Битовые операции. Битовые поля в структурах.



Битовые операции позволяют проверять и модифицировать отдельные биты в *целочисленных* и *символьных* данных.

Обозначение операций:

&	Побитовое «И»
	Побитовое «ИЛИ»
^	Исключающее «ИЛИ» (XOR)
<<	Сдвиг влево
>>	Сдвиг вправо
~	Дополнение до 1



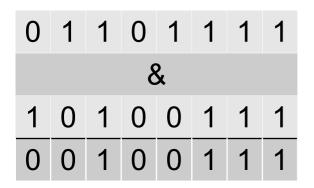
Определения функций:

Левый операнд (X)	Правый операнд (Y)	X & Y	X Y	X^Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

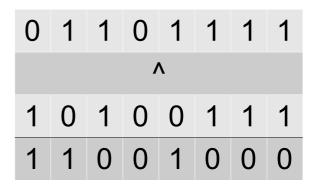


Примеры:

$$01101111 \rightarrow 111_{10}$$
$$10100111 \rightarrow 167_{10}$$



0	1	1	0	1	1	1	1
1	0	1	0	0	1	1	1
1	1	1	0	1	1	1	1



$$00100111 \rightarrow 39_{10}$$

$$111011111 \rightarrow 239_{10}$$

$$11001000 \rightarrow 200_{10}$$

Побитовое «И» (&) можно использовать для установки отдельных битов в «0», либо для проверки на «1».

Побитовая операция «ИЛИ» («|») полезна для установки в «1».



```
# include <stdio.h>
int main()
  char c;
   char mask;
   char res;
  c=' x7A';
  mask='\x77';
   res = c & mask;
   printf("%c %c %c\n", c, mask, res);
   return 0;
```

Результат: z w r

Символ «z» имеет код 7А (01111010), символ «w» имеет код 77 (01110111). В результате побитового «И» получим 01110010, т. е. 72, что соответствует символу «r».

См. пример lect-17-01.c



Побитовая операция «исключающее ИЛИ» (XOR, «^») полезна для определения состояния битов. Если биты операндов совпадают, то бит результата — 0, иначе — 1. (можно использовать для переключения значения младшего и старшего бита в байте).

```
Пример:
# include <stdio.h>
int main()
  char a='\x2C';
   char b='\x71';
   char c;
   c=a^b;
   printf("a=%c b=%c c=%c\n",a,b,c);
   return 0;
```

Rезультат: a=, b=q c=] (пример lect-17-02.c).

Побитовый сдвиг влево.

```
b1 << b2 - сдвиг влево.
b1 — сдвигаемое значение, b2 — количество позиций сдвига.
Освобождаемые справа позиции заполняются нулями.
Пример (lect-17-03.c):
# include <stdio.h>
int main()
   short x=1;
   printf("%d %d %d %d\n",x,x<<1,x<<2,x<<3);</pre>
   return 0;
                          0000001 x
                          00000010 x << 1
Результат: 1 2 4 8
                          00000100 x << 2
                          00001000 x << 3
```

Задача: написать функцию для переключения значения бита в n-й позиции байта.

```
Решение (пример lect-17-04.c):
# include <stdio.h>
                                       Исходный символ: у (код 121<sub>10</sub>)
char switchbit(char c, int n)
                                       При n=2 получаем
   return c^{(1 << n)};
                                       «4 New char: }» (позиции
                                       считаются справа с 0) — код 125<sub>10</sub>
int main()
                                       При n=3 получаем
   char nc;
   char c='\x79';
                                       «8 New char: q» — код 113<sub>10</sub>.
   short n;
   printf("Enter position from right: ");
   scanf("%d",&n);
   nc=switchbit(c,n);
   printf("%d New char: %c\n",1<<n,nc);</pre>
   return 0;
```

Побитовый сдвиг вправо.

b1 >> b2 - сдвиг вправо.

b1 — сдвигаемое значение, b2 — количество позиций сдвига. Освобождаемые слева позиции заполняются нулями для беззнаковых чисел (unsigned).

Для чисел со знаком в отрицательных числах позиции заполняются единицами (1), в положительных — нулями (0) (происходит «расширение» символа знака).



```
Побитовый сдвиг вправо.
Пример (lect-17-05.c):
# include <stdio.h>
int main()
   short x=0x8000;
   int y=8;
   unsigned int z=0x8000;
   printf("%x %x %x\n",x,x>>1,x>>2);
   printf("%x %x %x\n",y,y>>1,y>>2);
   printf("%x %x %x\n",z,z>>1,z>>2);
   return 0;
                         Результаты:
                         ffff8000 ffffc000 ffffe000
                        8 4 2
                        8000 4000 2000
```



Побитовый сдвиг вправо.

Объяснение:

х: 1000 0000 0000 0000 – для short старший бит является знаковым, слева дописываются единицы (1) до максимально возможной разрядности, наличие или отсутствие ffff зависит от компилятора.

x>>1: 1100 0000 0000 0000 \rightarrow c000

x>>2: 1110 0000 0000 0000 \rightarrow e000

y: 0000 0000 0000 1000

y>>1: 0000 0000 0000 0100

y>>2: 0000 0000 0000 0010

z: 1000 0000 0000 0000 – для unsigned int старший бит не считается знаковым, поэтому 1 просто перемещается вправо

z>>1: 0100 0000 0000 0000

z>>2: 0010 0000 0000 0000



- Побитовый сдвиг влево равносилен умножению на 2.
- Побитовый сдвиг вправо равносилен делению на 2.

Циклический сдвиг на k позиций влево или вправо (для n-разрядного двоичного числа):

- Если к ← n, то результат такой же как у обычного сдвига.
- Если k > n, то результат равносилен сдвигу на m позиций, где
 m = k%n



```
Унарная операция дополнения до 1 («~»).
(пример lect-17-06.c)
# include <stdio.h>
int main()
   unsigned short x=0x800;
   unsigned short y;
                                       Результат: 800 f7ff
   y = \sim x;
   printf("%x %x\n",x,y);
                                       Объяснение:
   return 0;
                                       x: 0000 1000 0000 0000
                                       y: 1111 0111 1111 1111
```



Применение битовых операций

Задача: определить количество единиц в двоичном представлении символа.

```
Peшeниe: функция numberOfOnes()
int numberOfOnes(char c)
   int i, count;
   char rest;
   count=0;
   for(i=0;i<8;i++)
        rest=(c>>i)&(0x1);
      if(rest==1) ++count;
   return count;
```

Важно! Начинать с «0» и префиксная форма увеличения count (или count=count+1).

Применение битовых операций

Задача: получить двоичное представление кода символа.

Решение: функция codeOfChar()

```
void codeOfChar(char c)
{
   int i;
   for(i=7;i>=0;i--) printf("%d",(c>>i)&(0x1));
}
```

Важно! Начинать с «7» и считать до «0» включительно!

Д/3: Написать функцию циклического сдвига вправо на заданное количество позиций для числа типа int.



- 1. Если объявлена struct s {int a, short b, char c, short d};, то sizeof(struct s) >= суммы размеров типов, составляющих структуру.
- 2. Если все возможные значения типов чисел в структурах не используются → память используется неэффективно.

Для типов, соответствующих целым числам (int, unsigned int, short, unsigned short, char, unsignet char) один из способов экономии памяти заключается в использовании битовых полей для представления совокупности данных целого типа.



Битовые поля могут описываться только в качестве элементов структурного шаблона аналогично объявлению целой знаковой или беззнаковой переменной, после имени которой через двоеточие записывается целая константа, определяющая размер битового поля.

В описании допускаются неименованные битовые поля (для них имя опускается, а указывается только двоеточие и размер), которые используются как заполнители пространства битового поля.

Битовые поля размещаются в направлении от младших к старшим битам в слове.



Форма описания:

Тип	Имя поля:	Ширина поля;
Целочисл. (без)знаковый	(Необязательно)	

```
struct BP
                      В зависимости от выбранного типа
   unsigned a1:1;
                             полей структуры
                      ДЛЯ
                                                 размер
   unsigned a2:1;
                      структуры будет разным.
   unsigned a3:1;
   unsigned a4:1;
                      Можно
                                управлять
                                             отдельными
   unsigned a5:1;
                      битами в двоичном представлении
   unsigned a6:1;
   unsigned a7:1;
                      данных.
   unsigned a8:1;
};
                      См. примеры lect-17-09.c
```



```
Пусть определено struct B {
   int i:2;
   unsigned j:2;
   int :2;
   int k:2;
   int d:8;
} m_s;
```

```
И установлены значения m_s.i=1;
```

```
m_s.i=1;
m_s.j=3;
m_s.k=-1;
m_s.d=0;
```

В памяти будет:

№ бита	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Значение	0	0	0	0	0	0	0	0	1	1			1	1	0	1
	d						k				j		i			

