элемент.

19. Дана целочисленная прямоугольная матрица. Определить номер первого из столбцов, содержащих хотя бы один нулевой элемент.

20. Переставляя строки заданной матрицы, расположить их в соответствии с убыванием суммы их отрицательных чётных элементов.

21. Осуществить циклический сдвиг элементов прямоугольной матрицы на *n* элементов вправо или вниз (в зависимости от введённого режима); *n* может быть больше количества элементов в строке или столбце.

22. Дана целочисленная квадратная матрица. Определить произведение элементов только в тех строках, которые не содержат отрицательных элементов.

23. Вычислить сумму элементов на главной диагонали.

24. Вычислить произведение элементов на побочной диагонали.

25. Найти максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

26. Вычислить сумму модулей элементов, расположенных ниже главной диагонали.

27. Вычислить сумму элементов в столбцах, не содержащих отрицательных элементов.

28. Найти и вывести на экран минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы.

29. Вычислить сумму элементов в строках, содержащих хотя бы один отрицательный элемент.

30. Найти сумму модулей элементов, расположенных выше главной диагонали.

31. Вывести на экран номера строк и столбцов всех седловых точек матрицы. Матрица A имеет седловую точку, если является минимальным элементом в i-й строке и максимальным в j-м столбце.

32. Для заданной матрицы размером $m \times m$ найти такие k, для которых k-я строка матрицы совпадает с k-м столбцом.

33. Найти сумму элементов в тех столбцах матрицы, которые содержат хотя бы один отрицательный элемент.

34. Соседями элемента A_{ij} в матрице назовём элементы A_{kl} с $i - 1 <= k <= i + 1$, $j - 1 <= l <= j + 1$, $(k, l) \neq (i, j)$. Построить результат сглаживания заданной вещественной матрицы размером 10×10. Операция сглаживания матрицы даёт новую матрицу того же размера, каждый элемент которой равен среднему арифметическому имеющихся соседей соответствующего элемента исходной матрицы.

35. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы размером 10×10.

36. Вычислить среднее арифметическое элементов трёхмерного целочисленного массива.

37. Организовать трёхмерные динамические структуры данных различными способами.

7. СПОСОБЫ СОРТИРОВКИ МАССИВОВ

Массив называют отсортированным, или *упорядоченным по возрастанию*, если каждый следующий элемент не меньше предыдущего.

Массив называют отсортированным, или *упорядоченным по убыванию*, если каждый следующий элемент не больше предыдущего.

Существует достаточное количество алгоритмов сортировки.

В данном курсе подробно рассмотрены наиболее простые для понимания и реализации алгоритмы.

При выборе алгоритма для решения практических задач сортировки стоит руководствоваться представлениями о количестве необходимых сравнений, сведениями о входных данных, возможности лучшего и худшего случая входных данных.

Поведение алгоритма сортировки называют *естественным*, если время сортировки наименьшее при отсортированной входной последовательности; время сортировки увеличивается по мере увеличения разупорядоченности и время сортировки наибольшее, если входная последовательность упорядочена в обратном порядке.

Сортировка выбором

Этот способ сортировки считается одним из самых простых в понимании и реализации, однако имеет невысокую скорость сходимости из-за большого количества сравнений элементов друг с другом и обменов элементов местами.

Способ рекомендуется применять, если заранее известно, что объём упорядочиваемой последовательности невелик.

При таком подходе отсортированная последовательность строится слева направо путём присоединения к отсортированной части минимального элемента из неотсортированной части.

На первом шаге в массиве находят минимальный элемент и производят его обмен местами с элементом, стоящим на первой позиции. Далее процесс повторяется для неотсортированной части массива до тех пор, пока последовательность не будет исчерпана.

Очевидно, что для реализации этого алгоритма необходимо два цикла. Во внешнем цикле необходимо организовать сдвиг границы

неотсортированной части массива. Во вложенном цикле необходимо осуществить поиск минимального элемента и поставить его на крайнюю левую неотсортированную позицию.

Алгоритм поиска минимального элемента был рассмотрен в разделе 4 «Операторы организации циклов *C*++».

При этом важно запомнить не только значение минимального элемента, но и его номер, так как эта информация необходима для обмена элементов местами.

При упорядочивании элементов последовательности не по убыванию, а по возрастанию осуществляют поиск максимального элемента и обмен его местами с элементом, стоящим на первой позиции. В остальном действия алгоритма остаются теми же самыми.

Переменная – параметр внешнего цикла (номер позиции, на которую будет поставлен минимальный элемент) – будет изменяться от 0 до $n - 2$, где $n - 1$ – номер последнего элемента.

Параметр вложенного цикла изменяется от границы отсортированной части до $n - 1$.

После выполнения i-го шага последовательность от нулевого до i-го элемента будет отсортированной. На рис. 30 представлена блок-схема алгоритма сортировки одномерного массива методом выбора.

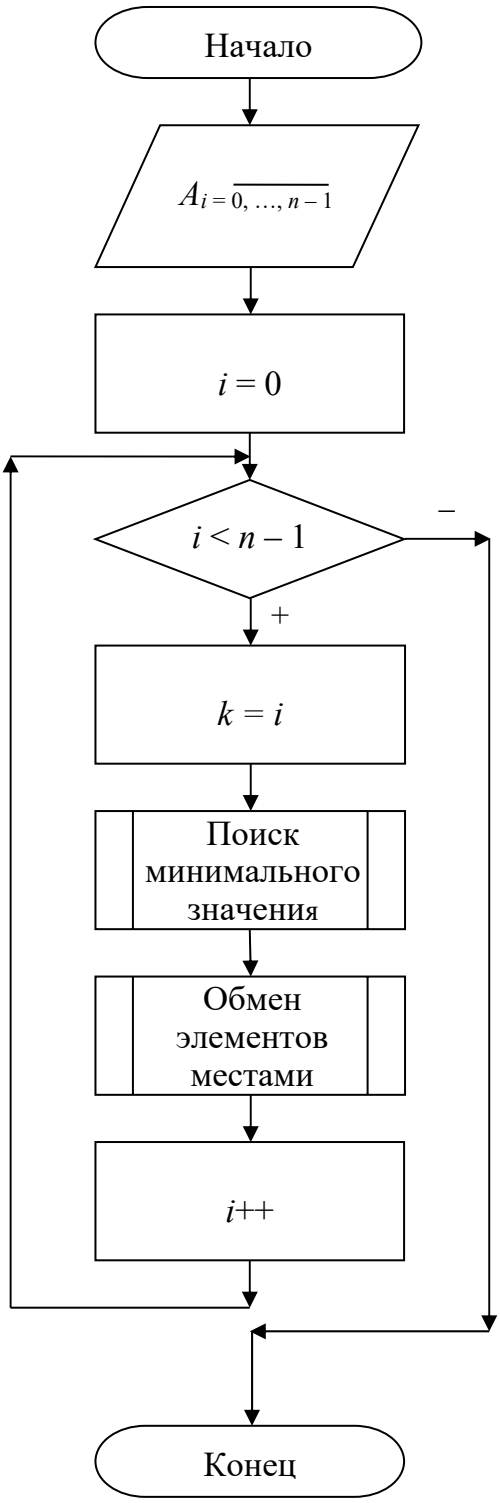


Рис. 30. Блок-схема алгоритма сортировки массива методом выбора

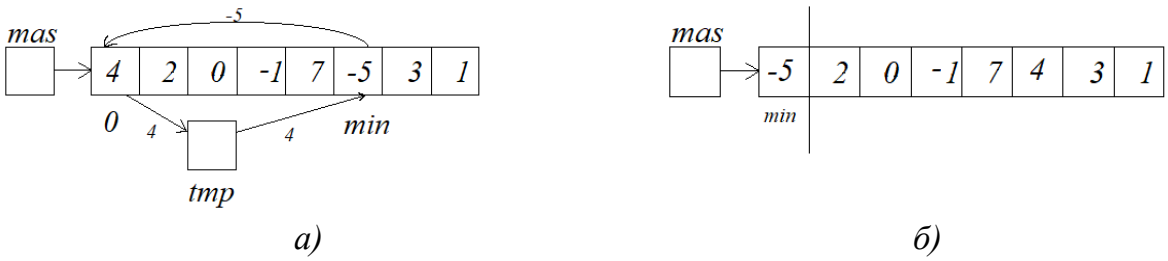
Для того чтобы поменять два элемента местами, необходимо задействовать дополнительную ячейку того же типа, что и элементы массива. Скопировать в эту ячейку

первое значение, минимальное значение записать на место первого элемента, значение из дополнительной переменной записать в ячейку на место минимального значения.

Ниже представлен фрагмент кода обмена местами нулевого значения массива и минимального значения массива:

```
int tmp = mas[0];
mas[0] = mas[min];
mas[min] = tmp;
```

На рис. 31 графически показан обмен элементов массива местами.



а) *б)*

Рис. 31. Обмен элементов местами: а – обмен элементов местами; б – результат обмена

```
for (int i = 0; i < n-1; i++)
{//внешний цикл
int k = i;//k – номер минимального элемента
for (int j = i+1; j < n; j++)
{/*вложенный цикл
Поиск номера минимального элемента от первого не-
отсортированного элемента до границы массива*/
}
//обмен элементов местами
}
```

На рис. 32 показан результат итераций внешнего цикла при сортировке массива методом выбора.

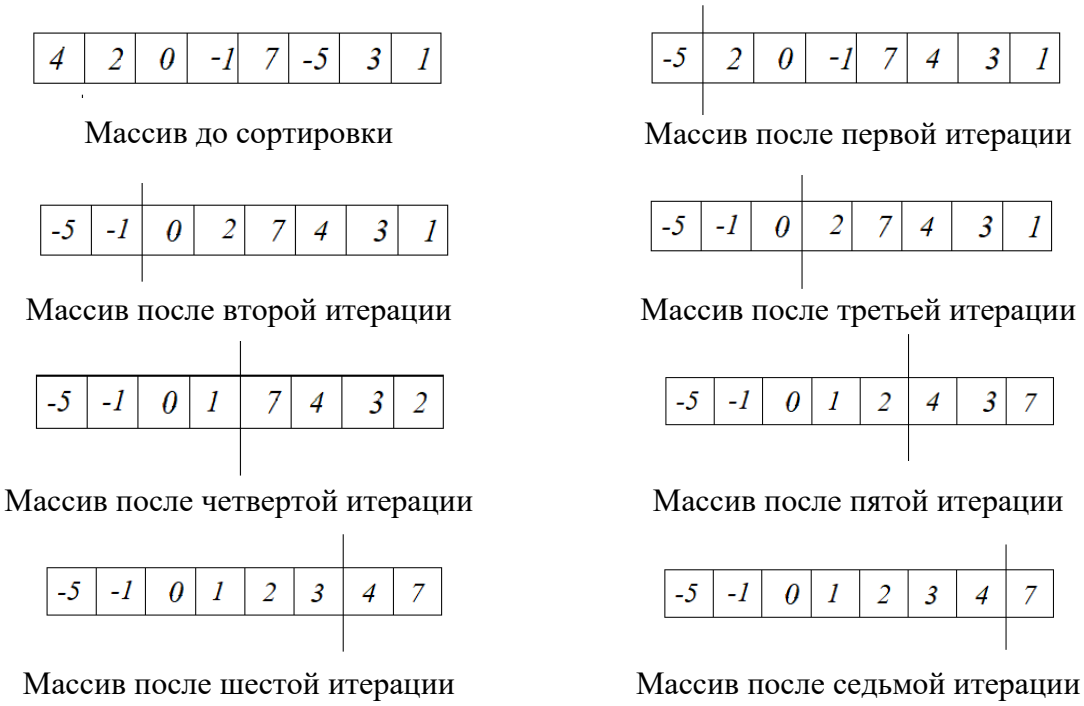


Рис. 32. Итерации внешнего цикла

Сортировка пузырьком

При такой сортировке массив представляют расположенным сверху вниз, номер верхнего элемента минимален, номер нижнего элемента максимален.

За один шаг сортировки осуществляется проход по массиву снизу вверх, при этом осуществляется попарное сравнение элементов и перестановка элементов местами, если два соседних элемента находятся не в правильном порядке. Неправильным считается порядок, при котором элемент с бóльшим индексом меньше предыдущего элемента, как показано на рис. 33.

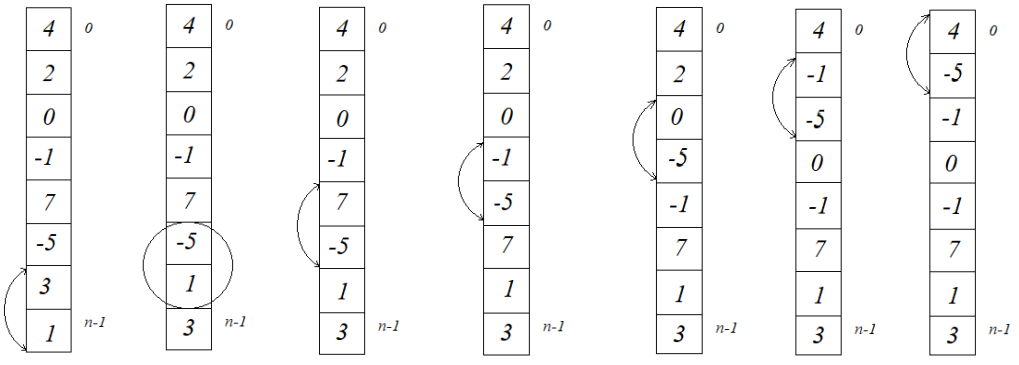


Рис. 33. Первая итерация внешнего цикла

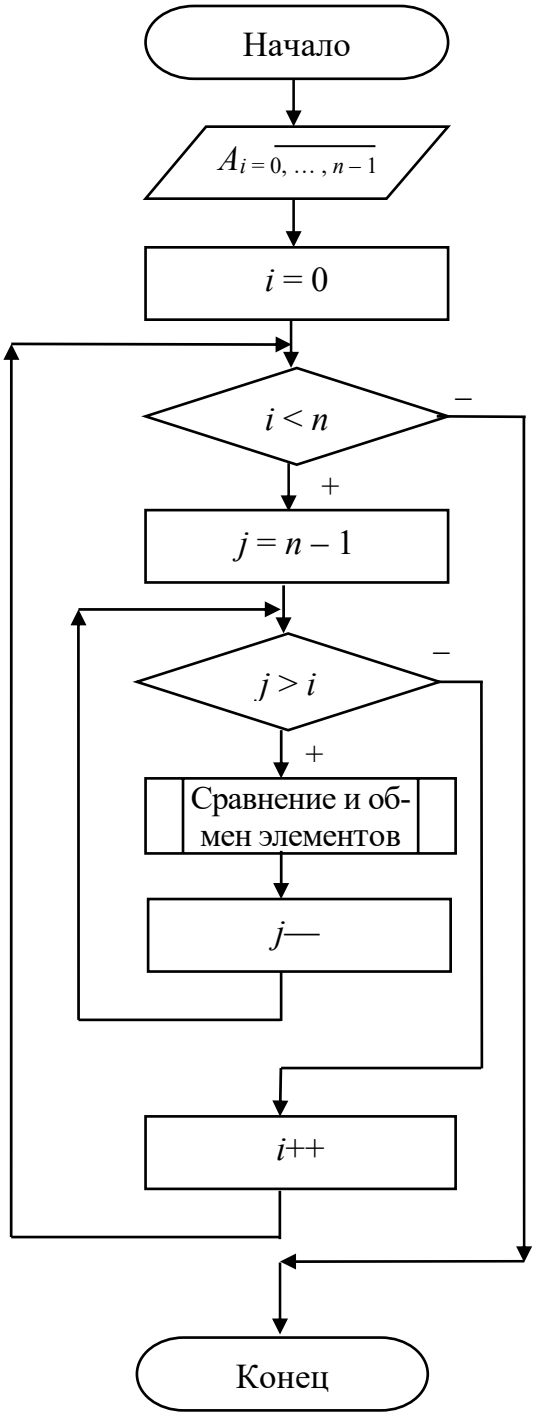


Рис. 34. Блок-схема сортировки пузырьком

После первого прохода по массиву наверху оказывается наименьший элемент (самый лёгкий элемент как бы всплывает наверх, отсюда и аналогия с пузырьком).

Следующий проход осуществляется снизу вверх до второго элемента, так как на первой позиции уже стоит минимальный элемент. Проходы по массиву снизу вверх продолжаются до тех пор, пока все элементы не будут стоять на своих местах, при этом верхняя граница с каждым шагом сдвигается на единицу вниз.

Во вложенном цикле осуществляется проход по массиву снизу вверх, сравнение и перестановка элементов, во внешнем – сдвиг верхней границы.

На рис. 34 представлена блок-схема алгоритма сортировки массива пузырьком.

Этот алгоритм достаточно прост в понимании и реализации, но имеет квадратичный порядок роста числа сравнений, поэтому достаточно медленен.

Скорость сортировки с использованием этого алгоритма можно увеличить, если внести в него несколько улучшений.

1. Если на одном из проходов по массиву не произошло ни одного обмена элементов местами, значит, массив отсортирован и продолжать процесс нет смысла.

Для реализации этого улучшения алгоритма сортировки пузырьком используют логическую переменную – *флаг*. Начальное значение этой переменной необходимо задать до входа во вложенный цикл, в цикле значение флага изменяется, если элементы стоят не в

правильном порядке и осуществляется перестановка. Перед следующей итерацией вложенного цикла необходимо произвести анализ значения флага, то есть проверить, была ли перестановка. Если перестановки не было, произвести досрочный выход из цикла.

2. Если при проходе по массиву запомнить индекс последней перестановки элементов, то при следующем проходе это значение можно рассматривать как верхнюю границу отсортированной части, так как все элементы выше уже стоят на своих местах.

3. Следующее улучшение заключается в изменении направления прохода по массиву.

В классическом алгоритме проходы осуществляются снизу вверх, при этом за один проход самый лёгкий элемент поднимается на самый верх на своё место, а самый тяжёлый опускается вниз только на одну позицию. Если после прохода снизу вверх осуществить проход сверху вниз, то самый тяжелый (максимальный) элемент встанет на свою позицию. Такое чередование направлений позволяет ускорить процесс упорядочивания массива. Алгоритм сортировки пузырьком с использованием смены направления называют *шейкер-сортировкой*.

Сортировка вставками

При сортировке вставками массив представляют разделённым на две части: отсортированную и неотсортированную. На первом шаге отсортированная часть состоит из одного элемента. За один шаг сортировки элемент, следующий за отсортированным, должен найти своё место в отсортированной части. Новый элемент как бы вставляется в отсортированную часть, отсюда и название метода.

Блок-схема алгоритма сортировки массива методом вставки представлена на рис. 35.

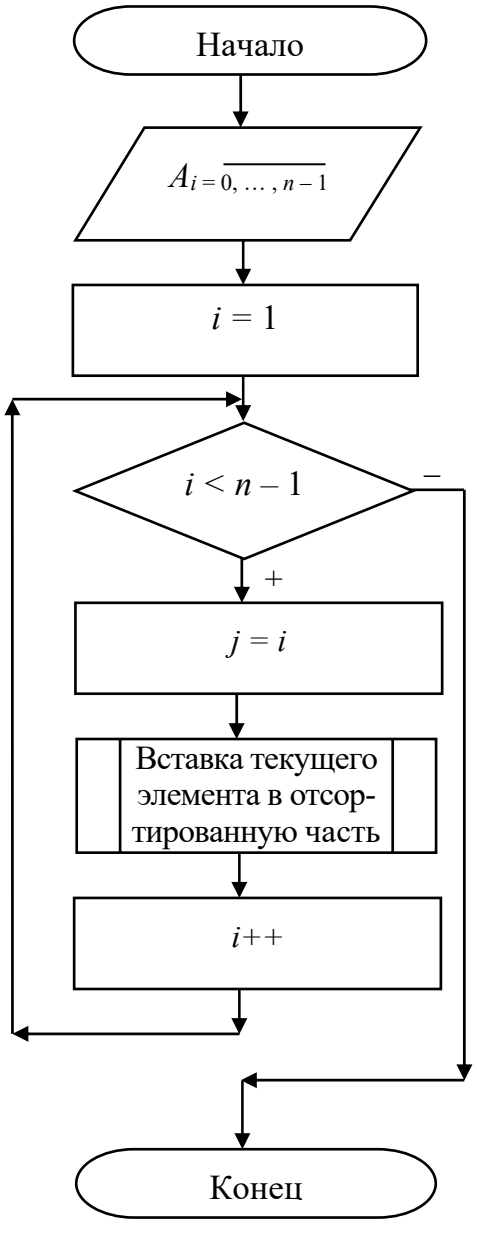


Рис. 35. Блок-схема алгоритма сортировки вставками

Во внешнем цикле осуществляется перебор элементов от второго до последнего, во вложенном цикле осуществляется вставка текущего элемента в отсортированную часть. Соответственно, параметр внешнего цикла изменяется от 1 до $n - 1$, параметр вложенного цикла изменяется от значения на единицу меньшего значения параметра внешнего цикла и до нуля. При этом если элемент нашёл своё место до завершения вложенного цикла, осуществляется досрочный выход из вложенного цикла.

Таким образом, текущий элемент считается вставленным в отсортированную часть массива, если он больше либо равен предыдущему элементу или перебрана вся отсортированная часть, и элемент становится первым элементом в массиве. Если элемент оказывается больше предыдущего, осуществляется досрочный выход из вложенного цикла, вставка текущего элемента считается завершённой, осуществляется переход на следующую итерацию внешнего цикла к поиску места в отсортированной части следующего элемента массива.

На рис. 36 представлен результат итерации внешнего цикла алгоритма сортировки вставками. Чертой обозначена граница отсортированной части массива.

Рис. 36. Массив после итераций внешнего цикла алгоритма сортировки вставками

Лабораторная работа № 7
СПОСОБЫ СОРТИРОВКИ МАССИВОВ

Цель работы. Изучение алгоритмов сортировки данных.

Постановка задачи

Составить блок-схемы и написать по ним программный код следующими способами сортировки данных:

1) методом пузырька без улучшений;
2) методом пузырька с тремя улучшениями: флаг перестановки; запоминание индекса последнего обмена; шейкер-сортировка;
3) простыми вставками;
4) выбором.

Вопросы для самоконтроля и подготовки к защите лабораторной работы

1. Какой массив называют упорядоченным по возрастанию?
2. Назовите известные вам алгоритмы сортировки данных.
3. В каком случае говорят о естественном поведении алгоритма сортировки?
4. Что влияет на скорость сортировки?
5. Какой случай называют лучшим случаем входных данных для алгоритма сортировки по возрастанию?
6. Какие алгоритмы называют алгоритмами внешней сортировки?
7. В чём заключается использование переменной-флага в алгоритме сортировки пузырьком?
8. Модернизацией какого метода является шейкер-сортировка?
9. В чём суть шейкер-сортировки?
10. Для чего в методе сортировки пузырьком используют информацию об индексе последнего обмена?

Задания для самостоятельной работы

Реализовать на языке программирования *C++*:

1) алгоритм сортировки пузырьком с использованием флага;

2) алгоритм сортировки пузырьком с использованием запоминания индекса последнего обмена;

3) шейкер-сортировку;

4) алгоритм сортировки пузырьком с использованием трёх улучшений, приведённых в теоретической части раздела;

5) алгоритм сортировки Шелла;

6) алгоритм гномьей сортировки;

7) алгоритм пирамидальной сортировки;

8) сортировку слиянием на двух упорядоченных целочисленных массивах;

9) пирамидальную сортировку;

10) сортировку подсчётом;

11) блочную сортировку.

8. ПОЛЬЗОВАТЕЛЬСКИЕ ТИПЫ ДАННЫХ. СТРУКТУРЫ

Пользовательский тип данных – это тип данных, который описывает программист по определённым правилам, исходя из условий решаемой задачи.

Структура – это пользовательский тип данных, позволяющий объединить разнотипные данные под одним именем.

Эти данные в рамках структуры называются *полями*.

Описание структуры начинается с ключевого слова `struct`, далее следует имя описываемого типа. Описание полей структуры заключается в фигурные скобки, описание заканчивается точкой с запятой.

Важно не путать имя структуры и имена переменных описанного типа:

```
struct имя типа {описание полей};
struct ST_Name
{
    int pole1;
    float pole2;
    char pole3[10];
};
```

Объявить переменную описанного типа «структура» можно по тем же правилам, что и переменную любого другого типа, то есть указав имя типа и имя переменной:

<имя типа> <имя переменной>`;`

```
ST_Name primer1, primer2;
```

Кроме того, объявить переменные типа «структура» можно сразу после описания типа, до точки с запятой:

```
struct ST_Name
{
    int pole1;
    float pole2;
    char pole3[10];
} primer;
```

В приведённом выше примере описана структура с именем типа `ST_Name` и объявлена переменная `primer` типа `ST_Name.`

Инициализация полей структуры возможна при объявлении переменной, значения полей необходимо передать списком в фигурных скобках, разделив значения запятой:

```
struct ST_Name
    {
        int pole1;
        float pole2;
        char pole3[10];
    } primer{1, 2.3, "name"};
    ST_Name primer1{ 10,2.4,"name1" }, primer2 =
{10,2.4,"name1"};
```

Для обращения к полям структуры необходимо указать имя переменной типа «структура» и через точку – имя соответствующего поля:

<имя переменной>`.`*<имя поля>*

```
primer1.pole1 = 9;
```

При необходимости скопировать поля одной структуры в поля другой структуры того же типа используют операцию присваивания либо копируют каждое поле в отдельности:

```
primer1 = primer2;
primer1.pole1 = primer2.pole1;
primer1.pole2 = primer2.pole2;
```

Объявление массива структур, указателя на структуру происходит по тем же правилам, что и для базовых типов:

```
ST_Name massiv[5];
ST_Name* pST = &primer1;
```

Лабораторная работа № 8
ПОЛЬЗОВАТЕЛЬСКИЕ ТИПЫ ДАННЫХ. СТРУКТУРЫ

Цель работы. Изучить способы создания структур и методы работы с ними.

Постановка задачи

1. Описать структуру с именем STUDENT, содержащую следующие поля: фамилия и инициалы, номер группы, успеваемость (массив из пяти элементов).

2. Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT;

– упорядочение элементов массива по возрастанию номера группы;

– вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4.0; если таких студентов нет, вывести соответствующее сообщение.

3. Объявить указатель на описанный тип, настроить его на элемент массива с наименьшим значением любого поля. Поле выбрать по своему усмотрению, за исключением поля, по которому проводилась сортировка.

Вопросы для самоконтроля и подготовки к защите лабораторной работы

1. Что такое структура в языке программирования *C++*?
2. Напишите синтаксис (правило) описания структуры.
3. Что такое поля структуры?
4. Назовите правило обращения к полям структуры.
5. Каким образом можно скопировать значение полей одной структуры в другую?
6. Каким образом можно объявить переменную типа «структура»?
7. Чем тип данных «структура» отличается от типа данных «массив»?
8. Каким образом можно объявить массив структур?
9. Можно ли объявить указатель на структуру?
10. Может ли структура содержать поля-указатели?

Задания для самостоятельной работы

1. Описать структуру с именем STUDENT, содержащую следующие поля:

– фамилия и инициалы;

– номер группы;

– успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT;

– упорядочение элементов массива по возрастанию среднего балла;

– вывод на дисплей фамилий и номеров групп для всех студентов, имеющих оценки «4» и «5»; если таких студентов нет, вывести соответствующее сообщение.

2. Описать структуру с именем STUDENT, содержащую следующие поля:

– фамилия и инициалы;

– номер группы;

– успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT;

– упорядочение элементов массива по алфавиту;

– вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку «2»; если таких студентов нет, вывести соответствующее сообщение.

3. Описать структуру с именем AEROFLOT, содержащую следующие поля:

– название пункта назначения рейса;

– номер рейса;

– тип самолета.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из семи элементов типа AEROFLOT;

– упорядочение элементов массива по возрастанию номера рейса;

– вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры; если таких рейсов нет, выдать на экран соответствующее сообщение.

4. Описать структуру с именем AEROFLOT, содержащую следующие поля:

– название пункта назначения рейса;

– номер рейса;

– тип самолета.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из семи элементов типа AEROFLOT;

– упорядочение элементов массива по названиям пунктов назначения;

– вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры; если таких рейсов нет, выдать на экран соответствующее сообщение.

5. Описать структуру с именем WORKER, содержащую следующие поля:

– фамилия и инициалы работника;

– название занимаемой должности;

– год поступления на работу.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из десяти элементов типа WORKER;

– упорядочение элементов массива по алфавиту;

– вывод на экран фамилий работников, чей стаж работы в организации превышает значение, введённое с клавиатуры; если таких работников нет, выдать на экран соответствующее сообщение.

6. Описать структуру с именем TRAIN, содержащую следующие поля:

– название пункта назначения;

– номер поезда;

– время отправления.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из восьми элементов типа TRAIN;

– упорядочение элементов массива по алфавиту названий пунктов назначения;

– вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени; если таких поездов нет, выдать на экран соответствующее сообщение.

7. Описать структуру с именем TRAIN, содержащую следующие поля:

– название пункта назначения;

– номер поезда;

– время отправления.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из шести элементов типа TRAIN;

– упорядочение элементов массива по времени отправления поезда;

– вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на экран соответствующее сообщение.

8. Описать структуру с именем TRAIN, содержащую следующие поля:

– название пункта назначения;

– номер поезда;

– время отправления.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из восьми элементов типа TRAIN;

– упорядочение элементов массива по номерам поезда;

– вывод на экран информации о поезде, номер которого введён с клавиатуры; если таких поездов нет, выдать на экран соответствующее сообщение.

9. Описать структуру с именем MARSH, содержащую следующие поля:

– название начального пункта маршрута;

– название конечного пункт маршрута;

– номер маршрута.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из восьми элементов типа MARSH;

– упорядочение элементов массива по номерам маршрутов;

– вывод на экран информации о маршруте, номер которого введён с клавиатуры; если таких маршрутов нет, выдать на экран соответствующее сообщение.

10. Описать структуру с именем MARSH, содержащую следующие поля:

– название начального пункта маршрута;

– название конечного пункта маршрута;

– номер маршрута.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из восьми элементов типа MARSH;

– упорядочение элементов массива по номерам маршрутов;

– вывод на экран информации о маршрутах, которые начинаются или оканчиваются в пункте, название которого введено с клавиатуры; если таких маршрутов нет, выдать на экран соответствующее сообщение.

11. Описать структуру с именем NOTE, содержащую следующие поля:

– фамилия, имя;

– номер телефона;

– дата рождения (массив из трёх чисел).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из восьми элементов типа NOTE;

– упорядочение элементов массива по датам рождения;

– вывод на экран информации о человеке, номер телефона которого введен с клавиатуры; если такого нет, выдать на экран соответствующее сообщение.

12. Описать структуру с именем NOTE, содержащую следующие поля:

– фамилия, имя;

– номер телефона;

– дата рождения (массив из трёх чисел).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из восьми элементов типа NOTE;

– упорядочение элементов массива по алфавиту;

– вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры; если таких нет, выдать на экран соответствующее сообщение.

13. Описать структуру с именем NOTE, содержащую следующие поля:

– фамилия, имя;

– номер телефона;

– дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из восьми элементов типа NOTE;

– упорядочение элементов массива по трём первым цифрам номера телефона;

– вывод на экран информации о человеке, чья фамилия введена с клавиатуры; если такого нет, выдать на экран соответствующее сообщение.

14. Описать структуру с именем ZNAK, содержащую следующие поля:

– фамилия, имя;

– знак зодиака;

– дата рождения (массив из трёх чисел).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив, состоящий из восьми элементов типа ZNAK;

– упорядочение элементов массива по датам рождения;

– вывод на экран информации о человеке, чья фамилия введена с клавиатуры; если такого нет, выдать на экран соответствующее сообщение.

Вопросы для самоконтроля и подготовки к защите лабораторной работы

1. Что представляет собой структура как тип данных языка программирования *C*++?

2. Напишите синтаксис объявления переменной описанного типа struct.

3. Объявите структуру с именем PRIMER, содержащую три разнотипных поля.

4. Объявите переменную типа, описанного в предыдущем вопросе, проинициализируйте поля при объявлении.

5. Измените значения полей переменной типа PRIMER.

6. Письменно приведите пример обращения к полям структуры.

7. Создайте и проинициализируйте массив структур PRIMER.

8. Объявите указатель на структуру PRIMER, настройте его на переменную типа PRIMER.

9. Объявите указатель на указатель на структуру. Настройте его на указатель, объявленный в предыдущем вопросе.

10. Измените значения полей переменной типа PRIMER, обращаясь к ним через указатели из вопросов 8 и 9.

9. РАБОТА СО СТРОКАМИ

Механизм работы со строками в языке *C++* унаследован от языка *C*.

С точки зрения языка *C*, ***строка*** – это массив символов, заканчивающийся признаком конца строки “\0”. Поэтому под элементы массива нужно выделить на один символ больше общего количества символов строки.

Проинициализировать объявленную строку можно при объявлении одним из двух продемонстрированных в примере способов:

```
char str[5]="abcd";
char str1[5] = { 'a','b','c','d','\0' };
```

Так как строка – это массив, то возможна динамическая инициализация и работа со строкой через указатель:

```
char* p_str = new char[5];
```

В библиотеке *C++* описан класс `string`, обеспечивающий работу со строками. В этом классе реализованы такие операции, как сравнение, присвоение, конкатенация, индексация, поиск подстроки и др. Ниже приведены примеры работы с переменными класса `string` и строками, реализованными по правилам языка *C*.

```
string s;
string s1 = "stroka1";
s = s1 + " + stroka2";
```

Зачение `s` равно `"stroka1 + stroka2"`.

```
string s2 = "abc";
s1 = "aaaaaaa";
string ss;
(s2 > s1) ? ss=s2 : ss=s1;
```

Значение `ss` равно `"abc"`

```
char c1[] = "abc";
char c2[] = "aaaaa";
int k = strcmp(c2,c1);
```

Функция `strcmp` возвращает целочисленное значение, это значение будет больше нуля, если первая строка больше второй, равно нулю, если строки равны, и меньше нуля, если первая строка меньше второй.

При этом производится сравнение не длины строк, а самих строк в лексикографическом порядке.

При таком подходе строка “*aaaaaa*” меньше строки “*bc*”.

```
cin.getline(c1,3); // ввод с клавиатуры значения строки c1
strcpy_s(c2, c1); /*в строку c2 будет скопировано значение из строки c1*/
```

При работе со строковыми данными важную роль играют алгоритмы поиска подстроки в строке. Рассмотрим алгоритм прямого (или последовательного) поиска, алгоритм довольно прост в понимании и реализации.

Суть алгоритма заключается в посимвольном сравнении символов подстроки с символами строки. Посимвольное сравнение производится до первого несовпадения или до завершения символов подстроки.

После несовпадения символа подстроки с символом строки производится сдвиг подстроки относительно строки на один символ, то есть индекс начала подстроки в строке смещается на единицу, индекс текущего элемента в подстроке перемещается в начало.

Если все символы подстроки совпали с символами строки, то номер символа строки, с которого подстрока начинается в строке, будет результатом работы алгоритма. Если строка исчерпана, а подстрока не была найдена, необходимо выдать соответствующее сообщение.

Наглядно работа описанного алгоритма представлена в табл.3.

Таблица 3

Поиск подстроки в строке

Сопоставление строки и подстроки									Результат сопоставления
a_0	a_1	b_2	a_3	c_4	a_5	b_6	c_7	d_8	–
a_0	b_1	c_2							Несовпадение на 1-м символе подстроки
	a_0	b_1	c_2						Несовпадение на 2-м символе подстроки
		a_0	b_1	c_2					Несовпадение на 0-м символе подстроки
			a_0	b_1	c_2				Несовпадение на 1-м символе подстроки
				a_0	b_1	c_2			Несовпадение на 0-м символе подстроки
					a_0	b_1	c_2		Подстрока в строке найдена, начиная с 5-го символа в строке

Для реализации алгоритма прямого поиска необходимо использовать два цикла. В теле вложенного цикла происходит сравнение *i*-го элемента подстроки с *j* + *i*-м элементом строки. В случае совпадения элементов осуществляется увеличение индекса *i*, иначе осуществляется досрочный выход из вложенного цикла и увеличение индекса *j* во внешнем цикле. Если все элементы подстроки совпали с элементами строки, то значение индекса *j* и есть номер элемента в строке, с которого начинается подстрока в строке, а следовательно, и результат работы алгоритма. Работа алгоритма завершается после того, как будет найдено первое вхождение подстроки в строку. Для нахождения последующих вхождений необходимо организовать повторное применение алгоритма.

Лабораторная работа № 9
СТРОКИ. АЛГОРИТМЫ ПОИСКА ПОДСТРОКИ В СТРОКЕ

Цель работы. Освоение способов работы со строками в языке программирования *C++*. Изучение алгоритмов поиска подстроки в строке.

Постановка задачи

1. Создать строку как массив символов. Проинициализировать этот массив при объявлении. Выполнить сортировку элементов массива по алфавиту. Вывести на экран отсортированный массив.

2. Объявить массив из десяти строк. Проинициализировать элементы массива вводом с клавиатуры. Выполнить сортировку элементов массива в алфавитном порядке. Вывести на экран отсортированный массив.

3. Объявить и проинициализировать три строки таким образом, чтобы из них можно было составить осмысленную фразу, каждую строку начинать с прописной буквы, заканчивать пробелом. Объединить строки в одну. Удалить из полученной строки последний пробел, заменив его на соответствующий знак препинания. Первую букву полученной строки сделать заглавной. Вывести на экран полученный результат.

4. Реализовать алгоритм прямого поиска.

Вопросы для самоконтроля и подготовки к защите лабораторной работы

1. Что такое строка с точки зрения языка *C*?
2. Какими способами ведётся работа со строками в языке *C*++?
3. Каким образом можно проинициализировать строку?
4. Что такое признак конца строки?
5. Что такое операция конкатенация строк?
6. Назовите тип возвращаемого значения функции `strcmp`.
7. Поясните смысл возвращаемого значения функции `strcmp`.
8. Каким образом можно скопировать значение одной строки в другую?
9. Что является входными значениями алгоритма прямого поиска?
10. Что является выходным значением алгоритма прямого поиска?

Задания для самостоятельной работы

1. Объявить строку как массив символов. Выполнить сортировку элементов объявленного массива по алфавиту; вывести на экран элементы отсортированного массива в обратном порядке; если элемент встречается более одного раза, вывести его только один раз.
2. Удалить из строки повторяющиеся символы.
3. Заменить в строке все заглавные буквы на строчные.
4. Дана строка, вывести на экран все цифры, содержащиеся в этой строке.
5. Дана строка, подсчитать и вывести на экран количество слов в ней.
6. Дана строка, подсчитать и вывести на экран количество гласных букв в ней.
7. Дана строка, среди символов строки содержатся цифры. Вычислить и вывести на экран сумму этих цифр и удалить их из строки.
8. Объединить три строки в одну, разделив их пробелом.
9. Реализовать алгоритм Бойера и Мура.
10. Реализовать алгоритм Кнута, Морриса и Пратта.

ЗАКЛЮЧЕНИЕ

Работа по подготовке специалистов в области информационных технологий, прикладной математики, компьютерных наук предполагает освоение ими дисциплин программирования и получение навыков в области разработки, отладки, тестирования и модификации программного обеспечения. Работа специалиста в этой области немыслима без перечисленных выше навыков.

Освоение курса «Основы программирования» даёт студенту возможность получения этих навыков в области разработки алгоритмических решений, представления решения в виде блок-схемы, самостоятельной разработки кода на языке программирования высокого уровня, использования и отладки для дальнейшей работы ранее написанного кода, в том числе и кода, написанного другими разработчиками.

Полученные знания и навыки могут быть применены студентами при решении широкого спектра учебных и профессиональных задач.

РЕКОМЕНДАТЕЛЬНЫЙ БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Павловская, Т. А. C/C++. Программирование на языке высокого уровня : учеб. для вузов / Т. А. Павловская. – СПб. : Питер, 2009. – 432 с. – ISBN 978-5-91180-174-8.

2. Страуструп, Б. Программирование. Принципы и практика с использованием C++ / Б. Страуструп. – М. : Вильямс, 2016. – 1328 с. – ISBN 978-5-8459-1949-6.

3. Шилдт, Г. Полный справочник по C++ : пер. с англ. / Г. Шилдт. – 4-е изд., стер. – М. : Вильямс, 2015. – 800 с. – ISBN 978-5-8459-2047-8.

4. Воронова, Л. М. Типовые алгоритмические структуры для вычислений : учеб. пособие / Л. М. Воронова ; Владим. гос. ун-т. – 2-е изд., перераб. и доп. – Владимир : Ред.-издат. комплекс ВлГУ, 2004. – 96 с. – ISBN 5-89368-503-2.

5. Прата, С. Язык программирования C++. Лекции и упражнения : пер. с англ. / С. Прата. – М. : Вильямс, 2017. – 1248 с. – ISBN 978-5-8459-2048-5.

ПРИЛОЖЕНИЯ

Приложение 1

Образец титульного листа отчёта по лабораторной работе

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)

Кафедра ФиПМ

ЛАБОРАТОРНАЯ РАБОТА № 1

по дисциплине «Основы программирования»
на тему: «Основные алгоритмические структуры»

Выполнил:
ст. группы ПМИ-120
Иванов И. И.
Принял:
ст. преподаватель
каф. ФиПМ
Шишкина М. В.

Владимир 2021

Приложение 2

Требования к форматированию отчёта по лабораторной работе

Отчёт по лабораторной работе должен быть создан в текстовом процессоре *Microsoft Word*. Отчёт должен быть распечатан на одной стороне чистого белого листа форматом А4, не допускается использование оборотной стороны использованных листов.

Титульный лист должен быть оформлен в соответствии с примером, приведенным в прил. 1.

При наборе текста используется шрифт *Times New Roman*, кегль 14 пт.

Поля задать следующим образом: левое поле – 30 мм; правое – 10 мм; верхнее и нижнее – 20 мм. Междустрочный интервал полуторный; режим выравнивания по ширине; отступ в начале абзаца (красная строка) 15 мм.

Все листы должны быть пронумерованы. Титульному листу номер присваивают, но не проставляют. Следовательно, номер страницы, следующий после титульного листа, – 2. Номер проставляют по центру листа, кегль 12 пт.

Для допуска к защите лабораторной работы необходимо продемонстрировать работоспособность написанной программы, представить отчёт, написанный и оформленный в соответствии с требованиями.

Для защиты работы необходимо ответить на вопросы по теоретической части, подробно пояснить строки кода программы, указанные преподавателем. Самостоятельно выполнить выданное преподавателем задание, по теме и уровню сложности соответствующее заданию лабораторной работы.

Приложение 3

Требования к содержанию отчёта

Отчёт по лабораторной работе должен содержать следующие части.

1. Титульный лист, оформленный по образцу, приведённому в прил. 1.

2. Цель работы.

3. Постановку задачи.

Формулировка цели работы и постановка задачи должны точно соответствовать указанным преподавателем.

4. Пять – семь ключевых слов на русском и английском языках. В качестве ключевых слов стоит выбирать слова, опираясь на которые можно составить исчерпывающий ответ по теме работы.

5. Краткая теоретическая часть должна содержать от двух до четырёх страниц связного самостоятельно написанного текста с примерами. Теоретическая часть не должна быть копией лекций, но лекционный материал должен быть взят за основу. При этом должны быть рассмотрены все вопросы, затронутые в работе, с теоретической точки зрения и на примерах, реализованных автором.

6. Ход работы должен содержать программный код, написанный на изучаемом языке программирования, и несколько тестов, результатов работы программы с объяснением полученных результатов.

7. Выводы по работе. В этой части необходимо указать, к какому выводу пришёл автор во время выполнения работы. Например, о достоинствах и недостатках изученного метода, разработанного алгоритма и т. п., для решения какого класса задач используют этот подход.

Приложение 4

Базовые типы данных *C++*

В таблице представлены основные типы данных языка *C++*.

В первом столбце указано зарезервированное слово, определяющее тип данных.

Во втором столбце указано количество байт, отводимое под переменную с соответствующим типом данных.

В третьем столбце приведён диапазон допустимых значений.

Тип	Байт	Диапазон принимаемых значений
Целочисленный (логический) тип данных		
bool	1	0 / 255
Целочисленный (символьный) тип данных		
char	1	0 / 255
Целочисленные типы данных		
short int	2	–32 768 / 32 767
unsigned short int	2	0 / 65 535
int	4	–2 147 483 648 / 2 147 483 647
unsigned int	4	0 / 4 294 967 295
long int	4	–2 147 483 648 / 2 147 483 647
unsigned long int	4	0 / 4 294 967 295
Типы данных с плавающей точкой		
float	4	–2 147 483 648.0 / 2 147 483 647.0
long float	8	–9 223 372 036 854 775 808.0 / 9 223 372 036 854 775 807.0
double	8	–9 223 372 036 854 775 808.0 9 223 372 036 854 775 807.0

Приложение 5

Таблицы кодов *ASCII*

0 -	16 - ▶	32 -	48 - 0	64 - @	80 - P	96 - '	112 - p
1 - ☺	17 - ◀	33 - !	49 - 1	65 - A	81 - Q	97 - a	113 - q
2 - ☻	18 - ↕	34 - "	50 - 2	66 - B	82 - R	98 - b	114 - r
3 - ♥	19 -!!	35 - #	51- 3	67 - C	83 - S	99 - c	115 - s
4 - ♦	20 - ¶	36 - $	52 - 4	68 - D	84 - T	100 - d	116 - t
5 - ♣	21 - §	37 - %	53 - 5	69 - E	85 - U	101 - e	117 - u
6 - ♠	22 - ▬	38 -&	54 - 6	70 - F	86 - V	102 - f	118 - v
7 -	23 - ↕	39 - '	55 - 7	71 - G	87 - W	103 - g	119 - w
8 -	24 - ↑	40 - (	56 - 8	72 - H	88 - X	104 - h	120 - x
9 -	25 - ↓	41 -)	57 - 9	73 - I	89 - Y	105 - i	121 - y
10 -	26 - →	42 - *	58 - :	74 - J	90 - Z	106 - j	122 - z
11 -	27 - ←	43 - +	59 - ;	75 - K	91 - [	107 - k	123 - {
12 -	28 - ∟	44 - ,	60 - <	76 - L	92 - \	108 - l	124 - |
13 -	29 - ↔	45 - -	61 - =	77 - M	93 - j	109 - m	125 - }
14 - ♫	30 - ▲	46 - .	62 ->	78 - N	94 - ^	110 - n	126 - ~
15 - ☼	31 - ▼	47 - /	63 - ?	79 - O	95 - ÷	111 - o	127 - ⌂
16 - ▶	32 -	48 -0	64 - @	80 - P	96 -	112 - p	

128 -А	144 - Р	160 - а	176 - ▒	192 - └	208 - ╨	224 -р	240 - Ё
129 - Б	145 - С	161 - б	177 - ▒	193 - ┴	209 - ╤	225 -с	241- ё
130 -В	146 - Т	162 - в	178 - ▓	194 - ┬	210 - ╥	226 -т	242 - Є
131 - Г	147 - У	163 - г	179 - |	195 - ├	211 - ╙	227 -у	243 - є
132 -Д	148 - Ф	164 - д	180 - ┤	196 - —	212 - ╘	228 -ф	244 - Ї
133 -Е	149 - Х	165 - е	181 - ╡	197 - ┼	213 - ╒	229 - х	245 - ї
134 - Ж	150 - Ц	166 - ж	182 - ╢	198 - ╞	214 - ╓	230 - ц	246 - Ў
135 -З	151 - Ч	167 - з	183 - ╖	199 - ╟	215 - ╫	231 - ч	247 - ў
136 -И	152 -Ш	168 - и	184 - ╕	200 -	216 - ╪	232 - ш	248 - °
137 - Й	153 -Щ	169 - й	185 - ╣	201 - ╔	217 - ┘	233 - щ	249 - ●
138- К	154 - Ъ	170 - к	186 - ║	202 - ╩	218 - ┌	234 - ъ	250 - .
139 -Л	155 - Ы	171 - л	187 - ╗	203 - ╦	219 - ▉	235 - ы	251 - √
140 -М	156 - Ь	172 - м	188 - ╝	204 - ╠	220 - ▄	236 - ь	252- №
141 -Н	157 - Э	173 - н	189 - ╜	205 - ═	221 - ▌	237 - э	253 - ¤
142 -О	158 - Ю	174 - о	190 - ╛	206 - ╬	222 - ▐	238 - ю	254 - ▪
143- П	159 - Я	175 - п	191 - ┐	207 - ╧	223 - ▀	239 - я	255 -
144 - Р	160 - а	176 - ▒	192 - └	208 - ╨	224 - р	240 - Ё	

ОГЛАВЛЕНИЕ

Учебное издание

ШИШКИНА Мария Викторовна

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Практикум

Редактор Е. А. Платонова
Технический редактор Ш. В. Абдуллаев
Корректор О. В. Балашова
Компьютерная верстка Е. А. Кузьминой
Выпускающий редактор А. А. Амирсейидова

Подписано в печать 27.09.2021.
Формат 60 × 84/16. Усл. печ. л. 6,51. Тираж 50 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.