

Побитовые (поразрядные) операторы

Поразрядные операции

- В отличие от многих других языков программирования в С определен полный набор *поразрядных операций*. Это обусловлено тем, что С был задуман как язык, призванный во многих приложениях заменить ассемблер, который способен оперировать битами данных
- *Поразрядные операции* — это тестирование (проверка), сдвиг или присвоение значений отдельным битам данных
- *Поразрядные операции* осуществляются над ячейками памяти, содержащими данные типа char или int. Данные типа float, double, long double, void или другие более сложные не могут участвовать в поразрядных операциях

Поразрядные операторы

<i>Оператор</i>	<i>Операция</i>
&	И
	ИЛИ
^	исключающее ИЛИ
~	НЕ (отрицание, дополнение к 1)
>>	Сдвиг вправо
<<	Сдвиг влево

Таблицы истинности

- Поразрядные операции И, ИЛИ, НЕ описываются теми же таблицами истинности, что и логические операции
- Поразрядные операции выполняются над отдельными разрядами (битами) операндов
- Операция "исключающее ИЛИ" имеет следующую таблицу истинности:

p	q	$p \oplus q$
0	0	0
1	0	1
1	1	0
0	1	1

Применение поразрядных операторов

- при программировании драйверов устройств, таких как модемы
- при программировании процедур, выполняющих операции над файлами
- при разработке стандартных программ обслуживания принтера

Побитовый оператор И: &

- Операция & может быть использована для *очистки (сбрасывания значения) бита*
- Любой бит одного из операндов, равный 0, обнуляет значение соответствующего бита в результате
- Например, следующая функция читает символ из порта модема и обнуляет бит контроля четности:

```
char get_char_from_modem(void)
{
    char ch;
    ch = read_modem(); /* чтение символа из порта модема */
    return(ch & 127);
}
```

Бит контроля четности, находящийся в 8-м разряде байта, обнуляется с помощью операции И. При этом в качестве второго операнда выбирается число, имеющее 1 в разрядах от 1 до 7, и 0 в 8-м разряде. Именно таким числом и является 127, поскольку все биты двоичного представления числа 127, кроме старшего, равны 1. Выражение `ch & 127` означает попарное применение операции И ко всем битам, составляющим значение переменной `ch`, и битам числа 127. В результате все биты, кроме старшего, остаются без изменения, а старший обнуляется:

Бит контроля четности - 8 бит

1100 0001 переменная `ch` содержит символ 'A' с битом четности

0111 1111 двоичное представление числа 127

& ----- поразрядная операция И

0100 0001 символ 'A' с обнуленным битом контроля четности

Поразрядная операция ИЛИ: |

- Поразрядная операция ИЛИ применяется для установки необходимых битов в 1
- В следующем примере выполняется операция $128 | 3$:

1000 0000 двоичное представление числа
128

0000 0011 двоичное представление числа 3

| ----- поразрядная операция ИЛИ

1000 0011 результат

Операция исключающего ИЛИ (XOR): \wedge

- Операция исключающего ИЛИ устанавливает бит результата в 1, если соответствующие биты операндов различны
- В следующем примере выполняется операция $127 \wedge 120$:
0000 0011 двоичное представление числа 127
0111 1000 двоичное представление числа 120
 \wedge ----- поразрядная операция XOR
0000 0111 результат

Необходимо помнить, что результат логической операции всегда равен 0 или 1. В то же время результатом поразрядной операции может быть любое значение, которое, как видно из предыдущих примеров, не обязательно равно 0 или 1.

- Результат логической операции всегда равен 0 или 1
- Результат поразрядной операции может быть любое равно любому целому числу, не обязательно равно 0 или 1.

Поразрядные операторы сдвига

>> и <<

- Поразрядные операторы сдвига >> и << смещают все биты переменной вправо или влево на указанное количество разрядов
- Общая форма оператора сдвига вправо:
переменная >> количество_разрядов
- Общая форма оператора сдвига влево:
переменная << количество_разрядов
- Как только сдвигаемые биты достигают края, с противоположного конца появляются нули
- Если число типа signed int отрицательно, то при сдвиге вправо левый конец заполняется единицами, так что знак числа сохраняется
- Если биты исчезают на одном краю числа, они не появятся на другом
- Поразрядные операции сдвига очень полезны при декодировании информации, поступающей с внешнего устройства
- Побитовые операторы сдвига можно использовать для быстрого умножения или деления целых чисел: сдвиг на один бит вправо делит число на 2, а на один бит влево — умножает на 2

Умножение и деление

операторами сдвига

<i>unsigned char x</i>	<i>x после операции</i>	<i>значение x</i>
$x = 7$	0000 0111	7
$x = x \ll 1$	0000 1110	14
$x = x \ll 3$	0111 0000	112
$x = x \ll 2$	1100 0000	192
$x = x \gg 1$	0110 0000	96
$x = x \gg 2$	0001 1000	24

Каждый сдвиг влево умножает на 2. Потеря информации произошла после операции $x \ll 2$ в результате сдвига за левую границу.

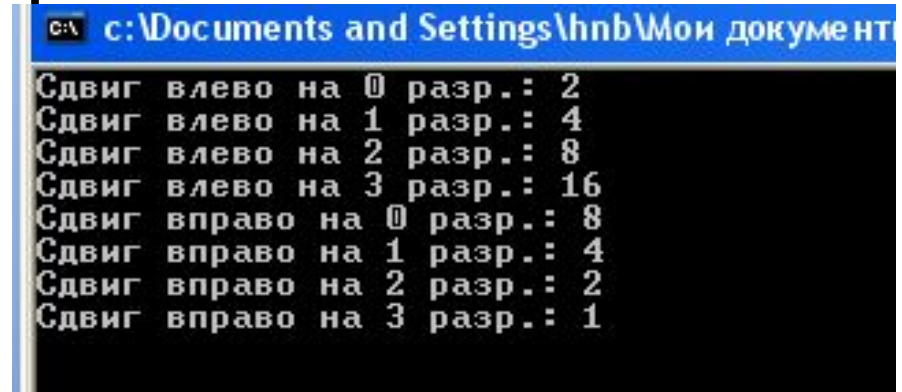
Каждый сдвиг вправо делит на 2. Сдвиг вправо потерянную информацию не восстановил.

Пример применения операторов

сдвига.

```
#include <stdio.h>
#include <conio.h>
#include "locale.h"
int main(void)
{
    setlocale(LC_ALL, "rus");
    unsigned int i;
    int j;
    i = 1;
    /* сдвиг влево */
    for(j=0; j<4; j++) {
        i = i << 1; /* сдвиг i влево на 1 разряд, что
                    равносильно умножению на 2 */
        printf("Сдвиг влево на %d разр.: %d\n", j, i);
    }

    /* сдвиг вправо */
    for(j=0; j<4; j++) {
        i = i >> 1; /* сдвиг i вправо на 1 разряд, что
                    равносильно делению на 2 */
        printf("Сдвиг вправо на %d разр.: %d\n", j, i);
    }
    getch();
    return 0;
}
```



```
c:\Documents and Settings\hnb\Мои документы
Сдвиг влево на 0 разр. : 2
Сдвиг влево на 1 разр. : 4
Сдвиг влево на 2 разр. : 8
Сдвиг влево на 3 разр. : 16
Сдвиг вправо на 0 разр. : 8
Сдвиг вправо на 1 разр. : 4
Сдвиг вправо на 2 разр. : 2
Сдвиг вправо на 3 разр. : 1
```

Поразрядная операция отрицания

~

- Поразрядная операция отрицания (дополнения до единицы) ~ инвертирует состояние каждого бита операнда.
- То есть, 0 преобразует в 1, а 1 — в 0.
- Поразрядные операции часто используются в шифровальных программах. Проведя с дисковым файлом некоторые поразрядные операции, его можно сделать нечитаемым. Простейший способ сделать это — применить операцию отрицания к каждому биту:
Исходный байт 0010100
После 1-го отрицания 1101011
После 2-го отрицания 0010100
- Обратите внимание, при последовательном применении 2-х отрицаний результатом всегда будет исходное число. Таким образом, 1-е отрицание кодирует состояние байта, а 2-е — декодирует

Пример использования операции

ОТРИЦАНИЯ

- В следующем примере оператор отрицания используется в функции шифрования символа:

```
#include <stdio.h>
#include <conio.h>
#include "locale.h"
int main(void)
{
    setlocale(LC_ALL, "rus");
    char ch;
    scanf("«%c", &ch);
    printf("«%c", ~ch); /* операция отрицания */
    getch();
    return 0;
}
```

