

В книге приведены многочисленные и реально работающие примеры, которые позволяют начинающим программистам взять за основу программу из примера и, модифицируя ее, более качественно осваивать язык программирования.

В конце каждой главы приведены контрольные вопросы. Большая часть глав имеет также практические задания, что делает книгу интересной для преподавателей, читающих курсы, связанные с программированием на языках высокого уровня.

Книга предназначена для студентов и аспирантов, изучающих вопросы, связанные с программированием, и преподавателей соответствующих курсов.

## ГЛАВА 1. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ ЯЗЫКА C++

### 1.1. Алфавит языка

Алфавитом языка называют присущий данному языку набор символов, из которых формируются все конструкции языка.

При использовании любого языка (в том числе и языка программирования) необходимо следить за тем, чтобы все произносимое или написанное было корректно с точки зрения алфавитных конструкций (т. е. *синтаксически*) и имело определенный смысл (т. е. обладало *семантикой*).

Переводом программы с языка, на котором она написана, на машинный язык и анализом ее синтаксиса занимается программа-компилятор. Иногда компилятор называют более общим термином – *транслятор*.

Для обозначения (идентификации) всех объектов, вводимых в программу, используются *идентификаторы*, или *имена*. Здесь под объектами понимаются переменные, константы, типы, подпрограммы и т. д. Идентификаторы начинаются с латинской буквы и могут содержать латинские буквы, цифры и знаки подчеркивания. Идентификаторы могут начинаться и со знака подчеркивания, но пользоваться такими именами нужно с большой осторожностью во избежание совпадения идентификаторов, создаваемых программистом, с именами, содержащимися в стандартных библиотеках.

*Ключевые (служебные, зарезервированные) слова* имеют однозначно определенный смысл и могут использоваться только так, как это задано в языке программирования. Ключевые слова не могут быть переопределены, т. е. их нельзя использовать в качестве имен, вводимых программистом.

Алфавит языка С (C++) включает:

- прописные латинские буквы A . . . Z;
- строчные латинские буквы a . . . z;
- арабские цифры 0 . . . 9;
- разделители: , . , ; ? ! , ! / , \ ~ , \_ , # , % , & , ^ , = , - , + , \* , ( , ) , { , } , [ , ] , < , > ;
- пробельные символы: SP, H\_TAB, CR, LF, V\_TAB, FF, Ctrl-Z (конец текстового файла);
- специальные символы, необходимые для представления символов, не имеющих графического обозначения, а также пробельных символов; наиболее употребительные специальные символы представлены в табл. 1.1.

**Таблица 1.1. Наиболее употребляемые специальные символы**

Символ	Описание
\n	Новая строка (LF)
\t	Горизонтальная табуляция (H_TAB)
\r	Возврат каретки (CR)
\f	Новая страница (FF)
\a	Звуковой сигнал
'	Апостроф
''	Кавычки
\\"	Обратная косая черта (back slash)

Компилятор языка С (C++) воспринимает одну и ту же прописную и строчную латинскую букву как различные символы. То есть имена MyProgram, MYPROGRAM и myprogram С-компилятор будет трактовать как три различных имени.

*Директива* препроцессора – это инструкция, записанная в исходном тексте C+-программы и выполняемая препроцессором. Препроцессор просматривает программу до процесса компиляции, подключает необходимые файлы, заменяет символические аббревиатуры в программе на соответствующие директивы и т. д. Стока, содержащая директиву препроцессора, начинается с символа «#» и не имеет точки с запятой в конце.

В программах на C++ предусмотрено использование *комментариев*, т. е. пояснительного текста, который не транслируется и, следовательно, не влияет на ход выполнения программы. Для оформления односторонних комментариев в языке C++ используются две наклонные черты, для многострочных комментариев – конструкция /\*...\*/:

```
// Односторонний комментарий в C++
/* Многострочный
   комментарий в C++*/
```

## 1.2. Структура программы на языке C++

По традиции первая программа на С (C++) пишет фразу «Здравствуй, мир!». Мы не будем нарушать эту традицию.

### Пример 1.1

```
#include "stdafx.h"           // директивы препроцессора
#include <iostream>
#include <conio.h>
using namespace std;          // директива using
int _tmain(int argc, _TCHAR* argv[]) // начало программы
{ cout<<"Hello, World!\n";      // оператор вывода на экран
  return 0;                   // конец программы
}
```

Директива #include <iostream> подключает заголовочный файл iostream, который обеспечивает работу с переменной cout и операцией << путем предоставления доступа к потокам ввода-вывода;

using namespace std; обозначает, что используется пространство имен с именем std. Пространство имен – это некоторая объявляемая область видимости имен. Она необходима, чтобы избежать конфликтов идентификаторов;

\_tmain( ) – имя основной программы, оно всегда присутствует и не может изменяться;

{ } – скобки, ограничивающие тело программы;

cout<<"Здравствуй, мир!\n"; – оператор, выводящий на экран строку, заключенную в кавычки;

return 0; – пока будем считать, что этот оператор необходим для корректного завершения программы.

Каждый оператор завершается символом «точка с запятой», который является атрибутом оператора, указывающим на его конец.

*Пример 1.2.* Рассмотрим программу, содержащую элементы вычислений.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{ int a, b, c;    // объявление переменных целого типа
  a = 5; b = 10; // определение переменных a и b
  c = a+b;       // присваивание переменной c суммы a и b
  cout<<"Value c = "<<c<<"\n";
  getch();        // подключается с помощью заголовочного файла conio.h,
  // останавливает выполнение программы до нажатия любой клавиши
  return 0;
}
```

В целом структура программы не изменилась, но добавились некоторые моменты:

int a,b,c; – объявление трех целочисленных переменных a, b и c; int – признак того, что переменные имеют целый тип;

общим правилом при программировании на C++ является то, что прежде, чем использовать в программе какую-либо переменную, ее необходимо объявить с указанием типа;

a=5, b=10; – определение переменных a и b. Объявление переменной приводит к тому, что для нее отводится место в памяти согласно объявлению типа. Определение переменной приводит к присваиванию ей конкретного значения.

Можно одновременно объявить и определить переменную. Такой способ называется *инициализацией*. То есть можно написать:

```
int a=5, b=10;
```

В результате работы программы будет напечатана строка  
Значение c=15

и курсор окажется на следующей строке благодаря наличию управляющего символа `\n` в операторе вывода на экран. Эффекта перевода курсора на новую строку можно добиться путем использования манипулятора `endl`, который, кроме перевода курсора, выполняет еще и очистку буфера строки. Использование манипулятора `endl` здесь более предпочтительно, чем управляющих символов. *Манипулятор* представляет собой особую инструкцию, которая предназначена для изменения формата вывода данных на экран.

Программа оказывается более гибкой, если значения переменным не присваивать в теле программы, а вводить с клавиатуры.

#### *Пример 1.3*

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{ double m, p, q; // объявление переменных вещественного типа
    cout<<"Input p ";
    cin>>p;
    cout<<"Input q ";
    cin>>q;
    m=p/q;
    cout<<"m = "<<m<<endl;
    _getch();
    return 0;
}
```

Можно этот пример переписать короче.

#### *Пример 1.4*

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{ double m, p, q;
    cout<<"Input p and q: ";
    cin>>p>>q;
    m=p/q;
    cout<<"m = "<<m<<endl;
    _getch();
    return 0; }
```

Здесь приведены простейшие программы, однако они позволяют сделать выводы о структуре программы на C++:

- любая программа может содержать (и скорее всего, содержит) заголовочные файлы, подключенные директивой `#include`;
- программа всегда начинается словом `main()`;
- перед использованием переменной ее необходимо объявить с указанием типа.

#### Контрольные вопросы

1. Что представляет собой алфавит языка?
2. Что понимают под словами «синтаксис» и «семантика»?
3. Дать определение компилятора.
4. Для чего нужны идентификаторы?
5. Перечислить особенности ключевых слов.
6. Что такое препроцессор?
7. Что содержится в заголовочных файлах и как можно подключить их?
8. Как ввести и как вывести данные в C++-программе?
9. Как объявляют переменные целого и вещественного типа?

#### Практические задания

1

1. Все ли правильно в приведенной ниже программе?

```
# include <iostream>
using namespace std;
int main()
{ int m, k;
    m = k+2;
    cout>>m;
}
```

2. Исправить ошибки, если они есть.

```
# include <iostream>
using namespace std;
int main()
{float r;
    cout<<"Введите значение a ";
    cin>>a;
    cout<<"Введите значение b ";
    cin>>b;
    r=a+b;
    cout<<"r = "<<r<<endl;
    return 0;
}
```

3. Вычислить среднее арифметическое четырех значений, введенных с клавиатуры.

4. Что будет выведено на экран в результате работы следующей программы, если введены значения 13 и 4?

```
# include <iostream>
using namespace std;
int _tmain()
{int p, l, m;
cout<<"Введите значение m ";
cin>>m;
cout<<"Введите значение l ";
cin>>l;
l = 2; m = 9;
p = m + l;
cout<<"p = "<<p<<endl;
return 0;
}
```

Брать со  
Пользова

Требова

По оконч  
докумен

## ГЛАВА 2. СТАНДАРТНЫЕ ТИПЫ ДАННЫХ

Язык C++ является типизированным языком с той точки зрения, что каждая используемая переменная должна быть объявлена с указанием типа, чтобы занять для нее достаточно места в оперативной памяти ЭВМ.

Переменная представляет собой имя, связанное с областью памяти, которая отведена для хранения значения этой переменной. Все переменные в программе, написанной на языке C++, должны быть объявлены до их использования этой программой.

### 2.1. Переменные целого типа

Переменные целого типа используются для работы с дискретными объектами. В языке C++ имеется несколько целых типов данных: short, int, long. Наиболее часто используется тип int. Этот тип является системно-зависимым: в MS DOS и в ранних версиях Windows он занимает 2 байта, в 32-разрядных ОС – 4 байта.

В языке C++ используются как знаковые, так и беззнаковые переменные. Все целые типы – char, short, int и long – являются знаковыми по умолчанию, но можно объявить их как беззнаковые, которые, занимая тот же объем памяти, что и их знаковые аналоги, представляют только положительные значения, включая нуль. Вследствие этого они могут представлять вдвое больше положительных значений (табл. 2.1).

Тип char (символьный) тоже может интерпретироваться как целый. Этот тип, хотя и предназначен для представления символов, может поддерживать целочисленные значения в диапазоне от -128 до 127 или от 0 до 255. Значения символьных переменных записываются в одинарных кавычках.

Таблица 2.1. Целочисленные типы данных

Тип данных	Размер, байт	Нижняя граница диапазона	Верхняя граница диапазона
[signed] char	1	-128	127
unsigned char	1	0	255
[signed] short	2	-32 768	32 767
unsigned short	2	0	65 535
[signed] int	2	-32 768	32 767
unsigned int	4	-2 147 483 648	2 147 483 647
[signed] long	2	0	65 535
unsigned long	4	0	4 294 967 295
[signed] long long	4	-2 147 483 648	2 147 483 647
unsigned long long	4	0	4 294 967 295

Если в процессе каких-либо арифметических действий, например при сложении, результат превосходит максимально допустимое значение для объявленного типа (т. е. при переполнении), значения целочисленных переменных обнуляются.

## 2.2. Переменные вещественного типа

Переменные вещественного (действительного) типа предназначены для хранения чисел, которыми измеряются непрерывные величины. Эти числа имеют дробную часть и могут быть представлены в форме записи с десятичной точкой или в экспоненциальной форме:

```
double d1=2.35; // представление числа в формате с десятичной точкой;
double d2=2.0E-5; // представление числа в экспоненциальной форме.
```

В языке C++ существует три вещественных типа данных: float, double, long double (табл. 2.2).

Таблица 2.2. Вещественные типы данных

Тип данных	Размер, байт	Точность	Нижняя граница диапазона	Верхняя граница диапазона
float	4	7	3.4E-38	3.4E+38
double	8	15	1.7E-308	1.7E+308
long double	10	15	3.4E-4932	1.1E+4932

Вообще, тип long double зависит от типа компилятора и его характеристики могут совпадать с типом double.

## 2.3. Переменные логического типа

Переменные логического типа (тип bool) могут иметь всего два значения: false (ложь) и true (истина). Эти переменные фактически занимают 1 бит, но в ЭВМ минимальной адресуемой единицей информации является байт, поэтому трансляторы выделяют под них по 1 байту памяти.

Обычно в логических переменных хранятся результаты разного рода сравнений.

## 2.4. Константы

Данные в языках программирования могут представляться также в виде констант. Константы используются в тех случаях, когда программе запрещено изменять значение какой-либо переменной. В этом случае ее объявляют как константу следующим образом:

```
const int intValue = 25;
const float floatVal = 2.531;
const char sym = '*';
```

Символьная константа состоит из одного символа, заключенного в одинарные кавычки: 'q', '2', '.' и т. д. К разряду символьных констант относятся и специальные символы.

Литеральные константы (литералы) – это те значения, которые вводятся непосредственно в текст программы. Поскольку после компиляции программы нельзя изменять значения литералов, их также называют константами.

В языке C++, помимо числовых и символьных констант, используются строковые константы. Строковая константа состоит из последовательности символов кода ASCII, заключенной в кавычки.

Для обработки любых констант компилятор использует допустимые типы данных с наименьшим возможным диапазоном значений, т. е. занимает минимально возможный объем памяти для хранения предлагаемой константы. Это происходит, поскольку осуществляется оптимизация по памяти. Например, для хранения таких чисел, как 1257 или -251, будет использован тип short (или int, если он в данной ОС занимает 2 байта). Однако если требуется, чтобы под константу было отведено столько памяти, сколько объявлено, а не столько, сколько она фактически занимает, необходимо явное указание ее длины. Для явного указания длинной константы используется буква L, записываемая после числа. Явное указание буквы U после числа делает число беззнаковым.

Добавление к константам, имеющим тип double, букв f или F приводит к тому, что они приобретают тип float, а буквы l или L – тип long double, при этом для их хранения будет отведено количество памяти в соответствии с табл. 2.2.

### Пример 2.1

```
const long int L1 = 1257L; // хранится как long int
const unsigned int UI = 5317U; // хранится как unsigned int
const unsigned long int ULI = 2UL; // хранится как unsigned long int
const float PI = 3.14159F; // хранится как float
const float PI = 3.14159f; // хранится как float
const long double M = 2.58316L; // хранится как long double
const long double M = 2.58316l; // хранится как long double
```

Константы можно определять с помощью директивы препроцессора #define.

```
#define PI 3.14159
```

Однако в C++ такой способ объявления констант является нежелательным.

## 2.5. Выражения

Выражение представляет собой объединение операций и операндов. Операндом называется то, над чем производится указанное действие, на-

зываемое *операцией*. Каждое выражение в языке C++ имеет значение, которое присваивается переменной, стоящей слева от знака присваивания.

### Пример 2.2

```
int b, a = 10;
b = a + 25;           // оператор, присваивающий переменной b
                      // значение выражения a+25
```

Здесь a и 25 – операнды, + – операция, a+25 – выражение, = – знак присваивания.

Знак «=>» – это знак присваивания, а не знак равенства. Присваивание является выражением, так как имеет значение, равное присваиваемому результату.

k = 5 – выражение

i=(j=k) – тоже является выражением, поскольку j=k – выражение, имеющее конкретный результат, так же, как i=j. Кроме того, скобки можно опустить, тогда выражение i=j=k присваивает значение k переменным i и j, а выражение i=j=k=5 – значение 5 всем трем переменным. Это очень удобно при инициализации нескольких переменных одним значением.

## 2.6. Операции

Типы операций, применяемые при построении выражений:

- арифметические;
- отношений;
- логические и поразрядные;
- инкремента и декремента;
- присваивания.

Арифметические операции:

\* – умножение;  
/ – деление;  
+ – сложение;  
- – вычитание;  
% – деление по модулю (взятие остатка от целочисленного деления).

Все перечисленные здесь операции выполняются традиционным образом, кроме операции деления. Особенность операции деления заключается в том, что она дает целый результат, если оба операнда целые. Чтобы получить действительный результат, необходимо иметь хотя бы один действительный operand.

### Пример 2.3

```
#include "stdafx.h"
#include <stdlib.h>
#include <iostream>
#include <conio.h>
```

```
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{ cout<<5 + 2<<endl;          // 7
  cout<<10 - 3<<endl;          // 7
  cout<<25 * 5<<endl;          // 125
  cout<<9 / 5<<endl;           // 1
  cout<<9 % 5<<endl;           // 4
  cout<<9 / 5.0<<endl;          // 1.8
  _getch();
  return 0;
}
```

Операции отношений в языках программирования, как и в математике, используются для сравнений. В языке C++ применяются следующие операции отношений:

< – меньше;  
<= – меньше или равно;  
> – больше;  
>= – больше или равно;  
== – равно;  
!= – не равно.

### Пример 2.4

```
bool b;
b=5>2; // b примет значение true
b=7<4; // b станет равным false
```

Ранние версии языка C++ логического типа не имеют. Там его заменяет тип int, для которого нулевое значение интерпретируется как ложь, ненулевое – как истина.

### Пример 2.5

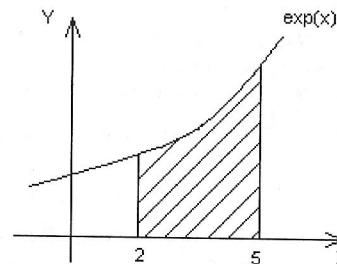
```
int b;
b=6>2; // b примет значение 1
b=7<4; // b станет равным 0
```

К логическим операциям относятся:

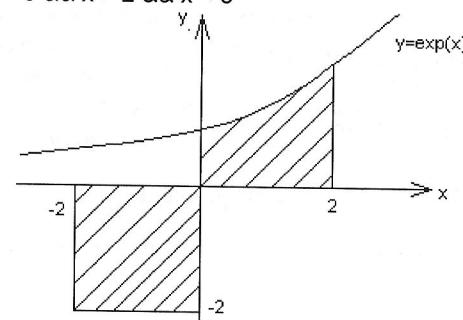
&& – логическое «И»;  
|| – логическое «ИЛИ»;  
! – логическое отрицание.

Логические операции используются в тех случаях, когда необходимо объединить несколько условных выражений.

**Пример 2.6.** Пусть нужно подтвердить или опровергнуть принадлежность точек к заштрихованной области:



```
bool b;
b = y < exp(x) && y > 0 && x > 2 && x < 5
```



```
bool b;
b = (y < exp(x) && y > 0 && x > 0 && x < 2) || (x < 0 && x > -2 && y < 0 && y > -2)
```

Свойства логических операций представлены в табл. 2.3.

Таблица 2.3. Свойства логических операций

$V_1$	$V_2$	$\neg V_2$	$V_1 \& V_2$	$V_1 \mid\mid V_2$
false	false	true	false	false
false	true	false	false	true
true	false		false	true
true	true		true	true

Поразрядные операции или операции с разрядами (битами) нельзя применять к переменным типа float и double. К таким операциям относятся:

- & – поразрядное «И»;
- | – поразрядное «ИЛИ»;
- $\neg$  – поразрядная инверсия;
- $\wedge$  – поразрядное «исключающее ИЛИ»;
- $\ll$  – сдвиг влево;
- $\gg$  – сдвиг вправо.

Поразрядные операции выполняются в соответствии с таблицами истинности (табл. 2.4), причем каждый разряд рассматривается независимо от других разрядов.

Таблица 2.4. Свойства поразрядных операций

$x$	$y$	$\neg y$	$x \& y$	$x \mid\mid y$	$x \wedge y$
0	0	1	0	0	0
0	1	0	0	1	1
1	0		0	1	1
1	1		1	1	0

Пример 2.7

$\&$	0011100110110101 1111100001111101 ----- 0011100000110101	0011100110110101 1111100001111101 ----- 1111100111111101
$\mid\mid$		$\wedge$
$\neg$		

В языке C++ существуют две операции, не встречающиеся в математике:

- $++$  – инкремент;
- $--$  – декремент.

Операция инкремента прибавляет к операнду единицу, а операция декремента вычитает из операнда единицу:

- $i++$  эквивалентно  $i = i + 1$ ;
- $i--$  эквивалентно  $i = i - 1$ .

Выражения, подобные  $i++$  и  $i--$ , имеют значения, которые зависят от расположения знаков  $++$  и  $--$ :

- оператор  $j = i++$ ; (постфиксная форма записи) эквивалентен  $j = i$ ;  $i++$ ;
- оператор  $j = ++i$ ; (префиксная форма записи) эквивалентен  $i++$ ;  $j = i$ .

Пример 2.8

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    int i=15;
    cout<<i<<endl;           // 15
    cout<<++i<<endl;          // 16
    cout<<i<<endl;           // 16
    cout<<i++<<endl;          // 16
    cout<<i<<endl;           // 17
    getch();
    return 0;
}
```

**Пример 2.9**

```
#include "stdafx.h"
#include <stdlib.h>
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    int sum, a=2, b=5;
    sum = a + b++;
    cout<<a<<' '<<b<<' '<<sum<<endl; // 2 6 7
    a=2; b=5;
    sum = a+++b;
    cout<<a<<' '<<b<<' '<<sum<<endl; // 3 5 7
    a=2; b=5;
    sum = (a++)+b;
    cout<<a<<' '<<b<<' '<<sum<<endl; // 3 5 7
    a=2; b=5;
    sum = a+(++b);
    cout<<a<<' '<<b<<' '<<sum<<endl; // 2 6 8
    _getch();
}
```

Запись `sum=a+b++`; означает «сложить a и b, присвоить результат sum и увеличить b на единицу», а запись `sum=a+++b`; – «увеличить b на единицу, сложить a и b и присвоить результат sum».

Операция присваивания в языке C++, помимо привычного вида, имеет форму представления, называемую операцией с присваиванием. Использование этого представления связано с ускорением работы программы. Например, выполнение оператора `a=a+25` приводит к двукратному определению адреса переменной a: для нахождения исходного значения и для занесения полученного результата. Эту избыточность можно ликвидировать, записав `a+=25`. Здесь адрес определяется всего один раз, так как оператор интерпретируется как «к содержимому переменной a прибавить 25».

Полный набор операций с присваиванием: `*=`, `/=`, `+=`, `-=`, `%=`, `<<=`, `>>=`, `&=`, `^=`, `|=`.

**Пример 2.10**

```
cnt+=50; // cnt=cnt+50;
cnt%=2; // cnt=cnt%2;
```

Операторы служат основными элементами, из которых строится программа на языке C++. Каждый оператор в языке C++ заканчивается точкой с запятой, которая является составной частью оператора, а не разделителем между операторами.

**2.7. Преобразование типов**

Язык C++, являясь типизированным языком, позволяет тем не менее весьма свободно обращаться с выражениями, оперирующими различными типами данных. В этом случае операнды приводятся к некоторому общему типу. Приведение выполняется в соответствии со следующими правилами:

1. Автоматически производятся лишь те преобразования, которые превращают операнды с меньшим диапазоном значений в операнды с большим диапазоном значений, поскольку это происходит без какой-либо потери информации, например:

```
int i_var = 5;
float f_var = 2.5, summa;
```

```
summa = i_var + f_var;
```

2. Выражения, не имеющие смысла (например, число с плавающей точкой в роли индекса), не пропускаются компилятором еще на этапе трансляции:

```
float f;
```

```
mas [f]=25; // вызывает ошибку трансляции (Error)
```

3. Выражения, в которых могла бы теряться информация (например, при присваивании длинных целых более коротким или действительных целым), могут вызвать предупреждение (Warning), но они допустимы:

```
int i;
float f=3.2;
i=f; // предупреждение (Warning) при трансляции
```

Для любого выражения можно явно указать преобразование его типа, используя унарный оператор, называемый *приведением типа*.

Оператор приведения типа имеет три возможные формы записи:

```
intV = (int) FloatV; // преобразование
intV = int FloatV; // вещественного
intV = static_cast <int>(FloatV); // в целое
```

Первые два способа считаются устаревшими и к использованию не рекомендуются, хотя транслируются и работают вполне корректно.

Приведение типов – потенциально опасная операция, поэтому применять ее следует только в том случае, если программист четко осознает необходимость этого и понимает, что получится в результате.

## 2.8. Манипулятор setw

Манипулятор setw позволяет задать ширину поля вывода данных. Ширина задается в скобках после названия манипулятора и определяет количество знакомест (символов), которые отводятся под выводимые данные. Для его подключения необходим заголовочный файл <iomanip>.

### Пример 2.11

```
#include "stdafx.h"
#include <stdlib.h>
#include <iostream>
#include <conio.h>
#include <iomanip>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    long lVar = 5312647;
    int iVar = 536;
    char cVar = '*';
    cout << setw(10) << lVar << setw(5) << iVar
        << setw(3) << cVar << endl;
    getch();
}
```

### Контрольные вопросы

1. Дать определение переменной.
2. Перечислить допустимые в языке C++ типы данных.
3. Почему беззнаковые типы допускают увеличение диапазона представляемых ими положительных значений?
4. Что происходит при инициализации переменной?
5. Можно ли определить константу без ее инициализации?
6. В чем заключается процесс оптимизации по памяти во время трансляции программы? Можно ли его предотвратить, и если можно, то как именно?

Перечислить операции, относящиеся к каждому из следующих типов:

- арифметические;
- отношений;
- логические и поразрядные;
- инкремента и декремента;
- присваивания.

## ГЛАВА 3. ЛИНЕЙНЫЕ И РАЗВЕТВЛЕННЫЕ АЛГОРИТМЫ

Программы бывают линейными, разветвленными, циклическими и сложными. Большинство программ являются сложными. Все остальные типы программ имеют несколько искусственный характер. Однако любую программу можно разбить на линейные, разветвленные и циклические фрагменты. Наиболее простыми являются линейные программы или линейные фрагменты сложных программ.

### 3.1. Линейные алгоритмы

Линейными называются программы, в которых операторы выполняются один за другим с первого до последнего, не повторяясь и не изменяя порядка их выполнения.

Линейные программы могут содержать операторы присваивания, математические функции, арифметические операции, действия, связанные с вводом-выводом, и другие операторы, не изменяющие порядка следования операторов в программе.

Основные математические функции, используемые в языке C++, представлены в табл. 3.1.

Таблица 3.1. Основные математические функции языка C++

Функция	Обозначение функции	Тип		Имя файла описания
		возвращаемого значения функции	аргумента	
Абсолютное значение	abs(x) cabs(x) fabs(x)	int double float	int double float	<stdlib.h> <math.h> <math.h>
Арккосинус	acos(x)	double	double	<math.h>
Арксинус	asin(x)	double	double	<math.h>
Арктангенс	atan(x)	double	double	<math.h>
Косинус	cos(x)	double	double	<math.h>
Синус	sin(x)	double	double	<math.h>
Быстроходный экспонента ex	exp(x)	double	double	<math.h>
Поверхностная функция xy	pow(x,y)	double	double	<math.h>
Логарифм натуральный	log(x)	double	double	<math.h>
Логарифм десятичный	log10(x)	double	double	<math.h>
Квадратный корень	sqrt(x)	double	double	<math.h>
Тангенс	tan(x)	double	double	<math.h>

Заголовочный файл, необходимый для работы этих функций, – `<math.h>` (или `<cmath.h>`).

Действия, связанные с вводом-выводом, обеспечиваются потоками стандартного ввода и стандартного вывода, а также связанными с ними манипуляторами.

**Пример 3.1.** Составить программу для вычисления объема  $V$  и площади поверхности  $S$  полого шара по заданным внешнему и внутреннему радиусам  $R$  и  $r$ , если известно, что  $S = 4\pi(R^2 + r^2)$ ,  $V = 4/3\pi(R^3 - r^3)$ .

```
#include "stdafx.h"
#include <conio.h>
#include <iostream>
#include <math.h>
using namespace std;
const double Pi=3.1415926;
int _tmain(int argc, _TCHAR* argv[])
{double S,V,R,r;
setlocale(LC_ALL, "Russian");      // подключение русификатора
cout << "Введите внешний радиус ";
cin >> R;
cout << "Введите внутренний радиус ";
cin >> r;
S = 4*Pi*(R*R - r*r);
V = 4.0/3*Pi*(pow(R,3)-pow(r,3));
cout << "S = " << S << endl;
cout << "V = " << V << endl;
_getch();
return 0;
}
```

**Пример 3.2.** Ввести координаты точек  $(x_1, y_1)$  и  $(x_2, y_2)$ . Определить расстояние между этими точками.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
#include <math.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{ double x1,x2,y1,y2,dist;
setlocale(LC_ALL, "Russian");
cout << "Введите x1 и y1 ->";
cin >> x1>> y1;
cout << "Введите x2 и y2 ->";
cin >> x2>> y2;
dist = sqrt(pow((x1-x2),2)+pow((y1-y2),2));
cout << "Расстояние равно " << dist << endl;
_getch();
return 0;
}
```

**Пример 3.3.** Поменять местами значения переменных  $x$  и  $y$ .

```
#include "stdafx.h"
#include <iostream>
#include <math.h>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");
int x, y, wrk;           // wrk - рабочая переменная
cout << "Введите x и y ->";
cin >> x >> y;
cout << "x = " << x << " y = " << y << endl;
wrk = x;                // запомнить в wrk значение переменной x
x = y;                  // поместить в x значение y
y = wrk;                // поместить в y значение x, хранящееся в wrk
cout << "x = " << x << " y = " << y << endl;
_getch();
return 0;
}
```

Редкая программа обходится без операций ввода-вывода. Язык C++ имеет большой набор функций и операций, осуществляющих ввод и вывод данных различных типов. Мы уже познакомились с потоковыми операциями ввода-вывода `cin` и `cout`. Однако наряду с ними все еще часто используются функции, применяемые в языке C: `printf()` и `scanf()`, которые предназначены для реализации форматного вывода и ввода данных.

Функция `printf()` имеет следующий синтаксис:

```
printf ("управляющая_строка", [список_аргументов])
```

*Список аргументов* – это последовательность констант, переменных или выражений, значения которых выводятся на экран дисплея в соответствии с форматом управляющей строки. Список аргументов в функции `printf()` может отсутствовать.

Управляющая строка содержит объекты трех типов: обычные символы, выводимые на экран без изменений, спецификации преобразования, каждая из которых вызывает вывод на экран значения очередного аргумента из последующего списка аргументов, и управляющие символьные константы. Каждая спецификация преобразования начинается с символа % и заканчивается символом преобразования. Между ними могут записываться:

- | знак минуса (-), указывающий на то, что выводимый текст выравнивается по левому краю; по умолчанию выравнивание происходит по правому краю;
- | строка цифр, задающая минимальный размер поля вывода;
- | точка, являющаяся разделителем;
- | строка цифр, задающая точность вывода;

- символ l, указывающий на то, что соответствующий аргумент имеет тип long.

Символ преобразования может быть следующим:

d – аргумент преобразуется в десятичное представление;  
o – аргумент преобразуется в восьмеричное представление;  
x – аргумент преобразуется в шестнадцатеричное представление;  
c – значением аргумента является символ;

s – значением аргумента является строка символов;

e – значением аргумента является величина типа float или double в экспоненциальной форме записи;

f – значением аргумента является величина типа float или double в форме записи с десятичной точкой;

g – один из форматов f или e;

i – значением аргумента является целое беззнаковое число;

p – значением аргумента является указатель (адрес).

Таким образом, «управляющая\_строка» определяет количество и тип аргументов.

Среди управляющих символов наиболее часто используются следующие:

\a – кратковременный звуковой сигнал;

\n – перевод строки;

\t – горизонтальная табуляция;

\b – возврат курсора на один шаг назад;

\r – возврат каретки.

Функция scanf описывается аналогично функции printf:

scanf ("управляющая\_строка", список\_аргументов);

но список аргументов здесь является обязательным и не может отсутствовать.

Аргументы функции scanf() должны быть указателями на соответствующие значения, для этого перед именем переменной записывается символ &. Управляющая строка содержит спецификации преобразования и используется для определения количества и типов аргументов, аналогично функции printf().

#### Пример 3.4

printf ("i=%d,\n j=%d,a=%6.2f.\n",i,j,a);

Если i=1234, j=127, a=86.531, то на экране увидим:

i=1234,

j=127, a = 86.53.

Так как после i=1234 стоит символ перевода строки \n, то дальнейший вывод происходит с начала следующей строки.

В управляющей строке допустимо использование символов заполнения, по умолчанию в качестве символа заполнения применяется пробел.

printf ("i=%06d,\n j=%d,a=%6.2f.\n",i,j,a);

Теперь значение i выглядит так:

```
i=001234,
scanf ("%d %f %c %s", &i,&a,&ch,r);
```

Здесь r – строка символов, имя которой само является указателем, поэтому перед ней знак амперсанда (&) не ставится.

Если входные данные при вводе с клавиатуры разделяются разделителями, например запятыми, то и в управляющей строке спецификации преобразования должны быть разделены такими же разделителями. Перешипим пример 3.2 для случая использования операций printf\_s() и scanf\_s().

#### Пример 3.5

```
#include "stdafx.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>
using namespace std;
int main(int argc, _TCHAR* argv[])
{
    float x1,x2,y1,y2,dist;
    printf("Input x1 and y1 ->");
    scanf_s ("%f %f", &x1, &y1);
    printf("Input x2 and y2 ->");
    scanf_s ("%f %f", &x2, &y2);
    dist = sqrt(pow((x1-x2),2)+pow((y1-y2),2));
    printf("Distance is %7.4f\n",dist);
    getch();
    return 0;
}
```

### 3.2. Разветвленные алгоритмы

Разветвленные алгоритмы предусматривают выбор маршрута выполнения программы в зависимости от истинности или ложности некоторых условий. Это обеспечивается наличием в программе специальных операторов, которые иногда называют *конструкциями принятия решений*:

#### 3.2.1. Условный оператор if

Синтаксис оператора if имеет вид:

**if (выражение) оператор;**

если оператор, выполняемый при истинности условия выражения, единственный или:

```
if (выражение) {оператор1;
    оператор2;
    ...
    операторN;
} //end if,
```

если таких операторов несколько.

Здесь под выражением понимается любое логическое выражение или любое выражение, значение которого приводимо к целочисленному значению. Если его значение истинно, т. е. отлично от нуля, то оператор будет выполняться. Схема алгоритма оператора if показана на рис. 3.1.

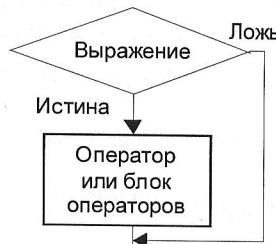


Рис. 3.1. Схема алгоритма оператора if

При необходимости сравнить выражение с некоторым значением следует использовать операции отношений в виде:

**if (выражение == значение) оператор;**

или

**if (выражение!=значение) оператор;**

или

**if (выражение>значение) оператор;**

«оператор;» может быть как простым, так и составным.

Оператор

**if (выражение!=0) оператор; эквивалентен оператору if (выражение) оператор;**

Нельзя писать

**if (выражение =значение) оператор;**

Это одна из наиболее распространенных и труднообнаруживаемых ошибок. Результатом такого использования операции присваивания «==» будет сравнение выражения со значением с последующим присваиванием выражению значения, с которым оно сравнивалось.

**Пример 3.6.** Использование оператора if для определения абсолютной величины введенного с клавиатуры целого значения.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "Russian");
    int value;
    cout << "Введите число: ";
    cin >> value;
    if (value < 0) value = -value;
    cout << value << endl;
    getch();
    return 0;
}
```

Выражение, служащее условием, заключается в круглые скобки.

**Пример 3.7.** По номеру у>0 некоторого года определить с – номер его столетия. Учесть, что, например, XXI в. начинается с 2001 г., а не с 2000-го.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "Russian");
    int y, c;
    cout << "Введите год: ";
    cin >> y;
    c = y/100;
    if (y%100 != 0) c+=1;
    cout << "Этот год принадлежит веку " << c << endl;
    getch();
    return 0;
}
```

### 3.2.2. Оператор if – else

Оператор if может иметь две ветви, одна из которых является альтернативной. Синтаксис оператора if – else имеет вид:

**if (выражение) оператор1;  
else оператор2;**

или после if и else находится по одному оператору, или

```
if (выражение) { оператор1;
    оператор2; }
else { оператор3; }
```

**оператор4;** },

если после if и else находится блок операторов, т. е. два или больше оператора, заключенные в фигурные скобки. Точка с запятой после фигурной скобки, закрывающей блок операторов, не ставится. Схема алгоритма оператора if – else показана на рис. 3.2.



Рис. 3.2. Схема алгоритма оператора if – else

### 3.2.3. Вложенные ветвления

В качестве внутренних операторов оператора if могут использоваться любые операторы, в том числе и условные. Другими словами, в операторе if допустимо применение вложенных конструкций:

```

if (выражение1) оператор1;
else if (выражение2) оператор2;
else if (выражение3) оператор3;
else if (выражениеN) операторN;
else // необязательная часть
      оператор_по_умолчанию;
  
```

В подобных конструкциях часть else связывается с ближайшим предыдущим if в том же блоке, не имеющем части else. Последний оператор else, за которым следует **оператор\_по\_умолчанию**, не является обязательным. Формально уровней вложенности операторов if может быть много, но реально при количестве таких вложенных конструкций, большем чем 4–5, программа становится трудноотлаживаемой.

#### Пример 3.8. [15]

```

#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{ int value;
cout << "Input value from 1 to 10: ";
  
```

```

cin >> value;
if (value >= 1)
    if (value >10)
        cout << "Error: value >10" << endl;
    else cout << "Error: value <1" << endl;
    getch();
return 0;
/* Поскольку else связывается с ближайшим if, эта программа будет работать неправильно */
  
```

```

#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{ int value;
cout << "Input value from 1 to 10: ";
cin >> value;
if (value >= 1)
{ if (value >10)
    cout << "Error: value >10" << endl;
}
else cout << "Error: value <1" << endl;
getch();
return 0;
/* Фигурные скобки скорректировали поведение программы. Теперь все работает так, как было задумано */
  
```

Приведенный пример наглядно демонстрирует, что во избежание неоднозначного толкования программы следует пользоваться фигурными скобками, не полагаясь в сомнительных случаях на компилятор. При этом отступы, как и комментарии, носят чисто декоративный характер и компилятором игнорируются.

**Пример 3.9.** Вычислить  $d = \max(a,b,c)$  – максимальное из трех введенных с клавиатуры чисел.

```

#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int main(int argc, _TCHAR* argv[])
{ int a, b, c, d;
cout << "a > ";
cin >> a;
cout << "b > ";
cin >> b;
cout << "c > ";
cin >> c;
if (a>b && a>c) d = a;
else if (b>c) d = b;
else d = c;
  
```

```

else d = c;
cout << "max = " << d << endl;
_getch();
return 0;
}

```

### 3.2.4. Условная операция

Условная операция является трехоперандной и имеет синтаксис `переменная = выражение? значение1: значение2;`

Такая запись является аналогом условного оператора

```

if (выражение) переменная = значение1;
else переменная = значение2;

```

Условный оператор и условное выражение в результате компиляции формируют практически идентичный код. Разница состоит в том, что в случае условного оператора обращение к переменной происходит дважды; следовательно, дважды вычисляется ее адрес, а в случае условной операции – лишь один раз. С другой стороны, с точки зрения понимания программы условный оператор намного лучше.

#### Пример 3.10

<pre> if (test == 'Y')     TestValue = 100; else     TestValue = 0; ... </pre>	<pre> TestValue = (test == 'Y')?100:0; ... </pre>
--	---

Это полностью эквивалентно приведенной слева записи оператора if

#### Пример 3.11

<pre> if (a&gt;b) max = a; else max = b; ... </pre>	<pre> max = (a&gt;b)?a:b; ... </pre>
---	--------------------------------------

### 3.2.5. Оператор множественного выбора

Если в программе необходимо выбрать один из многочисленных вариантов, то вместо вложенных конструкций if – else более целесообразно применять оператор-переключатель switch, иначе называемый *оператором множественного выбора*.

Его синтаксис:

```

switch (выражение)
{ case значение1: оператор1;
  break;
  case значение2: оператор2;
  break;
  case значение3: оператор3;
}

```

```

break;
default: // необязательный компонент
    оператор_по_несравнению; // если не было ни одного совпадения.
} //end switch (выражение)

```

Фигурные скобки, ограничивающие тело оператора switch, являются обязательными. Здесь для выполнения выбирается тот вариант (группа операторов), значение которого совпадает со значением выражения. Как выражение, так и метки (значения) должны иметь значения целого или символьного типа.

Оператор в каждом блоке выбора case может быть отдельным оператором или блоком операторов. В любом случае фигурные скобки, ограничивающие этот блок, не нужны, так как блок выбора всегда должен завершаться оператором break.

Оператор break в каждом блоке выбора case осуществляет выход из оператора switch. Если оператор break отсутствует, т. е. если оператор switch записан в виде

```

switch (выражение) { case значение1: оператор1;
                      case значение2: оператор2;
                      case значение3: оператор3;
                  } // end switch (выражение),

```

то первый же случай сравнения, давший в результате значение «истина», приведет к выполнению оператора «по признаку сравнения» и всех следующих за ним операторов, вплоть до оператора break или до конца оператора switch.

**Пример 3.12.** Ввести с клавиатуры символ. Если он является символом арифметической операции, то указать, какой именно, и привести соответствующий пример. Если не является, то выдать сообщение об этом.

```

#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int main(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "Russian");
    char sym;
    int op1, op2, res;
    cout << "Введите символ арифметической операции ";
    cin >> sym;
    switch (sym)
    {
        case '+':
            cout << "Сложение" << endl;
            cout << "1 слагаемое:" ;
            cin >> op1;
            cout << "2 слагаемое:" ;
            cin >> op2;
            res = op1 + op2;
            cout << op1 << '+' << op2 << '=' << res << endl;
        case '-':
            cout << "Вычитание" << endl;
            cout << "1 вычитаемое:" ;
            cin >> op1;
            cout << "2 вычитаемое:" ;
            cin >> op2;
            res = op1 - op2;
            cout << op1 << '-' << op2 << '=' << res << endl;
        case '*':
            cout << "Умножение" << endl;
            cout << "1 множитель:" ;
            cin >> op1;
            cout << "2 множитель:" ;
            cin >> op2;
            res = op1 * op2;
            cout << op1 << '*' << op2 << '=' << res << endl;
        case '/':
            cout << "Деление" << endl;
            cout << "1 делимое:" ;
            cin >> op1;
            cout << "2 делитель:" ;
            cin >> op2;
            res = op1 / op2;
            cout << op1 << '/' << op2 << '=' << res << endl;
        default:
            cout << "Неизвестная операция" << endl;
    }
}

```

```

break;
case '-':
cout << "Вычитание" << endl;
cout << "Уменьшаемое:" ;
cin >> op1;
cout << "Вычитаемое:" ;
cin >> op2;
res = op1 - op2;
cout << op1 << '-' << op2 << '=' << res << endl;
break;
case '*':
cout << "Умножение" << endl;
cout << "Множимое:" ;
cin >> op1;
cout << "Множитель:" ;
cin >> op2;
res = op1 * op2;
cout << op1 << '*' << op2 << '=' << res << endl;
break;
case '/':
cout << "Деление" << endl;
cout << "Делимое:" ;
cin >> op1;
cout << "Делитель:" ;
cin >> op2;
if (op2 != 0)
{ res = op1 / op2;
  cout << op1 << '/' << op2 << '='
    << res << endl; }
  else cout << "Деление на 0 запрещено" << endl;
break;
default:cout << "Не арифметическая операция" << endl;
} // end switch (sym)
_getch();
return 0;
}

```

### Контрольные вопросы

- Что такое линейные и разветвленные программы?
- В чем заключается понятие форматного вывода данных?
- Как осуществляется ввод данных?
- Каковы формы условного оператора?
- Каким образом строится условное выражение?
- Какую структуру имеет оператор-переключатель?
- С какой целью используется оператор разрыва break?
- Что происходит, если забыть поставить break?
- Определить значение переменной w после выполнения следующих операторов:

w = 100; u = 30;

switch (u/7)

{ case 0: w = 0; break;

```

case 1: w = 1; break;
case 2: w = 2; break;
case 3: w = 3; break;
default: w = 7;
}

```

- Определить значение переменной m после выполнения следующих операторов:

```

m = 5;
if (x>0) { if (y>0) m = 10;
            else m = 20;
      a) при x = -5, y = 7;
      b) при x = 2, y = -3;
      в) при x = 9, y = 3.

```

### Практические задания 3

- Вычислить S=A+B+C+D, если хотя бы одно из чисел A, B, C, D равно нулю, и P=A•B•C•D, если все числа отличны от нуля.

$$z = \begin{cases} a + tg^2 x^3, & a < 0; \\ \sqrt{|a+x^4|}, & a \geq 0. \end{cases}$$

$$g = \begin{cases} \frac{1}{a^{3/5} + x^2 \cdot a \cdot \sin(x+a)}, & \text{если } \sin(x) \geq \ln(a); \\ 2x/a, & \text{если } \sin(x) < \ln(a). \end{cases}$$

$$y = \begin{cases} \frac{x+3^2 - \sin^2 x}{\sqrt[3]{\sin(3a)}}, & \text{если } \sin(a) > x; \\ \frac{e^{\sin ax}}{\cos^2(ax)}, & \text{если } \sin(a) \leq x. \end{cases}$$

- Составить программу, которая при вводе оценки в виде цифры выводит оценку в буквенном виде: 5 – отлично, 4 – хорошо, 3 – удовлетворительно, 2 – неудовлетворительно.

- Составить программу, которая по введенному номеру месяца выводит его название и время года.

- Составить программу, которая при вводе символа определяет, скобка ли это, и указывает, какая именно, например фигурная открывающая (), квадратная закрывающая ()].

- Составить программу, которая при вводе символа выводит либо текст «цифра», если введена цифра, либо текст «латинская буква», если введена латинская буква, либо текст «не цифра и не латинская буква» в остальных случаях.

- Ввести с клавиатуры координаты точек (x1, y1) и (x2, y2) и определить расстояние между этими точками.

10. Поменять местами значения переменных  $x$  и  $y$ .
11. Значения переменных  $a$ ,  $b$ ,  $c$  поменять местами так, чтобы оказалось  $a \geq b \geq c$ .
12. Определить:  $y = \max(\min(a,b), \min(c,d))$ .
13. Определить:  $y = \min(a,b,c)$ .
14. Переменной  $k$  присвоить номер четверти координатной плоскости, в которой находится точка с координатами, введенными с клавиатуры. Отдельно учтеть случаи, когда точка попадает на одну из координатных осей или в начало координат.
15. Написать программу преобразования прописных латинских букв в строчные. При написании программы использовать условное выражение.
16. Написать программу решения квадратного уравнения с произвольными коэффициентами, которые вводятся с клавиатуры.
17. Определить  $d = \max(a, b, c)$ , если значения переменных  $a$ ,  $b$  и  $c$  введены с клавиатуры.

## ГЛАВА 4. ОРГАНИЗАЦИЯ ЦИКЛОВ В ЯЗЫКЕ C++

В практике программирования циклом называют многократное автоматическое выполнение некоторых действий (операторов). Повторяемый фрагмент называют *телом цикла*.

В языке C++ используются операторы цикла трех видов:

- с предварительным условием,
- с последующим условием,
- с параметром.

### 4.1. Оператор цикла с предварительным условием

Оператор цикла с предварительным условием (с предусловием) является наиболее общим по сравнению с другими конструкциями, которые легко моделируются с помощью оператора цикла с предусловием и, вообще говоря, могут считаться несколько избыточными, в основном служащими для удобства программирования.

Оператор цикла с предусловием выполняет повторяющиеся действия до тех пор, пока заданное условие истинно, и выглядит следующим образом:

```
while(условие входа в цикл) оператор;
  если тело цикла состоит из одного оператора или
  while(условие входа в цикл){ оператор1;
    оператор2;
    ...
    операторN;
  } //end while
```

Если тело цикла состоит из блока операторов.

Схема алгоритма цикла с предварительным условием показана на рис. 4.1.



Рис. 4.1. Схема алгоритма цикла с предварительным условием

Если условие входа в цикл сразу ложно, ни один из операторов, обра- зующих тело цикла, не будет выполнен ни одного раза, т. е. цикл будет полностью пропущен.

Тело цикла обязательно должно содержать действия, влияющие на условие входа в цикл. Иначе цикл станет «бесконечным».

**Пример 4.1.** Распечатать значение переменной sym, изменяющейся от 'A' до 'Z', с помощью оператора цикла с предусловием.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");
char sym;
cout << "Выводим на экран алфавит" << endl;
sym = 'A';
while(sym <= 'Z')
{ cout << sym << '';
  sym++;
}
_getch();
return 0;
}
```

**Пример 4.2.** Вычислить среднее арифметическое значение введенной с клавиатуры последовательности целых чисел. Признаком завершения процесса ввода чисел служит ввод нуля.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{ setlocale(LC_ALL, "Russian");
int dig, sum=0, cnt=0, average;
cout << "Введите число ";
cin >> dig;
cnt++;
sum = dig;
while(dig != 0)
{ cout << "Введите число ";
  cin >> dig;
  cnt++;
  sum += dig;
}
cnt--;
if (cnt != 0)
{ average = sum/cnt; // вычисление с точностью до целого
}
```

```
cout << "Среднее арифметическое = " << average << endl;
}
else cout << "Последовательность пустая" << endl;
_getch();
return 0;
}
```

## 4.2. Оператор цикла с последующим условием

Синтаксис оператора цикла с последующим условием (с постусловием) имеет вид:

```
do { оператор;} while (условие продолжения цикла);
или
do { оператор1;
      оператор2;
      ...
      операторN;
} while (условие продолжения цикла);
```

Если в теле цикла содержится только один оператор, то операторные скобки пишутся для устранения неоднозначности, обусловленной наличием while, так как это может интерпретироваться как заголовок структуры while с пустым телом цикла. Схема алгоритма цикла с последующим условием показана на рис. 4.2.



Рис. 4.2. Схема алгоритма цикла с последующим условием

Леттеры, определяемые оператором, выполняются до тех пор, пока условие продолжения цикла не станет ложным или равным нулю. Его основное отличие от оператора while состоит в том, что тело цикла выполняется хотя бы один раз, так как проверка условия продолжения происходит перед выполнением операторов, образующих тело цикла.

Оператор цикла с последующим условием обязательно должно содержать действия, влияющие на условие продолжения цикла, так как иначе цикл становится «бесконечным».

**Пример 4.3.** Вывести на экран значение переменной с, изменяющейся от 'A' до 'Z', с помощью оператора цикла с постусловием.

```
#include "stdafx.h"
#include <iostream>
#include <math.h>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");
char sym;
cout << "Выводим на экран алфавит" << endl;
sym = 'A'-1;
do { sym++;
    cout << sym << '';
} while (sym<'Z');
_getch();
return 0;
}
```

**Пример 4.4.** Забыли написать оператор counter++, влияющий на условие выхода из цикла, и программа «зациклилась».

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{int counter=1;
do { cout << counter << endl;
    // counter++;
} while (counter <10);
_getch();
return 0;
}
```

**Пример 4.5.** Вводить с клавиатуры последовательность символов, пока не будет введена точка. Подсчитать, сколько в этой последовательности скобок.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");
char sym;
int cnt = 0;
do { cout << "Введите символ ";
    cin >> sym;
    if (sym == '{' || sym == '}' || sym == '[' || sym == ']' || sym == '(' || sym == ')')
        cnt++;
}
```

```
} while (sym != '.');
cout << "Количество скобок = " << cnt << endl;
_getch();
return 0;
}
```

### 4.3. Оператор цикла с параметром

Оператор цикла с параметром (оператор for) обычно используется, когда известно, сколько раз должно быть выполнено тело цикла.

Синтаксис оператора:

```
for (выражение1; выражение2; выражение3) оператор;
в случае, если тело цикла состоит из единственного оператора или
for (выражение1; выражение2; выражение3)
(оператор1;
    оператор2;
    .....
    операторN;
```

если тело цикла содержит более одного оператора.

Здесь выражение1 называется *инициализирующим выражением* и выполняется один раз перед тем, как будет произведен анализ выражения2. Выражение2 называется *условием выполнения цикла*. Выражение3, как правило, определяет закон изменения параметра, но это не обязательно.

Ни одно из трех выражений в операторе цикла for не является обязательным. Схема алгоритма цикла с параметром показана на рис. 4.3, а и б.

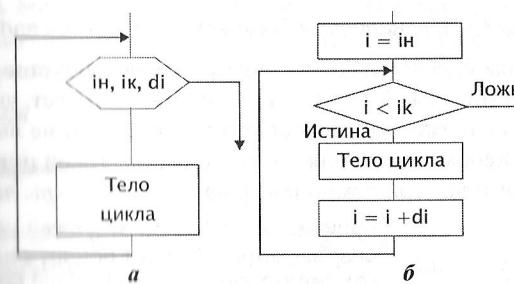


Рис. 4.3. Схемы алгоритмов цикла с параметром

На схемах, представленных на рис. 4.3,  $i$  является счетчиком цикла,  $i_0$  – начальное значение цикла,  $i_k$  – конечное значение цикла,  $di$  – шаг изменения счетчика цикла.

Номер 4.6. Вычислить и вывести на экран значение  $Y=X \cdot \sin X$  на промежутке  $[0, \pi]$  с шагом  $\pi/20$ .

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include <math.h>
#include <conio.h>
using namespace std;
const double Pi=3.14159265;
int _tmain(int argc, _TCHAR* argv[])
{double x, dx=Pi/20;
cout << setw(15) << "Y=X*sinX" << endl;
cout << setw(15) << "X" << setw(15) << "Y" << endl;
for(x=0; x<=Pi; x+=dx) cout << setw(15) << x << setw(15) << x*sin(x) << endl;
_getch();
return 0;
}
```

Можно сделать так, чтобы значение счетчика цикла возрастало в геометрической прогрессии, т. е. чтобы вместо прибавления фиксированного значения на каждом шаге цикла выполнялось умножение:

```
for (d=1; d<15; d=d*2) cout << d << endl;
```

В спецификации оператора цикла for может отсутствовать одно или более выражений (но символы «точка с запятой» должны присутствовать). При этом для корректной работы такого цикла необходимо включить в его тело операторы, которые рано или поздно приведут к завершению его работы:

```
a=2;
for ( n=3; a<=25;) a=a * n;
```

Тело цикла

```
for ( ; ; ) cout << "Мы написали бесконечный цикл" << endl;
```

будет выполняться бесконечное число раз, поскольку пустое условие всегда считается истинным. Этот цикл не инициализирует управляющую переменную, не содержит управляющего выражения и не выполняет никаких действий, необходимых для его завершения. Такой цикл, в котором оставлены пустыми все три компонента, называется *открытым*.

```
for (i=1; i<100; i++) // наличие знака ";" при отсутствии оператора -
    // оператор;
    // ошибка, которая будет обнаружена
    // компилятором С (но не Borland C++)
for (i=1; i<100; i++) ;
    // пустое тело цикла: перед знаком ";" //
    // стоит пробел.
```

### Операция «запятая»

Операция «запятая» увеличивает гибкость использования оператора цикла for, позволяя включать в его спецификацию несколько инициализирующих выражений. Операция «запятая» объединяет два выражения в одно и гарантирует их вычисление в порядке слева направо.

**Пример 4.7.** Пусть тарифы переговорного пункта таковы: первая минута разговора стоит 2 руб., а каждая следующая – 1,5 руб. Определить, сколько стоит 1-минутный, 2-минутный ... 10-минутный разговор.

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include <conio.h>
const int FIRST = 2;
const float NEXT = 1.5;
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{int minutes;
float cost;
for (minutes=1, cost=FIRST; minutes<=10; minutes++, cost+=NEXT)
    cout << setw(4) << minutes << setw(8) << cost << endl;
_getch();
return 0;
}
```

Переменная minutes будет инициализирована прежде переменной cost. Здесь это несущественно, но, если бы выражение для вычисления cost содержало переменную minut, порядок их написания был бы очень важен.

Существует несколько способов прервать выполнение цикла или изменить порядок следования операторов тела цикла.

### 4.4. Оператор break

Оператор break служит для продолжения работы программы после окончания текущего цикла и используется, если после выхода из цикла необходимо выполнить оставшуюся часть программы.

**Пример 4.8**

```
for(;;)
    if (выражение) break;
}
```

Открытый цикл for выполняется «бесконечно», но, став истинным, выражение в операторе if приведет к выполнению оператора break, который обрывает «бесконечное» выполнение цикла и переводит счетчик команд на оператор, следующий за последним из операторов, принадлежащих к телу цикла.

**Пример 4.9.** Найти первое отрицательное число в последовательности из 10 чисел.

```
#include "stdafx.h"
#include <iostream>
```

```
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");
int i, a;
for (i=0; i<10; i++) { cout << "Введите число ";
    cin >> a;
    if (a<0) break; }
if (i<10) // сюда передается управление
// после выполнения оператора break
    cout << "Первое отрицательное число: " << a << endl;
else
    cout << "В последовательности нет отрицательных чисел" << endl;
_getch();
return 0;
}
```

#### 4.5. Оператор continue

Оператор `continue` служит для завершения текущей итерации цикла и перехода к следующей итерации этого же цикла, но не является способом выхода из цикла.

**Пример 4.10.** Найти сумму положительных элементов последовательности из 100 чисел.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");
int i, a, s=0;
for (i=1; i<=100; i++) { cout << "Введите число ";
    cin >> a;
    if (a<=0) continue;
    s+=a;
}
cout << "Сумма положительных элементов: " << s << endl;
_getch();
return 0;
}
```

Это не единственный способ написания программы, но он удачно демонстрирует использование оператора `continue`.

**Пример 4.11.** Написать программу из примера 4.10 без использования оператора `continue`.

```
#include "stdafx.h"
#include <iostream>
```

```
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");
int i, a, s=0;
for (i=1; i<=10; i++) { cout << "Введите число ";
    cin >> a;
    s += a > 0 ? a : 0;
}
cout << "Сумма положительных элементов: " << s << endl;
_getch();
return 0;
}
```

**Пример 4.12.** Вводить с клавиатуры пары положительных чисел: делимое и делитель. Выдать на экран сообщения о каждой паре, где делимое кратно делителю. Признаком завершения работы программы служит ввод отрицательного числа.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");
int d1=1, d2=1;
while(d1>=0 && d2>=0)
    {cout << "Введите делимое ";
    cin >> d1;
    cout << "Введите делитель ";
    cin >> d2;
    if (d2 == 0) continue;
    if (d1 % d2 != 0) continue;
    cout << d1 << " кратно " << d2 << endl;
}
_getch();
return 0;
}
```

#### 4.6. Оператор goto

Оператор безусловного перехода `goto` не относится ни к условным операторам, ни к операторам цикла, однако он тоже позволяет изменить последовательность операторов в программе и передать управление оператору, номинированному **меткой** в соответствии с синтаксисом:

```
метка:
    операторы программы;
    операторы, к которым осуществляется переход;
```

Кажущиеся преимущества, которые дает оператор `goto`, на самом деле весьма сомнительны, так как очень быстро приводят к созданию программ-«спагетти», очень трудно поддающихся отладке и корректировке. Поэтому использование оператора `goto` считается признаком не слишком хорошего стиля программирования. Тем более что нет никаких оснований к использованию этого оператора.

#### 4.7. Останов программы с помощью оператора `exit`

При необходимости остановить программу можно воспользоваться функцией `exit()`, прототип которой определен в заголовочном файле `STDLIB.H`.

Выполнение оператора `exit(аргумент)`; немедленно завершает выполнение программы, закрывает все открытые файлы и выполняет некоторые другие завершающие действия. Значение **аргумента** передается окружающей среде программы и может анализироваться. Как правило, значение аргумента, равное нулю, служит признаком корректного выхода, другие значения являются признаком ошибки и могут служить кодом ошибки.

#### 4.8. Область видимости переменных внутри блока

Фрагмент программы, заключенный в фигурные скобки, называется **блоком**. Тело цикла, заключенное в фигурные скобки, также представляет собой блок.

Если объявить переменную внутри блока, то ее область видимости будет ограничена именно этим блоком, т. е. вне данного блока переменная не существует. Поэтому можно в разных блоках одной и той же программы объявлять разные переменные с одним именем.

Кроме того, аналогичный эффект имеет место при объявлении счетчика циклов непосредственно в операторе цикла:

```
for (int i=0; i<100; i++) тело цикла;
```

Область видимости переменной `i` ограничена оператором `for` и телом цикла. Это очень удобно, поскольку позволяет использовать одну переменную в качестве счетчика нескольких циклов в рамках одной программы.

Однако есть ограничения:

- нельзя использовать вложенные конструкции, изменяющие значение параметра цикла внутри тела этого цикла, например:

```
for (int i=0; i<100; i++)
    for (int i=0; i<25; i++) тело цикла;
```

- ряд трансляторов не ограничивает область видимости таких переменных телом цикла, а делает их видимыми от точки объявления до конца программы.

#### Контрольные вопросы

1. Дайте определение цикла.
2. Каким образом цикл `while` может имитировать цикл `for`?
3. Каким образом цикл `while` может имитировать цикл `do-while`?
4. Почему тело оператора с последующим условием нужно заключать в фигурные скобки, даже если в нем всего один оператор?
5. Почему открытый цикл всегда оказывается бесконечным?
6. Сколько раз будет выполнено тело цикла в приведенных далее фрагментах программ?
  - a) `i = 1;`  
`while (i<5) i = i - 1;`
  - b) `i = 2;`  
`while (i<=7) i = i + 3;`
  - c) `i = 2;`  
`do {i = i - 1;} while (i<=5);`
7. Определить значение переменной `m` после выполнения операторов:
  - a) `m = 0; n = 0;`  
`while (n<3) n++;`  
`m += n;`
  - b) `m = 0; n = 0;`  
`while (n<3) { n++; m += n; }`
  - c) `m = 0;`  
`do {m +=2;} while(m<12);`
8. В каких случаях используются операторы `break`, `continue`, `exit`?
9. Почему в языке C++ нет необходимости применять оператор `goto`?

#### Практические задания

1. Ввести с клавиатуры 30 целых чисел. Определить минимальное значение введенной последовательности.
2. Ввести с клавиатуры 10 чисел (положительных и отрицательных). Определить количество элементов последовательности, больших по абсолютному значению чем введенное первым.
3. Определить значение первого отрицательного элемента последовательности, если дано:  $a_0 = 5; a_i = a_{i-1} - i/2$ .
4. Вводить с клавиатуры отрицательные числа до тех пор, пока не будет введено положительное число или нуль. Найти значение максимального элемента введенной последовательности и его порядковый номер.
5. Ввести с клавиатуры 20 целых чисел. Определить максимальное значение среди четных элементов последовательности.

# ГЛАВА 5. ИСПОЛЬЗОВАНИЕ ОПЕРАТОРОВ ЦИКЛА ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

## 5.1. Вычисление суммы бесконечного ряда с заданной точностью

Часто в математических вычислениях возникает необходимость определить сумму бесконечного ряда с заданной точностью. На такие задачи налагается одно жесткое условие: ряд должен быть сходящимся, т. е. абсолютное значение текущего элемента ряда должно быть меньше абсолютного значения его предыдущего элемента.

В подобных случаях можно пользоваться итерационными циклами, реализуемыми операторами цикла с предварительным условием и с последующим условием. В противном случае условие выхода из цикла никогда не будет выполнено и цикл станет бесконечным.

**Пример 5.1.** Вычислить сумму бесконечного ряда с точностью  $\text{eps} = 0.001$ .

$$S(x) = \sum_{k=2}^{\infty} (-1)^{k-1} \frac{x^k}{(k-1)}, |x| < 1$$

```
#include "stdafx.h"
#include <iostream>
#include <math.h>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    int k;
    double sum=0, eps=0.001, slag, x;
    cout << "Введите значение x<1 ";
    cin >> x;
    k=2; // согласно условию
    slag=5*eps; // этот оператор нужен для входа в цикл
    while (fabs(slag)>eps)
    {
        slag = pow(x,k)/(k-1);
        if ((k-1) % 2 == 0) sum = sum + slag;
        else sum = sum - slag;
        k++;
    }
    cout << "Сумма равна " << sum << endl;
    getch();
    return 0;
}
```

## 5.2. Понятие о рекуррентных формулах

Если определен способ вычисления текущего значения заданной последовательности по предыдущему значению, то говорят, что задана рекуррентная формула вычисления элемента последовательности. В таком способе задания последовательности есть необходимый атрибут – явно заданное начальное значение этой последовательности.

При вычислении элементов заданной рекуррентно последовательности может использоваться любой из операторов цикла в зависимости от поставленной задачи.

**Пример 5.2.** Вычислить 7-й член последовательности, если дана формула  $x_i = x_{i-1} * 2 + i/2$  и известно начальное значение  $x_0 = 2$ .

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "Russian");
    int i;
    double x = 2.0;
    for (i=1; i<=7; i++) x = x * 2 + double(i) / 2;
    cout << "7-й элемент последовательности равен " << x << endl;
    getch();
    return 0;
}
```

Следующий пример, несмотря на внешнее сходство, решается по-другому.

**Пример 5.3.** Вычислить первый отрицательный член последовательности, если дана формула  $x_i = x_{i-1}/3 - i/2$  и известно начальное значение  $x_0 = 17$ .

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "Russian");
    int i=0;
    double x=17;
    while (x>0)
    {
        x = x / 3 - double(i) / 2;
        i++;
    }
}
```

Первый отрицательный элемент равен " << x << endl;  
Его номер " << i << endl;

### 5.3. Использование операторов цикла для решения задач численными методами

#### 5.3.1. Вычисление с заданной точностью $\varepsilon$ корня уравнения $F(x)=0$ методом простых итераций

Пусть корень уравнения находится на отрезке  $[a, b]$  (рис. 5.1). Для использования метода итераций исходное уравнение  $F(x) = 0$  нужно привести к виду  $x = f(x)$ . Если известно начальное приближение к корню  $x = x_1$ , то, подставив его в правую часть уравнения  $x = f(x)$ , получим новое приближение  $x_2 = f(x_1)$ . Затем аналогичным образом получим  $x_3 = f(x_2), \dots, x_{k+1} = f(x_k)$ .

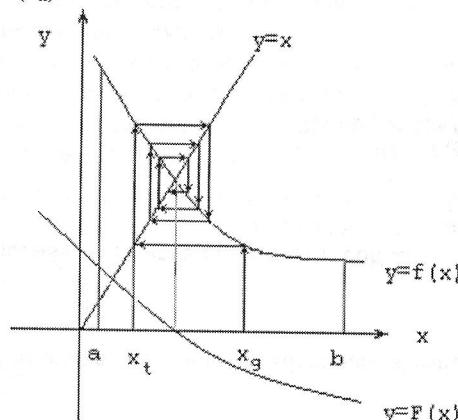


Рис. 5.1. Метод простых итераций

Итерационный процесс сходится к корню уравнения, если  $|f'(x)| < 1$  на отрезке, содержащем корень уравнения. Если выполняется неравенство  $-1 < f'(x) < 0$ , то корень уравнения всегда находится на отрезке  $[x_k, x_{k+1}]$  или  $[x_{k+1}, x_k]$  и условие окончания итерационного процесса имеет вид неравенства  $|x_{k+1} - x_k| < \varepsilon$ .

Переход от уравнения  $F(x) = 0$  к уравнению  $f(x) = 0$  можно осуществить следующим образом. Умножим левую и правую части уравнения  $F(x) = 0$  на произвольную константу  $h$  и добавим к обеим частям уравнения неизвестное  $x$ . Эти действия не изменяют корней уравнения:

$$\begin{aligned} hF(x) + x &= 0 * h + x; \\ hF(x) + x &= x. \end{aligned}$$

Обозначив  $f(x) = hF(x) + x$ , перейдем к уравнению  $x = f(x)$ . Величину  $h$  необходимо выбрать такой, чтобы выполнялись неравенства  $|f'(x)| < 1$ ,  $f'(x) < 0$  на отрезке, содержащем корень уравнения.

Исходными данными для программы, соответствующей приведенному алгоритму, являются грубое значение корня и точность вычисления. Условием выхода из итерационного процесса служит неравенство  $|x_g - x_t| < \varepsilon$ , при этом искомым значением является  $x_t$ .

**Пример 5.4.** Решение уравнения  $x^3 - 2x^2 - 3 = 0$  методом итераций.

Преобразуем уравнение к виду  $x = -0.2(x^3 - 2x^2 - 3) + x$

```
#include "stdafx.h"
#include <iostream>
#include <math.h>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "Russian");
    double xt, xg = 2.3, eps = 0.001, x = 5*eps;
    int step = 0;
    while (fabs(x)>=eps)
    {
        xt = -0.2*(xg*xg*xg - 2*xg*xg - 3) + xg;
        x = xt-xg;
        step++;
        xg = xt;
    }
    cout<<"Корень= "<<xt<< " и получен на шаге " << step;
    getch();
    return 0;
}
```

#### 5.3.2. Решение уравнения $f(x) = 0$ с заданной точностью $\varepsilon$ методом деления отрезка пополам

Метод деления отрезка пополам заключается в следующем. Проверяется наличие корня на отрезке  $[a, b]$  (рис. 5.2). Для этого вычисляются значения функций  $f(a)$  и  $f(b)$ . Если  $f(a)*f(b) > 0$ , то уравнение не имеет корней на заданном отрезке. Если  $f(a)*f(b) < 0$ , т. е. на концах отрезка  $[a, b]$  функция  $f(x)$  имеет противоположные знаки, то искомый корень лежит на этом отрезке. Поиск корня происходит следующим образом. Находим в точке  $a$  значение функции  $y_1 = f(a)$ . Затем определяем значение  $x$  как среднюю точку между  $a$  и  $b$ , вычисляем значение  $y_2 = f(x)$ . Теперь если  $y_1 * y_2 < 0$ , то корень находится на отрезке  $[x, b]$ . Перемещаем точку  $a$  вправо, выполняя присваивание  $a = x$ . Таким образом получаем второй отрезок  $[a, b]$ , но вдвое меньший предыдущего. Процесс деления отрезка продолжаем до тех пор, пока отрезок  $[a, b]$  не станет меньше заданной точности. После этого вычисляем значение  $x = (a+b)/2$ .

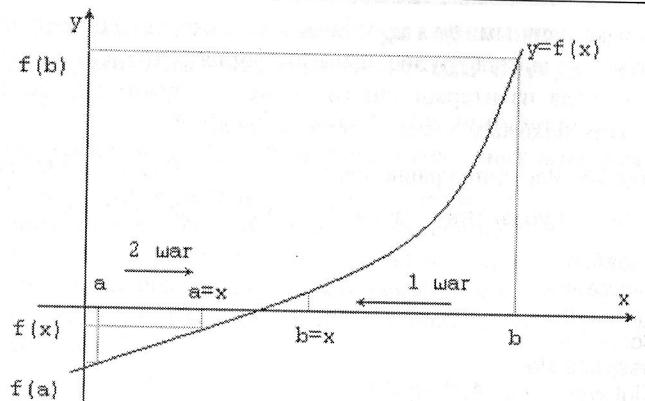


Рис. 5.2. Метод деления отрезка пополам

**Пример 5.5.** Решение уравнения  $x^3 - 2x^2 - 3 = 0$  с заданной точностью  $\varepsilon = 0.01$  методом деления отрезка пополам, если корень находится на отрезке  $[1, 3]$ .

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");
int n = 0;
double x, a = 1, b = 3, y1, y2, eps = 0.001, r, l;
l = a; r = b;
y1 = a*a*a - 2*a*a - 3;
y2 = b*b*b - 2*b*b - 3;
if (y1*y2>0) { cout << "Корней нет" << endl;
    getch();
    exit(1);
}
while ((b - a)>eps)
{ ++n;
    x = (a+b)/2;
    y1 = a*a*a - 2*a*a - 3;
    y2 = x*x*x - 2*x*x - 3;
    if (y1*y2>0) a = x;
    else b = x;
}
x = (a + b)/2;
cout << "Корень уравнения на отрезке " << l << ", " << r << " равен " << x;
cout << " и получен за " << n << " шагов" << endl;
_getch();
return 0;
}
```

## 5.4. Использование операторов цикла для вычисления определенных интегралов

### 5.4.1. Вычисление значения $\int f(x)dx$ с заданной точностью методом прямоугольников

Для вычисления первого приближения интеграла разделим отрезок  $[a, b]$ , отвечающий пределам интегрирования (рис. 5.3) на  $n$  равных частей ( $n = 4$ ), определим значения  $x_i = a + h*i - h/2$ ;  $h = (b - a)/n$ .

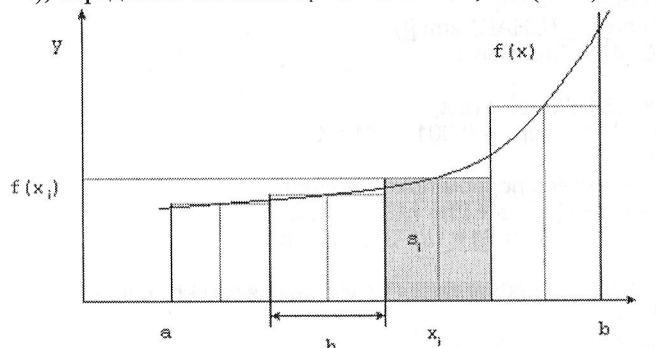


Рис. 5.3. Вычисление интеграла методом прямоугольников

Вычислим площадь одного прямоугольника  $s_i = h * f(x_i)$ . Сумма  $s_i$  площадей полученных прямоугольников является приближенным значением интеграла:

$$S_1 = \sum_{i=1}^n s_i = h \sum_{i=1}^n f(x_i).$$

Однако одно приближение не позволяет оценить точность, с которой вычислено значение интеграла; необходимо найти следующее приближение. Для этого увеличим  $n$  в два раза, т. е.  $n = 2n$ . Аналогично найдем

$$S_2 = h \sum_{i=1}^{2n} f(x_i).$$

Требуется вычислить значение интеграла с точностью  $\varepsilon$ , поэтому примем условие  $|S_1 - S_2| < \varepsilon$ . Если условие выполняется, то  $S_2$  принимается за искомое значение интеграла; если не выполняется, то последнее вычисленное значение  $S_2$  считается предыдущим, т. е.  $S_1 = S_2$ . После этого удвоим число точек деления отрезка и вычислим новое значение  $S_2$ . Процесс удвоения  $n$  и вычисления  $S_2$  будем продолжать до тех пор, пока модуль разности  $S_1$  и  $S_2$  не станет меньше  $\varepsilon$ .

**Пример 5.6.** Вычисление интеграла  $\int_0^{3/2\pi} (1/(5 - 3\cos x)) dx$  методом прямогоугольников.

```
#include "stdafx.h"
#include <iostream>
#include <math.h>
#include <conio.h>
using namespace std;
const double Pi = 3.14159;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");
int i, n = 4;
double a,b,x,h,S1,S2,eps,exact;
a = 0; b = 3/2*Pi; eps = 0.001; S1 = 0;
h = (b - a)/n;
// Вычисляем сумму в первом приближении
for (i = 1; i<=n; i++) { x = a + i*h - h/2;
    S1 = S1+ 1/(5-3*cos(x))*h;
}
// Вычисляем текущее приближение и сравниваем его с предыдущим
do { n = 2*n;
    h = (b - a)/n;
    S2 = 0;
    for (i = 1; i<=n; i++) { x = a + i*h - h/2;
        S2 = S2 + 1/(5-3*cos(x))*h;
    }
    exact = fabs(S1 - S2);
    S1 = S2;
} while(exact>eps);
cout << "S = " << S2 << endl;
_getch();
return 0;
}
```

#### 5.4.2. Вычисление $\int_a^b f(x)dx$ по формуле Симпсона

Для нахождения интеграла вычислим площадь под графиком функции, являющейся подынтегральным выражением. Здесь  $a$  и  $b$  – пределы интегрирования;  $x_i = a + i(b - a)/n$ .

Для использования формулы Симпсона разбиваем отрезок  $[a, b]$  на  $n$  (четное) более мелких отрезков (рис. 5.4).

Формула Симпсона имеет вид:

$$\int_a^b f(x)dx = \frac{b-a}{3n} (f(a) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \dots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(b))$$

где  $n$  – четное число, на которое разбивается интервал интегрирования;  $x_i = a + i(b - a)/n$ .

Алгоритм состоит в циклическом выполнении расчетов  $f(x_i)$ . При этом следует отдельно рассмотреть случаи для границ интегрирования  $f(a)$  и  $f(b)$  и учесть, что при нечетном номере вычисляемого элемента значение функции умножается на 4, при четном – на 2. При конечных значениях отрезка умножение не производится.

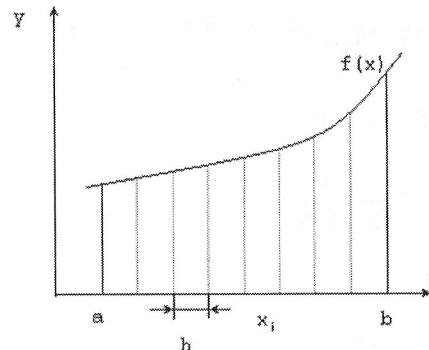


Рис. 5.4. Вычисление интеграла по формуле Симпсона

**Пример 5.7.** Вычисление интеграла  $\int_0^{1.8} 1/(1+\sqrt{x})dx$  по формуле Симпсона.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
const double Pi = 3.14159;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(LC_ALL, "Russian");
int n;
cout << "Четное количество делений ->";
cin << n;
double a,b,x,y,s;
a = 0; b = 1.8; s = 0; x = a;
a/n;
for (i = 1; i<=n; i++)
{ y = 1/(5-3*cos(x))*h;
if (i == 0) s = s + 4*y;
else if (i == 1 || i == n) s = s + y;
else s = s + 2*y;
}
cout << "S = " << s << endl;
_getch();
return 0;
}
```

### Практические задания 5

- Определить значение первого отрицательного элемента последовательности, если дано:  $a_0 = 5$ ;  $a_i = a_{i-1} - i/2$ .
- Ввести с клавиатуры  $x$  и точность вычисления  $Eps$ . Вычислить с заданной точностью сумму:
  - $S(x) = \sum_{k=1}^{\infty} (-1)^k x^{k+1}/k, |x| < 1$ ;
  - $S(x) = \sum_{k=1}^{\infty} (-1)^{k-1}/(x^2 k^3)$ ;
  - $S(x) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{k+1}{(k+1)^3 * x}$ ;
  - $S(x) = \sum_{k=1}^{\infty} \frac{x^{k+1}}{(k+1)!}$ ;
  - $S(x) = \sum_{k=1}^{\infty} (-1)^{2k-1} \frac{x^{2k+1}}{(2k)!}, |x| < 1$ .
- Методом деления отрезка пополам найти приближенное значение корня уравнения  $x^3 + x^2 - 3 = 0$  на интервале  $[0.5, 1.5]$ . Абсолютная погрешность не превышает 0.00001.
- Методом деления отрезка пополам найти приближенное значение корня уравнения  $x^5 - x - 0.2 = 0$  на интервале  $[0.9, 1.1]$ . Абсолютная погрешность не превышает 0.0001.
- По формуле Симпсона вычислить приближенное значение интеграла  $\int_{1.0}^{8.0} (x \sqrt[3]{1+x}) dx$ . Точность не превышает 0.001.
- Методом прямоугольников вычислить приближенное значение интеграла  $\int_0^{3/2} \frac{dx}{5 - 3\cos x}$ . Точность не превышает 0.001.
- Методом деления отрезка пополам и методом итераций найти приближенное значение корня уравнения  $2x^3 + 3x - 1 = 0$  на интервале  $[0, 0.5]$ . Абсолютная погрешность не превышает  $10^{-4}$ . Сравнить методы вычисления.
- По формуле Симпсона и методом прямоугольников вычислить приближенное значение интеграла  $\int_0^p \cos x / (x+2) dx$ . Точность не превышает 0.001. Сравнить методы вычисления.
- Методом деления отрезка пополам и методом итераций найти приближенное значение корня уравнения  $x^4 + 2x^3 - x - 1 = 0$  на интервале

- [0, 1]. Абсолютная погрешность не превышает 0.00015. Сравнить методы вычисления.
- По формуле Симпсона и методом прямоугольников вычислить приближенное значение интеграла  $\int_0^1 dx / (1+x^2)$ . Точность не превышает 0.001. Сравнить методы вычисления.