CSE1142 – Bitwise Operators

Sanem Arslan Yılmaz

Some of the slides are from:
CMPE150 – Boğaziçi University
Deitel & Associates

Agenda

- Bitwise Operators
  - AND
  - OR
  - Exclusive OR
  - Shift Left
  - Shift Right
- Bitwise Assignment Operators
- Examples

Motivation

- All data represented internally as sequences of bits
  - Each bit can be either 0 or 1
  - Sequence of 8 bits forms a **byte**
    - typical storage unit for a variable of type char.

- The following operations can be performed on bits
  - AND (&)
  - OR (|)
  - Exclusive OR (^)
  - Left shift (<<)
  - Right shift (>>)
  - One's complement (~)

Bitwise Operators

Operator		Description
&	bitwise AND	Compares its two operands bit by bit. The bits in the result are set to 1 if the corresponding bits in the two operands are *both* 1.
|	bitwise inclusive OR	Compares its two operands bit by bit. The bits in the result are set to 1 if *at least one* of the corresponding bits in the two operands is 1.
^	bitwise exclusive OR (also known as bitwise XOR)	Compares its two operands bit by bit. The bits in the result are set to 1 if the corresponding bits in the two operands are different.
<<	left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits.
>>	right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent when the left operand is negative.
~	complement	All 0 bits are set to 1 and all 1 bits are set to 0.

Fig. 10.6 | Bitwise operators.

AND Operator

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 10.8 | Results of combining two bits with the bitwise AND operator &.

OR Operator

Bit 1	Bit 2	Bit 1 | Bit 2
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 10.11 | Results of combining two bits with the bitwise inclusive OR operator |.

Exclusive OR Operator

Bit 1	Bit 2	Bit 1 ^ Bit 2
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 10.12 | Results of combining two bits with the bitwise exclusive OR operator ^.

Bitwise Assignment Operators

Bitwise assignment operators	
&=	Bitwise AND assignment operator.
|=	Bitwise inclusive OR assignment operator.
^=	Bitwise exclusive OR assignment operator.
<<=	Left-shift assignment operator.
>>=	Right-shift assignment operator.

Fig. 10.14 | The bitwise assignment operators.

Example 1 – Display Bits

```
void displayBits(unsigned int value)
{
   unsigned int c, displayMask = 1 << 31;
   char x;
   printf("%u = ", value);

   // loop through bits
   for (c = 1; c <= 32; ++c) {
      x = value & displayMask ? '1':'0';
      printf("%c",x);

      value <<= 1; // shift value left by 1
      if (c % 8 == 0) { // output space after 8 bits
         printf(" ");
      }
   }
   printf("\n");
}
```

`1 << 31 (10000000 00000000 00000000 00000000)`

Output is:

```
Enter a nonnegative int: 127
127 = 00000000 00000000 00000000 01111111

------------------------------
```

Example 2 – AND Operator

```
unsigned int number1 = 65535;
unsigned int mask = 1;
int number3;
puts("The result of combining the following");
displayBits(number1);
displayBits(mask);
puts("using the bitwise AND operator & is");
displayBits(number1 & mask);
```

```
number1=              00000000 00000000 11111111 11111111
mask=                 00000000 00000000 00000000 00000001
number1 & mask =      00000000 00000000 00000000 00000001
```

Example 3 – OR Operator

```
number1 = 15;
unsigned int setBits = 241;
puts("\nThe result of combining the following");
displayBits(number1);
displayBits(setBits);
puts("using the bitwise inclusive OR operator | is");
displayBits(number1 | setBits);

number1=              00000000 00000000 00000000 00001111
setBits=              00000000 00000000 00000000 11110001
number1 | setBits =   00000000 00000000 00000000 11111111
```

Example 4 – Exclusive OR Operator

```
number1 = 139;
unsigned int number2 = 199;
puts("\nThe result of combining the following");
displayBits(number1);
displayBits(number2);
puts("using the bitwise exclusive OR operator ^ is");
displayBits(number1 ^ number2);
```

```
number1=                 00000000 00000000 00000000 10001011
number2=                 00000000 00000000 00000000 11000111
number1 ^ number2 =      00000000 00000000 00000000 01001100
```

Example 5 – Complement Operator

```
number1 = 21845;
puts("\nThe one's complement of");
displayBits(number1);
puts("is");
displayBits(~number1);
```

```
number1=              00000000 00000000 01010101 01010101
~number1 =            11111111 11111111 10101010 10101010
~number1 = 4294945450
```

Question:

How to take two's complement of a number?

Example 6 – Two's Complement

```
unsigned int x;
 int y;
number1 = 21845;
puts("\nThe two's complement of");
displayBits(number1);
puts("is");
x=~number1+1;
displayBits(x);
y=x;

number1=              00000000 00000000 01010101 01010101
x =                   11111111 11111111 10101010 10101011
 x = 4294945451
y = -21845
```

Example 7 – Left Shift

```
unsigned int number1 = 960;
puts("\nThe result of left shifting");
displayBits(number1);
puts("8 bit positions using the left shift operator << is");
displayBits(number1 << 8);
```

```
number1=              00000000 00000000 00000011 11000000
number1 << 8 =        00000000 00000011 11000000 00000000
```

Example 8 – Right Shift

```
unsigned int number1 = 960;
puts("\nThe result of left shifting");
displayBits(number1);
puts("8 bit positions using the right shift operator >> is");
displayBits(number1 >> 8);

number1=              00000000 00000000 00000011 11000000
number1 >> 8 =        00000000 00000000 00000000 00000011
```