**Группа бинарных операций**

|  |  |
| --- | --- |
| Знак операции | Операция |
| \* | Умножение |
| / | Деление |
| % | Остаток от деления |
| + | Сложение |
| - | Вычитание |
| << | Сдвиг влево |
| >> | Сдвиг вправо |
| < | Меньше |
| > | Больше |
| <= | Меньше или равно |
| >= | Больше или равно |
| = = (=) | Логическое равно |
| != | Не равно |
| & | Поразрядное логическое И |
| | | Поразрядное логическое ИЛИ |
| ^ | Поразрядное исключающее ИЛИ |
| && | Логическое И |
| || | Логическое ИЛИ |
| , | Последовательное вычисление |

Операция ”остаток от деления” (%) имеет результатом остаток от деления первого операнда на второй. Знак результата зависит от конкретной реализации. В Visual C++ знак результата совпадает со знаком первого операнда.

**Пример:/\* Остаток от деления \*/**

int iVar1=49, iVar2=10,iVar3,iVar4,iVar5,iVar6;

iVar3=iVar1%iVar2;

iVar4=-iVar1%-iVar2;

iVar5=iVar1%-iVar2;

iVar6=-iVar1%iVar2;

printf("++=%d\t--=%d\t+-=%d\t-+=%d\n",iVar3,iVar4,iVar5,iVar6);

Операции сдвига сдвигают операнд влево (**<<**) или вправо (**>>**) на число битов, заданное вторым операндом. Оба операнда должны быть целыми величинами.

**Замечание:** В языке Си ”истина” считается любое число, не равное 0.

**Пример:**

bool bBool1,bBool2,bBool3,bBool4,bBool5;

bBool1=-31;

bBool2=0;

bBool3=!bBool1;

bBool4=true;

bBool5=false;

printf("bBool1=%d\tbBool2=%d\tbBool3=%d\n",bBool1,bBool2,bBool3);

printf("bBool4=%d\tbBool5=%d\n",bBool4,bBool5);

Операция поразрядного логического **И (&)** сравнивает каждый бит первого операнда с соответствующим битом второго операнда по правилу:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |

Операция поразрядного логического **ИЛИ (|)** сравнивает каждый бит первого операнда с соответствующим битом второго операнда по правилу:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |

Операция поразрядного исключающего **ИЛИ** (**^**)сравнивает каждый бит первого операнда с соответствующим битом второго операнда по правилу:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |

**Логические операции:**

Операция **&&** (логическое **И**) (значение обоих операндов - истина):

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |

Операция **||** (логическое **ИЛИ**)(значение хотя бы одного операнда -истина):

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |

Операция **!** (**НЕ)** (истина, если была ложь и наоборот).

Операция последовательного вычисления (,) используется обычно для вычисления двух или более выражений там, где по синтаксису допустимо только одно выражение. Эта операция вычисляет два операнда последовательно слева направо.

**Пример:**

for(i=0, j=10; i<MAX && 2\*j<MAX; i++)

#include<stdio.h>

int main()

{

int iVar1=8, iVar2;

printf("Right %d\t%d\t%d\n",iVar1>>1,iVar1>>2,iVar1>>3); // сдвиг вправо

printf("Left %d\t%d\t%d\n",iVar1<<1,iVar1<<2,iVar1<<3); // сдвиг влево

iVar1=-8;

printf("Right %d\t%d\t%d\n",iVar1>>1,iVar1>>2,iVar1>>3); // сдвиг вправо

printf("Left %d\t%d\t%d\n",iVar1<<1,iVar1<<2,iVar1<<3); // сдвиг влево

printf("Input iVar1, iVar2\t");

scanf("%d%d",&iVar1,&iVar2);

printf("iVar1&&iVar2=%d\n",iVar1&&iVar2); // логическое И

printf("iVar1||iVar2=%d\n",iVar1||iVar2); // логическое ИЛИ

printf("!iVar1=%d\n",!iVar1); // отрицание НЕ

return 0;

}

**Поразрядные логические операции**

int iVar1=0x5f, iVar2=0xa9;

int iResult;

iResult=iVar1&iVar2; // поразрядное логическое И

printf("iResult=%xh\n",iResult); // 9h

iResult=iVar1|iVar2; // поразрядное логическое ИЛИ

printf("iResult=%xh\n",iResult); // ffh

iResult=iVar1^iVar2; // поразрядное искл. ИЛИ

printf("iResult=%xh\n",iResult); // f6h

int iVar3=0x25,iVar4=0xa4;

iVar3+=iVar4;

printf("iVar3=%oo\tiVar3=%dd\tiVar3=%Xh\t",iVar3,iVar3,iVar3);

// 311 201 c9

**Операции присваивания**

|  |  |
| --- | --- |
| Знак операции | Название операции |
| ++ | Увеличение (инкремент) |
| -- | Уменьшение (декремент) |
| = | Простое присваивание |
| += | Сложение с присваиванием |
| -= | Вычитание с присваиванием |
| \*= | Умножение с присваиванием |
| /= | Деление с присваиванием |
| %= | Целочисленное деление с присваиванием |
| <<= | Сдвиг влево с присваиванием |
| >>= | Сдвиг вправо с присваиванием |
| &= | Поразрядное логическое И с присваиванием |
| |= | Поразрядное логическое ИЛИ с присваиванием |
| ^= | Поразрядное исключающее ИЛИ с присваиванием |

**Примеры**:

a+=20;

d%=3.0;

x\*=3\*y+12;

**Терарная операция.**

Терарная операция включает в себя выражение с тремя операндами и имеет формат:

**операнд1?орперанд2:операнд3;**

Операнд1 – арифметическое или логическое выражение-условие.

Операнд2 ­– результат по выполнению условию.

Операнд3 – результат по не выполнению условию.

**Пример:**

int x, y=-5;

x=(y<=0)?-y:y;

printf(“x=%d\n”,x); // x=5

Если y<=0 => x=-y

y>0 =>x=y

**Приоритеты операций и порядок выполнения**

|  |  |  |
| --- | --- | --- |
| **Приоритет** | **Знак операции** | **Тип операции** |
| 1 | **() [] . -> sizeof()** | Круглые и квадратные скобки, точка, стрелка |
| 2 (\*) | **- ~ ! \* & ++ --** | Унарные |
| 3 | **\* / %** | Мультипликативные |
| 4 | **+ -** | Аддитивные |
| 5 | **<< >>** | Побитовый сдвиг |
| 6 | **< > <= >=** | Неравенство |
| 7 | **== !=** | Равенство |
| 8 | **&** | Поразрядное И |
| 9 | **^** | Поразрядное исключающее ИЛИ |
| 10 | **|** | Поразрядное ИЛИ |
| 11 | **&&** | Логическое И |
| 12 | **||** | Логическое ИЛИ |
| 13 | **?:** | Условное выражение |
| 14 (\*) | **= \*= /= %= += -= <<= >>=** | Простое и составное присваивание |
| 15 | **,** | Последовательное вычисление |

**Примеры:**

y=x/2+a\*5%10; // y=(x/2)+((a\*5)%10);

a=b+++c/5; // a=(b++)+(c/5);

**Преобразование типов:**

Рассмотрим три правила преобразования типов.

1. Если операция выполняется над данными двух различных типов, обе величины приводятся к “высшему” из двух типов.
2. Применение ключевого слова “unsigned” повышает ранг соответствующего типа.
3. В операторе присваивания конечный результат вычисления приводится к типу переменной, которой должно быть присвоено это значение.

Существует возможность точно указать тип данных, к которому необходимо привести некоторую величину. Этот способ называется ”приведением типов” и состоит в следующем: перед данной величиной в круглых скобках записывается имя требуемого типа.

**Пример:**

int I Var;

iVar=1.6+1.7; // 3 (автоматическое преобразование типов)

iVar=(int)1.6+(int)1.7; // 2

(float)9 // 9.0