ОПЕРАТОРЫ В ЯЗЫКЕ С

ОПЕРАТОРЫ В ЯЗЫКЕ С .

Все операторв языка С можно разделить на пять групп:

- Оператор «**выражение**».
- Оператор **ветвления**.
- Операторы **цикла**.
- Оператор **множественного выбора**.
- Операторы **передачи управления**.

Оператор «выражение»

- Любое выражение, завершающееся точкой с запятой, рассматривается как оператор, выполнение которого заключается в вычислении выражения.
- Частным случаем выражения является пустой оператор (*он используется, когда по синтаксису оператор требуется, а по смыслу — нет*).

Примеры:

```
i++;      // выполняется операция
          // инкремента

a* = b + c;    // выполняется умножение
               // с присваиванием

fun(i, k);   // выполняется вызов функции
```

Оператор «выражение»

Оператор ветвления

- Условный оператор **if** используется для разветвления процесса вычислений на два направления.

Формат оператора:

if (выражение) оператор_1;

[else оператор_2;]

Оператор ветвления

Условный оператор if

- логичнее опускать вторую ветвь вместе с ключевым словом **else**

- Если в какой-либо ветви требуется выполнить несколько операторов, их необходимо заключить в блок.
- Блок может содержать любые операторы, в том числе описания и другие условные операторы, **но не может состоять из одних описаний**

Обратите внимание!

Примеры:

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

if (a<b && (a>d  || a==0)) b++; else {b *= a; a = 0;}                    // 2

if (a<b) {if (a<c) m = a; else m = c;}  else { if (b<c) m = b; else m = c; }   // 3

if (a++) b++;                       // 4

if (b>a) max = b;  else max = a;       // 5
```

Условный оператор if

1

2

3

4

5

Условный оператор if

Распространенная ошибка при записи условных операторов — использование в выражениях вместо проверки на равенство (==) простого присваивания (=), например, **if(a=1)b=0;**

- синтаксической ошибки нет, так как операция присваивания формирует результат, который и оценивается на равенство/неравенство нулю
- в данном примере присваивание переменной **b** будет выполнено независимо от значения переменной **a**
- поэтому в выражениях проверки переменной на равенство константе константу рекомендуется записывать слева от операции сравнения: **if (1==a) b=0;**.

Обратите внимание!

Вторая ошибка — неверная запись проверки на принадлежность диапазону

например, чтобы проверить условие **0<x<1**, нельзя записать его в условном операторе непосредственно

- так как будет выполнено сначала сравнение **0<x**, а его результат, преобразованный в **int** будет сравниваться с **1**.
- правильный способ записи: **if (0<x && x<1)…**

Обратите внимание!

Если какая-либо переменная используется только внутри условного оператора, рекомендуется объявить ее внутри скобок

- **if (int i = fun(t)) a -= i; else a += i;**

*Объявлять внутри оператора **if** можно только одну переменную*

- *область ее видимости начинается в точке объявления и включает обе ветви оператора*

Обратите внимание!

Объявление переменной в тот момент, когда она требуется, то есть когда ей необходимо присвоить значение, является признаком хорошего стиля и позволяет избежать случайного использования переменной до ее инициализации.

Операторы цикла

- Цикл с предусловием (**while**).
- Цикл с постусловием (**do while**).
- Цикл с параметром (**for**).

Цикл с предусловием (while)

Формат оператора:

while (выражение) оператор

- **Выражение** определяет условие повторения тела цикла, представленного простым или составным оператором.
- Выполнение оператора начинается с вычисления **выражения**.
  - Если оно истинно, то выполняется оператор цикла.
  - Если при первой проверке **выражение** равно **false**, цикл не выполнится ни разу.
- Тип **выражения** должен быть арифметическим или приводимым к нему.
- **Выражение** вычисляется перед каждой итерацией цикла.

Пример:

```
#include <iostream.h>
#include <conio.h>
int main(){
    clrscr();
    cout << "\nНажмите любую кнопку на клавиатуре";
    while(!kbhit());
    cout << "\nБлагодарю за сотрудничество";
    return 0;
}
```

Цикл с предусловием (while)

В круглых скобках после ключевого слова ***while*** *можно вводить описание переменной*

Областью ее действия является цикл

- **while (int x = 0){ … /* область действия x */ }**

Обратите внимание!

Формат оператора:

do оператор while выражение;

- Сначала выполняется простой или составной **оператор**, составляющий тело цикла.
- Затем вычисляется **выражение**.
  - Если оно истинно, тело цикла выполняется еще раз.
- Цикл завершается, когда **выражение** станет равным **false** или в теле цикла будет выполнен какой-либо **оператор передачи управления**.
- Тип **выражения** должен быть арифметическим или приводимым к нему.

Цикл с постусловием (do while)

Пример:

```
// программа находит все делители целого числа
#include <iostream.h>
int main(){
    int num;
    cout << "\nВведите число :  ";  cin >> num;
    int half = num / 2  // половина числа
    int div = 2    // кандидат на делитель
    do{
        if (!(num % div)) cout << div << "\n";
        div++;
    }while(div <= half);
    return 0;
}
```

Цикл с постусловием (do while)

Формат оператора:

for (инициализация; выражение; модификации) оператор;

Цикл с параметром (for)

Инициализация *используется для объявления и присвоения начальных значений величинам, используемым в цикле.*

- в этой части можно записать несколько операторов, разделенных запятой (операцией «последовательное выполнение»)

например, так:

```
for (int i  = 0,  j = 2;  ...
int k,  m;
for (k = 1, m = 0; ...
```

Областью действия переменных, объявленных в части инициализации цикла, является цикл

Инициализация выполняется один раз в начале исполнения цикла

Цикл с параметром (for)

Выражение *определяет условие выполнения цикла*

- если его результат, приведенный к типу **bool**, равен **true**, цикл выполняется

Цикл с параметром реализован как цикл с предусловием

Цикл с параметром (for)

Модификации *выполняются после каждой итерации цикла и служат обычно для изменения параметров цикла*

В части модификаций можно записать несколько операторов через запятую

Цикл с параметром (for)

Простой или составной

оператор

представляет собой тело цикла

Цикл с параметром (for)

Любая из частей оператора for может быть опущена

но точки с запятой надо оставить на своих местах!

Примеры:

```
// оператор, вычисляющий сумму чисел от 1 до 100
    for (int i = 1, s = 0; i<=100;  i++) s += i;

// программа находит все делители целого числа
#include <iostream.h>
int main(){
    int num, half, div;
    cout << "\nВведите число :  ";  cin >> num;
    for (half = num / 2, div = 2; div <= half; div++)
    if (!(num % div)) cout << div <<"\n";
    return 0;
}
```

Цикл с параметром (for)

Оператор множественного выбора

- Оператор **switch** (переключатель) предназначен для выбора одного из нескольких направлений вычислений.

Формат оператора:

```
switch ( выражение ){
    case константное_выражение_1: [список_операторов_1]
    case константное_выражение_2: [список_операторов_2]
    case константное_выражение_n: [список_операторов_n]
    [default: операторы ]
 }
```

Оператор множественного выбора

Оператор множественного выбора

Выход из переключателя обычно выполняется с помощью операторов **break** или **return**

- Оператор **break** выполняет выход из самого внутреннего из объемлющих его операторов **switch**, **for**, **while** и **do**
- Оператор **return** выполняет выход из функции, в теле которой он записан

Оператор множественного выбора

Все константные выражения должны иметь разные значения, но быть

одного и того же целочисленного типа

Несколько меток могут следовать подряд

Оператор множественного выбора

Пример:

```
// программа реализует простейший калькулятор
#include <iostream.h>
int main(){
     int a, b, res;
     char op;
     cout << "\nВведите 1й операнд:  ";  cin >> a;
     cout << "\nВведите знак операции:  ";  cin >> op;
     cout << "\nВведите 2й операнд:  ";  cin >> b;
     bool f= true;
     switch (op) {
     case '+'    : res=a+b;     break;
     case '-'    : res=a-b; break;
     case '/'    : res=a/b;     break;
     case '*'    : res=a*b;     break;
     default     : cout << "\nНеизвестная операция"; f=false
     }
     if (f) cout << "\nРезультат :" << res;
     return 0;
}
```

Оператор множественного выбора

В случае синтаксической ошибки в слове default
сообщение об ошибке не выдается, поскольку компилятор воспримет это слово как допустимую метку оператора.

Операторы передачи управления

- оператор безусловного перехода **goto**
- оператор выхода из цикла **break**
- оператор перехода к следующей итерации цикла **continue**
- оператор возврата из функции **return**

Формат оператора:

goto метка;

*В теле той же функции должна присутствовать ровно **одна** конструкция вида:* **метка: оператор;**

Оператор **goto** *передает управление на помеченный оператор*

- **Метка** — *это обычный идентификатор, областью видимости которого является функция, в теле которой он задан*

Оператор goto

*Не следует передавать управление внутрь операторов **if, switch** и циклов*

Нельзя переходить внутрь блоков, содержащих инициализацию переменных, на операторы, расположенные после нее, поскольку в этом случае инициализация **не будет выполнена**:

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

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

Оператор goto

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

Применение **goto** нарушает принципы структурного и модульного программирования

- *все блоки, из которых состоит программа, должны иметь только один вход и один выход*

Оператор goto

Оператор **break** используется внутри **операторов цикла** или **switch** для обеспечения перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится **break**

Оператор break

Оператор перехода к следующей итерации цикла **continue** пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации.

Оператор continue

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

Формат оператора:

return [выражение];

Выражение должно иметь скалярный тип

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

Оператор return