

4.4.5. Операторы циклов

Различают:

- 1) итерационные циклы;
- 2) арифметические циклы.

Группа действий, повторяющихся в цикле, называется его *телом*.

Однократное выполнение цикла называется его *шагом*.

В *итерационных циклах* известно условие выполнения цикла.

1. Цикл с предусловием. Цикл while:

Этот цикл имеет следующий формат:

while (условие)
 тело цикла

Условие заключается в круглые скобки. Оно может быть любым выражением, значение которого после вычисления может быть приведено к логическому типу. Схема выполнения цикла *while* следующая:

- вычисляется условие;
- если условие истинно (равно true), то выполняется одна инструкция или блок инструкций, составляющие тело цикла;
- предыдущие два пункта повторяются до тех пор, пока условие не станет ложным;
- если условие ложно, то выполнение цикла *while* заканчивается и начинает выполняться следующая за циклом инструкция.

Чтобы прервать выполнение цикла до того, как условие станет ложным, в теле цикла можно использовать инструкцию *break*.

Пример 34. Цикл с предусловием. Вычисление суммы чисел пока не будет введено нулевое значение

```
cin>>a;  
while (a!=0)  
{  
    s+=a;  
    cin>>a;  
}
```

Алгоритмическая структура, соответствующая оператору **while**, показана на рис. 3.3.

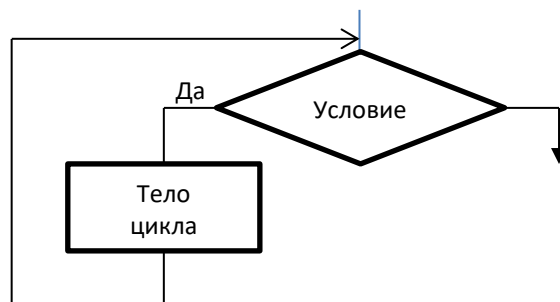


Рис. 3.3. Алгоритмическая структура выполнения оператора **while**

Пример. Пусть переменные x, s равны $x = 4, s = 0$. После выполнения оператора *while* ($x \leq 8$)
 $\{ s = s + x; x = x + 1; \}$
они получают значения $s = 0 + 4 + 5 + 6 + 7 + 8 = 30, x = 9$.

Пример. Вывести на экран таблицу чисел от 20 до 30, их квадраты и кубы, используя команду **while**, можно так:

```
int i = 20;
while (i <= 30)
{
    cout << i << " " << i * i << " " << i * i * i << "\n";
    i++;
}
```

Замечание. В теле цикла необходимо изменять параметр, иначе произойдет так называемое заикливание (бесконечное повторение тела цикла).

2. Цикл с постусловием. Цикл **do / while**

Инструкция цикла **do / while** подобна инструкции *while* и организует выполнение цикла с постусловием и используется в тех случаях, когда необходимо выполнить тело цикла хотя бы один раз. Этот цикл имеет следующий формат:

```
do
    тело цикла
while (условие)
```

Схема выполнения цикла *do / while* следующая:

- выполняется тело цикла (которое может быть одиночной инструкцией или блоком);
- вычисляется условие;
- если условие истинно, предыдущие два пункта повторяются до тех пор, пока условие не станет ложным;
- если условие ложно, то выполнение цикла *do / while* заканчивается и начинает выполняться следующая за циклом инструкция.

Чтобы прервать выполнение цикла до того, как условие станет ложным, можно использовать инструкцию *break*.

Пример 35. Цикл с постусловием.

```
do
{
    cin >> a;
    s += a;
}
while (a != 0);
```

Алгоритмическая структура, соответствующая оператору **do / while**, показана на рис. 3.4.

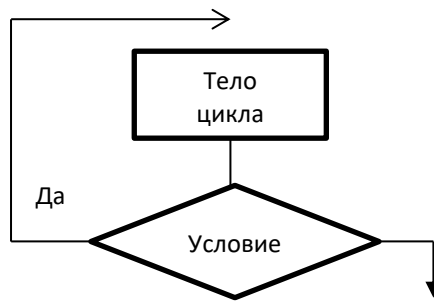


Рис. 3.4. Алгоритмическая структура выполнения оператора **do / while**

Циклы `while` и `do / while` могут быть вложенными:

Пример 36. Использование вложенных циклов

```

int i, j, k;
i = 0; j = 0; k = 0;
do
{
    i++;
    j--;
    while (a[k] < i) k++;
}
while (i < 30 && j > -30);
  
```

3. Цикл с параметром. Цикл for

Инструкция **for** - это наиболее часто используемое средство организации цикла. Инструкция цикла `for` предназначена для многократного (циклического) выполнения одной инструкции или блока инструкций в зависимости от того, истинно заданное условие или нет.

Цикл *for* имеет следующий формат:

**for (инициализирующее выражение; условие; модифицирующее выражение)
 тело цикла**

Инициализирующее выражение может состоять из одного или нескольких разделённых запятой выражений, в том числе и объявлений переменных. Инициализирующее выражение обычно используется для установления начального значения переменных, управляющих циклом, и выполняется только один раз, независимо от числа повторений цикла.

Условие не заключается в круглые скобки. Оно может быть любым выражением (включая список выражений, разделённых запятой), значение которого после вычисления может быть приведено к логическому типу. Проверка условия всегда выполняется в начале каждого цикла. Это значит, что тело цикла может ни разу не выполниться, если условие сразу будет ложным. **Модифицирующее выражение** также может состоять из списка выражений, разделённых запятой. Оно обеспечивает изменение переменных, управляющих циклом, после каждого выполнения тела цикла.

При получении управления инструкцией *for* схема работы цикла следующая:

- вычисляется инициализирующее выражение;
- вычисляется условие;
- если условие истинно (равно true), то выполняется простая инструкция или блок, составляющие тело цикла;
- вычисляется модифицирующее выражение и всё повторяется, начиная со второго пункта (вычисления условия);
- если условие ложно, то выполнение цикла for заканчивается и начинает выполняться следующая за циклом инструкция.

Пример 37. Вычислить квадраты чисел от 1 до 9.

```
int main()
{
    int i, b;
    for (i = 1; i < 10; i++)
    {
        b = i * i;
        cout << "\ni=" << i << " b=" << b;
    }
    return 0;
}
```

Алгоритмическая структура, соответствующая этому оператору, показана на рис. 3.1.

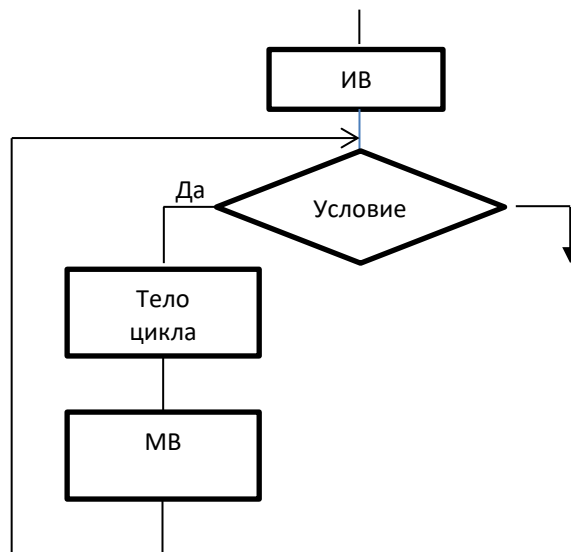


Рис. 3.1. Алгоритмическая структура выполнения оператора **for**

Другим вариантом использования инструкции for является бесконечный цикл. Для организации такого цикла можно использовать пустое условное выражение, а для выхода из цикла обычно используют дополнительное условие и инструкцию break. Например:

```
for (;;)
{
    ...
    break;
    ...
}
```

Цикл **for** может быть заменен циклом **while** следующим образом:

```
инициализирующее выражение;  
while (условие)  
{  
    инструкция  
    модифицирующее выражение;  
}
```

Тело цикла *for* может быть пустой инструкцией. Такая форма цикла может быть использована для организации поиска:

Пример 38. Поиск первого значения $t[i] \geq 10$
`for (i = 0; t[i] < 10; i++)`;

Пример 39. Использование цикла с параметром.

1) Уменьшение параметра:

```
for ( n=10; n>0; n--)  
{ <тело цикла>}
```

2) Изменение шага корректировки:

```
for ( n=2; n<60; n+=13)  
{ <тело цикла>}
```

3) Возможность проверять условие отличное от условия, которое налагается на число итераций:

```
for ( num=1; num*num*num<216; num++)  
{ <тело цикла>}
```

4) Коррекция может осуществляться не только с помощью сложения или вычитания:

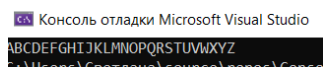
```
for ( d=100.0; d<150.0; d*=1.1)  
{ <тело цикла>}
```

```
for (x=1; y<=75; y=5*(x++)+10)  
{ <тело цикла>}
```

5) Можно использовать несколько инициализирующих или корректирующих выражений:

```
for (x=1, y=0; x<10; x++, y+=x)  
{ <тело цикла>}
```

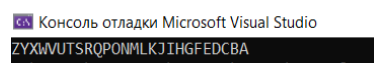
Пример 39. Заглавные латинские буквы в алфавитном порядке выведет цикл
`for (char i = 'A'; i <= 'Z'; ++i) cout << i;`



Консоль отладки Microsoft Visual Studio
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Заглавные латинские буквы в обратном порядке выведет цикл:

```
for (char i = 'Z'; i >= 'A'; --i)    cout << i;
```



Консоль отладки Microsoft Visual Studio
ZYXWVUTSRQPONMLKJIHGFEDCBA

```

for (char i = 'A'; i <= 'Z'; ++i)
{
    for (char j = 'A'; j <= i; ++j)
        cout << j;
    cout << '\n';
}

```

Консоль отладки Microsoft Visual Studio

A
AB
ABC
ABCD
ABCDE
ABCDEF
ABCDEFG
ABCDEFGH
ABCDEFGHI
ABCDEFGHIJ
ABCDEFGHIJK
ABCDEFGHIJKL
ABCDEFGHIJKLM
ABCDEFGHIJKLMN
ABCDEFGHIJKLMNO
ABCDEFGHIJKLMNOP
ABCDEFGHIJKLMNQP
ABCDEFGHIJKLMNPQR
ABCDEFGHIJKLMNPQRS
ABCDEFGHIJKLMNPQRST
ABCDEFGHIJKLMNPQRSTU
ABCDEFGHIJKLMNPQRSTUV
ABCDEFGHIJKLMNPQRSTUVW
ABCDEFGHIJKLMNPQRSTUVWX
ABCDEFGHIJKLMNPQRSTUVWXY
ABCDEFGHIJKLMNPQRSTUVWXYZ

4.4.6. Операторы перехода

Операторы перехода выполняют безусловную передачу управления.

1) Инструкция прерывания *break*

Инструкция прерывания **break** *обеспечивает прекращение выполнения самой внутренней из объемлющих её конструкций switch, while, do, for*. После выполнения инструкции **break** управление передается инструкции, следующей за объемлющей её конструкций.

Формат инструкции следующий:

```
break;
```

Пример 40.

```
// ищет сумму чисел вводимых с клавиатуры до тех пор, пока не будет введено 100
чисел или значение равное 0
```

```
for(s=0, i=1; i<=100; i++)
{
    cin>>x;
    if (x==0) break;           // если ввели 0, то суммирование заканчивается
    s+=x;
}
```

2) Инструкция продолжения *continue*

Инструкция продолжения **continue** используется только внутри циклов и *обеспечивает досрочное прекращение выполнения текущей итерации и переход к выполнению следующей итерации объемлющего её цикла*. Следует заметить, что, в

отличие от continue, инструкция break прекращает выполнение не только текущей итерации цикла, но цикла в целом.

Формат инструкции следующий:
continue;

Пример 41.

//ищет количество и сумму положительных чисел до ввода значения x=0

```
for( k=0,s=0; x!=0;)
{
    cin>>x;
    if (x<=0) continue;
    k++;s+=x;
}
```

Когда введенное число x меньше либо равно 0, инструкция *continue* передает управление на очередную итерацию цикла *for*, не выполняя инструкции расчета количества и суммы положительных чисел.

Инструкция continue прерывает итерацию самого внутреннего из объемлющих её циклов.

3) Инструкция goto

Оператор *goto* имеет формат:
goto метка;

В теле той же функции должна присутствовать конструкция:
метка: инструкция;

Метка – это обычный идентификатор, областью видимости которого является функция. Оператор *goto* передает управления оператору, стоящему после метки. Использование оператора *goto* оправдано, если необходимо выполнить переход из нескольких вложенных циклов или переключателей вниз по тексту программы или перейти в одно место функции после выполнения различных действий.

Применение *goto* нарушает принципы структурного и модульного программирования, по которым все блоки, из которых состоит программа, должны иметь только один вход и только один выход.

Нельзя передавать управление внутрь операторов *if*, *switch* и *циклов*. Нельзя переходить внутрь блоков, содержащих инициализацию, на операторы, которые стоят после инициализации.

Пример 42. Ввести несколько вариантов значений коэффициентов квадратного уравнения $ax^2+bx+c=0$, $a \neq 0$. Вывести сообщение о наличии действительных корней для каждого варианта:

```
#include <iostream>
#include <locale>      // файл, где определена функция setlocale (LC_CTYPE,"rus")
                        //для вывода текста кириллицей;
#include <conio.h>      // файл, где определена функция getch()
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
}

```

4) Инструкция возврата return

Инструкция возврата **return** *завершает выполнение функции, в которой она находится*, и возвращает управление в вызывающую функцию, в точку, непосредственно следующую за вызовом. Функция main() при выполнении инструкции return возвращает управление операционной системе.

Формат инструкции:

return выражение;

Значение выражения, если оно задано, возвращается в вызывающую функцию в качестве значения завершившей работу функции. Если выражение опущено, то возвращаемое значение не определено. Выражение может быть заключено в круглые скобки, хотя их наличие не обязательно.

Если в какой-либо функции отсутствует инструкция *return*, то передача управления в вызывающую функцию происходит после выполнения последней инструкции вызываемой функции. При этом возвращаемое значение не определено. **Если функция не должна иметь возвращаемого значения, то ее нужно объявлять с типом void.**

Пример 43.

```

int sum (int a, int b)
return (a+b);

```

Функция sum() имеет два формальных аргумента *a* и *b* типа *int*, и возвращает значение типа *int*, о чем говорит описатель, стоящий перед именем функции. Возвращаемое инструкцией **return** значение равно сумме фактических аргументов.

Пример 44.

```

int test(int a, double b)
{
    double c;
    if (a < 3) return 1;
    else if (b > 10) return 2;
    else

```



```

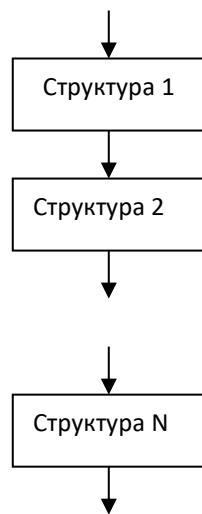
{
    c = a + b;
    if ((2*c - b) == 11) return 3;
}

```

В этом примере инструкция **return** используется для выхода из функции в случае выполнения одного из проверяемых условий.

1.1 Структурное программирование

А.
Пакетирование
структур



Б. Вложение структур

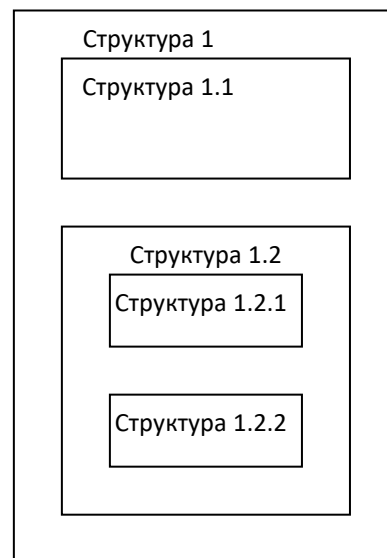
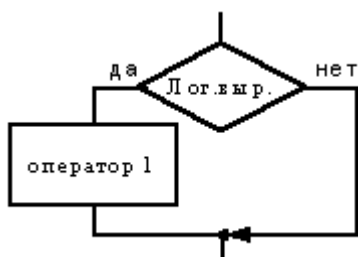
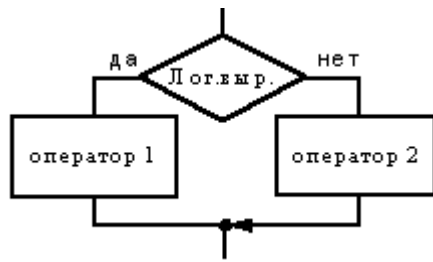


Рисунок 1.1 Пакетирование и вложение структур

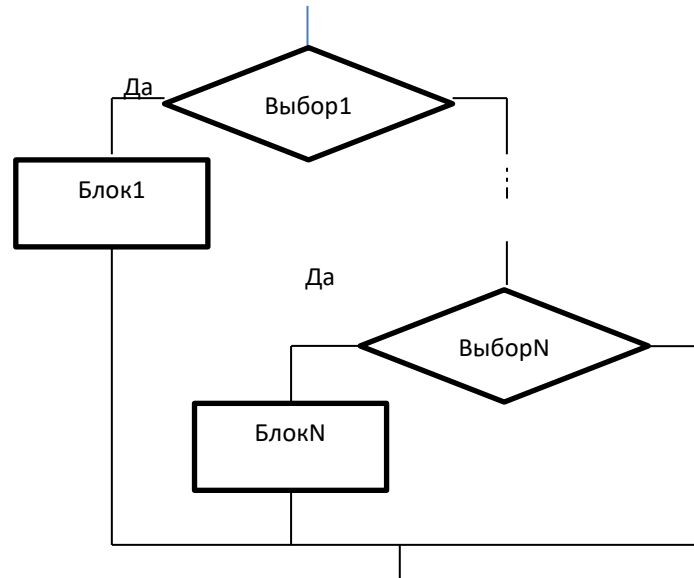
Структура выбора в языке C++ реализуется одной из трёх инструкций, графическое представление которых приведено на рис. 1.2.



Инструкция if...



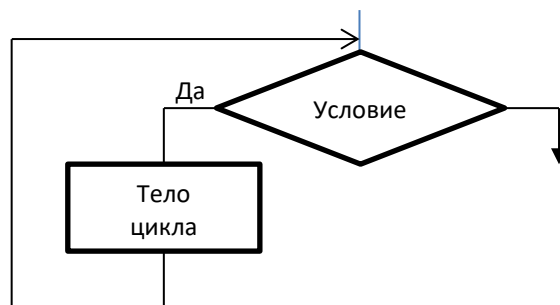
Инструкция if.....else



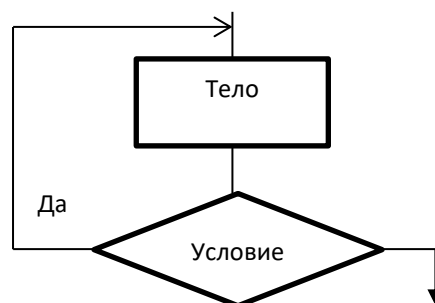
Инструкция оператора *switch*

Рисунок 1.2 Инструкции выбора в языке C++

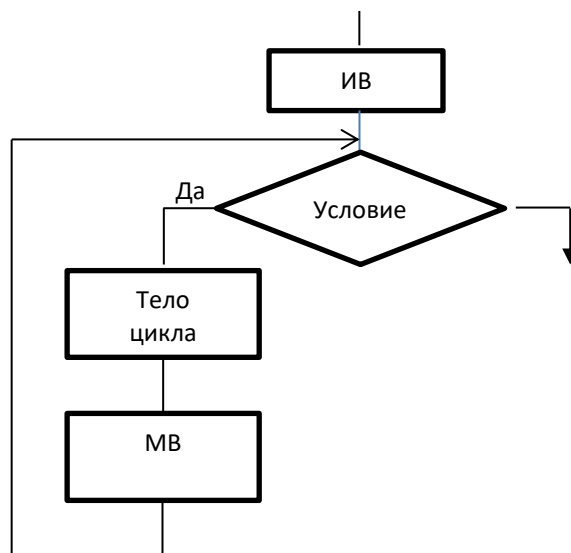
В языке C++ также имеется и три инструкции повторения, графическое представление которых показано на рис. 1.3.



Инструкция *while*



Инструкция *do / while*



Инструкция **for**

4.5. Примеры решения задач с использованием основных операторов C++

«Начинающие программисты, особенно студенты, часто пишут программы так: получив задание, тут же садятся за компьютер и начинают кодировать те фрагменты алгоритма, которые им удастся придумать сразу. Переменным дают первые попавшиеся имена типа *x* и *y*.

Когда компьютер зависает, делается перерыв, после которого все написанное стирается, и все повторяется заново. Периодически высказываются сомнения в правильности работы компилятора, компьютера и операционной системы. Когда программа доходит до стадии выполнения, в нее вводятся произвольные значения, после чего экран становится объектом пристального удивленного изучения. «Работает» такая программа обычно только в бережных руках хозяина на одном наборе данных, а внесение в нее изменений может привести автора к потере веры в себя и ненависти к процессу программирования.

Ваша задача состоит в том, чтобы научиться подходить к программированию профессионально. В конце концов, профессионал отличается тем, что может достаточно точно оценить, сколько времени у него займет написание программы, которая будет работать в полном соответствии с поставленной задачей. Кроме «ума, вкуса и терпения», для этого требуется опыт, а также знание основных принципов, выработанных программистами в течение более, чем полувека развития этой дисциплины. Даже к написанию самых простых программ нужно подходить последовательно, соблюдая определенную дисциплину» [3].

Решение задач по программированию предполагает ряд этапов:

1. Разработка математической модели. На этом этапе определяются исходные данные и результаты решения задачи, а также математические формулы, с помощью которых можно перейти от исходных данных к конечному результату.

2. Разработка алгоритма. Определяются действия, выполняя которые можно будет от исходных данных прийти к требуемому результату.

3. Запись программы на некотором языке программирования. На этом этапе каждому шагу алгоритма ставится в соответствие конструкция выбранного алгоритмического языка.

4. Выполнение программы (исходный модуль → компилятор → объектный модуль → компоновщик → исполняемый модуль)

5. Тестирование и отладка программы. При выполнении программы могут возникнуть ошибки 3 типов:

- а) синтаксические – исправляются на этапе компиляции;
- б) ошибки исполнения программы (деление на 0, логарифм от отрицательного числа и т.п.) – исправляются при выполнении программы;
- в) семантические (логические) ошибки – появляются из-за неправильно понятой задачи, неправильно составленного алгоритма.

Чтобы устранить эти ошибки программа должна быть выполнена на некотором наборе тестов. Цель процесса тестирования – определение наличия ошибки, нахождение места ошибки, ее причины и соответствующие изменения программы – исправление. Тест – это набор исходных данных, для которых заранее известен результат. Тест, выявивший ошибку, считается успешным. Отладка программы заканчивается, когда достаточное количество тестов выполнилось успешно, т.е. программа на них выдала правильные результаты.

Для определения достаточного количества тестов существует два подхода. При первом подходе программа рассматривается как «черный ящик», в который передают исходные данные и получают результаты.

Устройство самого ящика неизвестно. При этом подходе, чтобы осуществить полное тестирование, надо проверить программу на всех входных данных, что практически невозможно. Поэтому вводят специальные критерии, которые должны показать, какое конечное множество тестов является достаточным для программы. При первом подходе чаще всего используются следующие критерии:

- 1) тестирование классов входных данных, т.е. набор тестов должен содержать по одному представителю каждого класса данных:

$$X = 0 \ 1 \ 0 \ 1 \ -1 \ 1 \ -1$$

$$Y = 0 \ 1 \ 1 \ 0 \ 1 \ -1 \ -1$$

- 2) тестирование классов выходных данных, набор тестов должен содержать данные достаточные для получения по одному представителю из каждого класса выходных данных.

При втором подходе программа рассматривается как «белый ящик», для которого полностью известно устройство. Полное тестирование при этом подходе заканчивается после проверки всех путей, ведущих от начала программы к ее концу. Однако и при таком подходе полное тестирование программы невозможно, т.к. путей в программе с циклами бесконечное множество. При таком подходе используются следующие критерии:

- 1) тестирование команд. Набор тестов должен обеспечивать прохождение каждой команды не менее одного раза.

- 2) тестирование ветвей. Набор тестов в совокупности должен обеспечивать прохождение каждой ветви не менее одного раза. Это самый распространенный критерий в практике программирования.

4.5.1. Программирование ветвлений

Задача № 1. Определить, попадет ли точка с координатами (x, y) в заштрихованную область (рис. 19).

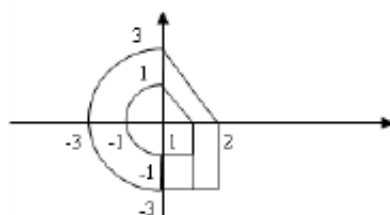


Рис. 19. Границы области для задачи №1

Исходные данные: x, y

Результат: да или нет

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:

$$1) \begin{cases} 1 = a \cdot 0 + b; \\ 0 = a \cdot 1 + b. \end{cases}$$

$$2) \begin{cases} 2 = a \cdot 0 + b; \\ 0 = a \cdot 3 + b. \end{cases}$$

Из этих систем получаем следующие уравнения прямых:

$$y = -x + 1;$$

$$y = -\frac{2}{3}x + 1;$$

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

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

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.

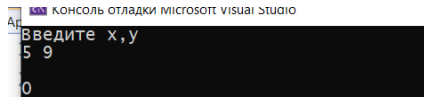
Условие попадания точки в IV квадрант будет выглядеть следующим образом:

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

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

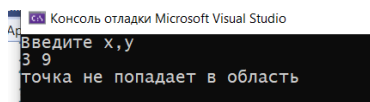
```
#include <iostream>
#include <math.h>
#include <windows.h>
using namespace std;
void main()
{
    SetConsoleOutputCP(1251);
    float x,y;
    cout<<"Введи x,y\n";
    cin>>x>>y;
    bool Ok=(y>=-x+1 && y<=-3/2*x+3 && x>=0 && y>=0)//
    (pow(x,2)+pow(y,2)>=1 && pow(x,2)+pow(y,2)<=9 && x<=0)//
    (x>=0 && x<=1 && y<=-1 && y>=-3)//
    (x>=1 && x<=2 && y<=0 && y>=-3);
    cout<<"\n"<<Ok;
```

}



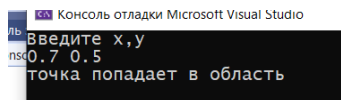
Вариант 2. Программа имеет вид (с использованием условной операции):

```
void main()
{
    SetConsoleOutputCP(1251);
    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()
{
    SetConsoleOutputCP(1251);
    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 |

4.5.2. Программирование арифметических циклов

Для арифметического цикла заранее известно сколько раз выполняется тело цикла.

Задача №1. Вычислить значение выражения, состоящее из суммы и произведения

$$y = \sum_{i=1}^{25} \frac{5i^2}{i!} + \prod_{i=1}^{25} i^2 [3].$$

$$1! = 1$$

$$2! = 1 * 2$$

$$3! = 1 * 2 * 3$$

Схема алгоритма решения этой задачи приведена на рис. 3.2.

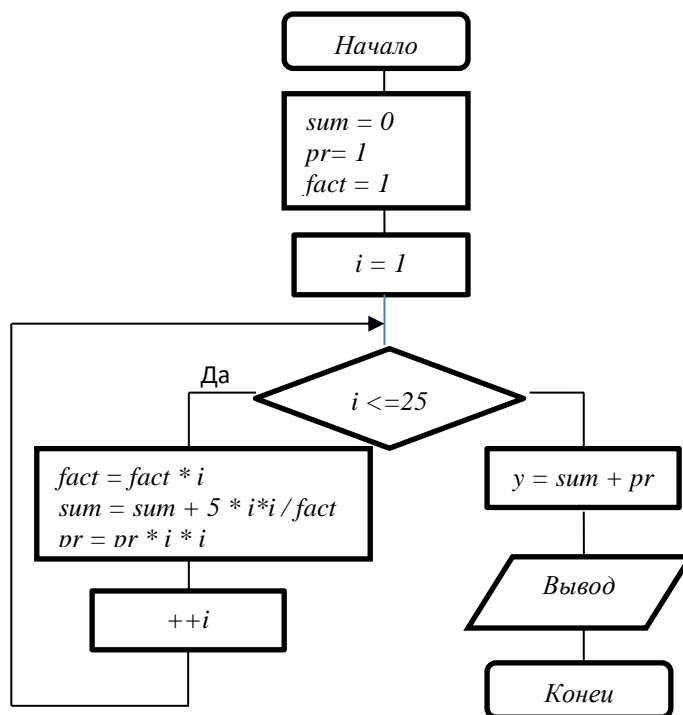


Рис. 3.2. Схема алгоритма решения задачи 3.1

Программа имеет вид:

```

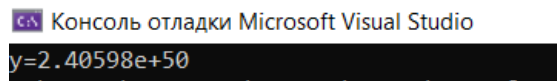
#include <iostream>
#include <windows.h>
#include <math.h>
using namespace std;
int main()
{
    SetConsoleOutputCP(1251);

```

```

//Сначала сумма (sum) равна нулю, а произведение (pr) единице
double sum = 0, pr= 1;
long int fact = 1;
for (int i = 1; i <=25; ++i)
{
    fact = fact * i;           // Вычисляем  $i! = 1 * 2 * 3 * \dots * i$ 
    sum = sum + 5 * i * i / fact; //Вычисляем сумму
    pr = pr * i * i;           // Вычисляем произведение
}
double y = sum + pr;
cout << "y=" << y;
}

```



Консоль отладки Microsoft Visual Studio

y=2.40598e+50

Задача № 2. Дана последовательность целых чисел из n элементов. Найти среднее арифметическое этой последовательности.

Программа:

```

#include <iostream> // подключить файл, где объявлены cin и cout
using namespace std;
void main()
{
    int a, n, i;
    double s = 0;
    cout << "\nEnter n\n";
    cin >> n;
    for (i = 1; i <= n; i++)
    {
        cout << "\nEnter a\n";
        cin >> a;
        s += a;
    }
    s = s / n;
    cout << "\nSr. arifm=" << s << "\n";
}

```

Тесты приведены в табл. 13.

Таблица 13

Тесты к задаче № 2

| Параметр | |
|-----------------------------|----------------|
| Количество цифр, n | 5 |
| Значение цифр, a | 1, 2, 3, 4, 5, |
| Среднее арифметическое, s | 3 |


```

Консоль отладки Microsoft Visual Studio

Enter n
5
Enter a
1
Enter a
2
Enter a
3
Enter a
4
Enter a
5
Sr. arifm=3

```

Задача № 3. Найти сумму чисел последовательности:
 $S = 1 + 2 + 3 + 4 + \dots + N$

Программа:

```

#include <iostream> // подключить файл, где объявлены cin и cout
using namespace std;
void main()
{
    int n,i,s=0;
    cout<<"\nEnter n";
    cin>>n;
    if(n<=0) {cout<<"\nN<=0"; return;}
    for (i=1; i<=n; i++) s+=i;
    cout<<"\nS="<<s;
}

```

Тесты приведены в табл. 14.

Таблица 14

Тесты к задаче № 3

| Значение параметра n | Значение параметра s |
|------------------------|------------------------|
| $n=-1$ | $N \leq 0$ |
| $n=0$ | $N \leq 0$ |
| $n=5$ | $S=15$ |

```

Консоль отладки Microsoft Visual Studio

Enter n
-4
N<=0

C:\Users\Светлана\source\repos\ConsoleApplication8\Debug\
Enter n
5
S=15

```

Задача №4. Найти сумму последовательности вида:
 $S = 15 - 17 + 19 - 21 + \dots$, всего n слагаемых.

Программа:

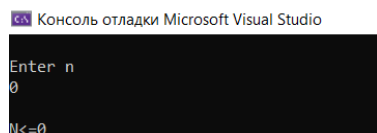
```
#include <iostream> // подключить файл, где объявлены cin и cout
using namespace std;
void main()
{
    int n,i,s=0, a=15;
    cout<<"\nEnter n";
    cin>>n;
    if(n<=0) {cout<<"\nN<=0"; return;}
    for(i=1; i<=n; i++)
    {
        if (i%2==1) s+=a;
        else s-=a;
        a+=2;
    }
    cout<<"\nS="<<s;
}
```

Тесты приведены в табл. 15.

Таблица 15

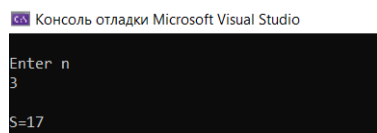
Тесты к задаче № 4

| Значение параметр n | Значение параметра s |
|---------------------|----------------------|
| $n = -1$ | $N \leq 0$ |
| $N = 0$ | $N \leq 0$ |
| $N = 3$ | $S = 17$ |



Консоль отладки Microsoft Visual Studio

```
Enter n
0
N<=0
```



Консоль отладки Microsoft Visual Studio

```
Enter n
3
S=17
```

Задача №5. Составить программу табулирования функции

$$y = \begin{cases} a \lg x + \sqrt{|x|}, & \text{если } x > 1; \\ 2a \cos x + 3x^2, & \text{если } x \leq 1, \end{cases}$$

для $a=0,9$ при изменении аргумента x в диапазоне $x \in [0,8;2]$ с шагом 0,1. Вывод значений x и y выполнить в виде таблицы.

```
#include <iostream>
#include <windows.h>
#include <math.h>
using namespace std;
int main()
{
    SetConsoleOutputCP(1251);
    double a, x, y, x0, xk, step;
    cout<<"Введите a, x0, xk, step\n"; //Ввести с клавиатуры
    cin>>a>>x0>>xk>>step; // a=0.9, x0=0.8, xk=2 и step=0.1
    cout<<"\n Таблица функции Y(X)\n"; //Заголовок таблицы
```

```

cout<<"\n X      Y\n";
x=x0;
while (x<=xk + step / 2)
//Слагаемое step / 2 обеспечивает включение последней точки xk в таблицу
{
    if (x > 1) y = a * log10(x) + sqrt(abs(x));
    else y = 2 * a * cos(x) + 3 * x * x;
    cout<< x<<" " <<y<<"\n";
    x += step;
}
}

```

Консоль отладки Microsoft Visual Studio

Введите a, x0, xk, step

0.9
0.8
2
0.1

Таблица функции Y(X)

| X | Y |
|-----|---------|
| 0.8 | 3.17407 |
| 0.9 | 3.5489 |
| 1 | 3.97254 |
| 1.1 | 1.08606 |
| 1.2 | 1.16671 |
| 1.3 | 1.24272 |
| 1.4 | 1.31473 |
| 1.5 | 1.38323 |
| 1.6 | 1.44862 |
| 1.7 | 1.51124 |
| 1.8 | 1.57139 |
| 1.9 | 1.62928 |
| 2 | 1.68514 |

Задача 6. Вычислить значения y , соответствующие каждому значению x ($x_n \leq x \leq x_k$, шаг

$$y = \frac{\sqrt[3]{|a - x^2| \ln(2 + a^2 + x^4)}}{2}$$

изменения x равен dx) по формуле

Вычислить среднее значение среди положительных элементов y , произведение ненулевых и количество отрицательных значений y . На экран выводить каждую вторую пару значений x и y . Контрольный расчёт провести при $a=2.17$, $x_n=-1.5$, $x_k=0.5$, $dx=0.2$.

```

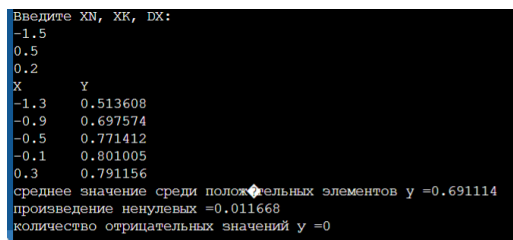
#include <iostream>
#include <windows.h>
#include <math.h>
using namespace std;
int main()
{
    SetConsoleOutputCP(1251);
    cout<<"Введите XN, XK, DX: ";
    double X, XN, XK, DX, Y, S=0, K=0, O=0, P=1, a=2.17;
    cin>>XN>>XK>>DX;
    int i=0;

```

```

cout<<"X"<<" " <<"Y"<<"\n";
for(X=XN; X<=XK; X+=DX)
{
    Y=pow(fabs(a-X*X)*log(2+a*a+pow(X,4)),1/3.0)/2;
    i++; //количество вычисленных значений y
    // вывод каждой второй пары значений x и y.
    if (i % 2 ==0)
        cout<<X<<" " <<Y<<"\n";
    if (Y>0) {K++; S=S+Y;} //количество и сумма положительных значений y
    if (Y!=0) P=P*Y; //произведение ненулевых значений y
    if (Y<0) O++; //количество отрицательных значений y
}
cout<<"среднее значение среди положительных элементов y ="<<S/K <<"\n";
cout<<"произведение ненулевых ="<<P<<"\n";
cout<<"количество отрицательных значений y ="<<O<<"\n";
}

```



```

Введите XN, XK, DX:
-1.5
0.5
0.2
X      Y
-1.3   0.513608
-0.9   0.697574
-0.5   0.771412
-0.1   0.801005
0.3    0.791156
среднее значение среди положительных элементов y =0.691114
произведение ненулевых =0.011668
количество отрицательных значений y =0

```

4.5.3. Итерационные циклы

Для итерационного цикла известно условие выполнения цикла и неизвестно точное число повторений.

Задача № 7. Дана последовательность целых чисел, за которой следует 0. Найти минимальный элемент этой последовательности.

Программа:

```

#include <iostream> // подключить файл, где объявлены cin и cout
using namespace std;
void main()
{
    int a, min;
    cout<<"\nEnter a\n";
    cin>>a;
    min=a;
    while (a!=0) //for(;a!=0;)
    {
        cout<<"\nEnter a\n";
        cin>>a;
        if (a!=0&& a<min) min=a;
    }
    cout<<"\nmin="<<min;
}

```

Тесты приведены в табл. 16.

Тесты к задаче № 5

| Последовательность a | min число |
|------------------------|-----------|
| 2 55 -3 -10 0 | -10 |
| 12 55 4 27 0 | 4 |

Задача № 8. Найти сумму чисел Фибоначчи, меньших заданного числа Q .

Примечание. Последовательность чисел Фибоначчи u_0, u_1, \dots образуется по закону $u_0=0, u_1=1, u_i=u_{i-1}+u_{i-2} (i=2,3,\dots)$.

Программа:

```
#include <iostream> // подключить файл, где объявлены cin и cout
using namespace std;
void main()
{
    int u0=0,u1=1, s=1,Q,ui;
    cout<<"\nEnter Q\n";
    cin>>Q;
    if(Q<=0)cout<<"Error in Q";
    else
        if(Q==1)cout<<"\nS=1";
        else
        {
            ui=u0+u1;
            while(ui<Q) //for(ui!=0;)
            {
                s+=ui;
                u0=u1;
                u1=ui;
                ui=u0+u1;
            }
            cout<<"\nS="<<s;
        }
}
```

Тесты приведены в табл. 17.

Таблица 17

Тесты к задаче № 6

| Значение числа Q | Сумма S |
|--------------------|------------|
| -1 | Error in Q |
| 0 | Error in Q |
| 1 | 1 |
| 2 | 2 |
| 10 | 20 |

4.5.4. Вложенные циклы

Задача № 9. Напечатать N простых чисел.

Программа:

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

```

using namespace std;
void main()
{
    int a=1, n, d;
    cout<<"\nEnter N\n";
    cin>>n;
    for ( int i=0; i<n; )           //внешний цикл
    {
        a++; d=1;
        do                       //внутренний цикл
            d++;
        while (a%d!=0);           //конец внутреннего цикла
        if(a==d)
        {
            cout<<a<<" ";
            i++;
        }
    }                               //конец внешнего цикла
}

```

```

Enter N
20
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 _

```