

## Лекция 8

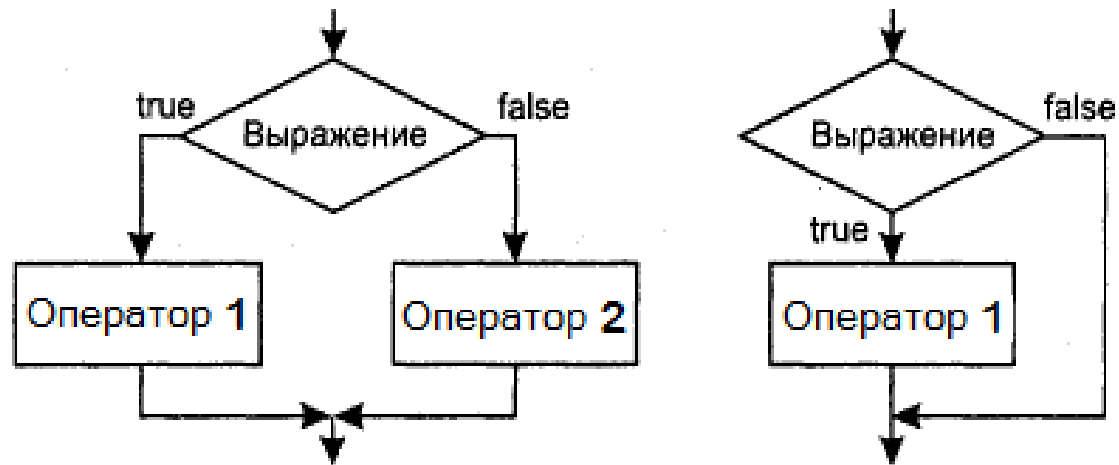
**Базовые конструкции структурного программирования. Операторы ветвления, операторы цикла, операторы передачи управления.**

# Операторы ветвления if-else

Инструкция **if-else** используется для принятия решения. Формально ее синтаксисом является:

**if (<выражение>) <оператор1>; [else <оператор2;>]**

причем **else**-часть может и отсутствовать.



**Составной оператор** – это последовательность операторов, заключенная в фигурные скобки **{ }**.

*Примеры:*

```
if (a<0) b = 1;
```

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

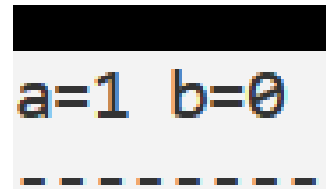
```
if (a<b) {if(a<c) m=a; else m=c;} else { if(b<c) m=b; else m=c;}
```

```
if (a++) b++; //условие не обязательно операция отношения
```

# Операторы ветвления if-else

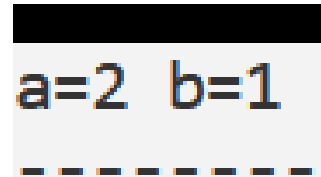
Пример: `if (a++) b++;` рассмотрим подробнее, при каких значениях переменной **a**, переменная **b** изменит значение?

```
int main () {  
    int a=0, b=0;  
    if (a++) b++;  
    cout<<"a="<<a<<" b="<<b;  
}
```



Terminal output for the first code block: a=1 b=0

```
int main () {  
    int a=1, b=0;  
    if (a++) b++;  
    cout<<"a="<<a<<" b="<<b;  
}
```

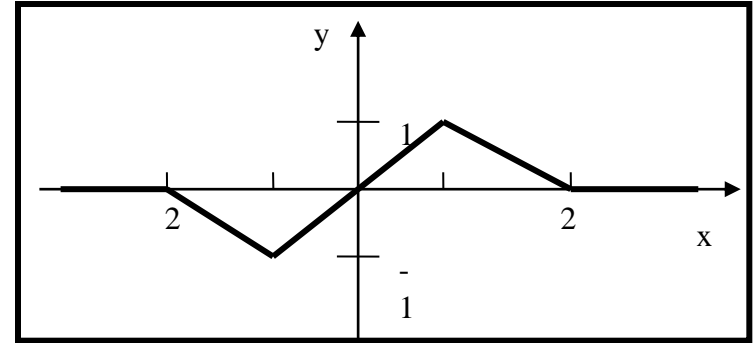


Terminal output for the second code block: a=2 b=1

**Значение выражения истинно, если оно отлично от нуля!**

# Пример вычисления функции

$$y = \begin{cases} 0, & x < -2 \\ -x - 2, & -2 \leq x < -1 \\ x, & -1 \leq x < 1 \\ -x + 2, & 1 \leq x < 2 \\ 0, & x \geq 2 \end{cases}$$



```
#include<stdio.h>
#include<windows.h>
int main() {
    SetConsoleOutputCP(65001);
    double x,y;
    printf("введите x");
    scanf("%lf",&x);
    if (x<-2) y=0;
    if (-2 <= x && x < -1) y= -x-2;
    if (-1 <= x && x < 1) y= x;
    if (1 <= x && x < 2) y= -x+2;
    if (x>=2) y= 0;
    printf("y=%g",y);
    return 0;
}
```

```
#include<stdio.h>
#include<windows.h>
int main() {
    SetConsoleOutputCP(65001);
    double x,y;
    printf("введите x");
    scanf("%lf",&x);
    if (x<-2) y=0;
    else if (x<-1) y=-x-2;
        else if (x<1) y= x;
            else if (x<2) y=-x+2;
                else y=0;
    printf("y=%g",y);
    return 0;
}
```

## Операторы ветвления ?:

Инструкция `if (a > b) z = a; else z = b; // z=max(a,b)` может быть записана с помощью условного выражения

**<выр1> ? <выр2> : <выр3>**

**(a>b) ? z = a : z = b;**

**z = (a>b) ? a : b;**

Тернарный оператор рекомендуется использовать только в простых выражениях, в сложных лучше использовать **if-else**.

Если **выр2** и **выр3** принадлежат разным типам, то тип результата определяется правилами преобразования.

Например, в выражении

**(n > 0) ? f : n;**

если **f** имеет тип **float**, а **n** — тип **int**, то типом выражения будет **float** вне зависимости от того, положительно значение **n** или нет.

# Операторы ветвления

```
int i; cin>>i;
```

```
//определим какое значение было введено
```

```
    //i=1
```

```
if (i==1) ...
```

```
if (i) ...
```

```
if (i!=0) ...
```

```
    //i=0
```

```
if (i==0) ...
```

```
if (!i) ...
```

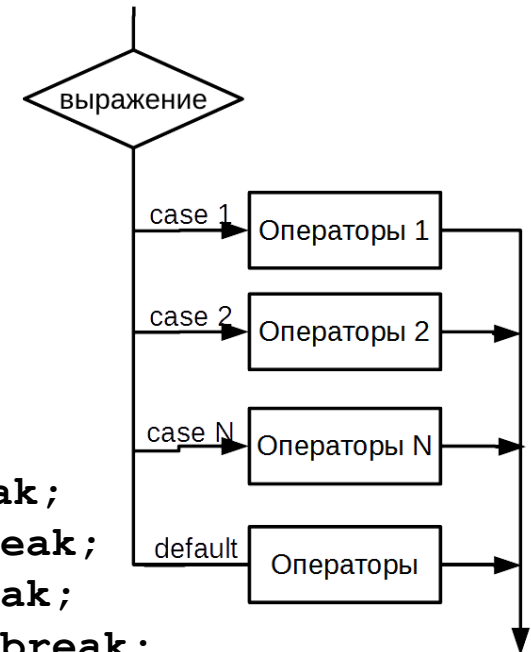
```
if (i!=1) ...
```

# Операторы ветвления switch

Инструкция **switch** используется для выбора одной из нескольких ветвей алгоритма. Формат вызова:

```
switch ( выражение ) {  
    case константное_выражение_1: [операторы_1];  
    case константное_выражение_2: [операторы_2];  
    case константное_выражение_n: [операторы_n];  
    [default: операторы ];  
}
```

```
#include<stdio.h>  
#include<windows.h>  
int main() {  
    SetConsoleOutputCP(65001);  
    int i;  
    printf("введите номер месяца");  
    scanf("%d",&i);  
    switch (i) {  
        case 1: case 2: case 12:printf("зима");break;  
        case 3: case 4: case 5: printf("весна"); break;  
        case 6: case 7: case 8: printf("лето"); break;  
        case 9: case 10: case 11:printf("осень"); break;  
        default: printf("error");  
    }  
    return 0;  
}
```



# Операторы ветвления switch

```
#include <stdio.h>

main() { /* подсчет цифр, пробелов и прочих символов */
    int c, i, nwhite, nother, ndigit[10];
    nwhite = nother = 0;
    for (i = 0; i < 10; i++)
        ndigit[i] = 0;
    while ((c = getchar()) != EOF) {
        switch (c) {
            case '0': case '1': case '2': case '3': case '4':
            case '5': case '6': case '7': case '8': case '9':
                ndigit[c - '0']++; break;
            case ' ': case '\n': case '\t':    nwhite++; break;
            default: nother++;
        }
    }
    printf ("цифр =");
    for (i= 0; i < 10; i++)
        printf (" %d", ndigit[i]);
    printf (" , пробелов= %d, прочих = %d\n", nwhite, nother);
    return 0;
}
```



# Операторы ветвления switch

Инструкция **break** вызывает немедленный выход из переключателя **switch**. Поскольку выбор ветви **case** реализуется как переход на метку, то после выполнения одной ветви **case**, если ничего не предпринять, программа *провалится вниз* на следующую ветвь.

Инструкции **break** и **return** – наиболее распространенные средства выхода из переключателя.

# Операторы цикла while

Цикл с предусловием

**while** ( <выражение C> ) <оператор A>;

Оператор выполняется ноль или более раз.

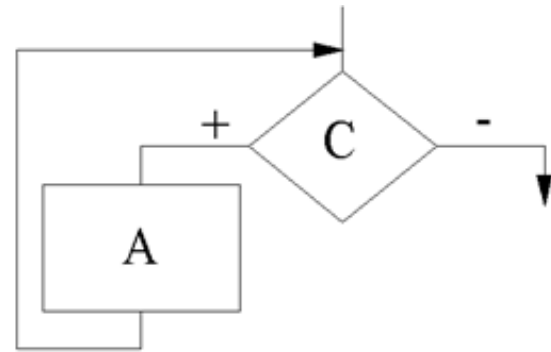
Условие (логическое выражение) выполнения цикла проверяется до выполнения оператора. Если его значение **true**, оператор выполняется.

Поскольку выражение вычисляется при каждой итерации, его следует делать насколько возможно простым.

*Пример:*

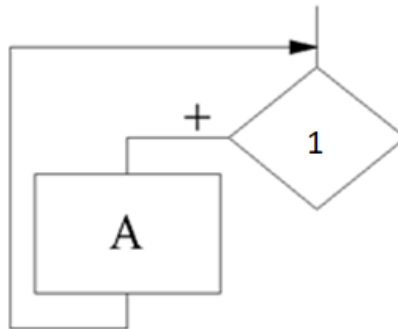
```
#include <stdio.h>
int main() {
    int i=0;
    while (i<10){
        printf(" %d",i);
        i++;
    }
    printf("\n %d",i);
    return 0;
}
```

0	1	2	3	4	5	6	7	8	9
10									



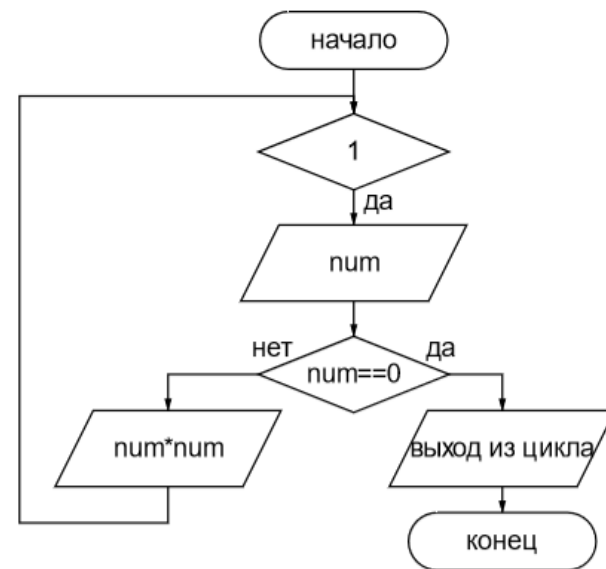
# Операторы цикла while

```
while (1) {  
    /*бесконечный цикл*/  
}
```



Пример с бесконечным циклом:

```
#include <stdio.h>  
#include <windows.h>  
int main() {  
    SetConsoleOutputCP(65001);  
    float num;  
    while (1) {  
        printf("Введите число (выход 0) -> ");  
        scanf("%f", &num);  
        if (num==0) break;  
        printf("\t%g^2=%g\n", num, num*num);  
    }  
    printf("\n Выход из цикла");  
    return 0;  
}
```

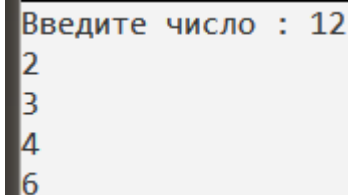


```
Введите число (выход 0) -> 2  
2^2=4  
Введите число (выход 0) -> 2.2  
2.2^2=4.84  
Введите число (выход 0) -> -15.5  
-15.5^2=240.25  
Введите число (выход 0) -> 1234.5  
1234.5^2=1.52399e+06  
Введите число (выход 0) -> 0  
  
Выход из цикла
```

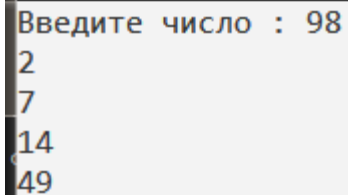
# Операторы цикла while

Какие действия выполняет программа?

```
#include <iostream>
#include <windows.h>
using namespace std;
int main() {
    SetConsoleOutputCP(65001);
    int num;
    cout<<"Введите число : ";
    cin>>num;
    int half = num / 2; // половина числа
    int div = 2; // кандидат на делитель
    while (div <= half) {
        if (!(num % div)) cout << div <<"\n";
        div++;
    }
    return 0;
}
```



```
Введите число : 12
2
3
4
6
```



```
Введите число : 98
2
7
14
49
```

# Операторы цикла for

Цикл с параметром имеет следующий формат:

**for (<выражение1>;<выражение2>;<выражение3>) <оператор>;**

**Выражение1** используется для присвоения начальных значений переменным цикла.

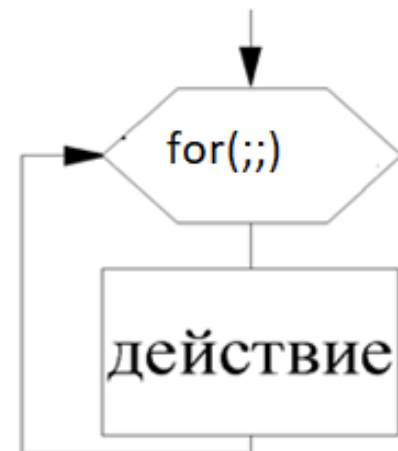
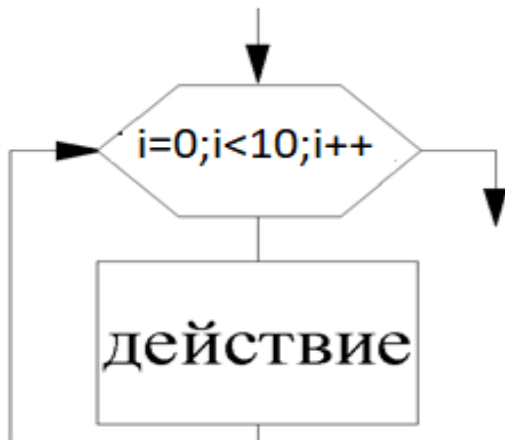
**Выражение2** определяет условие выполнения цикла: если его значение отлично от нуля цикл выполняется.

*Цикл с параметром реализован как цикл с предусловием.*

**Выражение3** вычисляется после каждой итерации цикла и служит обычно для изменения значений переменных цикла.

Любое из выражений может отсутствовать, но “;” пропускать нельзя:

**for (;;) { /\*бесконечный цикл\*/ }**



# Операторы цикла for

*Пример:*

```
#include <stdio.h>
int main() {
    for(int i=0; i<10; i++) printf(" %d",i*i);
    printf("\n");

    int j=10;
    for(; j>0; j--) printf(" %d",j);
    printf("\n");

    j=10; // объявлять уже не нужно
    for(;;) {
        if (j==0) break;
        printf(" %d",j--);
    }
    printf("\n");

    for(double x=65.05; x<66.3; x+=0.1) printf(" %g",x);
    return 0;
}
```

```
0 1 4 9 16 25 36 49 64 81
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
65.05 65.15 65.25 65.35 65.45 65.55 65.65 65.75 65.85 65.95 66.05 66.15 66.25
-----
```

# Операторы цикла for

Напишем функцию **atoi**, выполняющую преобразование строки в ее числовой эквивалент. Она игнорирует ведущие пробелы и обрабатывает знаки + и -, которые могут стоять перед цифрами. Преобразование заканчивается на первом символе, который не может быть частью числа.

```
#include <ctype.h>
int atoi(char s[ ]) { // atoi: преобразование s в целое число
    int i, n, sign;
    /* игнорировать символы-разделители */
    for (i = 0; isspace(s[i]); i++) ;
    sign = (s[i] == '-') ? -1: 1;
    if (s[i] == '+' || s[i] == '-') /* пропуск знака */
        i++;
    for (n = 0; isdigit(s[i]); i++)
        n = 10*n + (s[i] - '0');
    return sign*n;
}
main() {
    char s[]=" -123";
    cout<<"sizeof(s)="<<sizeof(s)<<endl;
    cout<<atoi(s);
    return 0;
}
```

```
sizeof(s)=6
-123
```

## Оператор “,”

“, ” (запятая) чаще всего используется в инструкции **for**.

Пара выражений, разделенных запятой, вычисляется слева направо. Типом и значением результата являются тип и значение правого выражения, что позволяет в инструкции **for** в каждой из трех компонент иметь по несколько выражений, например вести два индекса параллельно.

```
#include <string.h>
```

```
void reverse(char s[]) {// переворачивает строку s (результат в s)}
```

```
    int c, i, j;
```

```
    for (i=0, j=strlen(s)-1; i < j; i++, j-- ) {
```

```
        c = s[i];
```

```
        s[i] = s[j];
```

```
        s[j] = c;           c=s[i], s[i]=s[j], s[j]=c;
```

```
    }
```

```
}
```

```
main() {
```

```
    char s[]="123";
```

```
    cout<<"sizeof(s)="<<sizeof(s)<<endl;
```

```
    reverse(s);
```

```
    cout<<s;
```

```
}
```

```
sizeof(s)=4  
321
```



# Операторы цикла do-while

Цикл с постусловием и имеет вид:

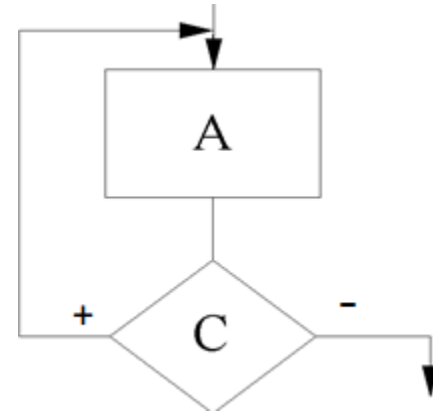
```
do <оператор>
while (<выражение>);
```

Цикл выполняется пока *выражение* истинно.

```
do { /*бескон. цикл*/ }
while (1);
```

*Пример:*

```
#include <iostream>
using namespace std;
int main() {
    char answer;
    do {
        cout << "Купи слоника! ";
        cin >> answer;
    } while (answer != 'y');
    return 0;
}
```



# Операторы цикла do-while

```
Купи слоника! r  
Купи слоника! e  
Купи слоника! h  
Купи слоника! n  
Купи слоника! u
```

# Операторы цикла do-while

```
/* itoa: преобразование n в строку s */
void itoa(int n, char s[]) {
    int i, sign;
    if ((sign = n) < 0)                /* сохраняем знак */
        n = -n;                        /* делаем n положительным */
    i = 0;
    do {                               /* генерируем цифры в обратном порядке */
        s[i++] = n % 10 + '0';         /* следующая цифра */
    } while ((n /= 10) > 0);
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}
```

# Прерывание циклов

Иногда бывает удобно выйти из цикла не по результату проверки, осуществляемой в начале или в конце цикла, а каким-то другим способом. Такую возможность предоставляет инструкция **break**. Эта инструкция вызывает немедленный выход из самого внутреннего из объемлющих ее циклов или переключателей.

*Пример /\* удаляет завершающие символы-пробелы \*/*

```
int trim(char s[]) {
    int n;
    for (n = strlen(s)-1; n >= 0; n--)
        if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n')
            break;
    s[n+1] = '\0' ;
    return n;
}

main() {
    char s[]="123 \n \t ";
    cout<<sizeof(s)<<endl;
    cout<<trim(s);
    return 0;
}
```

# Прерывание циклов

Инструкция **continue** вынуждает ближайший объемлющий ее цикл (**for**, **while** или **do-while**) начать следующий шаг итерации.

Для **while** и **do-while** это означает немедленный переход к проверке условия, а для **for** — к приращению шага.

Вот фрагмент программы, обрабатывающий только неотрицательные элементы массива **a** (отрицательные пропускаются).

```
for ( i = 0; i < n; i++) {  
    if (a[i] < 0) /* пропуск отрицательных элементов */  
        continue;  
    //... /* обработка положительных элементов */  
}
```

# Прерывание циклов

```
#include <stdio.h>
#include <stdlib.h>

int my_fun() {
    int x;
    while(1) {
        printf("x->");
        scanf("%d",&x);
        if (x==10) continue;
        if (x==20) break;
        if (x==30) return 1;
        if (x==40) exit(1);
        printf("in program\n");
    }
    printf("end function\n");
    return 0;
}

int main() {
    printf("%d\n", my_fun());
    printf("end program");
    return 0;
}
```

```
x->1
in program
x->10
x->20
end function
0
end program
-----
```

```
x->30
1
end program
```

```
x->40
-----
Process exited a
Для продолжения
```

# Операторы передачи управления

Обычно применяется когда нужно выйти сразу из двух или большего числа вложенных циклов.

```
for ( /* ... */  
    for ( /* ... */ ) {  
        if (disaster)      /* если бедствие */  
            goto error;    /* уйти на ошибку */  
  
    //...  
error:                    /* обработка ошибки */  
    //...
```

**Метка** имеет вид обычного имени переменной, за которым следует двоеточие. На метку можно перейти с помощью **goto** из любого места данной функции, т. е. метка видима на протяжении всей функции.

В любом случае **не следует передавать управление** внутрь операторов **if**, **switch** и **циклов**. Нельзя переходить внутрь блоков, содержащих инициализацию переменных, на операторы, расположенные после нее, т.к. она не будет выполнена.

# Операторы передачи управления

Использование оператора безусловного перехода оправдано в двух случаях:

- принудительный выход вниз по тексту программы из нескольких вложенных циклов или переключателей;
- переход из нескольких мест функции в одно (например, если перед выходом из функции всегда необходимо выполнять какие-либо действия).

В остальных случаях для записи любого алгоритма существуют более подходящие средства, а использование **goto** приводит только к усложнению структуры программы и затруднению отладки.



# Операторы передачи управления

```
int к; // ...  
goto metka; //...  
{  
    int a = 3, b = 4;  
    к = a + b;  
metka: int m = к + 1; // ...  
}
```

После выполнения этого фрагмента программы значение переменной **m** не определено.

# Оператор return

Оператор возврата из функции **return** завершает выполнение функции и передает управление в точку ее вызова.

Вид оператора:

```
return [ выражение ] ;
```

Выражение должно иметь скалярный тип.

Если тип возвращаемого функцией значения описан как **void**, то выражение должно отсутствовать.

*Пример:*

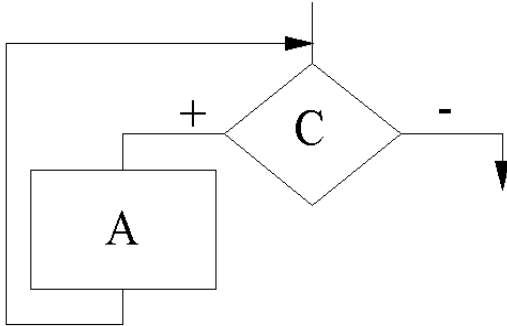
```
#include <stdio.h>
```

```
void my_fun() { // описание функции
    printf("Hello world\n");
    return;
}
```

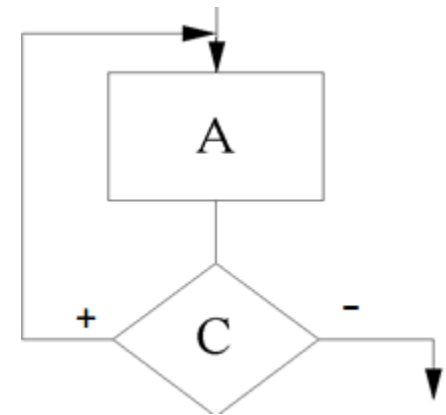
```
int main() {
    my_fun(); //вызов функции
    return 0;
}
```

# Структурные схемы циклов

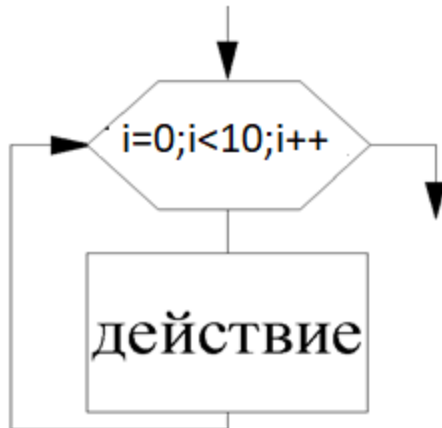
`while (<условие C>) <оператор A>;`



`do <оператор A> while (<условие C>);`

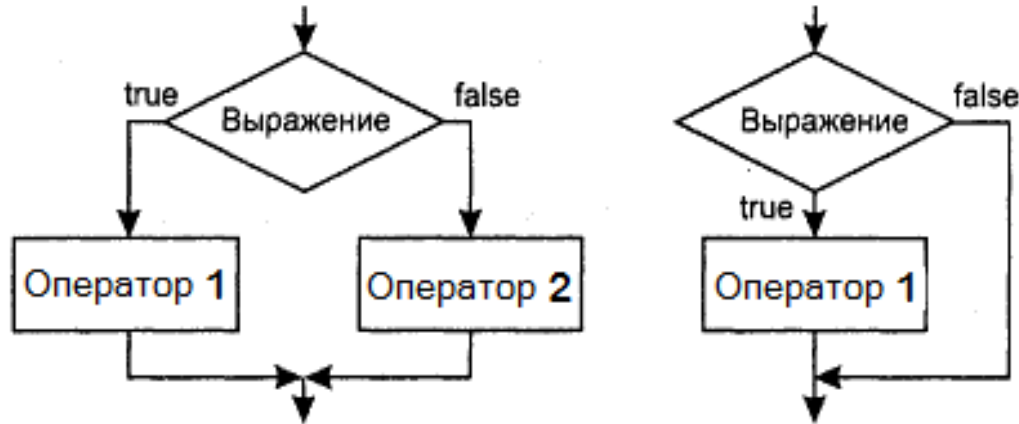


`for(i=0; i<10; i++) <действие>`

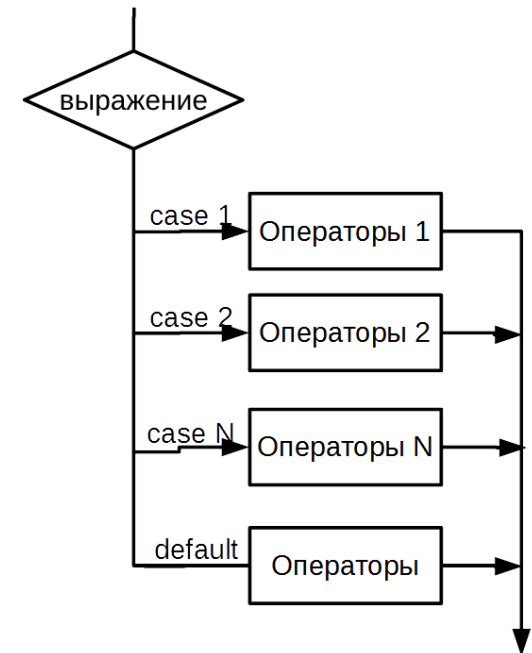


# Структурные схемы условных выражений

if (<выражение>) <оператор1>; [else <оператор2;>]



```
switch ( выражение ) {  
    case константное_выражение_1: [операторы_1];  
    case константное_выражение_2: [операторы_2];  
    case константное_выражение_n: [операторы_n];  
    [default: операторы ];  
}
```



# Пример оформления структурной схемы

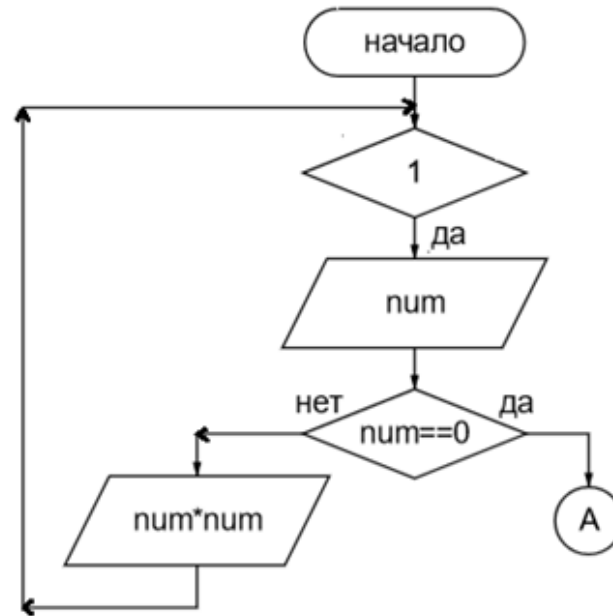


Рисунок 2.1 – Алгоритм основной программы

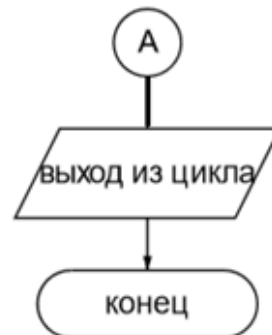


Рисунок 2.1 – Лист 2