ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА И ГОСУДАРСТВЕННОЙ
СЛУЖБЫ при ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

ОРЛОВСКИЙ ФИЛИАЛ

СЕМИНА Е.В., АРТЕМОВ А.В.

ОСНОВЫ АЛГОРИТМИЗАЦИИ И СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ НА C++.

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Орел – 2014

ББК
УДК
Ш

Рекомендовано к изданию Ученым Советом ОФ РАНХиГС

РЕЦЕНЗЕНТЫ:
Савина О.А. доктор экономических наук, профессор, зав. кафедрой «Информационные системы» Федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Госуниверситет — УНПК»
Новиков В.С. кандидат педагогических наук, доцент кафедры «Информатика» Федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Орловский государственный университет»

ISBN _____

Семина Е.В., Артемов А.В. Основы алгоритмизации и структурного программирования на С++. – Орел, 2014. – 48 с.

Данное пособие предназначено для приобретения практических навыков структурного программирования. В нем приводятся необходимые для этого сведения по основам алгоритмизации и базовые конструкции языка С++. На конкретных примерах рассмотрены алгоритмы решения типовых задач, для каждой из которых приведены математическая формулировка, подробное объяснение методики решения, схема алгоритма и программа на языке программирования С++, даны задания для самостоятельного выполнения.

Пособие может быть полезно студентам и аспирантам для закрепления базовых навыков программирования и использовано преподавателями при организации практических и лабораторных занятий, подкрепляющих теоретическую базу по структурному программированию.

ВВЕДЕНИЕ

В современном обществе основным техническим средством автоматизации и информационной поддержки различных сфер жизни человека служит персональный компьютер.

В настоящее время сформировался высокий спрос на специалистов в области программирования и настройки сложных ERP систем, таких как 1С, SAP, Oracle и т.п. В любой профессиональной области нередко возникают ситуации, когда функциональности существующих программ недостаточно. Умение написать нужную утилиту или слегка переделать имеющуюся резко повышает статус работника по сравнению с сослуживцами, а привлечение для этого профессиональных разработчиков зачастую нерационально с точки зрения временных и материальных затрат. Благодаря повсеместному внедрению телекоммуникационных средств связи и появлению безлимитных тарифов для доступа в Интернет в современном мире IT-технологий довольно часто появляются различные новые технологии. В будущем эта тенденция вряд ли снизится, скорее наоборот. В то же время основы, на которых строится все программирование, меняются гораздо реже. Знание фундаментальных основ программирования дает возможность освоения новых технологий с минимальными затратами времени и сил. Понимание основ является ключевым в достижении успеха в любом направлении, в том числе и в программировании.

В настоящее время большое внимание при обучении уделяется формированию алгоритмического мышления, творческих и исследовательских способностей студентов. Программирование для этого подходит как никакая другая наука.

Цель данного пособия — развитие у студентов практических навыков подготовки задач для решения на компьютере, формирование алгоритмического мышления, умения четко планировать свои действия и последовательно достигать результата по разработанному алгоритму, воспитание таких качеств, как четкость мышления и умение раскладывать поставленную задачу на подзадачи.

Особенностью освоения материала является необходимость сочетания изучения теоретических основ с одновременным решением задач на компьютере. Очень часто при решении задач на компьютере не отводится время на обдумывание и решение задачи без компьютера, тогда как проработка и оптимизация алгоритма на бумаге обычно приводит к значительной экономии сил, позволяет избежать многих ошибок. К любой задаче нужно подходить таким образом, чтобы сначала суметь придумать ее решение (алгоритм решения задачи), а затем уже пытаться реализовать это решение, путем написания кода программы.

В пособии рассматриваются примеры программирования типовых алгоритмов. Программы решения типовых задач являются основой для программирования подобных задач, предлагаемых для самостоятельного решения. Описания характерных приемов алгоритмизации выполнены по единой структуре и содержат краткие теоретические сведения, пример выполнения типовой задачи и задания для самостоятельной подготовки.

Пособие ориентировано на один из наиболее распространенных языков программирования – C++, и может быть использовано при работе в любой среде программирования C++.

Пособие предназначено для использования студентами, аспирантами и преподавателями при организации практических и лабораторных занятий, подкрепляющих теоретическую базу по программированию.

1. АЛГОРИТМЫ И СПОСОБЫ ИХ ОПИСАНИЯ

В основе решения любой задачи лежит понятие алгоритма. Алгоритм — конечная последовательность действий, однозначно определяющих процесс преобразования исходных и промежуточных данных в результат решения задачи.

Любой алгоритм должен обладать следующими основными свойствами:

1. Определенность (детерминированность) — алгоритм должен быть составлен так, чтобы каждое действие трактовалось однозначно.

2. Результативность — алгоритм порождает процесс, который обязательно должен приводить к результату после конечного числа шагов преобразований.

3. Массовость — алгоритм должен быть применима к любому ряду исходных данных.

4. Дискретность — алгоритм должен быть составлен так, чтобы его можно было разбить на подзадачи.

Существуют три способа задания алгоритмов: описательный (словесный), графический (блок-схемный) и операторный (программный).

При словесном способе алгоритм задается в произвольном изложении на естественном языке. Рассмотрим словесный способ описания алгоритмов на примере вычисления наибольшего общего делителя (НОД) двух произвольно заданных положительных чисел (известный как алгоритм Евклида):

1. Ввести два произвольных положительных целых числа A и B и перейти к пункту 2.
2. Сравнить числа A и B. Если A>B, то перейти к пункту 3. Если A<B, то перейти к пункту 4. Если A=B, то перейти к пункту 5.
3. Вычислить новое значение A=A-B и перейти к пункту 2.
4. Вычислить новое значение B=B-A и перейти к пункту 2.
5. Найденное значение A — есть искомый результат, т.е. НОД двух заданных чисел. Задача решена, и процесс вычислений прекращается.

Однако такой способ словесного описания часто приводит к громоздким и трудно воспринимаемых текстам и применяется лишь для представления простейших алгоритмов.

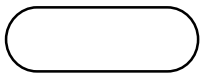
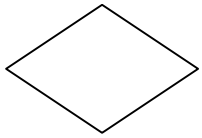
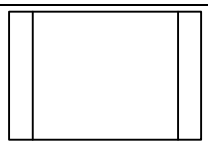
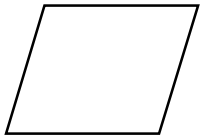
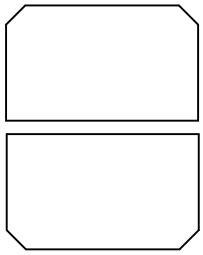
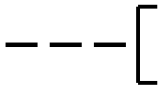
Большей наглядностью обладает графический способ задания алгоритмов.

Каждому типу действий в блок-схеме соответствует определённый символ - блок. Изображение блоков регламентируется ГОСТ 19.701-90. "ЕСПД. Схемы алгоритмов, программ данных и систем. Условные обозначения и правила выполнения". Наиболее часто используемые символы действий для описания алгоритмов представлены в табл.1.1.

Алгоритм представляется последовательностью блоков. Внутри блоков указывается информация, характеризующая выполняемые ими функции. Метод блок-схем независим от специфики языков программирования, поэтому в описаниях блоков не рекомендуется использовать резервированные слова и символы языков программирования.

Таблица 1.1

Условные обозначения в блок-схемах

Название	Обозначение	Описание
Блок начало-конец (пуск-остановка)		Обозначение начала и конца алгоритма (входа и выхода в подпрограммы)
Блок действия		Вычислительное действие или последовательность вычислительных действий
Логический блок (блок условия)		Проверка условий, в логическом операторе указывается оператор сравнения или условный оператор
Предопределенный процесс		Вызов подпрограммы
Данные (ввод-вывод)		Ввод/вывод данных, например, из файла/в файл, на принтер и др.
Граница цикла		Символ состоит из двух частей – соответственно начало и конец цикла. Операции, выполняемые внутри цикла, размещаются между ними. Условия цикла и приращения записываются внутри символа начала или конца
Внутристраничный соединитель		Обрыв линии потока для переноса ее в другое место одной страницы. Соответствующие соединительные символы должны иметь одинаковое, уникальное для пары обозначение.
Межстраничный соединитель		Обрыв линии потока для переноса ее на следующую страницу.
Комментарий		В комментарии указываются пояснения и дополнительная информация.

Блоки в блок-схемах соединяются линиями потока. Линии потока используются для обозначения порядка выполнения действий и изображаются только вертикально или горизонтально. Направления сверху вниз и слева направо — основные, поэтому могут не обозначаться стрелками. В остальных случаях направление линий потока указывается стрелками. Линии потока рекомендуется выполнять в два раза тоньше линий обводки блоков.

При использовании программного способа описания — алгоритм записывается на одном из языков программирования. Алгоритм, записанный на языке программирования, называется программой. В этом случае алгоритм представляется в виде последовательности операторов языка.

По структуре связей между действиями алгоритмы классифицируют на линейные, разветвляющиеся и циклические. Эти определения относятся чаще к отдельным ветвям (частям) алгоритма, в то время , как общий алгоритм имеет более сложную комбинированную структуру.

Алгоритм линейной структуры — алгоритм, символы которого изображены на схеме в той последовательности, в которой должны быть выполнены предписываемые ими действия.

На рис.1.1 представлен пример линейного алгоритма для задачи нахождения значения $y = \dfrac{24{,}36a - 2{,}1b^2}{2{,}3 - c}$ при $a = 0{,}241$, $b = 1{,}907$, $c = 0{,}2927$

.

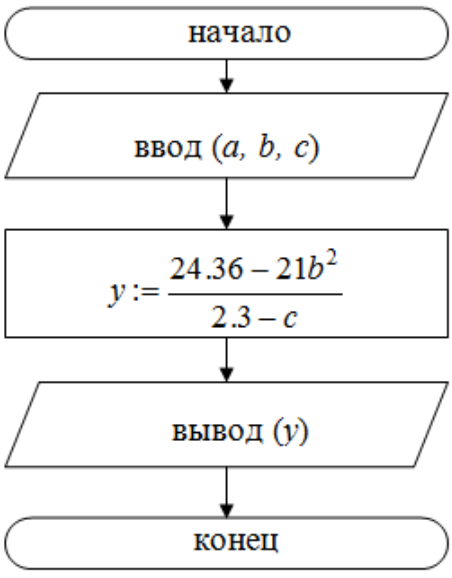


Рис. 1.1. Блок-схема алгоритма линейной структуры

Разветвленный алгоритм — алгоритм, в котором предусмотрено разветвление последовательности действий по результату проверки некоторого условия.

Возможные структуры алгоритма разветвления показаны на рис. 1.2 и рис. 1.3.

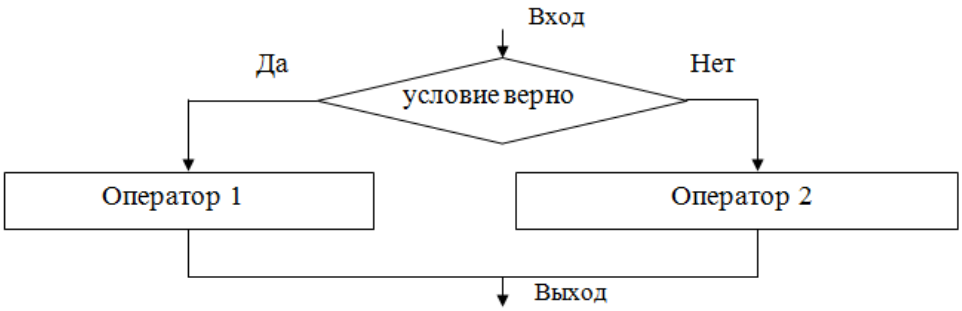


Рис. 1.2. Структура Если-То-Иначе

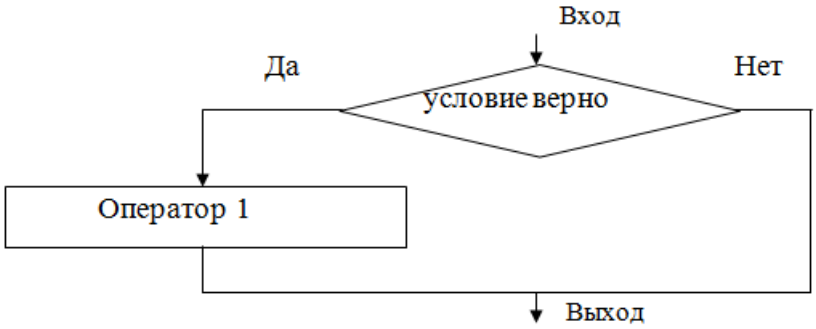


Рис. 1.3. Структура Если-То

На рис. 1.4 приведен пример разветвляющегося алгоритма для задачи вычисления значения функции

$$y=\begin{cases}\dfrac{1}{\pi}\ln\left(a^2+\sin\pi x\right), & x\le 0{,}75\\ a\sin\left(\pi\dfrac{\cos\pi x}{2}\right)+e^x, & x>0{,}75\end{cases}$$

при $a=1{,}51$, $x=0{,}39tg\dfrac{a}{x}$, $\pi=3{,}1416$.

При решении многих задач предусматривается многократное выполнение определенных последовательностей действий над исходными данными и промежуточными результатами. Эти последовательности называют циклами. Алгоритм, содержащий цикл, называется циклическим.

Переменная, изменяющаяся в цикле, называется параметром цикла. В одном цикле может быть несколько параметров.

Для организации цикла необходимо выполнить следующие действия:

1) задать перед циклом начальное значение параметра цикла;
2) изменять параметр цикла перед каждым новым повторением цикла;
3) проверять условие выхода из цикла и переходить к его началу, если он не закончен, или выходить из него по окончании.

Последние две функции выполняются многократно.

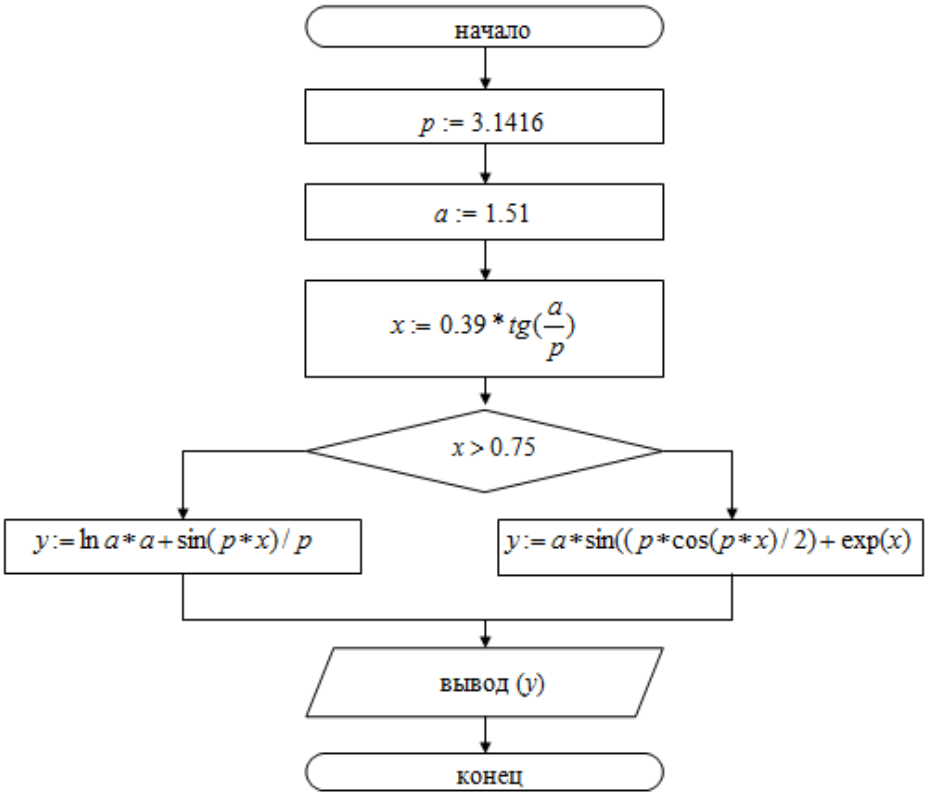


Рис. 1.4. Блок-схема разветвляющегося алгоритма

По количеству повторений цикла, циклические алгоритмы делятся на:

– циклы с заранее известным числом повторений;
– циклы с заранее неизвестным числом повторений (итерационные).

Циклы могут быть реализованы с предусловием и с постусловием.

Возможны случаи, когда внутри тела одного цикла необходимо повторять некоторую последовательность операторов, т.е. организовать внутренний цикл. Такая структура получила название цикл в цикле или вложенные циклы. Цикл, охватывающий другие циклы, называется внешним, а остальные циклы — внутренними. Правила организации как внешнего, так и внутренних циклов аналогичны правилам организации простого цикла. При одном значении параметра внешнего цикла параметр внутреннего последовательно принимает все возможные значения. При организации вложенных циклов необходимо следить за тем, чтобы область действия внутреннего цикла не выходила за область действия внешнего цикла.

Алгоритмы решения сложных задач могут включать все перечисленные структуры, которые используются для реализации отдельных участков общего алгоритма. Для решения любой нетривиальной задачи существует несколько алгоритмов, приводящих к получению результата. Из возможных алгоритмов следует выбирать

наилучший по некоторому критерию. Чаще всего в качестве критерия выбирается либо оценка точности решения задачи, либо затраты времени на ее решение, либо некоторый интегральный критерий, включающий оценки точности и затраты времени.

2. БАЗОВЫЕ КОНСТРУКЦИИ И ОПЕРАТОРЫ ЯЗЫКА С++

2.1. Структура программы

Программа на **С++** состоит из одной или нескольких функций. В программах на С++ не допускается вложенность функций. Любая программа на С должна содержать главную функцию с именем main, с которой начинается выполнение программы. Вызов всех остальных функций прямо или косвенно инициируется главной функцией.

На рис. 2.1 приведена графическая иллюстрация функции `main ()`.

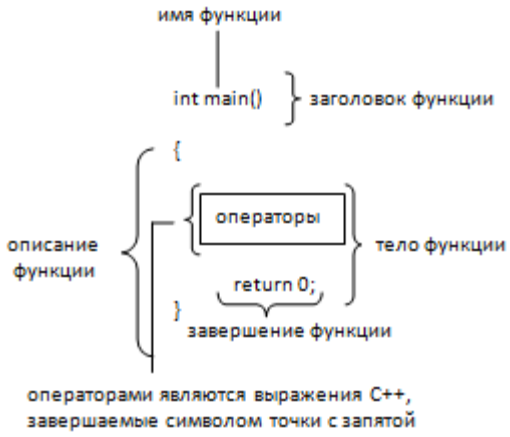


Рис. 2.1. Функция main

2.2. Данные в языке С++

Для решения задачи в программе, как правило, выполняется обработка каких-либо данных. Данные могут быть самых различных типов: целые и вещественные числа, символы, строки и др.

2.2.1. Типы данных в языке С++

В языке С++ определены пять основных типов данных: char – символьные, int – целые, float – с плавающей точкой, double – двойной точности, void – без значения (бестиповый). На базе этих типов формируются другие. Данные типа char всегда занимают один байт. Размер и диапазон представления остальных типов определяется

конкретной системой программирования, операционной системой и процессором.

В табл. 2.1 приведены основные типы данных языка C++ с указанием минимально допустимого диапазона значений.

Таблица 2.1

Типы данных языка C++

Типы данных	Размер в байтах	Минимально допустимый диапазон значений	Комментарий
Целочисленные значения			
char	1	-128…127	
unsigned char	1	0…255	
signed char	1	-128…127	
int	2 или 4	-32768…32767	
unsigned int	2 или 4	0…65535	
signed int	2 или 4	-32768…32767	
short int	2	-32768…32767	
unsigned short int	2	0…65535	
signed short int	2	-32768…32767	
long int	4	-2147483648…2147483647	
long long int	8	$-(2^{63}-1)\ldots(2^{63}-1)$	
signed long int	4	-2147483648…2147483647	
unsigned long int	4	0…4294967295	
unsigned long long int	8	$0\ldots2^{64}-1$	
Вещественные значения			
float	4	типичным является диапазон 3.4E-38 ÷ 3.4E+38	точность мантиссы не менее 6 значащих цифр
double	8	типичным является диапазон 1.7E-308…1.7E+308	точность мантиссы не менее 10 значащих цифр
long double	10	типичным является диапазон 3.4E-4932…1.1E+4932	точность мантиссы не менее 10 значащих цифр
Логические значения			
bool	1	true, false	

Как видно из таблицы, базовые типы могут быть расширены с помощью спецификаторов (модификаторов) signed, unsigned, long, short.

2.2.2. Переменные языка C++

Переменная – поименованный участок памяти, в котором хранится значение. Имя (идентификатор) в языке C++ – совокупность букв, цифр и символа подчеркивания, начинающаяся с буквы или символа подчеркивания. В C++ строчные и прописные буквы считаются разными (т.е. abc и Abc – разные переменные).

Чтобы переменную можно было использовать в программе, она должна быть предварительно объявлена. Данные в языке C++ можно объявлять в любом месте программы, но обычно их описывают в начале функции. В объявлении указывается тип переменной, и, при необходимости начальное значение, например:

```
int A;
int B,C,D;
int a=0;
double p=3.14159;
```

Вещественные числа можно выводить на консоль в естественном и в экспоненциальном форматах mE±p, где m – мантисса (целое или дробное число с десятичной точкой), p – порядок (целое число).

Например,
$-6.42E+2 = -6.42 \cdot 10^2 = -642$
$-3.2E-6 = -3.2 \cdot 10^{-6} = -0.0000032$

В C++ могут обрабатываться структурированные типы данных: массивы, строки, файлы и др.

Массив – совокупность данных одного и того же типа. Число элементов массива фиксируется при описании массива и в процессе выполнения программы не изменяется. Для доступа к элементу необходимо указать имя массива и его номер в квадратных скобках. Описание одномерного массива имеет вид:

тип имя_массива [n];

где *n* – количество элементов в массиве; элементы в массиве нумеруются с нуля, таким образом, элементы в массиве нумеруются от *0* до *n-1*.

Например: double a[25];

Описан массив a из 25 вещественных чисел (типа double), элементы нумеруются от 0 до 24 (a[0]…a[24]).

Двумерный массив (матрицу) можно объявить так:

тип имя_массива [n][m];

где *n* – количество строк в матрице(строки нумеруются от *0* до *n-1*), *m* – количество столбцов (столбцы нумеруются от *0* до *m-1*).

Например: int h[10][15];

Описана матрица h, состоящая из 10 строк и 15 столбцов (строки нумеруются от 0 до 9, столбцы от 0 до 14).

Для обращения к элементу матрицы необходимо указать ее имя, и в квадратных скобках номер строки, а затем в квадратных скобках – номер столбца.

Например, h[2][4] – элемент матрицы h, находящийся в третьей строке и пятом столбце.

В C++ можно описать многомерные массивы, которые можно объявить с помощью оператора следующей структуры:

тип имя_массива [n1][n2]…[nk];

Строка – последовательность символов. Если в выражении встречается одиночный символ, он должен быть заключен в одинарные кавычки. При использовании в выражениях строка заключается в двойные кавычки: cout << "Welcome to C \n";

2.2.3. Константы в языке C++

Константы не изменяют своего значения в процессе выполнения программы.

Например:
A=A+20; //20-константа
B=25; //25-константа

Если при объявлении переменной используется модификатор const, то она не может изменять свое значение, т.е. превращается в переменную-константу.

const double pi=3.141592653589793

По умолчанию (при отсутствии описания типа) константа будет принадлежать к типу наименьшего возможного размера. Однако, используя суффикс (символ после значения константы) можно явно указать тип. Если после вещественного числа в экспоненциальной форме присутствует символ F, то константа принадлежит к типу float, а если символ L – то к типу long double. Для целых чисел суффикс U обозначает unsigned, L – long.

Например:
pi=3.141592653589793L //тип long double
A=678L //тип long

2.3. Ввод-вывод в языке C++

Операторы ввода обеспечивают программу исходными данными, необходимыми для решения задачи, а операторы вывода осуществляют вывод результатов решения.

2.3.1. Вывод данных с помощью функции printf

Обращение к функции printf имеет следующий вид:

printf (s1, s2);

Здесь *s1* – строка вывода, *s2* – список выводимых переменных.

В строке вывода указывается строка преобразования следующего вида (табл. 2.2, табл. 2.3):

%[флаг][ширина][.точность][модификатор]тип

Таблица 2.2

Символы управления форматированием

Параметр	Назначение
-	Выравнивание числа влево. Правая сторона дополняется пробелами. По умолчанию выравнивание вправо.
+	Перед числом выводится знак «+» или «-»
Пробел	Перед положительным числом выводится пробел, перед отрицательным – «-»
#	Выводится код системы счисления: 0 – перед восьмеричным числом, 0x (0X) перед шестнадцатеричным числом.
Ширина	
n	Ширина поля вывода. Если n позиций недостаточно, то поле вывода расширяется до минимально необходимого. Незаполненные позиции заполняются пробелами.
0n	То же, что и n, но незаполненные позиции заполняются нулями.
Точность	
ничего	Точность по умолчанию
n	Для типов e, E, f выводить n знаков после десятичной точки
Модификатор	
h	Для d, i, o, u, x, X тип short int
l	Для d, i, o, u, x, X тип long int
Тип	
c	Символ
d	Десятичное целое число со знаком

i	Десятичное целое число со знаком
o	Восьмеричное число без знака
u	Десятичное int unsigned
x, X	Шестнадцатеричное число, при x используются символы a-f (строчные буквы), при X – A-F (прописные буквы).
f, F	Значение со знаком вида [-]dddd.dddd (естественный формат)
e	Значение со знаком вида [-]d.ddddе[+|-]ddd (экспоненциальный формат)
E	Значение со знаком вида [-]d.ddddE[+|-]ddd (экспоненциальный формат)
g	Использует более короткий из форматов %e или %f
G	Использует более короткий из форматов %E или %F
s	Строка символов

Таблица 2.3

Некоторые специальные символы

Символ	Назначение
\b	Сдвиг текущей позиции влево
\n	Перевод строки
\r	Перевод в начало строки, не переходя на новую строку
\t	Горизонтальная табуляция
\’	Символ одинарной кавычки
\’’	Символ двойной кавычки
\?	Символ ?

2.3.2. Ввод данных с помощью функции scanf

Обращение к функции *scanf* имеет следующий вид:

scanf(s1, s2);

Здесь *s1* – список форматов вывода; *s2* – список адресов вводимых переменных.

Все переменные должны передаваться посредством своих адресов, например, с помощью символа &.

Формат вывода представляется в следующем виде (табл. 2.4):

%mun

Таблица 2.4

Спецификаторы формата функции scanf()

Код	Формат
%c	Читает один символ

%d	Читает десятичное целое число
%i	Читает целое число в любом формате (десятичное, восьмеричное или шестнадцатеричное)
%u	Читает десятичное целое число типа short int
%e	Читает число с плавающей точкой (и в экспоненциальной форме)
%E	Аналогично коду %e
%f	Читает число с плавающей точкой
%lf	Читает десятичное число с плавающей точкой типа double
%F	Аналогично коду %f (для стандарта C99)
%g	Читает число с плавающей точкой.
%G	Аналогично коду %g
%o	Читает восьмеричное число
%x	Читает шестнадцатеричное число
%X	Аналогично коду %x
%s	Читает строку
%p	Читает указатель
%n	Принимает целое значение, равное количеству прочитанных до сих пор символов
%[ ]	Просматривает набор символов
%%	Читает знак процента

Например:

```
//пример использования операторов scanf и printf
#include <iostream> // библиотека ввода-вывода
int main()
{
int a;
double b;
// прочесть два числа
scanf ("%d %lf",&a,&b);
// вывести значения переменных
printf ("\n a=%i \t b=%5.2f \n",a,b);
}
```

Значения переменных a и b можно ввести на одной строке, отделяя их пробелом и нажать клавишу Enter, либо каждое число с новой строки, нажимая после набора каждого числа клавишу Enter.

2.3.3. Ввод/вывод с помощью cin и cout

В C++ существуют стандартные потоки для ввода информации с клавиатуры, вывода данных на экран, а также для вывода в случае

возникновения ошибки. В общем случае каждый из перечисленных потоков может быть представлен как некоторый виртуальный файл (байт-поток), закрепленный за определенным физическим устройством.

В C++ стандартный поток ввода связан с константой cin, а поток вывода - с константой cout (для использования этих констант подключается заголовочный файл iostream).

Для вывода информации в стандартный поток используется формат:

cout « выражение;

где *выражение* может быть представлено переменной или некоторой символьной строкой.

Для консольного ввода данных используют формат записи:

cin » переменная;

При этом *переменная* служит приемником вводимого значения:

Например:

```
//пример использования операторов cin и cout
#include <iostream>
using namespace std;
int main()
{
int i;
double f;
// прочесть два числа из cin
cin >> i >> f;
// вывести в cout
cout << "i=" << i << " f=" << f << endl;
}
```

Обозначить новую строку при выводе в C++ можно посредством обозначения \n, принятого в языке C, или использовать манипулятор endl, как показано в предыдущем примере.

2.4. Операция присваивания и выражения

Составляющими элементами выражения являются *данные* и *операторы*. Данные могут быть представлены переменными, константами или значениями, возвращаемыми функциями.

Операции представляют собой некоторое действие, выполняемое над одним (унарная) или несколькими операндами (бинарная операция), результатом которого является возвращаемое значение.

Для эффективного использования возвращаемого операциями значения предназначен *оператор присваивания* (=).

2.4.1. Оператор присваивания

В операторе присваивания слева всегда стоит имя переменной, а справа – значение, например:

<переменная>=<значение>;

где <переменная> – имя переменной или элемента массива, <значение> – выражение, переменная, константа или функция.

В результате выполнения оператора переменной присваивается значение (константа, переменная, результат вычисления выражения). Если в операции присваивания встречаются переменные разных типов, происходит преобразование типов. В операции присваивания значение в правой части преобразуется к типу переменной левой части. Следует учитывать, что при этом можно потерять информацию или получить другое значение.

2.4.2. Арифметические операции

Представим основные арифметические операции языка в виде таблицы (табл. 2.5).

Таблица 2.5

Основные арифметические операции

Операция	Действие	Тип операнда	Тип результата
+	сложение	целый вещественный	целый вещественный
-	вычитание, унарный минус	целый вещественный	целый вещественный
*	умножение	целый вещественный	целый вещественный
/	деление	вещественный	вещественный
/	целочисленное деление	целый	целый
%	остаток от деления	целый	целый
--	декремент (уменьшение на 1)	целый	целый
++	инкремент (увеличение на 1)	целый	целый

Кроме указанных, в C++ имеется еще ряд арифметических операций.

2.4.3. Логические операции и операции отношения

Логические операции выполняются над логическими значениями ИСТИНА (true) и ЛОЖЬ (false). В языке Си ложью является 0, а истина – любое значение, отличное от нуля. В С++ появился тип bool.

Результатами операций отношения (<, <=, >, >=, ==, !=) или логической операции является ИСТИНА (true, 1) или ЛОЖЬ (false, 0).

В Си определены следующие логические операции ИЛИ (||), И(&&), НЕТ (!).

2.4.4. Стандартные математические функции

Некоторые стандартные математические функции показаны в табл. 2.6.

Таблица 2.6

Основные математические функции

Обозначение	Действие
abs(x)	Модуль целого числа
fabs(x)	Модуль вещественного числа
sin(x)	Функция синус
cos(x)	Функция косинус
tan(x)	Функция тангенс
atan(x)	Арктангенс $\left[-\frac{\pi}{2};\frac{\pi}{2}\right]$
atan2(y,x)	Арктангенс $\frac{y}{x}$ в диапазоне $[-\pi,\pi]$
sinh(x)	Гиперболический синус x
cosh(x)	Гиперболический косинус x
tanh(x)	Гиперболический тангенс x
exp(x)	e^x
log(x)	Функция натурального логарифма ln(x), x>0
log10(x)	Функция десятичного логарифма log10(x), x>0
pow(x,y)	x^y. Ошибка области определения, если x=0, y≤0 или x<0 и y – не целое
sqrt(x)	$\sqrt{x}$, x≥0

2.4.5. Пробелы и круглые скобки

Для повышения удобочитаемости программы при записи выражений можно использовать пробелы и символы табуляции.

Лишние скобки, если они не изменяют приоритет операций, не приводят к ошибке и не замедляют вычисление выражения. Дополнительные скобки часто используют для прояснения порядка вычислений. В следующем примере 2-я строка читается значительно легче:

```
x = y/3-34*temp+127;
x = (y/3) - (34*temp) + 127;
```

2.4.6. Приоритет операций

Операции в выражении выполняются в соответствии с приоритетом, в первую очередь выполняются операции с наивысшим приоритетом. Если операции имеют одинаковый приоритет, то они выполняются последовательно слева направо. Изменить порядок выполнения операций можно с помощью скобок.

Перечислим операции в порядке убывания приоритета их выполнения:

высший приоритет — унарные операции (постфиксный инкремент, постфиксный декремент, префиксный декремент, преобразование типа, унарный минус, унарный плюс);
— умножение, деление, остаток от деления;
— сложение, вычитание;
— операции сравнения;
— логическое И, логическое ИЛИ;

низший приоритет — умножение с присваиванием, деление с присваиванием, сложение с присваиванием, вычитание с присваиванием.

2.5. Условный оператор if

Для программной реализации структуры Если-То, представленной на рис.3, используется оператор if. Условие (expression) в операторе if заключается в круглые скобки.

Если результат выполнения условия будет истинным, то возможно выполнить один оператор:

```
if (expression)
   statement;   // тело if из одного оператора
```

или несколько операторов (для этого следует использовать фигурные скобки):

```
if (expression)
   {
   statement;
   statement;        // тело if из нескольких операторов
   statement;
   }            // точка с запятой не нужна
```

Структура Если-То-Иначе (рис. 2) реализуется конструкцией if–else. Общая форма записи конструкции if–else:

```
if (expression)
   statement1;   // тело if из одного оператора
else
   statement2;   // тело elseиз одного оператора
```

Если условие expression выполняется, то будет выполняться фрагмент программы statement1, в противном случае будет выполняться statement2.

Каждое из утверждений может быть множественным. В таком случае применяются фигурные скобки:

```
if (expression)
   {
   statement;
   statement;        // тело if из нескольких операторов
   }
else
   {
   statement;
   statement;        // тело else из нескольких операторов
   }
```

Приведем пример на использование оператора if, решим задачу:

Составим блок-схему (рис. 2.2) и программу решения квадратного уравнения $ax^2 + bx + c = 0.$

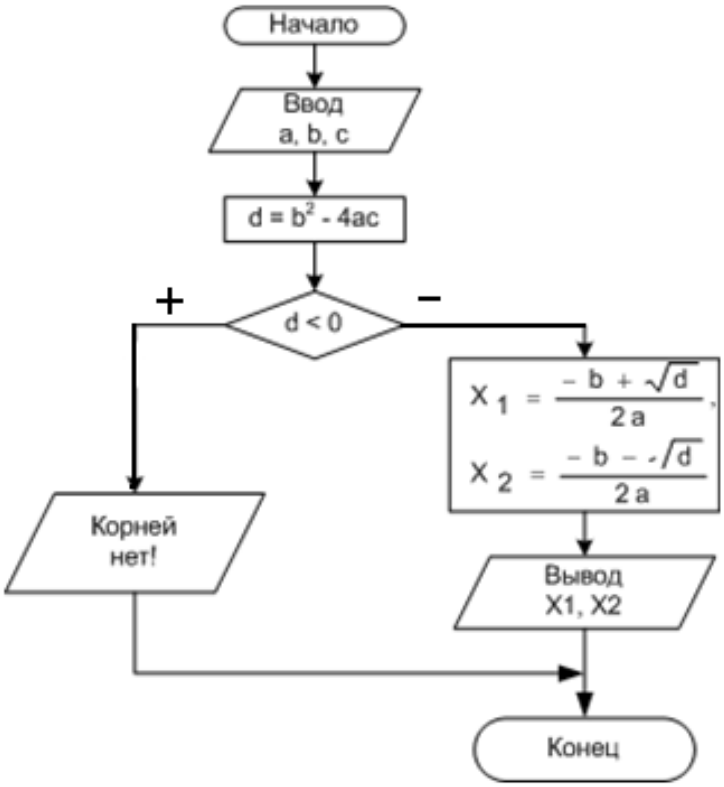


Рис. 2.2. Блок-схема решения квадратного уравнения

```
//листинг программы решения квадратного уравнения
#include <iostream>
#include <math.h>
#include <clocale>
using namespace std;
int main()
{
setlocale(LC_CTYPE,"Russian");
float a,b,c,d,x1,x2;
cout << "\nВведите a,b,c \n\n";
cin >> a >> b >> c;
cout << "\na="  << a << "\t b="  << b << "\t c="  <<c;
d=b*b-4*a*c;
if (d<0) cout << "\n\nРешения нет \n\n";
else
{
x1=(-b+sqrt(d))/2/a;
x2=(-b-sqrt(d))/2/a;
cout << "\n\nКорни уравнения: x1=" << x1 << "\t x2="<< x2<< "\n\n";
}
}
```

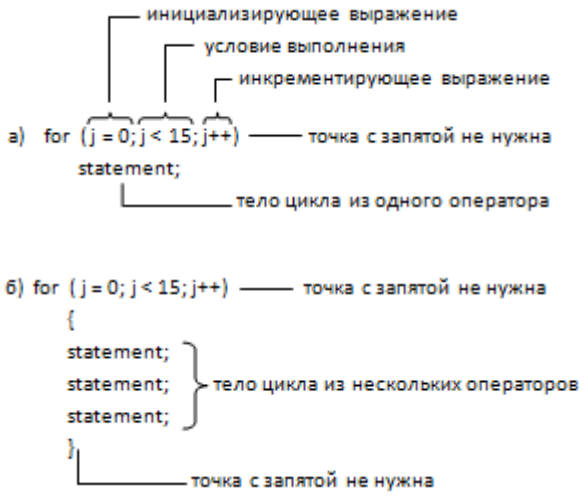
2.6. Операторы цикла

Цикл задает многократное прохождение по одному и тому же коду программы (итерации). Он имеет точку вхождения, проверочное условие и (необязательно) точку выхода. Цикл, не имеющий точки выхода, называется *бесконечным.* Для бесконечного цикла проверочное условие всегда принимает истинное значение.

В C++ имеются 3 оператора цикла: for, while и do. Проверка условия может осуществляться перед выполнением (циклы for, while) или после окончания (do) тела цикла. Когда значение выражения, задающего условие, становится ложным, выполнение цикла прекращается, а управление передается оператору, следующему непосредственно за циклом.

2.6.1. Оператор цикла for

Покажем работу оператора for на примерах:



Все три выражения (инициализирующее выражение, условие выполнения, инкрементирующее выражение) не обязательно должны присутствовать в конструкции, однако синтаксис не допускает пропуска символа точка с запятой (;).

Рассмотрим пример программы распечатки на консоль простых чисел из диапазона от 2 до N, где N – число, вводимое пользователем с клавиатуры.

Как известно, простое число – это целое положительное число больше единицы, которое не делится без остатка ни на одно другое целое положительное число, кроме единицы и самого себя. Единица не считается простым числом.

Возможный программный код решения примера:

```
//пример использования оператора for
#include <iostream>
int main ()
{
        int i, j;
        int N;
   using namespace std;
        cout << "Input N ";
        cin >> N;
        for ( i = 2; i <= N; i++ )
      {
                int ok;
                ok = 1;
                for ( j = 2; j < i; j++ )
                        if ( (i%j) == 0 )
                        ok = 0;
                if ( ok )
                printf(" %3d", i);
        }
   cout << endl;
}
```

Оператор for в C++ позволяет изменять значение управляющей переменной произвольным образом. Эта переменная может быть вещественного типа, и ее значение может меняться не только по закону арифметической прогрессии, но и любым сколь угодно сложным способом.

2.6.2. Цикл while (цикл с предусловием)

Оператор while можно использовать как при программировании циклов с известным числом повторений, так и итерационных циклов.

Покажем механизм работы цикла while в виде блок-схемы (рис. 2.3):

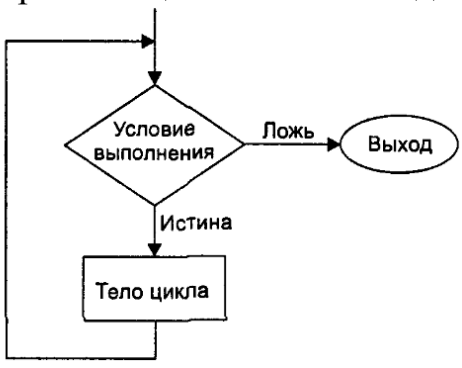


Рис. 2.3. Принцип работы оператора while

Цикл while имеет следующий формат (синтаксис) записи:

```
while (expression)
        statement;
```

Производится расчет выражения expression, заключенного в круглые скобки. Если получается истинный результат (TRUE), то выполняется утверждение statement. После выполнения этого утверждения вновь рассчитывается выражение expression. Цикл повторяется до тех пор, пока в результате расчета выражения expression не будет получено значение FALSE (ложный), после чего выполнение программы продолжается с утверждения, следующего за утверждением statement. Когда требуется выполнить группу утверждений, то она (группа) располагается в фигурных скобках:

```
while (expression)
        {
        statement1;
        statement2;
        statement3;
        ...
        }
```

Если expression представляет собой константу с истинным значением, тело цикла будет выполняться всегда и, следовательно, мы имеем дело с бесконечным оператором. Цикл также окажется бесконечным, когда условие, определенное в expression изначально, истинно и нигде далее в теле цикла не изменяется.

Например, итерационная схема вычисления корня квадратного из x имеет следующий вид: $y_n+1=0.5*(y_n +x/y_n)$.
Для ее реализации можно воспользоваться следующим циклом:

```
//пример работы оператора while
#include <iostream>
#include <clocale>
#include <math.h>
using namespace std;
int main()
{
setlocale(LC_CTYPE,"Russian");
double x,y;
cout  <<  "Введите x " ;
cin >> x;
y = x; //начальное приближение
while (fabs(x-y*y)>1e-6)
   y = 0.5*(y+x/y);
```

```
cout << "Корень квадратный из " << x <<" = "<< y << endl;
}
```

2.6.3. Цикл do-while (цикл с постусловием)

Оператор do-while (как и while) можно использовать как при программировании циклов с известным числом повторений, так и итерационных циклов. В отличие от оператора while, цикл do-while сначала выполняет тело (оператор или блок), а затем уже осуществляет проверку выражения на истинность. Такая конструкция гарантирует, что тело цикла будет обязательно выполнено хотя бы один раз.

Механизм работы цикла do-while показан на рис. 2.4.

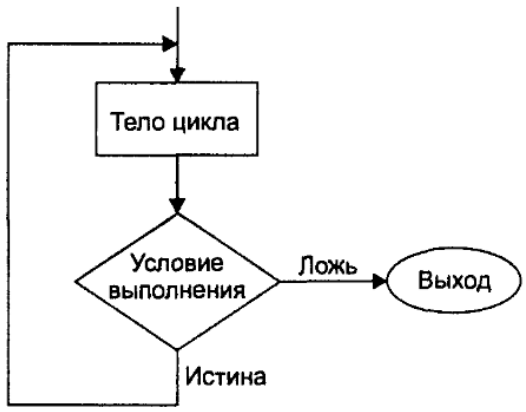


Рис. 2.4. Принцип работы оператора do-while

Оператор цикла do–while имеет следующий формат записи:

```
do
    statement;
while (expression);
```

Выполнение цикла do–while происходит следующим образом: сначала выполняется утверждение statement, затем производится проверка условия выполнения цикла expression с помощью оператора while. Если результатом проверки будет значение TRUE (истина), то выполнение цикла продолжится, и утверждение statement всякий раз будет выполняться вновь. Повторение цикла будет продолжаться до тех пор, пока в результате проверки условия выполнения цикла expression будет получаться значение TRUE. Когда в результате проверки условия будет вычислено значение FALSE (ложь), то выполнение цикла прекратится и произойдет переход к утверждению (следующему фрагменту программы), непосредственно следующему за циклом.

Таким образом, цикл do–while гарантированно выполнится хотя бы один раз.

В случае выполнения нескольких утверждений используются фигурные скобки для выделения тела цикла:

```
do {
    statement1;
    statement2;
    statement3;
    } while (expression);
```

Программа вычисления корня квадратного из x будет иметь вид:

```
//пример работы оператора do-while
#include <iostream>
#include <clocale>
#include <math.h>
using namespace std;
int main()
{
setlocale(LC_CTYPE,"Russian");
double x,y;
cout  <<  "Введите x " ;
cin >> x;
y=x; //начальное приближение
do
   y=0.5*(y+x/y);
while(fabs(x-y*y)>1e-6);
cout << "Корень квадратный из " << x <<" = "<< y << endl;
}
```

3. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ РАЗЛИЧНЫХ СТРУКТУР

3.1. Программирование алгоритмов линейной структуры

В программах линейной структуры операторы выполняются последовательно друг за другом, в порядке заданном алгоритмом. Такой порядок выполнения называется естественным. Для организации программы линейной структуры используются операторы присваивания, ввода исходных данных и вывода результатов обработки данных.

Пример 3.1. Заданы длины двух катетов в прямоугольном треугольнике. Найти длину гипотенузы, площадь треугольника и величину его углов.

Входные данные: **a, b** - длины катетов.

Выходные данные: **c** - длина гипотенузы, **S** - площадь треугольника, α, β - углы.

Блок-схема для задачи представлена на рис.3.1.

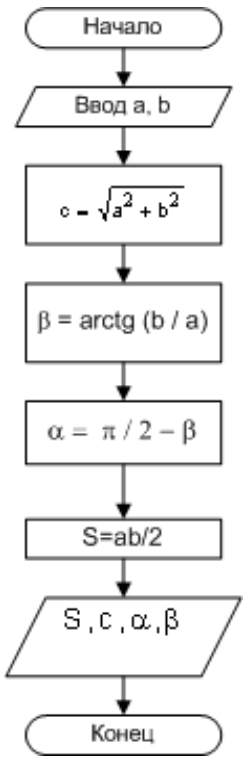


Рис. 3.1. Блок-схема для примера 3.1

Программа имеет вид:

```
#include <math.h>
#include <iostream>
#include <clocale>
int main()
{
   using namespace std;
   setlocale(LC_CTYPE,"Russian");
   const double p=3.14159;
   double a,b,c, alf,bet,s;
   cout<<"A=";
   cin>>a;
   cout<<"B=";
   cin>>b;
   s=a*b/2;
   c=pow(a*a+b*b,0.5);
   alf=atan(a/b);
   bet=p/2-alf;
   cout<<"\n A="<<a<<" B="<<b;
   printf("\n C=%5.2f \t S=%5.2f",c,s);   printf("\n alf=%3.0f град. \t
```

```
    bet=%3.0f град.\n",alf*180/p,bet*180/p);
}
```

Задания для самостоятельного выполнения.

Составить блок-схемы и программы для следующих задач:

1. Вычислить площадь круга и длину окружности по заданному радиусу, вводимому пользователем с клавиатуры.
2. Вычислить высоты треугольника со сторонами a, b, c.
3. Вычислить площадь треугольника со сторонами a, b, c.
4. Вычислить длину окружности описанной около треугольника со сторонами a, b, c.
5. Вычислить длину окружности вписанной в треугольник со сторонами a, b, c.

3.2. Программирование алгоритмов разветвляющейся структуры

Разветвления в программах возникают при необходимости выбора одного из нескольких возможных путей в решении задачи, который может зависеть от исходных данных или промежуточных результатов.

Пример 3.2. Составить блок-схему и программу, которая по введенному значению аргумента t вычисляет значение переменной M в соответствии с заданным условием:

$$M=\begin{cases}\dfrac{t^2+2}{C}, & \text{если } 2\le t\le 14\\ (t+2)^2, & \text{если } t>14\\ \ln|t|, & \text{иначе}\end{cases}$$

Блок-схема вычисления переменной представлена на рис.3.2.

Программа имеет вид:

```
#include <iostream>
#include <clocale>
#include <math.h>
int main()
{
   using namespace std;
   setlocale(LC_CTYPE,"Russian");
   float t, C, M;
```

```
    cout << " Введите значения t и C" << endl;
    cin >> t >> C;
    if (t>=2 && t<=14) M = (t*t+2)/C;
    else if (t>14) M = (t+2)*(t+2);
    else M = log(abs(t));
    cout <<"Для t = "<<t<< " и C = " << C << " значение функции M = " << M
    << endl;
}
```

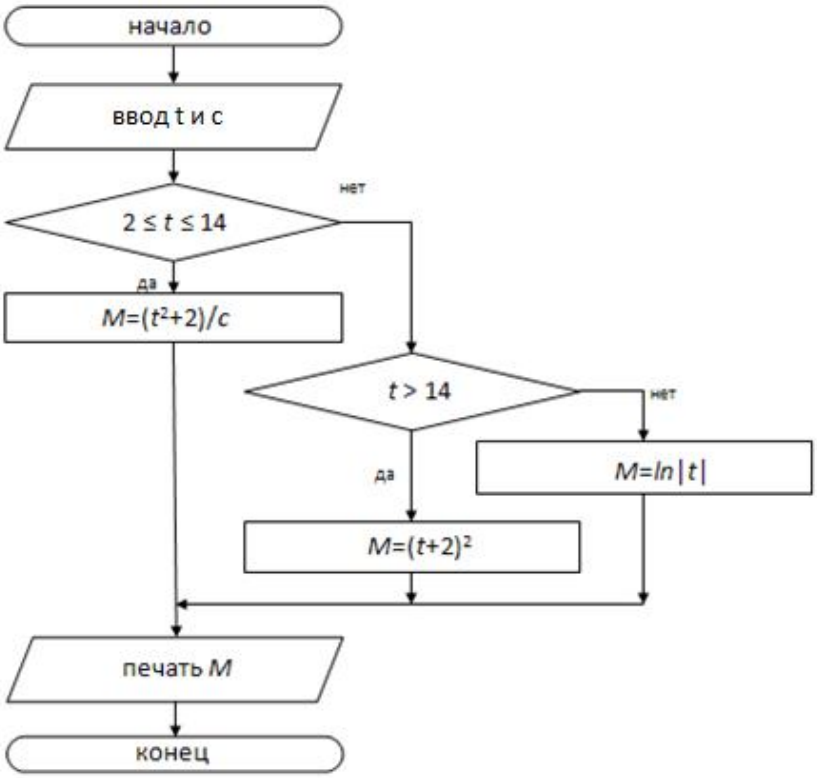


Рис. 3.2. Блок-схема для примера 3.2

Задания для самостоятельного выполнения.

1. Составить блок-схему и программу, которая по введенному значению аргумента вычисляет значение функции в соответствии с заданным условием:

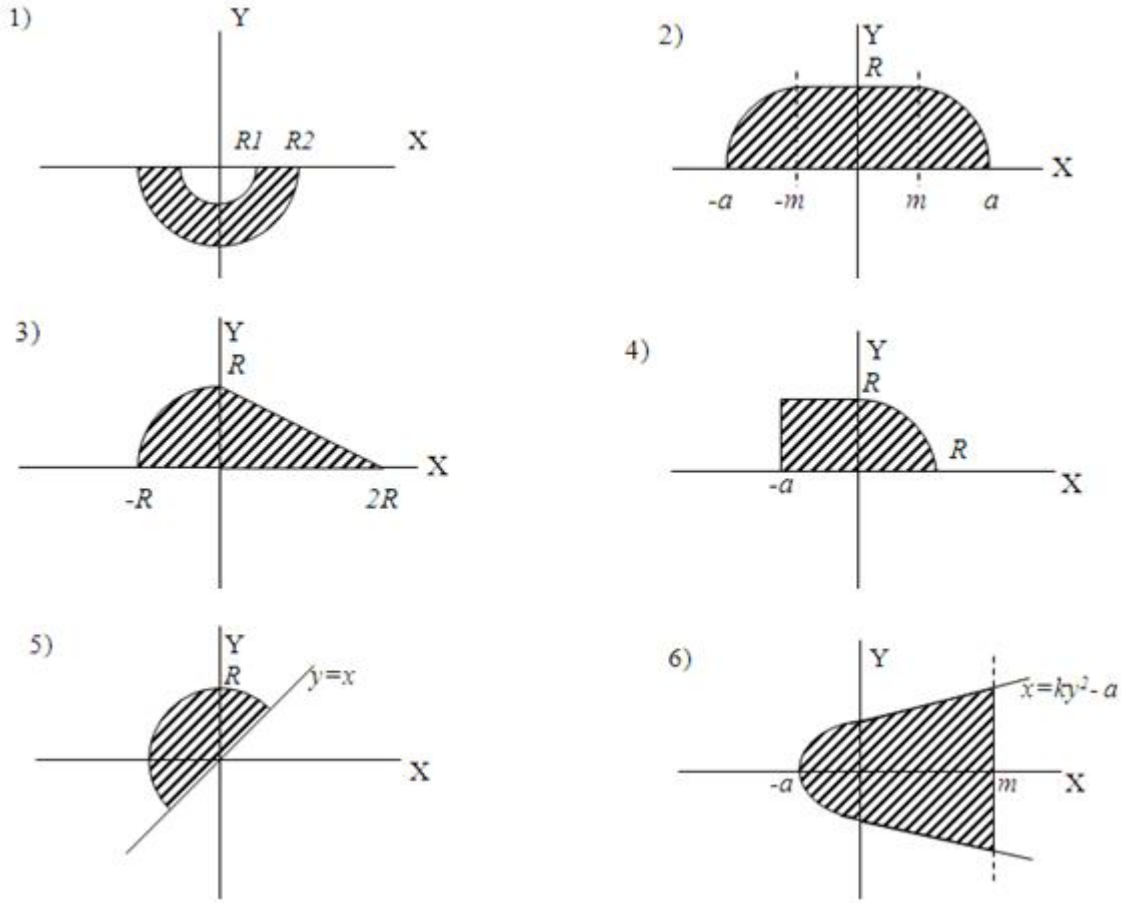
$$1)\quad Z=\begin{cases}\sqrt[5]{ax+b}, & \text{если } 0\le ab\le 7\\ \sin(ax+b), & \text{если } ab>7\\ |ax+b|, & \text{иначе}\end{cases}\qquad 2)\quad Z=\begin{cases}e^{kx+2}, & \text{если } 2\le x\le 7\\ \ln(kx-2), & \text{если } x>7\\ \cos(kx), & \text{иначе}\end{cases}$$

$$3)\quad C=\begin{cases}\sqrt[3]{x^2+y^2}, & \text{если } xy<0\\ \dfrac{x^2+y^2}{x+y}, & \text{если } 0\le xy\le 6\\ \sin(x^2+y^2), & \text{иначе}\end{cases}\qquad 4)\quad P=\begin{cases}|ax|, & \text{если } x<0\\ \dfrac{ax+b^2}{x+5}, & \text{если } 0\le x\le 7\\ \sin x, & \text{иначе}\end{cases}$$

5) $$Z=\begin{cases}\ln|xy|, & \text{если } xy<0\\ e^{xy}, & \text{если } 0\le xy\le 5\\ \lg(xy), & \text{иначе}\end{cases}$$

6) $$Y=\begin{cases}x^a, & \text{если } x<0\\ e^{ax+b}, & \text{если } 0\le x\le 2{,}5\\ \ln|ax+b|, & \text{иначе}\end{cases}$$

2. Составить блок-схему и программу для определения того, попадает ли точка с координатами (x, y) в заштрихованную область. Результат вывести в виде текстового сообщения.



3.3. Характерные приемы программирования циклических алгоритмов

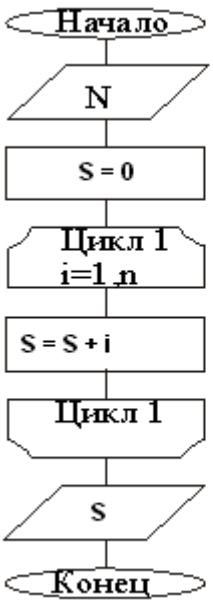
Циклическая структура программы позволяет производить многократные повторения группы операторов при изменении одного параметра цикла или нескольких параметров цикла одновременно. Решение практических задач сводится к использованию характерных приемов, облегчающих процесс программирования. Рассмотрим характерные приемы для циклических алгоритмов, наиболее часто использующиеся при решении практических задач.

3.3.1. Вычисление суммы и произведения натуральных чисел

Вычисление суммы натуральных чисел выполняется в цикле, на каждом шаге которого происходит накопление суммы путем прибавления числа к сумме предыдущих. Формула, используемая для накопления имеет вид $S_i = S_{i-1} + i$. Начальное значение суммы перед циклом, как правило, следует обнулить.

Аналогично накапливается и произведение, с той лишь разницей, что для его накопления используется формула $P_i = P_{i-1} \cdot i$, а начальное значение произведения должно быть равно единице.

Пример 3.3. Блок-схема и программа вычисления суммы N первых натуральных чисел показаны на рис. 3.3.



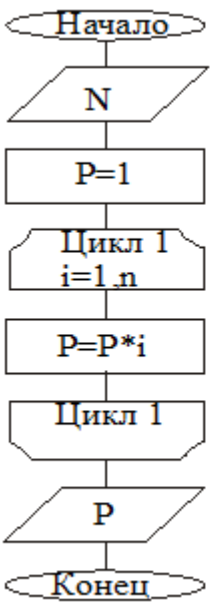

```
#include <iostream>
#include <clocale>
int main ()
{
int S =0, N;
using namespace std;
setlocale(LC_CTYPE,"Russian");
cout << "Введите целое число: ";
cin >> N;
for(int i=1; i<=N; i++)
S+= i; cout << "Сумма составит: " << S << endl;
}
```

Рис. 3.3. Вычисление суммы натуральных чисел

Задания для самостоятельного выполнения.

1. Составить блок-схему и программу вычисления суммы натуральных чисел от 10 до 20.
2. Составить блок-схему и программу вычисления среднего арифметического N первых натуральных чисел.
3. Составить блок-схему и программу вычисления суммы натуральных чисел от 3 до M.
4. Составить блок-схему и программу вычисления суммы натуральных чисел от M до N.
5. Составить блок-схему и программу вычисления среднего арифметического натуральных чисел от 5 до N.

Пример 3.4. Блок-схема и программа вычисления произведения N первых натуральных чисел (N!) показаны на рис.3.4.




```
#include <iostream>
#include <clocale>
using namespace std;
int main ()
{
int N;
unsigned long P = 1;

setlocale(LC_CTYPE,"Russian");
cout << "Введите целое число: ";
cin >> N;
for(int i=1; i<=N; i++)
P*= i; cout << "Произведение составит: " << P << endl;
}
```

Рис. 3.4. Вычисление произведения натуральных чисел

Задания для самостоятельного выполнения.

1. Составить блок-схему и программу вычисления среднего геометрического N первых натуральных чисел.
2. Составить блок-схему и программу вычисления произведения четных натуральных чисел от 1 до 50.
3. Составить блок-схему и программу вычисления произведения нечетных натуральных чисел от 20 до 60.
4. Составить блок-схему и программу вычисления произведения четных натуральных чисел от M до N.
5. Составить блок-схему и программу вычисления произведения нечетных натуральных чисел от M до N.

3.3.2. Вычисление суммы членов бесконечного ряда с заданной точностью

Для вычисления суммы членов бесконечного ряда с заданной точностью используется итерационный цикл, так как заранее не известно, при каком члене ряда будет достигнута требуемая точность. Выход из цикла осуществляется по условию достижения требуемой точности $y \le Ɛ$, где y – член ряда. Для вычисления суммы членов ряда используется рассмотренный ранее прием накопления суммы, параметром цикла при этом является номер члена ряда. Для уменьшения затрат на вычисление текущего члена ряда целесообразно использовать рекуррентную формулу

члена ряда. Начальное значение суммы не всегда равно нулю, например оно может быть принято равным значению первого члена ряда, если вычислять его нет необходимости.

Пример 3.5. Вычислите с точностью 10^{-6} значение суммы числового ряда:

$$S = x + \frac{\cos^2 x}{2} + \frac{\cos^3 x}{3} + \frac{\cos^4 x}{4} + \ldots + \frac{\cos^n x}{n} + \ldots$$

Программный код решения примера:

```
#include <iostream>
#include <clocale>
#include <math.h>
using namespace std;
int main()
{
setlocale(LC_CTYPE,"Russian");
double x, s;
int n;
cout << "Введите x " ;
cin >> x;
s = x; n = 2;
while (fabs(pow(cos(x),n)/n) > 1e-6)
   {s = s + pow(cos(x),n)/n;
    n = n+1;}
cout << "Сумма ряда = "<< s << endl;
}
```

В данной программе сумма ряда вычисляется как значение переменной s. Ее начальное значение s=x, т.к. первое слагаемое в приведенном примере не является членом числового ряда. Погрешность вычисления суммы ряда не превышает первого из отброшенных членов ряда, поэтому цикл прекращается, как только будет найден член ряда не превышающий 10^{-6}.

Задания для самостоятельного выполнения.

Разработайте блок-схему алгоритма и программу для вычисления суммы ряда:
а) с использованием оператора while;
б) с использованием оператора do…while.

1) Найти сумму ряда: $Y = x^3 + \frac{x^2}{2!} + x + \frac{x^2}{4!} + \frac{x^3}{6!} + \ldots + \frac{x^n}{(2n)!} + \ldots$, x<1 с точностью Ɛ.

2) Найти сумму ряда: $S = \frac{\cos x}{2} + \frac{\cos^2 x}{3} + \frac{\cos^3 x}{4} + ... + \frac{\cos^n x}{n+1} + ...$, с точностью 10^{-3}.
3) Найти сумму ряда: $Y = x + \sin x + \frac{\sin x^2}{2!} + \frac{\sin x^3}{3!} + ... + \frac{\sin x^n}{n!} + ...$, с точностью Ɛ.
4) Найти сумму ряда: $Y = \frac{\sin(x+1)}{2!} + \frac{\sin(x+2)}{3!} + ... + \frac{\sin(x+n)}{(n+1)!} + ...$, с точностью Ɛ.
5) Найти сумму ряда: $Y = \frac{\sin^2 x}{x} + \frac{\sin^2 x}{2} + \frac{\sin^2 x}{3} + ... + \frac{\sin^2 x}{n+1} + ...$, с точностью Ɛ.

3.3.3. Вычисление суммы и произведения элементов одномерного массива.

Пример 3.6. Блок-схема и программа нахождения количества положительных элементов массива A(10) показаны на рис. 3.5.

Вычисление количества элементов массива в соответствии с заданным условием выполняется в цикле аналогично нахождению суммы. На каждом шаге цикла происходит увеличение количества на единицу. Начальное значение количества перед циклом следует обнулить.

```
#include <iostream>
#include <clocale>
using namespace std;
int main()
{
setlocale(LC_CTYPE,"Russian");
double k = 0;
int i;
double A[10];
for (i = 0 ;  i < 10 ; i++)
 {
 cout  <<  "Введите " << i << " элемент массива: ";
 cin >> A[i];
 if (A[i]>0) k=k+1;
 };
cout << "Количество положительных элементов: " << k << endl;
}
```

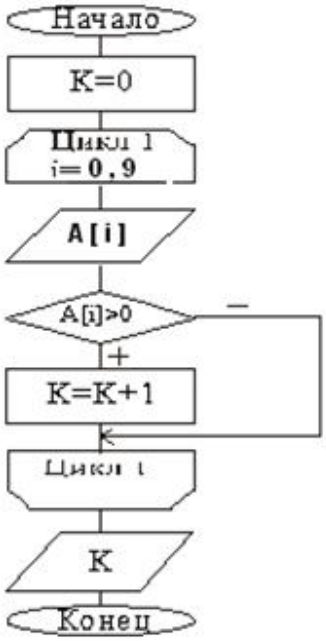


Рис. 3.5. Вычисление количества элементов массива

Задания для самостоятельного выполнения.

1) Найти и вывести на печать сумму и количество элементов массива A(10), кратных 5.

2) Найти среднее арифметическое элементов массива Y(N), удовлетворяющих условию $2 \le Y_i \le 3{,}5$.
3) Найти сумму и среднее арифметическое элементов массива X(N), кратных 3.
4) Найти сумму и количество элементов массива A(N), удовлетворяющих условию $0 \le A_i \le 7{,}2$.
5) Найти и вывести на печать среднее арифметическое элементов массива B(15), кратных двум.

Пример 3.7. Блок-схема м программа вычисления произведения элементов массива A(10) представлены на рис. 3.6.

```
#include <iostream>
#include <clocale>
using namespace std;
int main()
{
setlocale(LC_CTYPE,"Russian");
double P = 1;
int i;
double A[10];
for(i=0;i<10;i++)
 {
 cout << "Введите " << i << " элемент массива: ";
 cin >> A[i];
 P*=A[i];
 };
cout << "Произведение равно: " << P << endl;
}
```

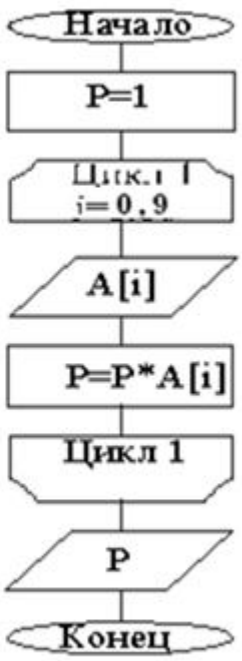


Рис. 3.6. Вычисление произведения элементов одномерного массива

Задания для самостоятельного выполнения.

1) Найти произведение и индексы элементов массива C(M), больших пяти.
2) Найти и вывести на печать среднее геометрическое элементов массива A(X), кратных 6.
3) Найти и вывести на печать произведение и индексы элементов массива X(8), кратных 3 и больших7.
4) Вывести на печать произведение и индексы элементов массива X(N), больших 0.
5) Найти и вывести на печать среднее геометрическое положительных элементов массива A(N), кратных 3.

3.3.4. Организация цикла с несколькими одновременно изменяющимися параметрами

На практике часто встречаются задачи, в которых одновременно изменяются несколько параметров. Цикл с несколькими одновременно изменяющимися параметрами организуется по схеме, аналогичной схеме организации цикла с одним параметром. В операторе цикла осуществляется контроль за одним параметром. Для остальных параметров необходимо перед циклом задавать начальные значения, а внутри цикла вычислять текущие значения.

Пример 3.8. Вычислить $y_x = \frac{a + \sin(x)}{\cos(2x)} + 2a$, если x изменяется от x_1 до x_2 с шагом h, а изменяется от a_1 до a_2.

Блок-схема и программа решения задачи показаны на рис. 3.7.

```
#include <iostream>
#include <clocale>
#include <math.h>
using namespace std;
int main()
{
setlocale(LC_CTYPE,"Russian");
double x,a,x1,x2,h,a1,a2,ha,y,k;
cout <<"Введите x1,x2,h,a1,a2: ";
cin >> x1>>x2>>h>>a1>>a2;
k = (x2-x1)/h+1;
ha = (a2-a1)/(k-1);
a = a1;
x = x1;
while (x<=x2)
 {
 y=(a+sin(x))/cos(2*x)+2*a;
 cout<<"При x="<<x<<"; a="<<a<<"; y="<<y<<endl;
 x=x+h;a=a+ha;
 }
}
```

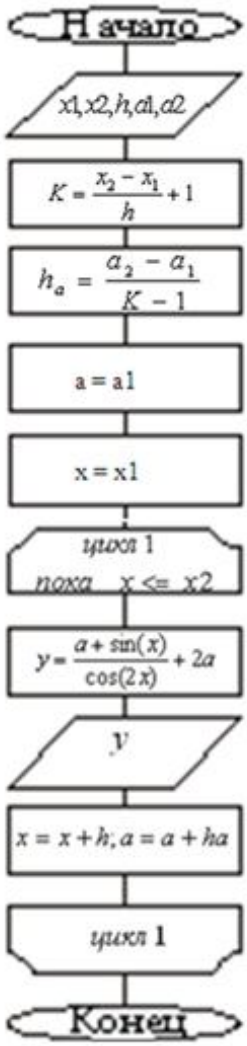


Рис. 3.7. Цикл с несколькими одновременно изменяющимися параметрами

В приведенном примере для переменной x даны начальное значение, конечное значение и шаг изменения, что позволяет определить количество вычислений переменной y: $K = \frac{x_2 - x_1}{h} + 1$, а затем шаг изменения переменной a: $h_a = \frac{a_2 - a_1}{K - 1}$.

Задания для самостоятельного выполнения.

1) Определить значения: $A = \frac{ax + cx^3}{\sin(x) + 2x^5}$, где a – изменяется от a_0 до a_1 с шагом h; c – от c_0 до c_1; x – от 0,1 с шагом 0,2.
2) Определить значения: $Y = \frac{\sqrt{ax^2 + b} - \sin a}{\cos^2 ax + 2}$, где x – изменяется от x_0 до x_1 с шагом h; a – от a_0 до 5; b – от 0 с шагом 0,1.
3) Определить значения: $D = \frac{x^2 + \sqrt{ax + b}}{\sin ax + 2}$, где x – изменяется от 0 до x_1 с шагом h; a – от 0 до 2; b – от b_0 до b_1.
4) Определить значения: $Y = \frac{\cos(ax + b) + \sqrt[3]{2}}{a + bx^3}$, где a – изменяется от a_0 до a_1 с шагом h; b – от b_0 до 5; x – от 2 до x_1.
5) Определить значения: $M = \frac{P + 2q^2}{|tgP| + 4}$, где P – изменяется от P_0 до P_1 с шагом h; q – изменяется от q_0 до q_1.

3.3.5. Запоминание результата

По завершению цикла в памяти может сохраняться либо последнее значение результата, либо все значения результатов на каждом шаге цикла. Для запоминания всех значений результатов необходимо:

1) выделить для хранения результатов требуемое число ячеек памяти (массив);
2) вычислить результат как переменную с индексом.

Пример 3.9. Дан массив A(15), записать элементы с чётными индексами в массив B, с нечётными в массив C. Блок-схема и программа для данного примера представлены на рис. 3.8.

Задания для самостоятельного выполнения.

1) Дан массив A(N). Переписать элементы массива A, кратные трём в массив B, а остальные – в массив C.
2) Дан массив X(N). Переписать элементы массива, удовлетворяющие условию $0 \le X_i \le 2{,}7$ в массив Y, а остальные – в массив Z.

3) Дан массив A(N). Переписать отрицательные элементы массива A, кратные пяти в массив B, а остальные – в массив C.
4) Дан массив X(N). Переписать положительные элементы массива A, удовлетворяющие условию $5{,}2 \le X_i \le 7{,}4$ в массив Y, а остальные – в массив Z.
5) Дан массив A(N). Переписать отрицательные элементы массива A, удовлетворяющие условию
$2 \le |A_i| \le 4{,}5$ в массив X, а остальные – в массив Y.

```
#include <iostream>
#include <clocale>

int main()
{
setlocale(LC_CTYPE,"Russian");
using namespace std;
int i,K=-1,M=-1;
double A[15],B[15],C[15];
for(i=0;i<15;i++)
  {
  cout  <<  "Введите " << i << " элемент массива: ";
  cin >> A[i];
  };
for(i=0;i<15;i++)
  if (i%2 == 0)
      {K = K+1; B[K] = A[i];}
    else
       {M = M+1; C[M] = A[i];}
for(i=0;i<=K;i++)
  cout  <<  "B[" << i << "] = " << B[i]<<endl;
for(i=0;i<=M;i++)
  cout  <<  "C[" << i << "] = " << C[i]<<endl;
}
```

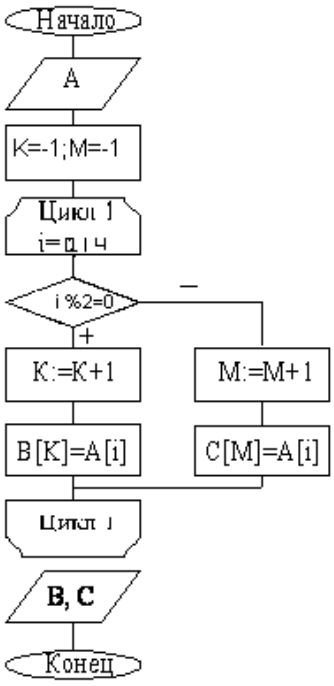


Рис. 3.8. Задача запоминания результата

3.3.6. Нахождение наибольшего и наименьшего значений функции

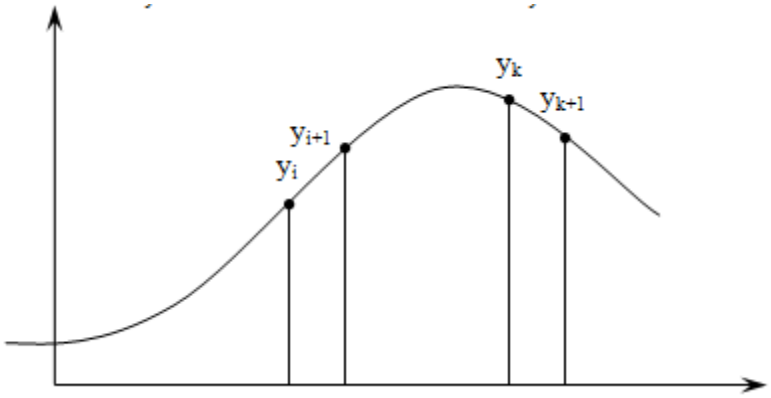
Нахождение наибольшего или наименьшего значения функции y=f(x) выполняется в цикле, в котором вычисляется текущее значение функции и сравнивается с наибольшим или наименьшим из всех предыдущих значений этой функции. Если текущее значение функции окажется больше наибольшего из предыдущих, то его надо считать новым значением наибольшего. В противном случае наибольшее сохраняет своё старое значение. Математически это можно описать:

$$y_{max} = \begin{cases} y_i, \text{если } y_i > y_{max} \\ y_{max}, \text{если } y_i \le y_{max} \end{cases}$$

Аналогично, для наименьшего значения:

$$y_{min} = \begin{cases} y_i, \text{если } y_i < y_{min} \\ y_{min}, \text{если } y_i \ge y_{min} \end{cases}$$

Следует отметить, что речь идёт не о максимуме функции, а именно о наибольшем или наименьшем из вычисленных значений функции. Это объясняется тем, что в соответствии с условием задачи вычисляются дискретные значения функции и истинный максимум или минимум может находиться между ними.



Пример 3.10. Блок-схема и программа нахождения наибольшего значения функции y(x)= 2sin(x) + tg(x) при изменении x от 0 до 1,5 с шагом 0,1 показаны на рис. 3.9.

```
#include <iostream>
#include <math.h>
int main()
{
using namespace std;
double x = 0.0,ymax = 2*sin(0.0)+tan(0.0),y;
while(x <= 1.5)
   {
    y = 2*sin(x)+tan(x);
    if (y>ymax) ymax = y;
    x = x+0.1;
   };
cout << "ymax = " << ymax << endl;
}
```

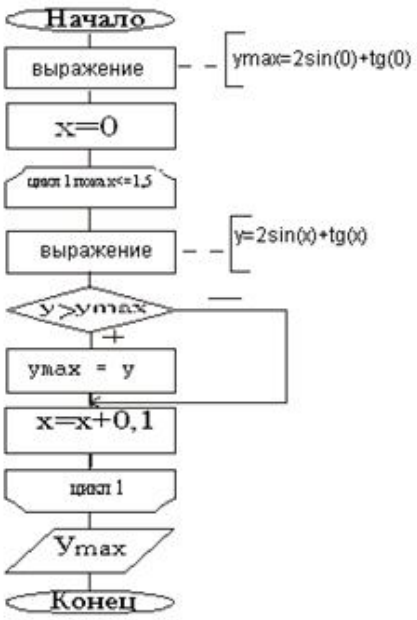


Рис. 3.10. Нахождения наибольшего значения функции

Задания для самостоятельного выполнения.

1) Определить наибольшее значение функции $Y = \frac{\sin(2x+2)}{X^2}$ при изменении X от π/12 до π с шагом π/12.

2) Определить наименьшее значение функции $Z = \frac{\cos(0{,}25y+4)^2}{2a+2}$, где a - константа, при изменении Y от 0 до π с шагом π/24.

3) Найти наибольшее значение функции $Z = \frac{\sin(7{,}25x+x^2)}{2} + x$ при изменении X от 0 до π/2 с шагом π/12.

4) Найти наименьшее значение функции $Y = 2x^2 + x + a$, где a-константа, при изменении X от –B до B с шагом H.

5) Найти наибольшее и наименьшее значения функции $Y = \frac{\sin(x+0{,}25)}{x} + 5$ при изменении X от 0,1 до A с шагом H.

3.3.7. Нахождение наибольшего и наименьшего элемента одномерного массива и его индекса

Пример 3.11. Блок-схема определения индекса наибольшего элемента массива A(10) представлена на рис. 3.11.

Значения элементов массива A после ввода размещаются в памяти компьютера. Для нахождения наибольшего элемента массива (и/или его индекса) используется тот же алгоритмический прием, что и при нахождении наибольшего/наименьшего значения функции. В качестве начального значения наибольшего целесообразно взять значение первого элемента массива (в C++ это элемент с индексом 0). Сравнение выбранного элемента следует проводить со вторым, третьим и т.д. элементом массива. При выполнении условия A[i]>Amax необходимо выполнять операторы присваивания Amax = A[i] и Nmax = i. Так как значение первого элемента может оказаться наибольшим, то перед циклом, наряду с оператором Amax = A[0], необходимо записать Nmax = 0.

Рис. 3.11

Программный код решения примера:

```
#include <iostream>
#include <clocale>
int main()
{
setlocale(LC_CTYPE,"Russian");
using namespace std;
int i,Nmax;
double A[10],Amax;
for(i=0;i<10;i++)
  {
  cout  <<  "Введите " << i << " элемент массива: ";
  cin >> A[i];
  };
Amax = A[1]; Nmax = 1;
for(i=0;i<10;i++)
  if (Amax < A[i])
      {Amax = A[i]; Nmax = i;}
cout  <<  "индекс наибольшего элемента - " << Nmax <<endl;
}
```

Задания для самостоятельного выполнения.

1) Найти наибольший элемент массива X(N и его порядковый номер.
2) Найти сумму наибольшего и наименьшего среди значений элементов массива X(A).
3) Найти наименьший элемент массива N(X) и его порядковый номер.
4) Найти наименьший и наибольший элементы массива X(A).
5) Найти наибольший по модулю элемент массива A(B) и его порядковый номер.

3.3.8. Сортировка элементов одномерного массива

Сортировка - это способ, с помощью которого неупорядоченные данные можно расположить упорядоченно в определенном порядке (по возрастанию, убыванию, по алфавиту и т.д., например, сотрудников отсортировать по возрасту или отсортировать названия какой-либо продукции по алфавиту). Существует много способов сортировки: пузырьковая сортировка, сортировка выбором, блочная сортировка и т.д. Рассмотрим на примере (рис.3.12) сортировку выбором.

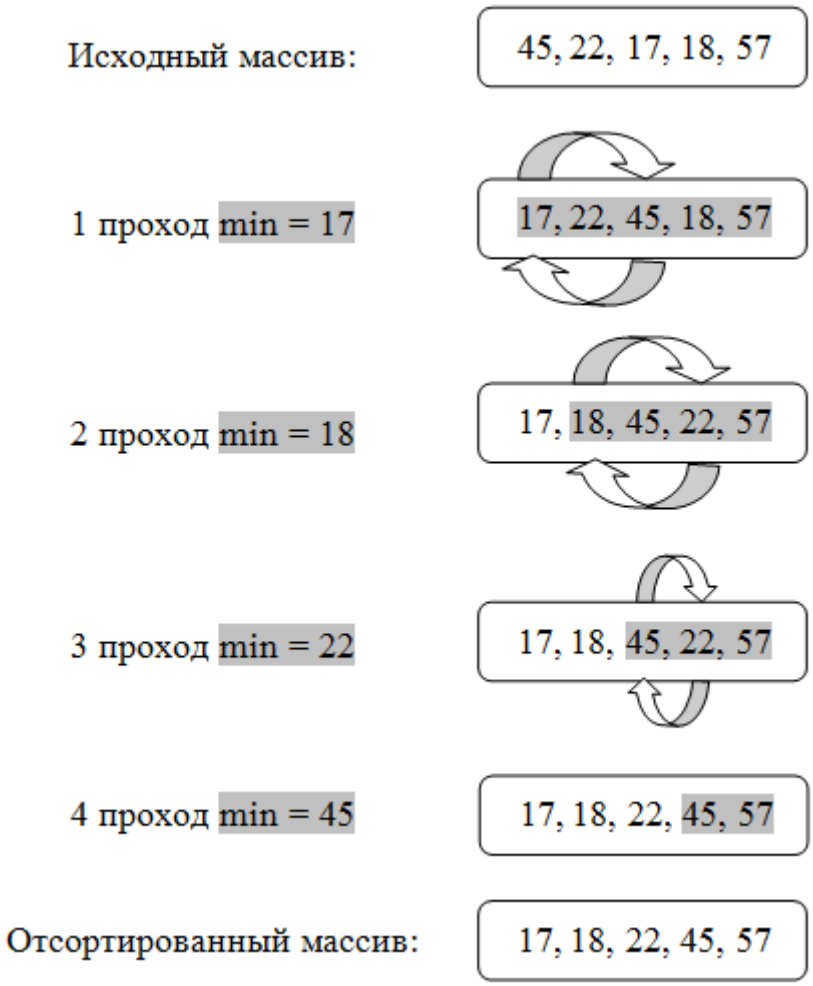


Рис. 3.12. Схема сортировки выбором

Пример 3.12. Блок-схема сортировки элементов массива A(10) в порядке возрастания методом выбора приведена на рис. 3.13.

Программный код решения примера:

```
#include <iostream>
#include <clocale>
int main()
{
setlocale(LC_CTYPE,"Russian");
using namespace std;
int i, k, hold;
double a[10];
for(i=0;i<10;i++)
  {
  cout  <<  "Введите " << i << " элемент массива: ";
  cin >> a[i];
  };
for (i = 1; i < 10; i++)
```

```
        for (k = 0; k < 10 - 1; k++)
                if (a [k] > a [k + 1])
                {
                hold = a [k];
                a [k] = a [k + 1];
                a [k + 1] = hold;
                }
cout << "Отсортированный массив:"<< endl;
for (i = 0; i < 10; i++)
        cout << a [i] << "; ";
cout << endl << endl;
}
```

Сортировка выбором заключается в том, что программа сначала анализирует массив, находя в нем элемент с минимальным значением. Затем этот наименьший элемент обменивается местами с первым элементом массива. Далее процесс повторяется для подмассива, начиная со второго элемента. Затем для подмассива, начиная с третьего элемента и так далее. В результате каждого прохода минимальные значения занимают свои позиции последовательно, в итоге получаем отсортированный массив. По производительности данный тип сортировки сравним с алгоритмом пузырьковой сортировки, т.к. для массива из n элементов нужно выполнить n - 1 проход, а для каждого подмассива нужно выполнить n - 1 сравнение.

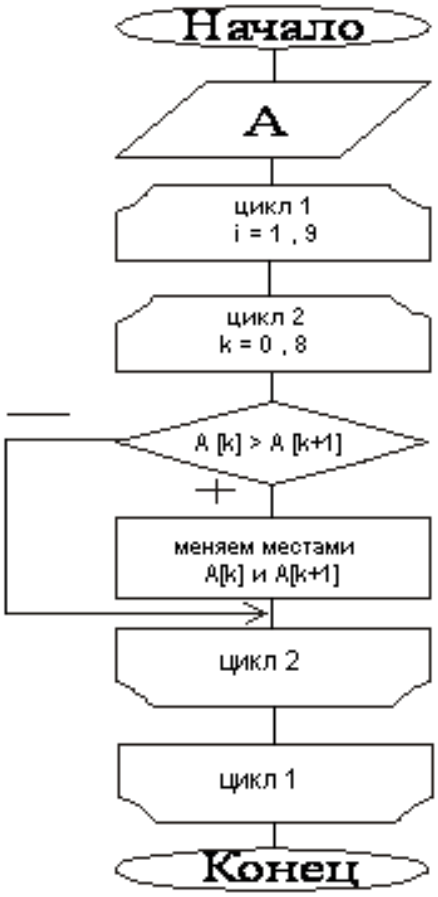


Рис. 3.13

Когда обрабатываемый подмассив будет содержать всего лишь один элемент, значит массив отсортирован.

Изложенный метод называется сортировкой выбором, поскольку он работает по принципу выбора наименьшего элемента из числа неотсортированных.

Задания для самостоятельного выполнения.

1) Дан массив X(N). Вывести на экран элементы этого массива кратные пяти в порядке возрастания.
2) Вывести на экран четные элементы массива X(A) в порядке убывания.
3) Дан массив A(X). Вывести на экран элементы этого массива кратные трем в порядке возрастания.
4) Вывести на экран четные элементы массива A(N) в порядке возрастания.
5) Дан массив X(N). Вывести на экран элементы этого массива кратные семи в порядке убывания.

3.3.9. Программирование алгоритмов со структурой вложенных циклов

При решении достаточно сложных задач появляется необходимость использования в одной программе различных приемов программирования. В программе со структурой вложенных циклов необходимо определить, какой из приемов следует использовать в данном цикле.

В цикл, называемый внешним, могут входить один или несколько вложенных циклов, называемых внутренними. Организация как внешнего, так и внутреннего цикла осуществляется по тем же правилам, что и простого цикла. Параметры внешнего и внутреннего циклов разные и изменяются не одновременно, при одном значении параметра внешнего цикла параметр внутреннего цикла принимает поочередно все свои значения.

Структура вложенных циклов применяется, в частности, при программировании задач на двумерные массивы.

ЛИТЕРАТУРА

1. Александреску А. Современное проектирование на C++.: Пер. с англ. — М.: Издательский дом «Вильямс», 2012 — 336с.
2. Архангельский А.Я., Тагин М.А. Программирование на C++Builder. М.: ООО "Бином-Пресс", 2007 г. – 1184 с.
3. Березин Б.И., Березин С.Б. Начальный курс С и С++. – М.: ДИАЛОГ-МИФИ, 2011. 288 с.
4. Богуславский А.А., Соколов С.М. Основы программирования на языке Си++. – Коломна: КГПИ, 2012. – 490 с.
5. Бондарев В.М. Программирование на С++. 2-е изд. – Харьков: "Компания СМИТ", 2005.- 284 с.
6. Влиссидес Д. Применение шаблонов проектирования. Дополнительные штрихи.: Пер. с англ. — М.: Издательский дом "Вильямс", 2003. — 144 с.
7. Давыдов В.Г. Visual C++. Разработка Windows-приложений с помощью MFC и API-функций. — СПб.: БХВ-Петербург, 2008. —- 576 с.
8. Давыдов В.Г. Программирование и основы алгоритмизации: Учеб. пособие/В.Г. Давыдов. — М.: Высш. шк., 2013. — 447 с.
9. Джамса К. Учимся программировать на языке С++.: Пер. с англ. — М.: Мир, 2008 — 320с.
10. Динман М.И. С++. Освой на примерах. — СПб.: БХВ-Петербург, 2006. —- 384с.
11. Дэвис Стефан Р. С++ для "чайников", 5-е издание.: Пер с англ. – М.: Издательский дом "Вильямс", 2007. – 384 с.
12. Ишкова Э.А. С++. Начала программирования. Изд. 2-е, перераб. и доп. — М.: ООО «Бином-Пресс», 2004 — 368с.
13. Культин Н. Б. С/С++ в задачах и примерах. — СПб.: БХВ-Петербург, 2005. — 288 с.
14. Лаптев В.В., Морозов А.В., Бокова А.В. С++. Объектно-ориентированное программирование. Задачи и упражнения. — СПб.: Питер, 2007. —- 288с.
15. Страуструп Б. Язык программирования С++. М.; СПб.: БИНОМ – Невский диалект, 2004 – 378 с.
16. Франка П. С++: учебный курс. — СПб.: Питер, 2013. — 521 с.
17. Халпери П. Стандартная библиотека С++ на примерах. : Пер. с англ. — М.: Издательский дом «Вильямс», 2011 — 336с.
18. Холзнер С. Visual C++. Учебный курс. . — СПб.: Питер, 2007. — 570с.
19. Шилдт Г. С++. Руководство для начинающих 2-е издание. : Пер. с англ. — М.: Издательский дом «Вильямс», 2005 — 672с.
20. Язык С++: Учеб. Пособие / И.Ф. Астахова, С.В. Власов, В.В. Фертиков, А.В. Ларин. – Мн.: Новое знание, 2013. – 203 с.

СОДЕРЖАНИЕ