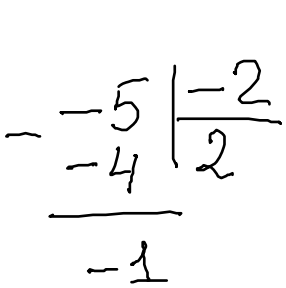
ДРУГИЕ ОПЕРАЦИИ (ДОПОЛНЕНИЕ)

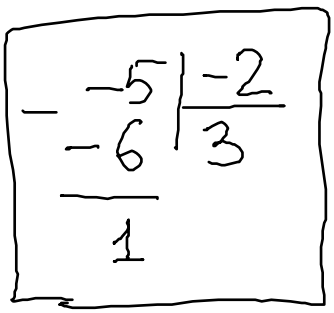
Операция нахождения остатка от деления %

Математический вариант операции взятия остатка:

```
A = B * [A/B] + A mod B  => A mod B = A - B * [A/B]

5%2     = 5 - 2 * [5/2]          = 5 - 2 * 2     = 1
5%(-2)  = 5 - (-2) * [5/(-2)]    = 5 + 2 * (-2)  = 1
(-5)%2  = -5 - 2 * [-5/2]        = -5 - 2 * (-2) = -1
-5%(-2) = -5 - (-2) * [-5/(-2)] = -5 + 2 * 2    = -1
```





Операции инкремента и декремента

Пример.

```
int a = 20, b = 20;
cout << "a = " << a << ": b = " << b << "\n";
cout << "a++ = " << a++ << " : ++b = " << ++b << "\n";
cout << "a = " << a << ": b = " << b << "\n";
```

Результат:

```
a =20: b = 20
a++ = 20: ++b = 21
a =21: b = 21
```

Пример.

```
x = 2 * x++ * (3 - ++x); //не поступайте так
```

Битовые операции

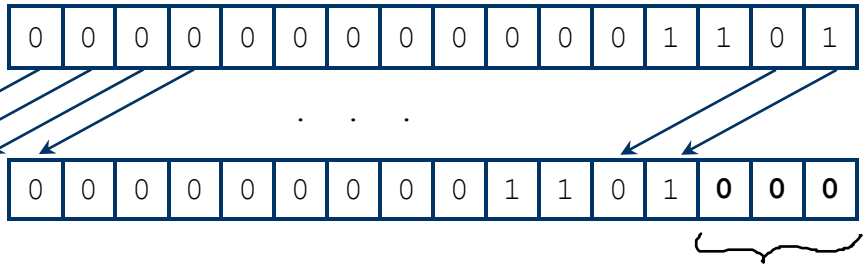
↓

для целочисленных значений

①*Сдвиг*

влево: *значение* << *величина*

```
short int a = 13;
```

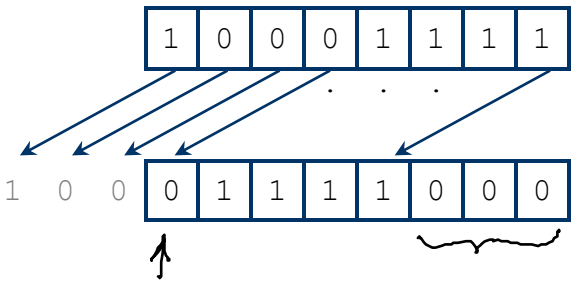



```
a = a << 3;
a <<= 3;
```

a << 1

a · 2

```
char a = -113;
```



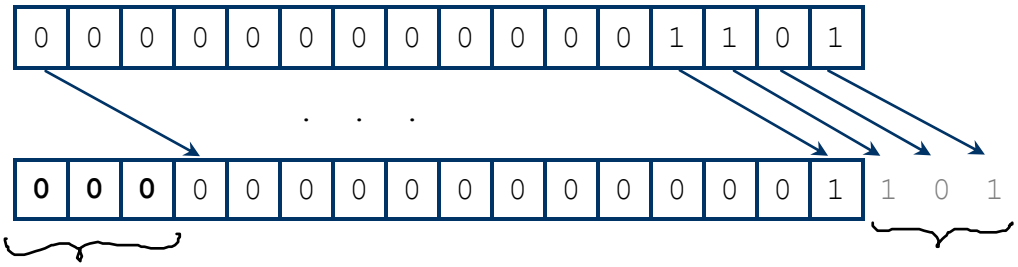

```
a = a << 3;
a <<= 3;
a *= 8;
```

a << n

$a \cdot 2^n$

вправо: *значение >> величина*

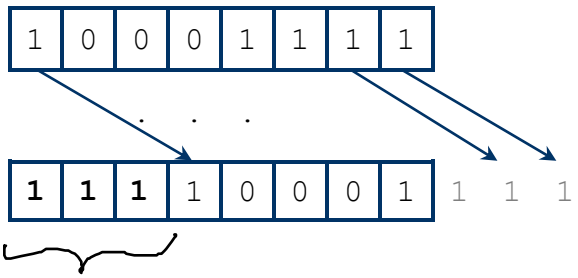
```
short int a = 13;
```




```
a = a >> 3;
a >>= 3;
a /= 8;
```

2^3

```
char a = -113;
```

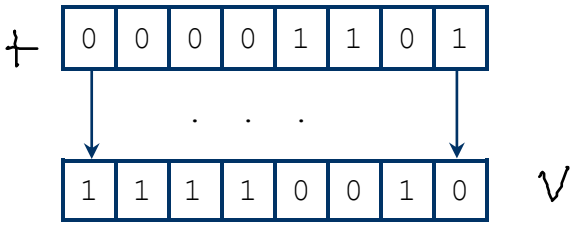



```
a = a >> 3;
a >>= 3;
~~a /= 8;~~
```

②*Логические битовые операции*

битовое отрицание: *~ значение*

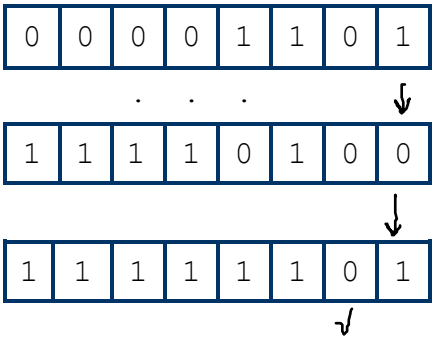
```
unsigned char a = 13;
```




```
a = ~a;
//  242 или
//  int(~a) = (-14)доп
```

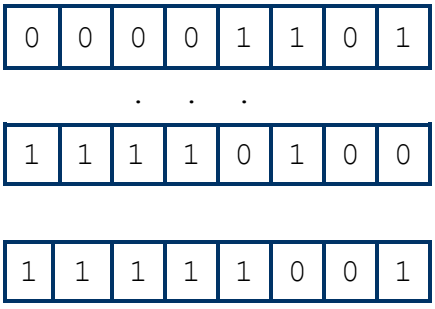
битовая операция "ИЛИ" *значение | значение*

```
char a = 13,
b = -12;
a = a | b;
a |= b;
```



битовая операция исключающего "ИЛИ" *значение ^ значение*

```
char a = 13,
b = -12;
a = a ^ b;
a ^= b;
```



<u>битовая операция "И"</u> (&): *значение & значение*

```
char a = 13,
```

0	0	0	0	1	1	0	1

&

```
b = -12;
```

1	1	1	1	0	1	0	0

```
a = a & b;
```

0	0	0	0	0	1	0	0

```
a &= b;
```

XOR ^

1 1 1 1 1 0 0 1 a^b

a^=b

Альтернативные представления битовых операций

Стандартное представление	**Альтернативное представление**	**Стандартное представление**	**Альтернативное представление**
&	bitand	!	not
&=	and_eq	!=	not_eq
|	bitor	&&	and
|=	or_eq	||	or
~	compl		
^	xor		
^=	xor_eq		

```
b = compl a bitand b; // то же, что и b = ~a & b;
c = a xor b;          // то же, что и c = a ^ c;
```

$\overline{\overline{x}} = x$

!!x ≠ x

if (!!13) == ?

true, false

x = 13 0 → false

13 → true

!13 → f

!!13 → t → 1

Применения битовых операций

- Вычисление модуля числа без использования условного оператора
- Нахождение минимума и максимума из двух чисел без использования условного оператора
- Нахождение младшего/старшего единичного бита
- Циклический сдвиг
- Подсчет количества единичных битов

Пример. Вывести двоичное представление числа.

```
unsigned short A=5,
     bit, pos, vesBit;
vesBit = USHRT_MAX+1 >> 1; // его вес
while (vesBit > 0) {
     bit = A & vesBit;     // получаем значение бита B в позиции,
                           // соответствующего весу vesBit
     if (bit == 0)  cout << 0;
     else           cout << 1;
     vesBit /= 2;
}
```

Шаг 1.

USHRT_MAX		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	+															1
USHRT_MAX+1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
USHRT_MAX+1 >> 1		**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	&															
A		**0**	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Шаг 14.

USHRT_MAX+1 >> 14		0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
	&																
A		0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	1
		0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0

Операция запятая

составной оператор {} → операция запятая (,)

Пример. Оператор запятая – составной оператор

```
for (int i, ; i<6, j<6; ++j , --i)
    {...}
```

Пример. Оператор запятая – разделитель

```
for (int i, j;  i<6, j<6; ++j , --i)
    {...}
```

Особенности операции запятой:

– Первое выражение вычисляется перед вторым

```
i = 20, j = 2 * i // i присваивается 20, затем j получает значение 40
```

– Значение выражения – значение второй части

– Приоритет – наименьший из всех операций

```
cats = 17,240;   →   (cats = 17), 240;   →   cats = 17
```

НО:

```
cats = (17,240); →   cats = 240
```

Операции отношений

Операция	Описание
<	Меньше чем
<=	Меньше или равно
==	Равно
>	Больше чем
>=	Больше или равно
! =	Не равно

```
x + 3 > y - 2
```

```
(x + 3) > (y - 2)
```

```
x + (3 > y) - 2
```

true 1 false 0

```
-1 < x < 10  // true
-10 < x < -9 // false
```

-1 < 100 < 10

true

1 < 10 → true

-1 < -100 < 10

false

0 < 10 → true

Присваивание, сравнение и вероятные ошибки

```
x == 4 // сравнение
x =4   // присваивание
```

Пример. Риск допустить ошибку.

```
  // равенство:
  int scores[10] = { 20, 20, 20, 20, 20, 19, 20, 18, 20, 20};
  int i;
  for (i = 0; scores [i] == 20; i++)
      cout << " score " << i << ": 20\n";

  // присваивание:
  for (i = 0; scores [i] = 20; i ++ )
      cout << " score " << i << ": 20\n";
```

score 0: 20
1:
2:
3:
4:

Библиотека <cctype>

Функция	Проверка, является ли символ
`isalnum`	буквенно-цифровым: 0..9, A..Z, a..z
`isalpha`	буквенным: A..Z или a..z
`islower`	в нижнем регистре: a..z
`isupper`	прописной буквой: A..Z
`isdigit`	цифрой
`isxdigit`	шестнадцатеричным: 0123456789abcdefABCDEF
`iscntrl`	управляющим символом: символами с кодами 0x00-0x1F и 0x7F
`isgraph`	графическим символом: цифра (0123456789), прописная буква (ABCDEFGHIJKLMNOPQRSTUVWXYZ), строчными буквами (abcdefghijklmnopqrstuvwxyz), или знак пунктуации (!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~)
`isspace`	пробельным символом: пробел (0x20, ' '), смена страницы (0x0c, '\f'), перевод строки (0x0a, '\n'), возврат каретки (0x0d, '\r'), горизонтальный таб (0x09, '\t'), вертикальный таб (0x0b, '\v')
`isblank`	«пустым» символом: пробел или табуляция (C++11)
`isprint`	является печатным символом: 0..9, буквой в верхнем регистре A..Z или в нижнем регистре a..z, знак препинания (!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~), пробел
`ispunct`	символом пунктуации !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
`tolower`	преобразует символ в нижний регистр
`toupper`	преобразует символ в верхний регистр