ФГБОУ ВО ЧГУ ИМ. И.Н. УЛЬЯНОВА
ФАКУЛЬТЕТ РАДИОЭЛЕКТРОНИКИ И АВТОМАТИКИ
КАФЕДРА АВТОМАТИКИ И УПРАВЛЕНИЯ В ТЕХНИЧЕСКИХ СИСТЕМАХ

ОСНОВНЫЕ ОПЕРАТОРЫ ЯЗЫКА С++

Лекция 2.2.

доцент Васильева Л.Н.

Ввод-вывод в C++

Ввод в формате языка С

scanf – форматный ввод

формат ввода

адреса ячеек, куда записать введенные числа

```
scanf ("%i%i", &a, &b);
```

Формат – символьная строка, которая показывает, какие числа вводятся (выводятся).

- `%i,%d` – целое число
- `%f` – вещественное число
- `%c` – 1 символ
- `%s` – символьная строка

&a – адрес переменной **a**

```
7652
12
```

a – значение переменной **a**

ждать ввода с клавиатуры двух целых чисел (через пробел или *Enter*), первое из них записать в переменную **a**, второе – в **b**

Сложение двух чисел

Задача. Ввести два целых числа и вывести на экран их сумму.

```
#include <stdio.h>
main()
{
   int a, b, c;
   printf("Введите два целых числа\n");
   scanf ("%i%i", &a, &b);
   c = a + b;
   printf("%d", c);
}
```

подсказка для ввода

ввод двух чисел с клавиатуры

вывод результата

Что неправильно?

```
int a, b;
scanf ("%i", a);
scanf ("%i", &a, &b);
scanf ("%i%i", &a);

scanf ("%i %i", &a, &b);
scanf ("%f%f", &a, &b);
```

&a

%d%d

&a, &b

убрать пробел

%i%i

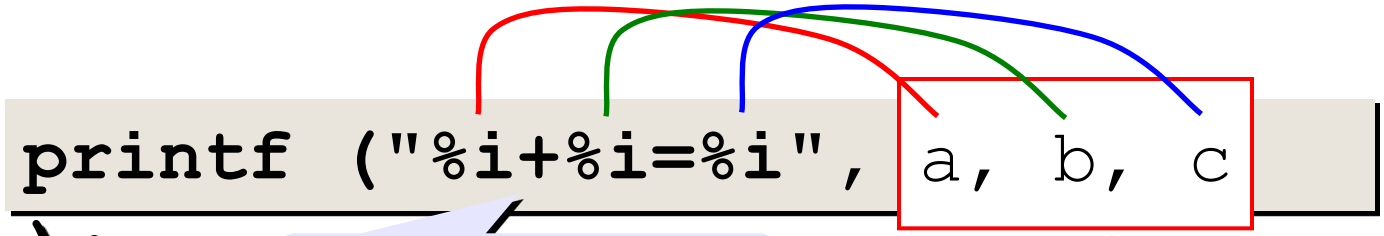
Вывод чисел на экран

здесь вывести целое число

это число взять из ячейки **c**

```
printf ("%i", c);
```

```
printf ("Результат: %i", c);
```



```
printf ("%i+%i=%i", a, b, c
);
```

формат вывода

список значений

```
printf ("%i+%i=%i", a, b, a+b
);
```

арифметическое выражение

Вывод целых чисел

```
int x = 1234;
printf ("%d", x);
```

или "%i"

```
1234
```

минимальное число позиций

или "%9i"

```
printf ("%9d", x);
```

```
    1234
```

всего 9 позиций

5 4

Вывод вещественных чисел

```
float x = 123.4567;
printf ("%f", x);
```

```
123.456700
```

минимальное число позиций, **6 цифр** в дробной части

```
printf ("%9.3f",
x);
```

```
  123.456
```

всего 9 позиций, **3 цифры** в дробной части

```
printf ("%e", x);
```

```
1.234560e+02
```

стандартный вид: **$1{,}23456 \cdot 10^2$**

```
printf ("%10.2e", x);
```

```
  1.23e+02
```

всего 10 позиций, **2 цифры** в дробной части мантиссы

Полное решение

```
#include <stdio.h>
main()
{
   int a, b, c;
   printf("Введите два целых числа\n");
   scanf("%d%d", &a, &b);
   c = a + b;
   printf("%d+%d=%d", a, b, c);

}
```

Протокол:

```
Введите два целых числа
25 30
25+30=55
```

это выводит компьютер

это вводит пользователь

В ФОРМАТЕ С++

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

Пример:

```
#include <iostream>
using namespace std;
int main()
{
int n;
cout << "Введите количество элементов:\n ";
cin >> n;
. . . . . . .
}
```

<< - операция записи данных в поток

>> - операция чтения данных из потока

БАЗОВЫЕ КОНСТРУКЦИИ

линейная, разветвляющаяся и циклическая

Операторы управления работой программы называются управляющими конструкциями программы:

- **составные операторы;**
- **операторы выбора(управления);**
- **операторы циклов;**
- **операторы перехода.**

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.

Например:

составной оператор	блок

{
n++;
summa+=n;
}
	{
	int n=0;
	n++;
	summa+=n;
	}

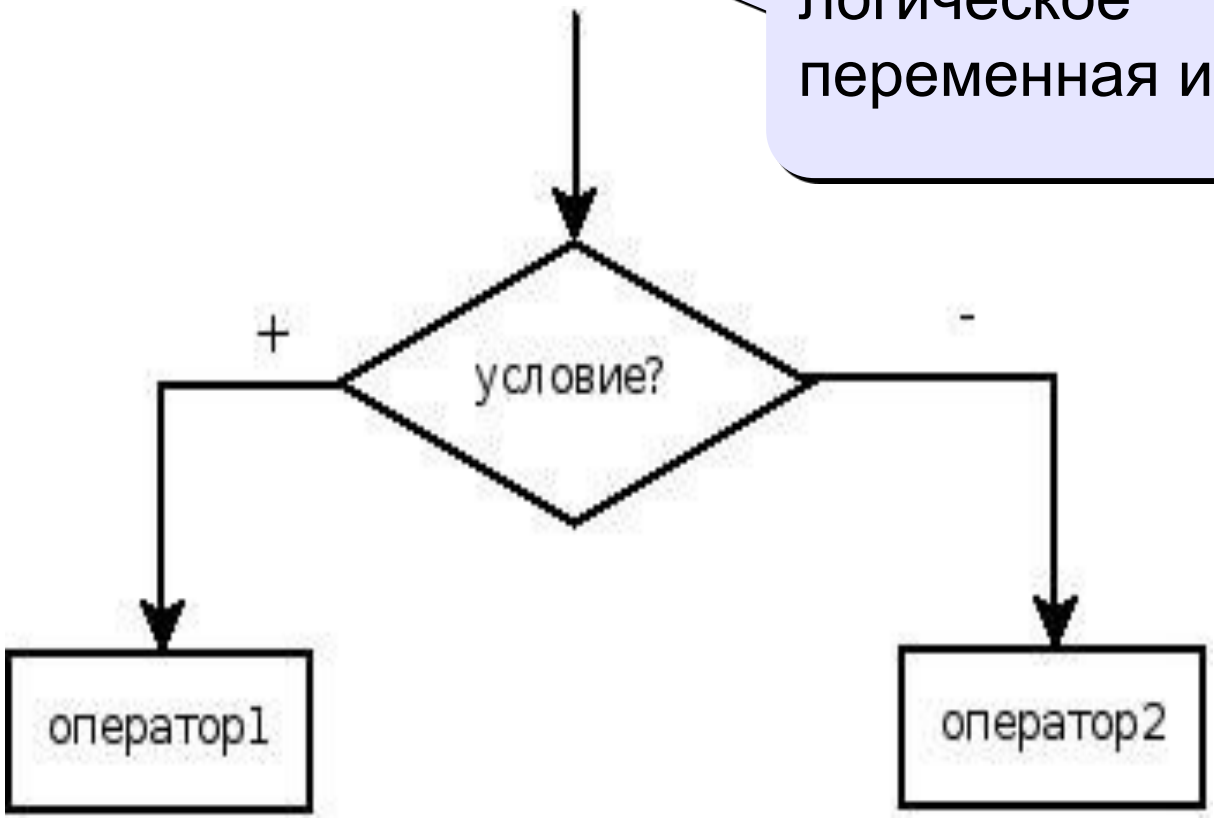
Транслятор воспринимает составной оператор как одно целое.

ОПЕРАТОРЫ УПРАВЛЕНИЯ

УСЛОВНЫЙ ОПЕРАТОР

```
if (условие) оператор_1; else
оператор_2;
```

логическое выражение, переменная или константа.



В операторе if требуется, чтобы выполнялся не один оператор, а несколько	В операторе может отсутствовать ветвь else
if (условие) **{** оператор**_1;** оператор**_2;** **...** **}** **else** **{** оператор**_1;** оператор**_2;** **...** **}**	**if (условие) оператор;** **или так** **if (условие)** **{** оператор**_1;** оператор**_2;** **...** **}**

Что неправильно?

```
if ( a > b ) {
  a = b;
  }
else
  b = a;
```

```
if ( a > b ) {
  a = b; }
else
  b = a;
```

```
if ( a > b ) a = b;
else
  b = a;
```

```
if ( a > b ) {
  a = b;
  c = 2*a; }
else
  b = a;
```

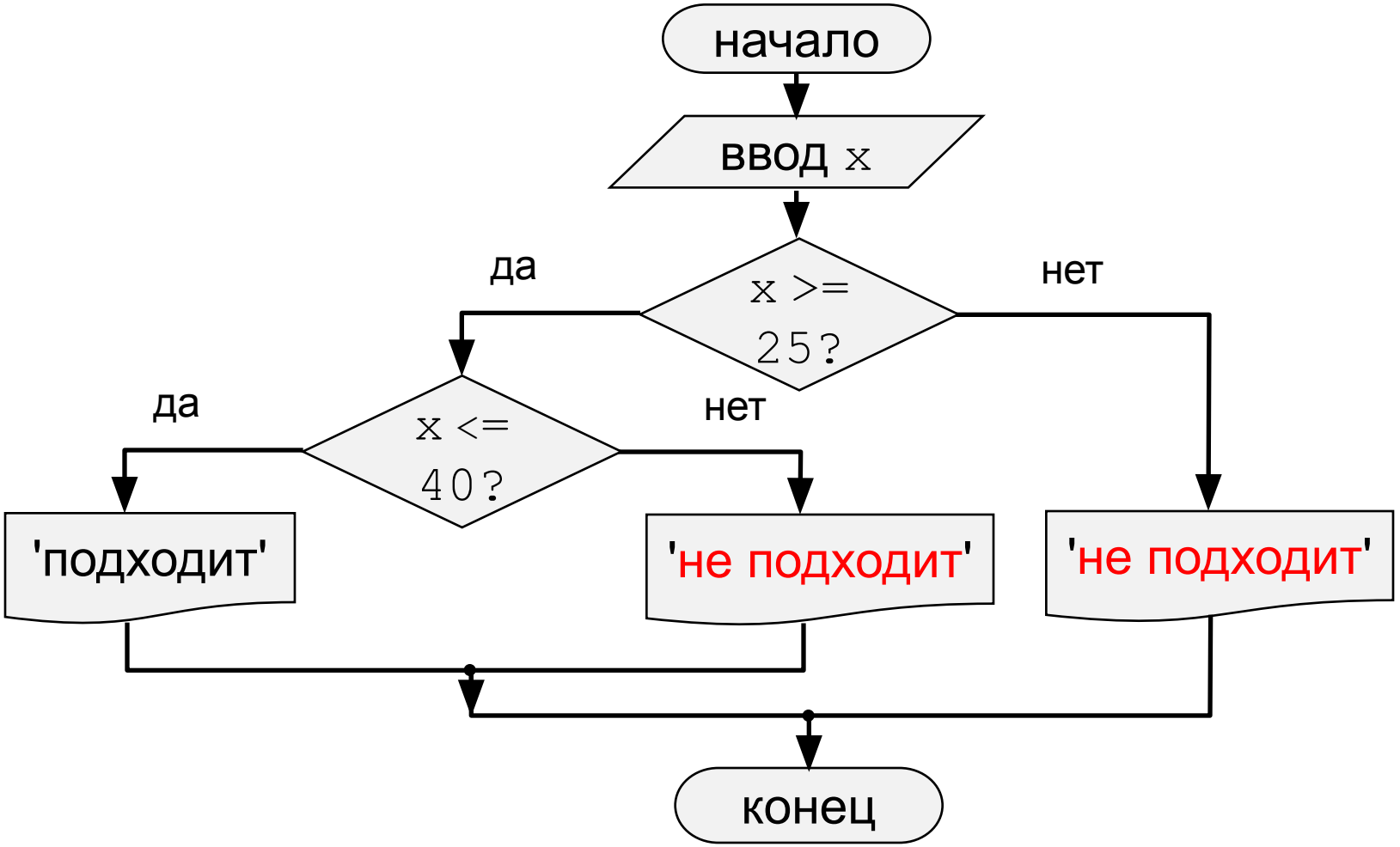
Сложные условия

Сложные условия

Задача. Фирма набирает сотрудников от 25 до 40 лет включительно. Ввести возраст человека и определить, подходит ли он фирме (вывести ответ «подходит» или «не подходит»).

Особенность: надо проверить, выполняются ли два условия одновременно.

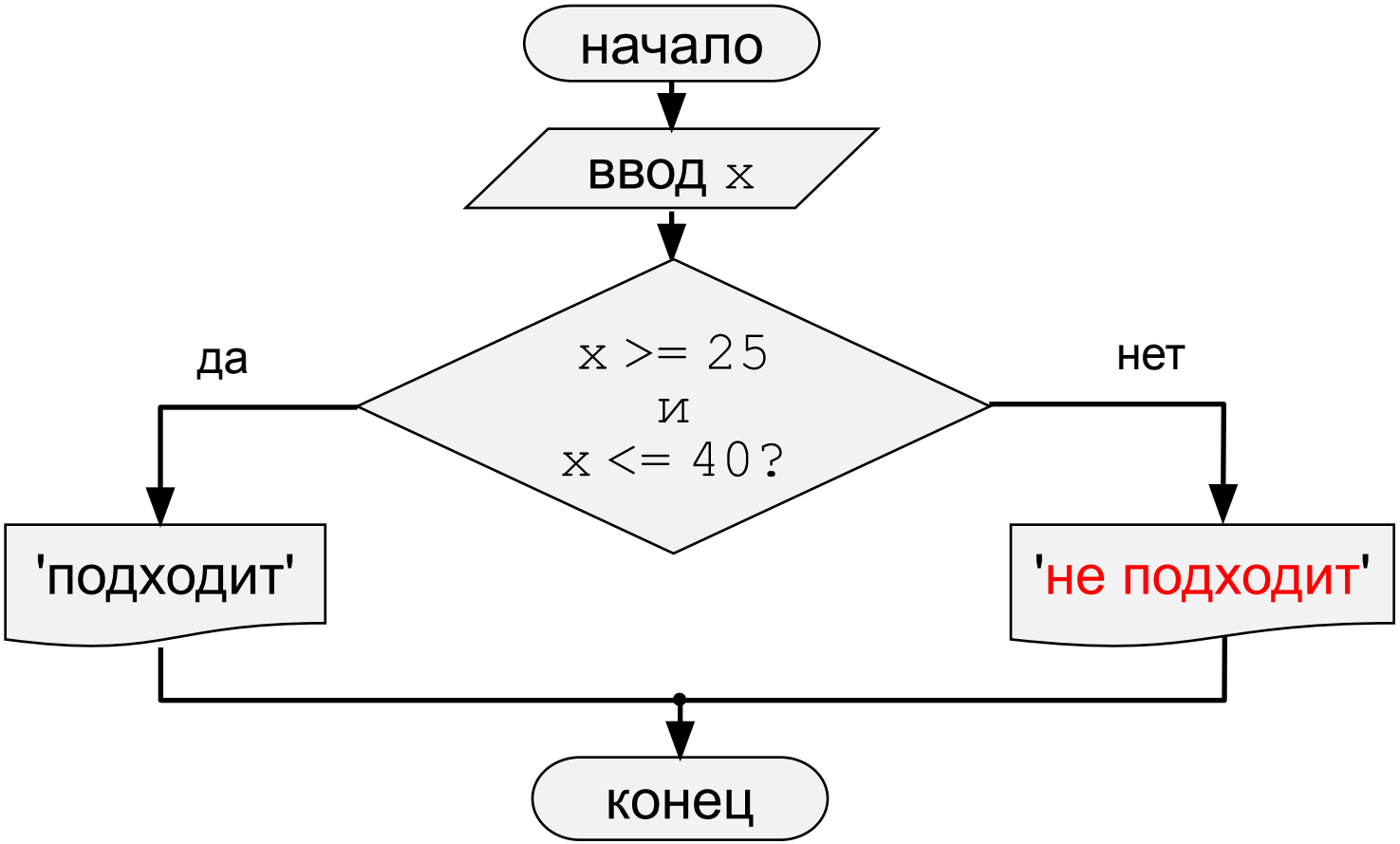
Вариант 1. Алгоритм



Вариант 1. Программа

```
main()
{
 int x;
 cout<<"Введите возраст\n";
 cin>>x;
 if (x >= 25)
   if (x <= 40)
        cout<<"Подходит";
   else cout<<"Не подходит";
 else
   cout<<"Не подходит";
}
```

Вариант 2. Алгоритм



Вариант 2. Программа

```
main()
{
int x;
cout<<"Введите возраст\n";
cin>>x;
if ( x >= 25 && x <= 40 )
      cout<<"Подходит";
else cout<<"Не подходит";
}
```

сложное условие

Сложные условия

Сложное условие – это условие, состоящее из нескольких простых условий (отношений), связанных с помощью логических операций:

- `!` – НЕ (*not*, отрицание, инверсия)
- `&&` – И (*and*, логическое умножение, конъюнкция, одновременное выполнение условий)
- `||` – ИЛИ (*or*, логическое сложение, дизъюнкция, выполнение хотя бы одного из условий)

Простые условия (отношения)

`<` `<=` `>` `>=` `==` `!=`

равно

не равно

Сложные условия

Порядок выполнения сложных условий:

- выражения в скобках
- `!` (НЕ, отрицание)
- `<, <=, >, >=`
- `==, !=`
- `&&` (И)
- `||` (ИЛИ)

Пример:

```
       2    1       6    3      5       4
if ( !(a > b) || c != d && b == a)
   {
   ...
   }
```

Сложные условия

Истинно или ложно при `a = 2; b = 3; c = 4;`

```
!(a > b)
a < b && b < c
!(a >= b) || c == d
a < c || b < c && b < a
a > b || !(b < c)
```

1

1

1

1

0

Для каких значений `x` истинны условия:

```
x < 6 && x < 10
x < 6 && x > 10
x > 6 && x < 10
x > 6 && x > 10
x < 6 || x < 10
x < 6 || x > 10
x > 6 || x < 10
x > 6 || x > 10
```

$(-\infty, 6)$	$x < 6$
$\varnothing$	
$(6, 10)$	
$(10, \infty)$	$x > 10$
$(-\infty, 10)$	$x < 10$
$(-\infty, 6) \cup (10, \infty)$	
$(-\infty, \infty)$	
$(6, \infty)$	$x > 6$

ОПЕРАТОР ВЫБОРА

Оператор **switch** необходим в тех случаях, когда в зависимости от значений какой-либо переменной надо выполнить те или иные операторы:

```
switch (выражение)
{
case значение_1: Операторы_1; break;
case значение_2: Операторы_2; break;
case значение_3: Операторы_3; break;
...
case значение_n: Операторы_n; break;
default: Операторы; break;
}
```

Если оператор **break** не указан, то будут выполняться следующие операторы из списка, не смотря на то, что значение, которым они помечены, не совпадает со значением выражения.

Программа

```
main()
{
 int M, D;
 printf("Введите номер месяца:\n");
 scanf("%d", &M);
 switch ( M ) {
   case 2:  D = 28; break;
   case 4: case 6: case 9: case 11:
            D = 30; break;
   case 1: case 3: case 5: case 7:
   case 8: case 10: case 12:
            D = 31; break;
   default: D = -1;
   }
 if (D > 0)
      printf("В этом месяце %d дней.", D);
 else printf("Неверный номер месяца");
}
```

выйти из **switch**

ни один вариант не подошел

Задача: Ввести букву и вывести название животного на эту букву.

Особенность: выбор по символьной величине.

```
main()
{
 char c;
 cout<<"Введите первую букву названия животного:\n";
 cin>>c;
 switch ( c ) {
   case 'а': cout<<"Антилопа"; break;
   case 'б': cout<<"Бизон"; break;
   case 'в': cout<<"Волк"; break;
   default:  cout<<"Я не знаю!";
   }
}
```

Оператор выбора

Особенности:

- после `switch` может быть имя переменной или арифметическое выражение целого типа (`int`)

```
switch ( i+3 ) {
   case 1: a = b; break;
   case 2: a = c;
   }
```

  или символьного типа (`char`)
- нельзя ставить два одинаковых значения:

```
switch ( x ) {
   case 1: a = b; break;
   case 1: a = c;
   }
```

ОПЕРАТОРЫ ЦИКЛА

ОПЕРАТОР ЦИКЛА С ПРЕДУСЛОВИЕМ

while (выражение) оператор;

или

while (условие)

{

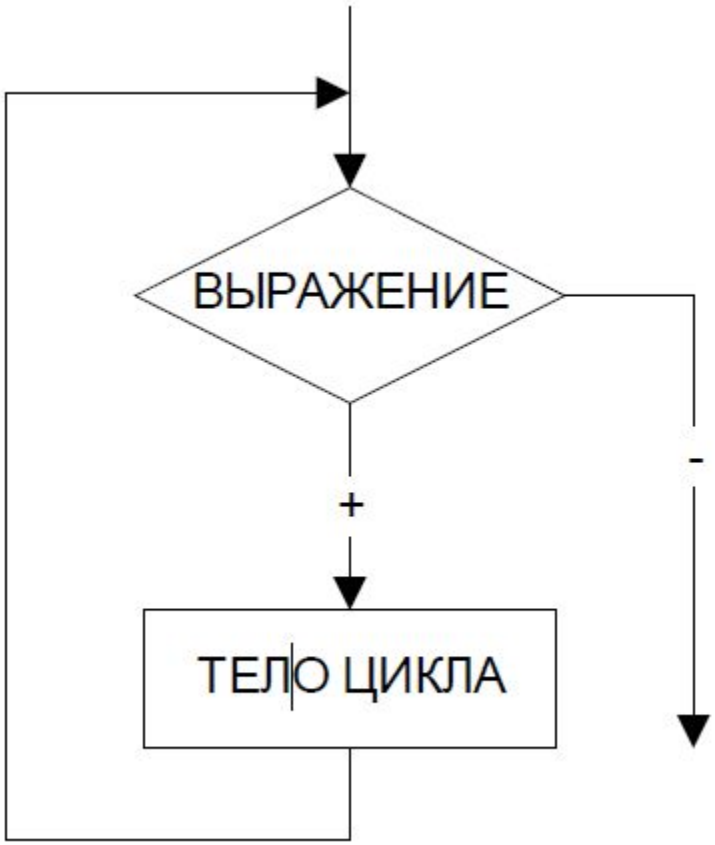
оператор **1;**

оператор **2;**

…

оператор **n;**

}

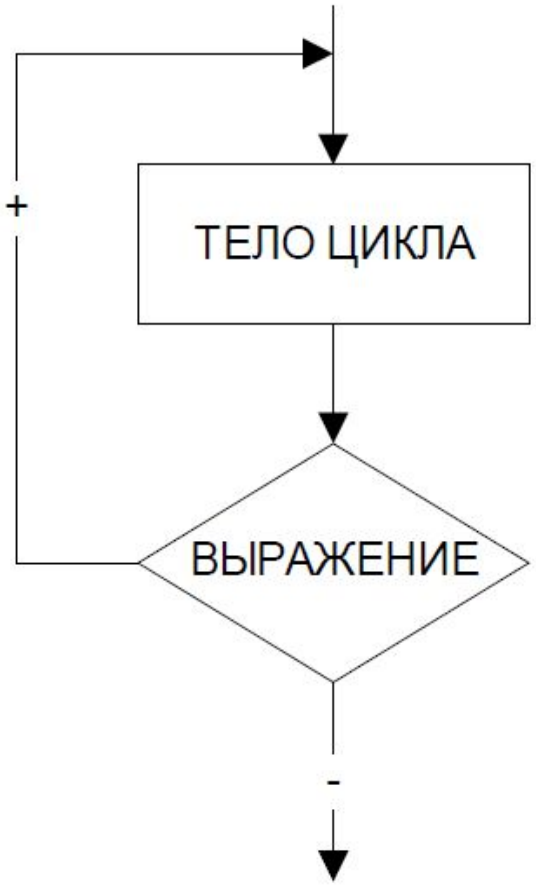


ОПЕРАТОР ЦИКЛА С ПОСТУСЛОВИЕМ

do оператор while (выражение);

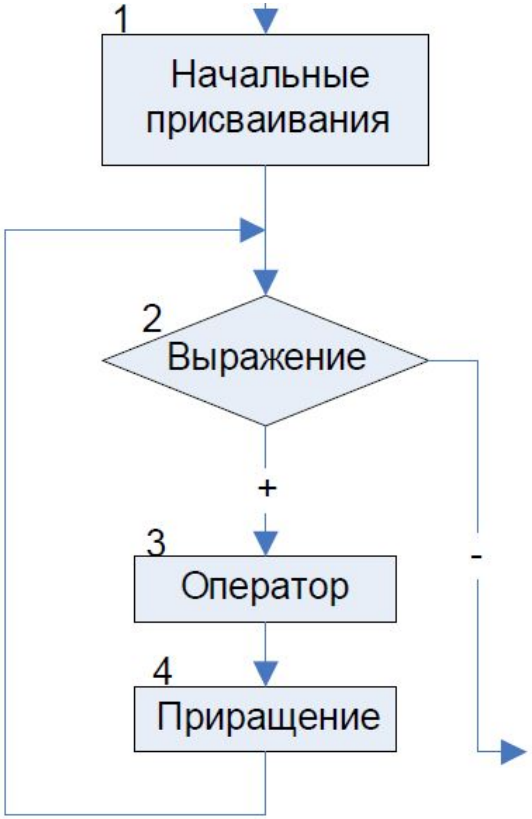
или

```
do
{
оператор_1;
оператор_2;
...
оператор_n;
} while (выражение);
```



ОПЕРАТОР ЦИКЛА С ПАРАМЕТРОМ

```
for (нач_присваивание; выражение; приращение)
{
оператор1;
оператор2;
…
}
```



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<216; num++)
{ тело цикла};
```

ИСПОЛЬЗОВАНИЕ ЦИКЛА С ПАРАМЕТРОМ

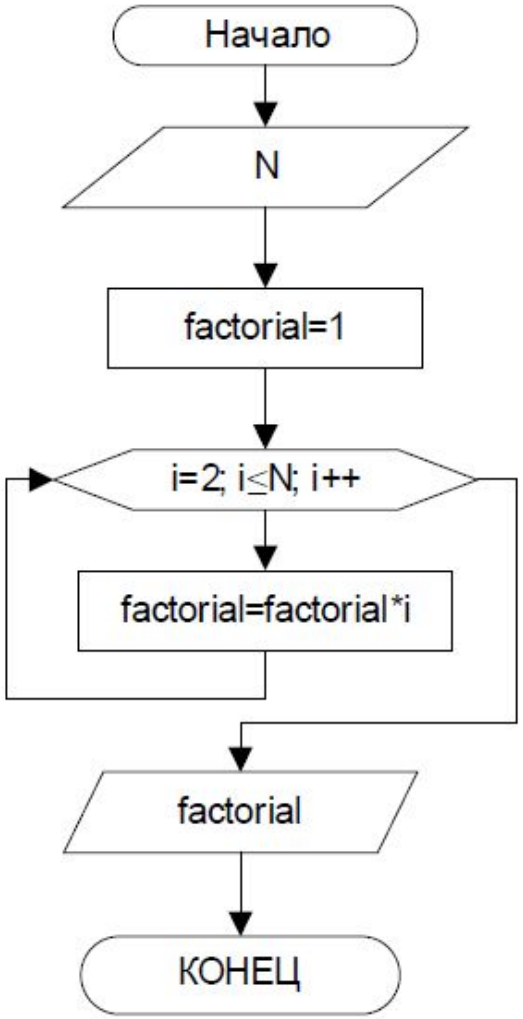
4) Коррекция может осуществляться не только с помощью сложения или вычитания:

```
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 цикл};
```

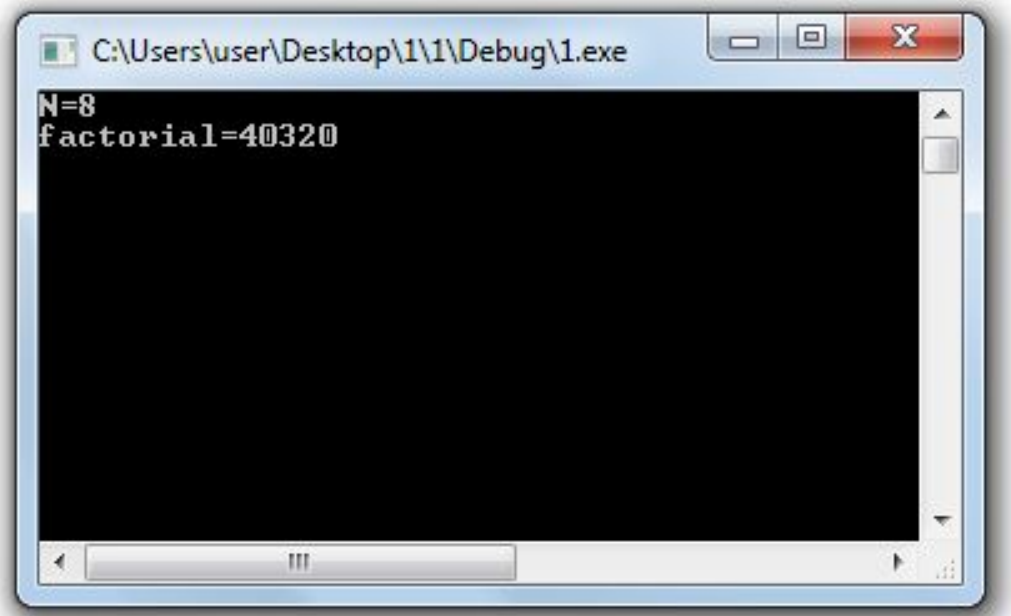
ВЫЧИСЛИТЬ ФАКТОРИАЛ ЧИСЛА N (N!=1×2×3 ×… ×N).



1.cpp

(Глобальная область)

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
unsigned int factorial, N, i;
for (cout<<"N=",cin>>N,factorial=1,i=2;i<=N;factorial*=i,i++);
cout<<"factorial="<<factorial<<"\n";
getch();
return 0;
}
```

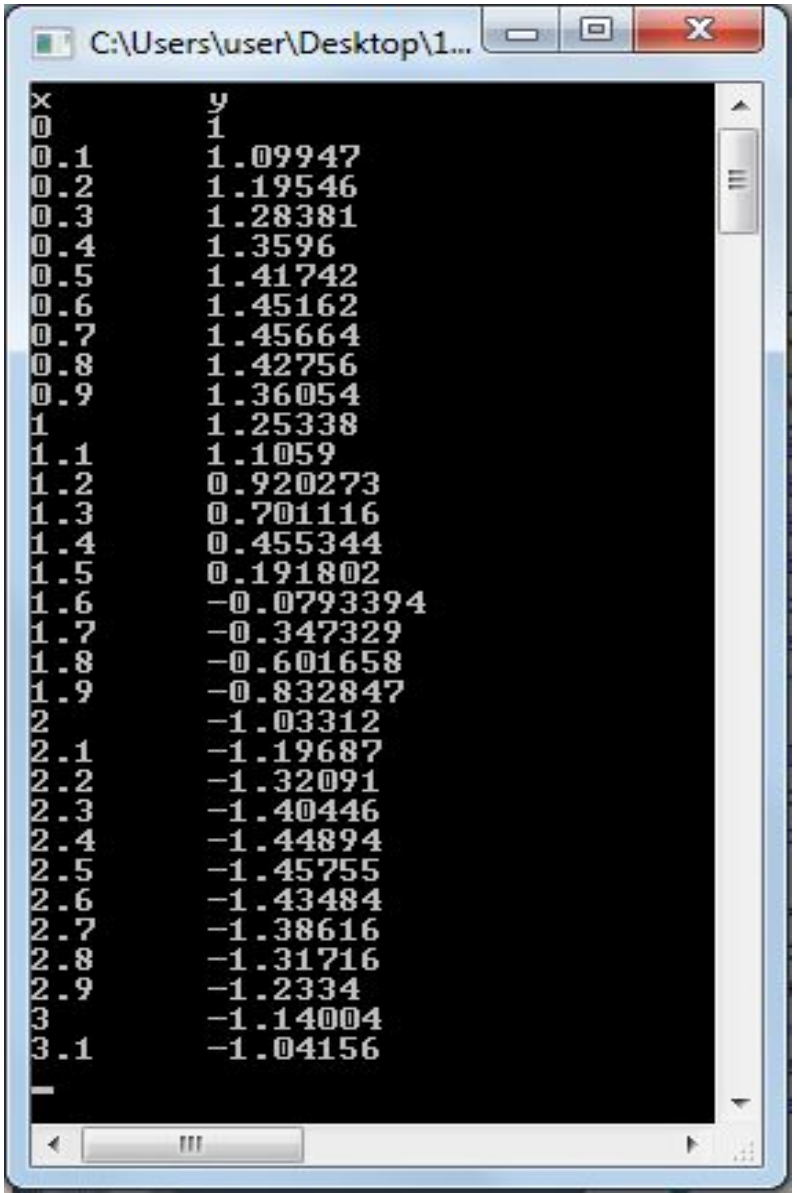


ВЫВЕСТИ НА ЭКРАН ТАБЛИЦУ ЗНАЧЕНИЙ ФУНКЦИИ $y = e^{\sin(x)}\cos(x)$ НА ОТРЕЗКЕ [0;pi] С ШАГОМ 0.1

1.cpp

(Глобальная область)

```
#include "stdafx.h"
#include <iostream>
#include <math.h>
#include <conio.h>
#define PI 3.14
using namespace std;
int main()
{
float x=0, y;
cout<<"x\ty\n";
do
{
y=exp(sin(x))*cos(x);
cout<<x<<"\t"<<y<<"\n";
x+=0.1;
}while(x<=PI);
getch();
return 0;
}
```



ОПЕРАТОРЫ ПЕРЕХОДА

Операторы, которые принудительно изменяют порядок выполнения команд

1. goto метка;

…

метка**: оператор;**

обычный идентификатор

2. break осуществляет немедленный выход из циклов while, do - while и for, а также из оператора выбора switch.

3. continue прерывает выполнение данного шага цикла.

4. return завершает выполнение функции и передает управление в точку ее вызова.

Отладка программ

Отладка программ

Отладка – поиск и исправление ошибок в программе.

Англ. *debugging*, *bug = моль, жучок*

Методы:

- трассировка – вывод сигнальных сообщений
- отключение части кода (в комментарии)
- пошаговое выполнение – выполнить одну строчку программы и остановиться
- точки останова – выполнение программы останавливается при достижении отмеченных строк (переход в пошаговый режим)
- просмотр и изменение значений переменных в пошаговом режиме

Трассировка

```
main()
{
 int i, X;
 printf("Введите целое число:\n");
 scanf("%d", &X);
 printf("Введено X=%d\n", X);
 for(i=1; i<10; i++)
    {
    printf("В цикле: i=%d, X=%d\n", i, X);
    ...
    }
 printf("После цикла: X=%d\n", X);
 ...
}
```

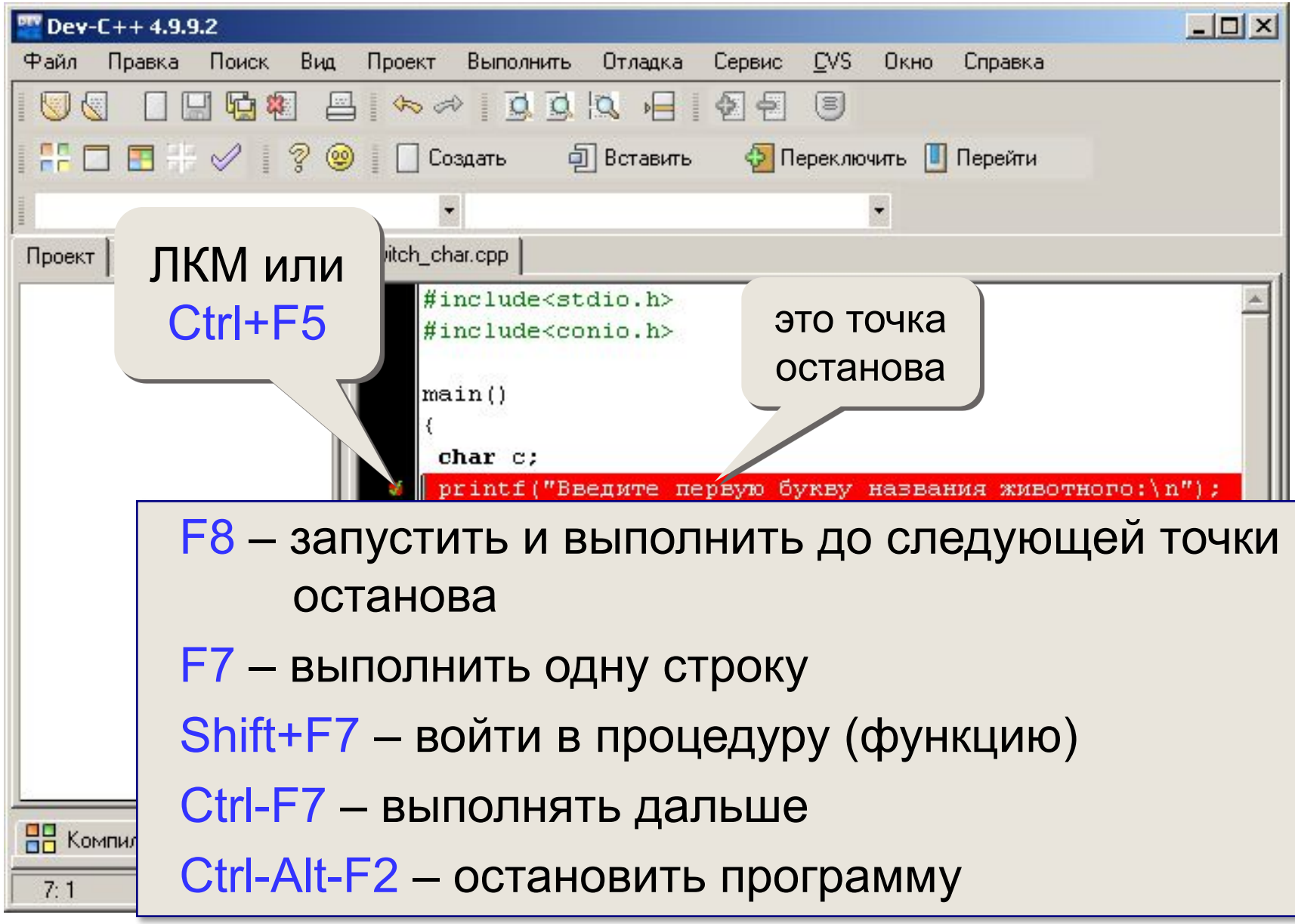
Отключение части кода (комментарии)

```
main()
{
 int i, X;
 printf("Введит  целое число:\n");
 scanf("%d", &X);
 // X *= X + 2;
 for(i=1; i<10; i++) X *= i;
 /* while ( X > 5 ) {
    ...
    } */
 ...
}
```

комментарий до конца строки //

закомментированный блок /* … */

Точки останова



F8 – запустить и выполнить до следующей точки останова

F7 – выполнить одну строку

Shift+F7 – войти в процедуру (функцию)

Ctrl-F7 – выполнять дальше

Ctrl-Alt-F2 – остановить программу

Просмотр значений переменных