поразрядные операторы (побитовые)

& **поразрядная коньюнкция (побитовое умножение) (and, И)**

| **поразрядная дизъюнкция (побитовое сложение) (or, ИЛИ)**

^ **поразрядное исключающее ИЛИ (xor)**

операторы сдвига

>> **сдвиг вправо**

<< **сдвиг влево**

Побитовые операторы

Побитовые операторы выполняются над целыми числами: над каждым битом операндов. Результатом является целое число.

Таблица значений результатов

Операнды		and &	or |	xor ^
Бит1	Бит2			
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Оператор & сравнивает каждый бит первого операнда с соответствующим битом второго операнда. Если оба бита =1, то соответствующий бит результата =1, иначе = 0. Оператор & часто используется для выделения некоторого множества битов.

```
234567 & 255
```

В этом примере выделяется последний байт числа.

Упр. Чему он равен?

С помощью операции & можно определить остаток от деления операнда типа **`unsigned int`** на 2, 4, 8, 16 и т.д.

Для этого достаточно применить операцию & к делимому с масками **`0x01, 0x03, 0x07, 0x0f, 0x1f`** и т.д.

Например:

`127&0x03` дает **`3 (остаток от деления 127 на 4).`**

`127=0x7f=01111111`

`01111111&111 \\ 00000011=3`

Оператор | сравнивает каждый бит 1-го операнда с соответствующим битом 2-го операнда; если хотя бы один из них или равен 1, то соответствующий бит результата =1, иначе = 0.

Операция | используется для включения битов:

`x | 7` устанавливает в 1 три последних бита переменной x (число 7 =00000111_2), остальные биты переменной не изменяются.

Оператор ^ сравнивает каждый бит первого операнда с соответствующим битом второго операнда; если значения битов одинаковое, то соответствующий бит результата устанавливается в 0, если разное - бит результата устанавливается в 1.

Оператор ^ часто используется для обнуления переменных:

```
n=n^n;
```

Значение переменной n будет равно 0.

Используя этот оператор можно обменять значения двух переменных, не используя третью:

```
x=x^y;
y=x^y;
x=x^y;
```

Оператор ~ меняет в битовом представлении операнда 0 на 1, а 1 - на 0.

Например:

```
 short int a=0x45ff, b=0x00ff;
~ a;     //  -0x3a00
//0100 0101 1111 1111
//1011 1010 0000 0000
//-0x3a00
```

Упр. Разобрать след. примеры:

```
~ b;     //  -0x7f00
a | b    //  0x45ff
a & b    //  0x00ff
a ^ b    //  0x4500
```

Операторы сдвига

Оператор << (сдвиг влево) выполняет побитовый сдвиг влево левого операнда на количество разрядов, соответствующее значению правого операнда. Сдвиг на 1 бит эквивалентен умножению на 2. Результатом является целое число.

Оператор >> (сдвиг вправо): выполняет побитовый сдвиг вправо левого операнда на количество разрядов, соответствующее значению правого операнда. Если число без знака, то левые биты = 0. Сдвиг на 1 бит вправо эквивалентен делению на 2. Результатом является целое число.

Сдвиг вправо может быть арифметическим (т. е. освобождающиеся слева разряды заполняются значениями знакового разряда) или логическим в зависимости от реализации, однако гарантируется, что при сдвиге вправо целых чисел без знака освобождающиеся слева разряды будут заполняться нулями.

Например:

```
unsigned short m=2000;
m = m>>4;  // 2000=0x7d0
//0111 1101 0000;
//0111 1101=0x7d=7*16+13=125
//2000:16=125
```

int m=2000;

m=m>>4; // 125 (2000: $2^{4)}$

m=m<<4; // 32000 (2000* $2^{4)}$

2000_{10} &4 =$01111101000 0_2$ &100_2=0

2000_{10} |4 =$01111101000 0_2$ |100_2=2004_{10}

2000_{10} ^4 =$01111101000 0_2$ ^100_2=2004_{10}

Оператор присваивания

Оператор присваивания (=) не обязан стоять в отельной строке и может входить в более крупные выражения.

В качестве результата оператор возвращает значение, присвоенное левому операнду. Например, следующее выражение вполне корректно:

```
value1 = 8 * (value2 = 5);
```

Составные операторы присваивания объединяют логические и арифметические операторы с оператором присваивания.

Побитовые присваивания

<<= X <<= y // x = x<<y

>>= X >>= y // x = x>>y

&= X &= y // x = x&y

^= X ^= y // x = x^y

|= X |= y // x = x|y