East West University Bangladesh
Computing Science and Engineering Department

CSE-105: Bitwise Operators in C

Objectives:

• To understand bit-wise operations, and to use C bit-wise operators to manipulate binary data.

In this lab, you will practice using C language constructs and operators to do bitwise operations. The most common language constructs used to implement bitwise operations are:

Operator (In context)	Usage	Description
&	a & b	bitwise AND two values
|	a | b	bitwise OR two values
~	~a	bitwise invert/compliment a value
^	a ^ b	bitwise XOR two values
<<	b << n	shift the bits in b to the left by n bits (the upper n bits of b are lost).
>>	b >> n	shift the bits in b to the right by n bits (the lower n bits of b are lost). The shift is an arithmetic for signed integers, but it is a logical shift right for unsigned integers.

Task-1:

a) Manually, do the bitwise and, or, xor, complement, left shift and right shift of two given binary numbers **a= 00000101** and **b=00001001** and note the results of each operation.

b) Write the following C program as test1.c. Compile and run test1.c. Trace the results of the program and correct (if necessary).

```
/* C Program to demonstrate use of bitwise operators*/
#include<stdio.h>
int main()
{
    unsigned char a = 5, b = 9; // a = 5(00000101), b = 9(00001001)
    printf("a = %d, b = %d\n", a, b);
    printf("a&b = %d\n", a&b);
    printf("a|b = %d\n", a|b);
    printf("a^b = %d\n", a^b);
    printf("~a = %d\n", a = ~a);
    printf("b<<1 = %d\n", b<<1);
    printf("b>>1 = %d\n", b>>1);
    return 0;
}
```

Interesting facts about bitwise operators

The bitwise operators should not be used in-place of logical operators.

The result of logical operators (&&, ||, !) is either 0 or 1, but bitwise operators return an integer value. Also, the logical operators consider any non-zero operand as 1. For example: consider the following program, the results of & and && are different for same operands.

```
int main()
{
   int x = 2, y = 5;
   (x & y)? printf("True ") : printf("False ");
   (x && y)? printf("True ") : printf("False ");
   return 0;
}
```

Task-2:

a) Write the above C program as test2.c. Compile and run test2.c. Trace the results of the program. Write the differences of two operators & and &&.
b) Change the code and replace the & operator to | and && to ||. Trace the results of the program. Write the differences of two operators | and ||.

The left-shift and right-shift operators are equivalent to multiplication and division by 2 respectively.

```
int main()
{
   int x = 5;
   printf ("x << 1 = %d\n", x << 1);
   printf ("x >> 1 = %d\n", x >> 1);
   return 0;
}
```

Task-3:

a) Write the above C program as test3.c. Compile and run test3.c. Trace the results of the program. Write the differences of two operators << and >> in your report.
b) Replace all 1s to 2s and then all 2s to 3s and change the code. Recompile and run the code. Trace the results of the program after every change. Write your understanding about the results in your report.
c) Replace all 3s to -1s and check the result. Write your understanding about the results in your report.
d) Replace all -1s to 1 and all x to digit -5 and check the result. Write your understanding about the results in your report.
e) If the number is shifted more than the size of integer, the behavior is undefined. Change inside the code and check 1<<33. In report, explain your understanding.

The & operator can be used to quickly check if a number is odd or even

The value of expression (x & 1) would be non-zero only if x is odd, otherwise the value would be zero.

```
int main()
{
   int x = 19;
   (x & 1)? printf("Odd"): printf("Even");
   return 0;
}
```

Task-4:

a) Write the above C program as test4.c. Compile and run test4.c. Write the result of the program in your report.
b) Replace the line **int x = 19;** to **int x; scanf("%d",&x);** in your code and recompile and run the code to trace the odd/even numbers from the following inputs.
   3, 17, 29, 305, 200, 502, 56

The ~ operator should be used carefully

```
// Note that the output of following program is compiler dependent
int main()
{
   unsigned int x = 1;
   printf("Signed Result %d \n", ~x);
   printf("Unsigned Result %ud \n", ~x);
   return 0;
}
```

Task-5:

a) Write the above C program as test5.c. Compile and run test5.c. Write the result of the program in your report.
b) Convert the code so that it takes an input integer x using scanf() and outputs the 2's complement value of x.
c) Check the 2's complement value of **0**, and **MinINT**. Why are they acting different than others? Explain.

Masking

Bitwise operations are particularly useful for **masking**. In this case each bit in a byte represents a value which may be either on or off, i.e. true or false. In this case we wish to be able to access the bits individually, to turn each bit on or off as desired.

This is particularly common when accessing hardware devices at a low level.

- We can turn a bit on by a bitwise OR (|) with 1 in the relevant position.
- We can test whether a bit is set by a bitwise AND (&) with 1 in the relevant position.
- We can turn a bit off by a bitwise AND (&) with 0 in the relevant position.
- We can toggle a bit by a bitwise XOR (∧) with 1 in the relevant position.

Assignments

1. Write a program that takes an input from user and do the following task.
   a) Check the third bit is on or off?
   b) If the third is off, then turns on the third bit.
   c) If the third bit is on, then turn off the third bit.
   d) Toggle the third bit using XOR operator.