**Операторы языка Си**

Все операторы условно могут быть разделены на категории:

1. Условные операторы (**if, switсh**);
2. Операторы цикла (**for, while, do**);
3. Операторы переходов (**break, continue, return, goto**);
4. Операторы “выражение” ,”,”, “пустой оператор”, “составной оператор”.

Операторы в программе могут объединяться в составные операторы с помощью фигурных скобок.

Любой оператор языка Си может быть помечен меткой, состоящей из имени и следующего за ним двоеточия.

Все операторы языка Си, кроме составных операторов, заканчиваются точкой с запятой (;).

1. **Оператор ”выражение”**

Этот оператор заключается:

* 1. в вычислении выражения и присваивания;
  2. в вызове функций.

**Пример**:

++i;

fltVar=cos(a\*5);

swap(&x,&y)**;**

1. **Пустой оператор**

Состоит только из точки с запятой. При выполнении ничего не происходит. Используется в операторах **do, for, while, if** в случаях, когда тело оператора не требуется, хотя по синтаксису требуется хотя бы один оператор. А также, когда необходимо пометить фигурную скобку меткой. Синтаксис требует, чтобы после метки обязательно следовал оператор. Фигурная скобка оператором не является. Поэтому, если надо передать управление на фигурную скобку, необходимо использовать пустой оператор.

**Пример:**

void main()

{

…

{

goto end:;

…

}

end:;

}

1. **Составной оператор**

Формат:

**{**[объявление]

…

оператор

[оператор]

…

**}**

В конце составного оператора точка с запятой не ставится. Выполнение составного оператора заключается в последовательном выполнении составляющих его операторов.

1. Оператор **if.**

**if** (выражение)

оператор1

[**else** оператор2]

**Пример**: **if (i<j)**

**i++;**

**else {**

**j=i-3;**

**i++;**

**}**

Вычисление оператора **if** начинается с вычисления **выражения**.

Если **выражение** истинно (не ноль), то выполняется **оператор1,** иначе **оператор2**.

Если **выражение** ложно и отсутствует фраза **else**, то выполняется следующий за **if** **оператор**.

Допускается использование вложенных операторов **if**

**Пример**:

|  |  |
| --- | --- |
| int main()  {  int iVar1=2, iVar2=7, iVar3=3;  if (iVar1>iVar2)  {  if ( iVar2<iVar3)  iVar3=iVar2;  }  else  iVar3=iVar1;  printf(“iVar3=%d\n”,iVar3);  } | int main()  {  int iVar1=2, iVar2=7, iVar3=3;  if (iVar1>iVar2)    if ( iVar2<iVar3)  iVar3=iVar2;    else  iVar3=iVar1;  printf(“iVar3=%d\n”,iVar3);  } |

1. Оператор **switch**

предназначен для организации выбора из множества вариантов.

**Формат**:

switch (выражение){

[case константное\_выражение:]

…

[список операторов]

[case константное\_выражение:]

…

[список операторов]

…

[default: список оператор]

**}**

**Выражение**, следующее за ключевым словом **switch**, может быть произвольным выражением, значение которого должно быть целым.

Константное выражение вычисляется во время компиляции. Оно не может содержать переменные или вызовы функций. Обычно используются целые или символьные константы.

**Замечание:** Программист должен сам позаботиться о выходе из **case**, иначе все выполняется до конца.

**Пример:**

int a=2;

switch(a) {

case 1: func1();

case 2: func2();

case 3: func3();

case 4: func4();

default: printf(“Good bye\n”);

}

Выполнение будет идти от **func2()** и до конца.

1. Оператор **break**

Обеспечивает прекращение выполнения самого внутреннего из объемлющих его операторов **switch, do, for** и **while.** После выполнения оператора **break** управление передается оператору, следующему за прерванным.

**Пример:**

#include<stdio.h>

#include<stdlib.h>

int main()

{

char cSign;

float x, y = 2, z = 5;

printf("Enter the operator sign: (+ - \* / ) :");

scanf\_s("%c", &cSign, 1);

switch (cSign) {

case '-':x = y - z;

break;

case '+':x = y + z;

break;

case '\*':x = y \* z;

break;

case '/':x = y / z;

break;

default: printf("Unknown operation.");

x = 0;

}

printf("x=%f\n", x);

system("pause");

return 0;

}

1. Оператор **for**

Оператор цикла **for** имеет следующий формат:

for([выражение\_инициализация];[условное выражение];[выражение\_итерация])

оператор

Выражение\_инициализация используется для установки начального значения переменных, управляющих циклом.

Условное выражение — выражение, определяющее условие, при котором оператор цикла будет исполняться.

Выражение\_итерация определяет изменение переменных, управляющих циклом после каждого выполнения цикла.

Схема выполнения оператора **for**:

1. вычисляется выражение\_инициализация;
2. вычисляется условное выражение;
3. если значение условного выражения не равно нулю, выполняется оператор;
4. вычисляется выражение\_итерация;
5. вновь вычисляется условное выражение;
6. если условное выражение равно нулю, управление передается на оператор, следующий за оператором **for**.

**Замечание:** Проверка условия всегда выполняется в начале цикла. Это значит, что цикл может ни разу не выполниться, если условное выражение сразу будет ложным.

**Пример:**

#include<stdio.h>

#include<stdlib.h>

#include<windows.h>

int main()

{

char cStr[] = "Квадрат числа ";

int i;

CharToOem(cStr, cStr);

for (i = 0; i <= 10; i++)

printf("%s%2d=%3d\n", cStr, i, i\*i);

system("pause");

return 0;

}

Результат — квадрат чисел от 0 до 10.

Проверить работу программы, если изменить 7-ую строку на:

**for( i=0; i<=10; i++); //** добавили **;**

1. Оператор **while**

**Формат:**

while(выражение)

оператор

Схема выполнения:

1. вычисляется выражение;
2. если выражение ложно (=0), то тело оператора **while** не выполняется, а управление передается на следующий за **while** оператор;
3. если выражение истинно (не равно нулю), то тело оператора **while** выполняется;
4. процесс повторяется с шага 1.

**Пример:**

int i=0;

while (i < 10)

{

printf("i = %d\n", i);

i++;

}

1. Оператор **do**

Используется в тех случаях, когда тело цикла должно выполняться хотя бы один раз.

**Формат:**

do

{

оператор;

}

while (выражение);

Схема выполнения оператора **do**:

1. выполняется оператор;
2. вычисляется выражение. Если выражение не равно нулю, то выполнение продолжается с шага 1.
3. Оператор **continue**

Этот оператор работает подобно оператору break, но в отличие от break прерывает выполнение тела цикла и передает управление на следующую итерацию.

Формат: continue

**Пример: /\* вывод четных чисел\*/**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int iVar;

for (iVar = 0; iVar < 100; iVar++)

{

if (iVar % 2) continue;

printf("%d\n", iVar);

}

system("pause");

return 0;

}

Оператор **return**

Оператор завершает выполнение функции, в которой он задан, и возвращает управление в вызывающую функцию. Управление передается в вызывающую функцию в точку, непосредственно следующую за вызовом.

Формат: **return [выражение];**

Значение выражения возвращается в вызывающую функцию в качестве значения вызываемой функции. Если выражение опущено, то возвращаемое функцией значение не определено.

Если функция не возвращает значения, ее следует объявить с типом void.

То есть, имеем два случая использования:

1. надо немедленно выйти из функции;
2. функция должна возвращать значение.

**Пример:**

int sum(int iVar1, int iVar2)

{

return (iVar1+iVar2);

}

**Пример:**

void print(char x)

{

if (x==0)

{ printf(“Это плохой аргумент\n”,);

return;

}

printf(“Введите аргумент %с\n”,x);

}

Оператор **return** используется для выхода из функции в случае, если аргумент равен нулю.

1. Оператор **goto**

**Рекомендуется не использовать.**

**Формат:**

**goto имя;**

**…**

**имя: оператор;**

Метка — просто идентификатор.

**Программы – примеры использования операторов языка Си**

**/\* Метод дихотомии \*/**

#include<stdio.h>

#include<stdlib.h>

double fun(double);

int main()

{

double s, a = 0., b = 1.8, dt = 1.5, e = 1E-6;

while (dt > e)

{

s = (a + b) / 2;

if (fun(a)\*fun(s) < 0)

b = s;

else

a = s;

dt = b - a;

printf("dt=%f\ta=%f\tb=%f\n", dt, a, b);

}

printf("x=%f\tdt=%f\n", a, dt);

printf("s=%10.9lf\n", fun((b + a) / 2));

system("pause");

return 0;

}

double fun(double c)

{

return (c\*c - 3 \* c + 1);

}

#include<stdio.h>

#include<stdlib.h>

sum(int, int);

int main()

{

int s, a = 10, b = 20;

s = sum(a, b);

printf("s=%d\n", s);

int i = 0, j = 0, j1 = 0;

while (j < 100)

{

j += ++i;

printf("i=%d\tj=%d\n", i, j);

if (j > 50)

{

j1 = 1;

goto m;

}

}

m:;

printf("j1=%d\n", j1);

system("pause");

return 0;

}

int sum(int iVar1, int iVar2)

{

return (iVar1 + iVar2);

}