

#### 4.3.6. Ввод и вывод данных

В языке C++ нет встроенных средств ввода и вывода – он осуществляется с помощью функций, типов и объектов, которые находятся в стандартных библиотеках. Существует два основных способа: функции, унаследованные из C и объекты C++.

##### 1.1 Функции ввода – вывода языка C

Язык C++ унаследовал от языка C библиотеку стандартных функций потокового ввода – вывода.

Под потоком понимается процесс ввода или вывода информации из или во внешний источник информации, представляющий собой развёрнутую во времени последовательность байтов.

Когда программа начинает выполняться, для неё автоматически открываются три стандартных потока ввода – вывода, представленных в табл. 3.6.

Таблица 3.6

Стандартные потоки ввода - вывода		
Стандартный поток	Назначение	Устройство по умолчанию
stdin	Стандартное устройство ввода	Клавиатура
stdout	Стандартное устройство вывода	Экран
stderr	Стандартное устройство выдачи сообщений об ошибках	Экран

Для того, чтобы можно было использовать любой из этих потоков и описанные ниже функции, в программу должна быть включена директива:

**#include <stdio.h>**

Функции *printf()* для вывода и *scanf()* для ввода дают возможность при выполнении операций вывода или ввода преобразовывать числовые величины в их символьное представление и обратно, а также работать с текстовой информацией. Преобразование информации осуществляется этими функциями под управлением форматных строк.

##### 1.1.1 Форматный вывод для стандартных потоков вывода

###### 1. *printf* (форматная строка, список аргументов);

*Форматная строка* – строка символов, заключённых в кавычки, которая показывает, как должны быть напечатаны аргументы.

###### Пример 19

*printf* (“Значение числа  $\Pi$  равно %f\n”, pi);

Форматная строка может содержать

- 1) символы печатаемые текстуально;
- 2) спецификации преобразования;
- 3) управляющие символы.

Каждому аргументу соответствует своя спецификация преобразования:

**%d, %i** – десятичное целое число;

**%f** – число с плавающей точкой;

**%e, %E** – число с плавающей точкой в экспоненциальной форме;

**%u** – десятичное число в беззнаковой форме;

**%c** – символ;

**%s** – строка.

В форматную строку также могут входить **управляющие символы**:

**\n** – управляющий символ новая строка;

**\t** – табуляция;

**\a** – звуковой сигнал и др.

Также в форматной строке могут использоваться модификаторы формата, которые управляют шириной поля, отводимого для размещения выводимого значения. Модификаторы – это числа, которые указывают минимальное количество позиций для вывода значения и количество позиций для вывода дробной части числа:

**%[j]m[.p]C**,

где **j** – задает выравнивание по левому краю; **m** – минимальная ширина поля; **p** – количество цифр после запятой (точность) для чисел с плавающей точкой (если точность не указана, выводится шесть цифр) и минимальное количество выводимых цифр для целых чисел (если цифр в числе меньше, чем значение p, то выводятся начальные нули);

**C** – спецификация формата вывода.

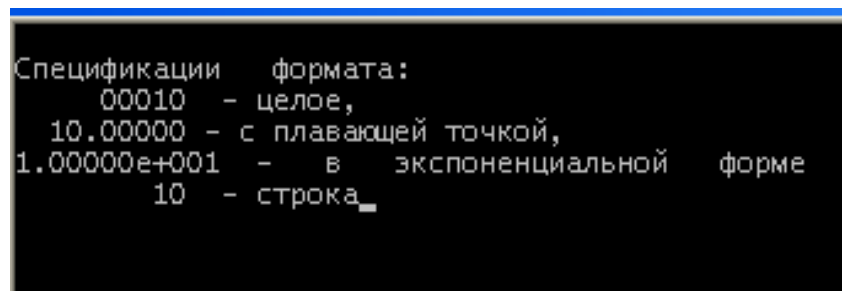
Второе необязательное поле **m** спецификации задаёт ширину поля для вывода. Если поле задано, оно должно содержать ненулевое десятичное целое, задающее минимальное число символов, которые должны быть выведены для представления значения соответствующего аргумента. Если ширина больше, чем нужно, необходимое число пробелов добавляется к выводу – справа или слева в зависимости от установленного выравнивания. Если значение ширины начинается с цифры 0, тогда эта цифра будет использоваться вместо пробела как символ-заполнитель.

Если заданная ширина недостаточна для вывода значения, выводятся все символы, не глядя на заданную ширину.

### Пример 20

```
printf("\nСпецификации формата:\n%10.5d - целое,\n%10.5f – с плавающей  
точкой,\n%10.5e – в экспоненциальной форме \n%10s – строка",10,10.0,10.0,"10");
```

Будет выведено:



```
Спецификации формата:
00010 - целое,
10.00000 - с плавающей точкой,
1.00000e+001 - в экспоненциальной форме
10 - строка_
```

## 1.1.2 Форматный ввод для стандартных потоков ввода

Функция **scanf** представляет собой аналог функции **printf** для ввода; в ней предусмотрены многие из описанных преобразований, но выполняемых в обратном направлении. Вид функции:

**scanf (форматная строка, список аргументов);**

В качестве аргументов используются адреса переменных.

### Пример 21

```
scanf(" %d%f ", &x,&y);
```

Функция читает символы из стандартного потока ввода, интерпретирует их в соответствии с форматом и записывает результаты в соответствующие параметры. Все параметры должны задаваться с помощью указателей на соответствующий тип данных.

Формат обычно содержит спецификации преобразования и используется для непосредственной интерпретации входной последовательности.

**Пример:**

```
int i;  
float x;  
char name[50];  
scanf("%d%f%s",&i, &x, name);
```

С входной строкой

25      54.32E-1      Thompson

присвоит 25 – i,      5.432 – x,      "Thompson\0" – name.

Функция *scanf* останавливается по исчерпанию строки формата либо при несоответствии входной информации спецификациям формата.

Если данные, прочитанные с помощью *scanf*, не соответствуют строке формата, то функция может вести себя непредсказуемо. Поэтому часто программисты делают выбор в пользу функции *gets* (описывается в библиотечном файле *stdio.h*) вместо мощной, но требовательной *scanf*.

Функции *gets* читает вводимые данные в указанный буфер. Данные представляются в виде строки. Если должно быть введено число, то можно затем вызвать функцию *atoi*, *atol*, *atof* (описывается в библиотечном файле *stdlib.h*) для преобразования строки соответственно в целое, длинное целое или вещественное число.

**Пример 22** Пример, иллюстрирующий ввод-вывод.

```
int main()  
{  
    SetConsoleOutputCP(1251); //функция для вывода текста кириллицей  
    char name[80], ageStr[80];  
    int age=0;  
    printf("Пожалуйста введите Ваше имя:");  
    gets(name);  
    printf("Привет, %s! Сколько Вам лет?", name);  
    gets(ageStr);  
    age=atoi(ageStr);  
    if (age>0 && age<=30) printf(" %s! Да Вам всего лишь %d лет?", name,age);  
    else printf(" %s! Вы большой молодец?", name);  
    return 0;  
}
```

Будет выведено:

Пожалуйста введите Ваше имя: Иван  
Привет, Иван! Сколько Вам лет? 25  
Иван! Да Вам всего лишь 25 лет?

## Операторы ввода – вывода C++

При использовании библиотеки классов C++, используется библиотечный файл *iostream*, в котором определены стандартные потоки ввода данных от клавиатуры *cin* и вывода данных на экран дисплея *cout*, а также соответствующие операции:

- 1) << – операция записи данных в поток;
- 2) >> – операция чтения данных из потока.

**Операторы ввода (>>) и вывода (<<)** не являются встроенными для языка C++, а обеспечиваются стандартной библиотекой с помощью потоков ввода – вывода. При запуске программы на выполнение автоматически открываются три стандартных потока языка C++:

**cin** – стандартный поток ввода (с клавиатуры);

**cout** – стандартный поток вывода (на дисплей);

**cerr** – стандартный поток для выдачи сообщений (на дисплей). Вместо потока *cout* для вывода сообщений об ошибках может использоваться поток *cerr*.

Для выполнения операторов потокового ввода – вывода в программу должна быть включена инструкция препроцессора:

```
#include <iostream> // подключить файл, где объявлены cin и cout
```

Для возможности использования кириллицы в программу должна быть включена инструкция препроцессора:

```
#include <windows.h> // файл, где определена функция SetConsoleOutputCP(1251)
```

или

```
#include <locale> // файл, где определена функция setlocale(LC_CTYPE, "rus")
```

**Стандартным потоком для оператора ввода (>>) является поток cin**, который обычно представляет клавиатуру компьютера. Имя этого потока размещается слева от оператора ввода. Тип операнда справа от оператора определяет способ интерпретации вводимых с клавиатуры символов, а сам операнд задаёт то, куда будут записываться значения из входного потока.

**Пример 16.** Следующая функция считывает набранное на клавиатуре целое число (например, 4321) и помещает его в переменную *i*, затем вводит в переменную *d* число с плавающей точкой (например, 12.34E5), после чего считывает набранную на клавиатуре строку символов (например, имя “Валентин”).

```
void f()
{
    int i;
    double d;
    char name[20];
    cin >> i;      // считывание в i целого числа
    cin >> d;      // считывание в d числа с плавающей точкой удвоенной
                  // точности
    cin >> name;   // считывание в массив name строки символов
}
```

Три последние инструкции приведенной выше функции, если не учитывать комментарии, можно заменить одной строкой:

```
cin >> i >> d >> name;
```

В языке C++ допускается, задав один раз поток ввода, использовать несколько операторов ввода, каждый раз задавая справа от них по одной переменной, возможно, различных типов.

При вводе данных нет необходимости ставить перед именами переменных оператор взятия адреса **&**. Оператор ввода **>>** берёт на себя задачу вычисления адреса, определение формата вводимых данных, преобразование символьного представления информации в двоичное.

**Стандартным потоком для оператора вывода (<<) является поток cout,** представляющий собой дисплей компьютера. Вместо потока **cout** для вывода сообщений об ошибках может использоваться поток **cerr**. Имя потока вывода размещается слева от оператора вывода, а справа от оператора помещается выражение, значение которого требуется поместить в поток. Перед выдачей значение выводимого выражения преобразуется в последовательность символов.

Например:

```
void f()
{
    cout << 10;
}
```

Эта функция поместит символ 1, а затем символ 0 в стандартный поток вывода, т. е. на экран в строку, на которую указывает курсор.

Можно выводить значения различных типов, например:

**Пример 17.** Вывод значений различных типов:

```
void h(int i)
{
    cout << "Значение i равно "; // вывод текста
    cout << i;                    // вывод целого
    cout << '\n';                 // помещает в выходной поток один символ –
                                //переход на новую строку
}
```

Три последние инструкции приведенной выше функции, если не учитывать комментарии, можно заменить одной строкой:

```
cout << "Значение i равно " << i << '\n';
```

**При выводе отдельных символов** каждый символ должен заключаться в **одиночные кавычки** в отличие от вывода строк символов, когда **выводимая строка** заключается в **двойные кавычки**.

**Пример 18.** Совместное использование операторов ввода и вывода:

```
#include <iostream> // подключить файл, где объявлены cin и cout
using namespace std;
void main()
{
    char name[20];
    cout << "Пожалуйста, введите Ваше имя.\n"; cin >> name;
    cout << "Привет, " << name << "!\n";
}
```

## 4.4. Основные операторы языка C++

### 4.4.1. Базовые конструкции структурного программирования

В теории программирования доказано, что программу для решения задачи любой сложности можно составить только из трех структур: линейной, разветвляющейся и циклической. Эти структуры называются базовыми конструкциями структурного программирования.

**Линейной** называется конструкция, представляющая собой последовательное соединение двух или более операторов.

**Ветвление** – задает выполнение одного из двух операторов, в зависимости от выполнения какого либо условия.

**Цикл** – задает многократное выполнение оператора.

Целью использования базовых конструкций является получение программы простой структуры. Такую программу легко читать, отлаживать и при необходимости вносить в нее изменения. Операторы управления работой программы называют управляющими конструкциями программы. К ним относят:

- 1) составные операторы;
- 2) операторы выбора;
- 3) операторы циклов;
- 4) операторы перехода.

### 4.4.2. Оператор «выражение» (*пустой оператор*)

Любое выражение, заканчивающееся точкой с запятой, рассматривается как оператор, выполнение которого заключается в вычислении этого выражения.

Частным случаем выражения является *пустой оператор* «;».

Пустая инструкция состоит только из точки с запятой. При выполнении этой инструкции ничего не происходит. Она обычно используется в следующих случаях:

- в инструкциях *do, for, while, if* в строках, когда по смыслу инструкция не нужна, но по синтаксису требуется хотя бы одна инструкция;
- при необходимости пометить фигурную скобку.

Синтаксис языка C++ требует, чтобы после идентификатора с двоеточием обязательно следовала инструкция. Фигурная же скобка инструкцией не является. Поэтому, если надо передать управление на блок инструкций, необходимо использовать пустую инструкцию.

#### Пример 22

```
int main( )
{
    ...
    {
        if(...) goto a; // переход на скобку
    }
    ...
}
a;;
```

```

}
return 0;
}

```

#### 4.4.3. Составные операторы

К составным операторам относят собственно составные операторы и блоки. В обоих случаях это последовательность операторов, заключенная в фигурные скобки. *Блок отличается от составного оператора наличием определений в теле блока.*

**Пример 23.** Составные операторы.

```

{
n++;           // это составной оператор
summa+=n;
}
{
int n=0;
n++;           // это блок
summa+=n;
}

```

*Блок* представляет собой последовательность инструкций, включая, возможно, объявления, заключенных в фигурные скобки:

```

{
[ объявления ]
...
инструкция [ инструкция ]
...
}

```

В конце блока точка с запятой не ставится.

Объявления в блоке могут располагаться не обязательно перед исполнительными инструкциями. Объявление локальной переменной можно разместить в любом месте программы перед первым обращением к этой переменной.

*Блоки инструкций могут быть вложены друг в друга на любую глубину.* На переменную, объявленную внутри блока, можно ссылаться только внутри этого или внутри вложенных блоков (если эта переменная не скрыта переменной с таким же именем внутри вложенного блока).

Выполнение блока заключается в последовательном выполнении составляющих его инструкций, включая вложенные в него блоки:

**Пример 23**

```

int main ()
{
int q, b;
double t, d;
...
if (...)
{
int e, g;
double f, q;

```

```

...
}
...
return (0);
}

```

Переменные e, g, f, q будут уничтожены после выполнения блока. Отметим, что переменная q является локальной в блоке, т. е. она никоим образом не связана с переменной q, объявленной в начале функции main() с типом int. Отметим также, что выражение, стоящее после return, может быть заключено в круглые скобки, хотя их наличие необязательно.

Заметим, что тело любой функции представляет собой одну инструкцию или блок инструкций, включая случаи пустой инструкции и вложенных на несколько уровней блоков инструкций.

#### 4.4.4. Операторы выбора

*Операторы выбора* – это условный оператор и переключатель.

1. **Условный оператор** имеет полную и сокращенную форму.

- **Сокращённая форма:**

**if (выражение-условие)**  
**инструкция;**

В качестве **выражения-условия** могут использоваться арифметическое выражение, отношение и логическое выражение. Если значение *выражения-условия* отлично от нуля (т.е. истинно), то выполняется **инструкция (блок)**.

Алгоритмическая структура, соответствующая этому оператору, показана на рис.2.2.

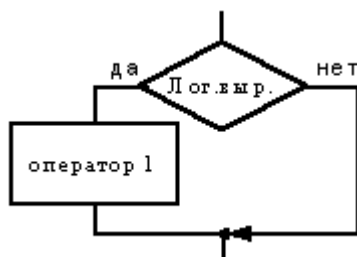


Рис. 2.2. Алгоритмическая структура выполнения краткой формы оператора ветвления

**Пример 24.** Сокращённая форма условного оператора. Поиск минимального значения  
**if** (x<y&&x<z) min=x;

- **Полная форма:**

**if (выражение-условие)**  
**инструкция 1;**  
**else**  
**инструкция 2;**



Если значение **выражения-условия** отлично от нуля, выполняется **инструкция1 (блок1)**, при нулевом значении выражения-условия выполняется **инструкция 2 (блок2)**.

Алгоритмическая структура, соответствующая этому оператору, показана на рис.2.3.

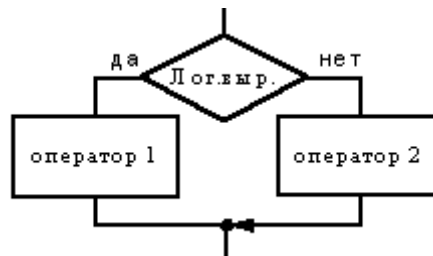


Рис. 2.3. Алгоритмическая структура выполнения полной формы оператора ветвления

**Пример 28.** Полная форма условного оператора. Вычисление корней квадратного уравнения

```
if (d>=0)
{
    x1=(-b-sqrt(d))/(2*a);
    x2=(-b+sqrt(d))/(2*a);
    cout<< "\nx1="<<x1<<"x2="<<x2;
}
else cout<<"\nРешения нет";
```

Под *инструкциями* в приведенных форматах понимаются любые одиночные инструкции или блоки инструкций (возможно, включающие в себя вложенные блоки инструкций).

Условие в обоих вариантах инструкции заключается в круглые скобки. Оно может быть любым выражением, значение которого после вычисления может быть приведено к логическому типу.

**Допускается использование вложенных инструкций if.** Инструкция if может быть включена в конструкцию if или в конструкцию else другой инструкции if:

**Пример 25.**

```
int main ( )
{
    int t = 2, b = 7, r = 3;
    if (t > b)
    {
        if (b < r) r = b;
    }
    else r = t;
    return 0;
}
```

В результате выполнения программы перед инструкцией возврата значение переменной r станет равным 2.

## 2 Переключатель (инструкция выбора).

Инструкция **switch** предназначена для организации выбора из множества различных вариантов.

Формат инструкции следующий:

```
switch (выбирающее выражение)
{
    объявления
    case константа 1:
        последовательность инструкций 1;
    case константа 2:
        последовательность инструкций 2;
    ...
    case константа N:
        последовательность инструкций N;
    default:
        последовательность инструкций;
}
```

Схема выполнения инструкции **switch** следующая:

- вычисляется выбирающее выражение в круглых скобках;
- вычисленное значение последовательно сравнивается с константами, следующими за ключевыми словами **case**;
- если одна из констант совпадает со значением выражения, то начинает выполняться последовательность инструкций, следующая за соответствующим ключевым словом **case**; если ни одна из констант не равна выражению, то начинает выполняться последовательность инструкций, следующая за ключевым словом **default**, а в случае его отсутствия управление передается на следующую после **switch** инструкцию.

Выбирающее выражение в круглых скобках, следующее за ключевым словом **switch**, может быть любым выражением, допустимым в языке C++. *Значение этого выражения должно быть целым числом*, по которому осуществляется выбор из нескольких вариантов.

Тело инструкции **switch** состоит из нескольких последовательностей инструкций, начинающихся ключевым словом **case** с последующим константным выражением, завершающимся двоеточием.

Так как константное выражение вычисляется во время компиляции, оно не может содержать переменные или вызовы функций. *Обычно в качестве константного выражения используются целые или символьные константы.*

*Все значения константных выражений в инструкции switch должны быть уникальны.* Кроме последовательностей инструкций, помеченных ключевым словом **case**, может быть, но обязательно одна, последовательность с начальным ключевым словом **default** с двоеточием.

*Каждая из последовательностей инструкций может быть пустой, либо содержать одну или более инструкций; не требуется заключать эти последовательности инструкций в фигурные скобки.*

Отметим также, что в инструкции **switch** можно использовать свои локальные переменные, объявления которых должны находиться перед первым ключевым словом **case**; однако в объявлениях не должна использоваться инициализация.

*Все последовательности инструкций, следующие за case, должны заканчиваться одной из инструкцией: break, return или goto; как правило, используется break.* Если же удалить эту инструкцию, допустим, из последовательности инструкций 1, то после неё начнёт выполняться последовательность 2 и т. д.

### Пример 26.

```
int i = 2;
switch (i)
{
    case 1: i += 2;
    case 2: i *= 3;
    case 0: i /= 2;
    case 4: i -= 5;
    default: ;
}
```

Выполнение инструкции *switch* начинается с инструкции, помеченной *case 2*. Таким образом, переменная *i* получает значение, равное 6, далее выполнится инструкция, помеченная ключевым словом *case 0*, а затем - *case 4*. Переменная *i* примет значение 3, а затем значение -2. Инструкция, помеченная ключевым словом *default*, в данном случае не изменит значения переменной.

Последовательности инструкций (точнее, их отсутствие) без прерывающих инструкций могут использоваться для выполнения одних и тех же действий для нескольких вариантов выбора:

### Пример 27.

```
char ch;
int aCount = 0;
int bCount = 0;
...
switch (ch)
{
    case 'a':
    case 'A':
        ++aCount;
        break;
    case 'b':
    case 'B':
        ++bCount;
        break;
}
```

Использование инструкции ***break*** позволяет в необходимый момент прервать последовательность выполняемых инструкций в теле *switch* путем передачи управления инструкции, следующей за *switch*.

### Пример 32.

```
char OP;
int x, y, z;
switch (OP)
{
    case '+': x = y + z; break;
    case '-': x = y - z; break;
    case '*': x = y * z; break;
    case '/': x = y / z; break;
    default: ;
}
```

Отметим, что в теле инструкции *switch* можно использовать вложенные инструкции *switch*, при этом с ключевыми словами *case* можно использовать одинаковые константные выражения.

**Пример 33.**

```
...
switch (a)
{
    case 1: b = c; break;
    case 2:
        switch (d)
        {
            case 0: f = s; break;
            case 1: f = 9; break;
            case 2: f -= 9; break;
        }
    case 3: b -= c; break;
    ...
}
```

## 2. ПРИМЕРЫ ПРОГРАММИРОВАНИЯ АЛГОРИТМОВ РАЗВЕТВЛЯЮЩЕЙСЯ СТРУКТУРЫ

**Задача 2.1.** Определить, попадет ли точка с координатами (x, y) в ограниченную область (рис. 2.3).

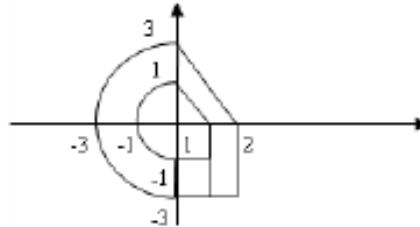


Рис. 2.3. Границы области для задачи 1.1.

Исходные данные: x, y

Результат: да (1) или нет (0)

### Математическая модель:

$Ok = I \parallel II \parallel III \parallel VI$ , где I, II, III, IV – условия попадания точки в заштрихованную область для каждого квадранта.

**Квадрант I:** Область формируется прямыми  $OX$  и  $OY$ , прямой, проходящей через точки  $(0,1)$  и  $(1,0)$  и прямой, проходящей через точки  $(0,3)$  и  $(2,0)$ .

Необходимо определить уравнения прямых  $y = a x + b$ . Решаем две системы уравнений:

$$\begin{cases} 1 = a * 0 + b \\ 0 = a * 1 + b \end{cases} \quad \begin{cases} 3 = a * 0 + b \\ 0 = a * 2 + b \end{cases}$$

Из этих систем получаем следующие уравнения прямых:

$$y = -x + 1;$$

$$y = -\frac{3}{2}x + 3;$$

Тогда условие попадания точки в I квадрант будет выглядеть следующим образом:

$$y \geq -x + 1 \ \&\& \ y \leq -\frac{3}{2}x + 3 \ \&\& \ y \geq 0 \ \&\& \ x \geq 0.$$

**Квадранты II и III:** Область формируется прямыми  $OX$  и  $OY$  и двумя окружностями, описываемыми формулами  $x^2 + y^2 = 1$ ,  $x^2 + y^2 = 9$ .

Тогда условие попадания точки во II и III квадранты будет выглядеть следующим образом:

$$x^2 + y^2 \geq 1 \ \&\& \ x^2 + y^2 \leq 9 \ \&\& \ x \leq 0.$$

### Квадрант IV:

Область формируется двумя прямоугольниками. Точка может попадать либо в первый прямоугольник, либо во второй.

Условие попадания точки в IV квадрант будет выглядеть следующим образом:

$$(x \geq 0 \ \&\& \ x \leq 1 \ \&\& \ y \leq -1 \ \&\& \ y \geq -3) \parallel (x \geq 1 \ \&\& \ x \leq 2 \ \&\& \ y \leq 0 \ \&\& \ y \geq -3).$$

Вариант 1. Программа имеет вид (без использования разветвляющейся структуры):

```
#include <iostream>
#include <math.h>
using namespace std;
void main()
```

```

{
    float x,y;
    cout<<"Введите x,y\n";
    cin>>x>>y;
    bool Ok=(y>=-x+1 && y<=-3/2*x+3 && x>=0 && y>=0)//
    (pow(x,2)+pow(y,2)>=1 && pow(x,2)+pow(y,2)<=9 && x<=0)//
    (x>=0 && x<=1 && y<=-1 && y>=-3)//
    (x>=1 && x<=2 && y<=0 && y>=-3);
    cout<<"\n"<<Ok;
}

```

Вариант 2. Программа имеет вид (с использованием условной операции):

```

void main()
{
    float x,y;
    cout<<"Введите x,y\n";
    cin>>x>>y;
    ((y>=-x+1 && y<=-3/2*x+3 && x>=0 && y>=0)//
    (pow(x,2)+pow(y,2)>=1 && pow(x,2)+pow(y,2)<=9 && x<=0)//
    (x>=0 && x<=1 && y<=-1 && y>=-3)//
    (x>=1 && x<=2 && y<=0 && y>=-3)) ?
    cout<<"точка попадает в область\n":
    cout<<"точка не попадает в область\n";
}

```

Вариант 3. Программа имеет вид (с использованием оператора ветвления):

```

void main()
{
    float x,y;
    cout<<"Введите x,y\n";
    cin>>x>>y;
    if ((y>=-x+1 && y<=-3/2*x+3 && x>=0 && y>=0)//
    (pow(x,2)+pow(y,2)>=1 && pow(x,2)+pow(y,2)<=9 && x<=0)//
    (x>=0 && x<=1 && y<=-1 && y>=-3)//
    (x>=1 && x<=2 && y<=0 && y>=-3))
    cout<<"точка попадает в область\n";
    else cout<<"точка не попадает в область\n";
}

```

Тесты приведены в табл. 2.1.

Таблица 2.1

Тесты к задаче 2.1		
Квадрант	Исходные данные (X,Y)	Результат (Ok)
I	0.2, 0.2	0
I	0.7, 0.5	1
II	-0.5, 0.5	0
II	-2, 0	1
III	-0.5, -0.5	0
III	-2, -1	1
IV	0.5, -0.5	0
IV	1.5, -1	1
Центр системы координат	0, 0	0

**Пример.** Пусть  $a = 5$ . Тогда в результате выполнения операторов:

**if** ( $a < 7$ )

{  $b = a - 2$ ;  $c = 1 + 2 * a$ ; }

**else**

{  $b = 2 + 5 * a$ ;  $c = 12 - 4 * (a - 3)$ ; }

получим  $b = 3$ ,  $c = 11$ .

**Задание 2.1.** Вычислите значение  $b$  и  $c$ , если  $a = 8$ .

**Задача 2.2.** Вычислить и вывести значения функции  $y$  в некоторой заданной точке  $x$ , если

$$y = \begin{cases} \ln|x|, & x < -1, \\ \sin(x), & -1 \leq x < 1, \\ \cos(x), & x \geq 1. \end{cases}$$

Схема алгоритма решения представлена на рис. 2.2.

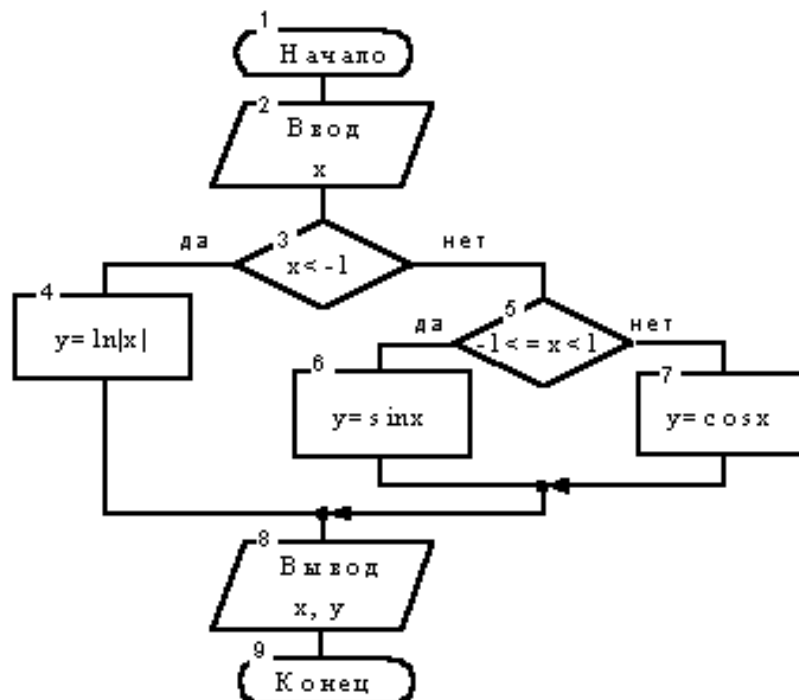


Рис. 2.2. Схема алгоритма решения задачи 2.1

Программа имеет вид:

```
#include <iostream>
#include <math.h>
using namespace std;
void main()
{
    double x,y;
    cout<<"\nEnter x";
    cin>>x;
    if (x<-1) y = log(fabs(x));
    else
        if (x>= -1 && x< 1) y = sin(x);
        else y = cos(x);
    cout<<"\n" << "x =" << x << " y =" << y;
}
```

**Пример.** Пусть  $x = 25$ . Тогда в результате выполнения операторов  
`if (x > 12) y = 2 * x; z = 10;`  
`if (x < 5) z = x / 2;`  
получим  $y = 50, z = 10$ .

**Задание 2.2.** Решить задачу 2.1, используя краткую форму условного оператора.

**Задача 2.3.** Значения переменных X, Y, Z поменять местами так, чтобы они оказались упорядоченными по возрастанию.

Программа имеет вид:

```
#include <iostream>
#include <windows.h>
using namespace std;
int main()
{
    SetConsoleOutputCP(1251);
    cout<<"Введите X, Y, Z: ";
    int X, Y, Z;
    cin>>X>>Y>>Z;
    if(Y < X)
    {
        int tmp = X;
        X = Y;
        Y = tmp;
    }
    if(Z < X)
    {
        int tmp = Z;
        Z = Y;
        Y = X;
        X = tmp;
    }
    if(Z < Y)
    {
        int tmp = Y;
        Y = Z;
        Z = tmp;
    }
    cout<<"Упорядоченные по возрастанию числа: "<<X<<" "<<Y<<" "<<Z<<"\n";
    return 0;
}
```

**Задание 2.3.** Составьте схему алгоритма задачи 2.2.

**Задача 2.4.** Написать программу, которая определяет, является ли введенное число – счастливым (Счастливым считается шестизначное число, у которого сумма первых 3 цифр равна сумме вторых трех цифр). Если пользователь ввел не шестизначное число, сообщить об ошибке.

Программа имеет вид:

```
#include <iostream>
#include <locale>
using namespace std;
void main()
{
    setlocale (LC_CTYPE,"rus");
    int num, d1,d2,d3,d4,d5,d6;
    cout<<"Введите шестизначное число = ";
    cin>>num;
```



```

if (num/1000000!=0||num/100000==0)
    cout<<"Вы ввели не шестизначное число\n";
else
{
    d1=num/100000;
    d2=num/10000%10;
    d3=num/1000%10;
    d4=num/100%10;
    d5=num/10%10;
    d6=num%10;
    if (d1+d2+d3==d4+d5+d6)
        cout<<"Число счастливое\n";
    else
        cout<<"Число несчастливое\n";
}
}

```

**Задача 2.5.** Ввести несколько вариантов значений коэффициентов квадратного уравнения  $ax^2+bx+c=0$ ,  $a \neq 0$ . Вывести сообщение о наличии действительных корней для каждого варианта.

Программа имеет вид:

```

#include <iostream>
#include <conio.h>    // файл, где определена функция getch()
#include <locale>
using namespace std;
void main()
{
    setlocale (LC_CTYPE, "rus");
    double a, b, c, d;
    mes:cout<<"Введите коэффициенты квадратного уравнения = ";
    cin>>a>>b>>c;
    if (a == 0)  { cout<<"Данное уравнение не является квадратным\n"; goto finish; }
    d=b*b-4*a*c;
    if (d >=0) cout<<"Данное уравнение имеет действительные корни\n";
    else
    {
        cout<<"Данные введены некорректно\n";
        cout<<"Уравнение действительных решений не имеет\n";
        goto mes;
    }
    finish: getch();    // ждать нажатия любой клавиши
}

```

**Задача 2.6.** Клиент покупает билеты в количестве N штук в определенную зону (А В С D), по 20, 30, 50 и 100 руб. соответственно, если клиент купит, больше чем на 2000 руб, то скидка 3%, больше 5000 – 5%.

Программа имеет вид:

```

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

```

```

void main()
{
    setlocale (LC_CTYPE,"rus");
    char a;  int pb, numb;  double oplata, disc=0;
    cout<<"В какую зону хотите купить билеты? A,B,C,D?";
    cin>>a;
    if (a=='a' || a=='A' || a=='b' || a=='B' || a=='c' || a=='C' || a=='d' || a=='D')
    { switch (a)
        { case 'a':
          case 'A':
              cout<<"Вы выбрали зону A\n";
              pb=20;
              break;
          case 'b':
          case 'B':
              cout<<"Вы выбрали зону B\n";
              pb=30;
              break;
          case 'c':
          case 'C':
              cout<<"Вы выбрали зону C\n";
              pb=50;
              break;
          case 'd':
          case 'D':
              cout<<"Вы выбрали зону D\n";
              pb=100;
              break;
        }
        cout<<"Сколько вы хотите купить билетов? ";
        cin>>numb;
        if (numb*pb>=5000)
        { disc=numb*pb*0.05; cout<<"Ваша скидка = "<<disc;}
        else
            if (numb*pb>=2000)
            { disc=numb*pb*0.03; cout<<"Ваша скидка = "<<disc;}
            else { disc=0; cout<<"Ваша скидка = "<<disc;}
        cout<<"\nВнесите оплату в размере = "<<numb*pb-disc<<"руб.\n";
        cin>>opлата;
        if (opлата>=numb*pb-disc)
            cout<<"Ваша сдача = "<<opлата-(numb*pb-disc)<<" руб."<<"\nВы выбрали зону =
"<<a<<"\nКоличество билетов = "<<numb<<"\nСтоимость билетов = "<<numb*pb-disc<<"
руб.\n";
            else
                cout<<"Не хватает "<<numb*pb-disc-opлата<<" руб. для оплаты\n";
        }
        else
            cout<<"Нет такой зоны\n";
    }
}

```