|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | **Programming Fundamentals** | |  | **(CL214)** | |  | **LABORATORY MANUAL** | |  | **Spring 2021** | |  | **C:\Users\Aamer\Desktop\nu-new.png**  **LAB 02** | |  | **Bitwise & Logical Operators** | |  | **Engr. Ibrar Khan**  **Engr. Sana Saleh** |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | \_\_\_\_\_\_\_\_\_\_ | | | \_\_\_ | | STUDENT NAME | | ROLL NO | | | SEC | |  | | | | | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | | LAB ENGINEER SIGNATURE & DATE | | | | | | | **MARKS AWARDED: /10** | | | | | | |  | | | | | | | **NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), ISLAMABAD** | | | | | | |  | | | | | | | Last Edited by: | Engr. Sana Saleh | | Version: | 2.01 | | | Prepared by: | Engr. Sana Saleh | | Date: | 03 Sep, 2017 | | | Verified by: | Engr. Shahid Qureshi | | Date: | 03 Feb, 2019 | | | | | |
| **LAB02** | **Bitwise & Logical Operators** |

**Lab Objectives:**

1. Understand how to use bitwise operators
2. Understand differences between logical and bitwise AND, OR and NOT.

**Software Required:**

* Dev C++

**Introduction:**

1. **Bitwise Operations**

C++ has six operators intended to directly manipulate bits within integer variables:

|  |  |
| --- | --- |
| Operator | Operation |
| & | bitwise AND |
| | | bitwise OR |
| ^ | bitwise XOR |
| << | left shift |
| >> | right shift |
| ~ | 1's complement (invert all bits) |

Table 1. Bitwise Operators

## 1.1 Bitwise AND

## A bitwise AND takes two binary-represented numbers of equal binary length and performs the logical AND operation bit by bit. For example:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 25 |
| & | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 216 |
| = | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 24 |

## 1.2 Bitwise OR

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C++ Programming, bitwise OR operator is denoted by |.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 25 |
| | | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 216 |
| = | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 217 |

## Bitwise XOR

## The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 25 |
| | | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 216 |
| = | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 193 |

## Bitwise Complement

## Bitwise compliment operator is a unary operator (works on one operand only). It changes the 1 to 0 and 0 to 1. It is denoted by ~.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ~ | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 216 |
| = | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 39 |

## For any integer n, bitwise complement of n will be - (n+1). To understand this, you should have the knowledge of 2's complement.

## Bit Shifting

Another bitwise operation that is very useful is bit shifting. You can shift to the left or to the right. There are two types of shifts:

* + Logical shifts
  + Arithmetic shifts

In a right arithmetic shift, the most-significant bit is preserved to maintain the sign of the number. In logical shifts of either direction, no bits are preserved; instead, the bits shifted out are replaced with zeros. A left arithmetic shift is the same as a left logical shift. In C++, all shifts of **unsigned numbers are logical** and all shifts **of signed numbers are arithmetic**. A right shift will divide a number by 2-*n*, where *n* is the number of shifts. For example, if we had the number 20 and we did a right shift by 3 bits (20 >> 3), we would obtain 20 \* 2-3: 20 / 23 = 20 / 8 = 2.5 = 2. Examples of right shifting follow:

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Binary | | Decimal |
| Arithmetic right shift | Signed | 1110 1100 >> 1 = 1111 0110 | -20 >> 1 = -10 |
| Logical right shift | Unsigned | 1000 0010 >> 1 = 0100 0001 | 130 >> 1 = 65 |

A shift left has the opposite effect. Instead of dividing by powers of 2, we are multiplying by powers of 2. If we have a number, say, 10, and we shift to the left *n* times, it is equivalent to obtaining 10 \* 2*n*, where *n* is the number of shifts. For example, say we have the number 10 and we shift it left 3 times (10 << 3). The operation is equivalent to obtaining 10 \* 23 = 10 \* 8 = 80.

|  |  |  |
| --- | --- | --- |
| Operation | Binary | Decimal |
| Left shift | 0000 1010 << 3 = 0101 0000 | 10 << 3 = 80 |

## Note that vacancies in the right most bits are filled by zeros in case of Left Shift operator.

## Example 1:

# include <iostream>

#include <bitset>

using namespace std;

int main ()

{

unsigned int a = 0x11001, b = 0x11011000;//in decimal, a=25, b=216

unsigned int c = -50;

signed int d = -50;

cout << "a AND b: " << bitset<32>(a & b) << endl;

cout << "a OR b: " << bitset<32>(a | b) << endl;

cout << "a XOR b: " << bitset<32>(a ^ b) << endl;

cout << "b Compliment: " << bitset<32>(~ b) <<endl;

return 0;

## }

## Example 2: (Left shift)

|  |
| --- |
| #include <iostream>  #include <bitset>  using namespace std;  int main() {  unsigned int int1 = 4;  cout<<bitset<32> (int1)<<endl ;  unsigned int int2 = int1 << 1;  cout << bitset<32>(int2) << endl;    unsigned int int3 = int1 << 2;  cout << bitset<32>(int3) << endl; } |

## Example 3: (Right shift )

|  |
| --- |
| #include <iostream>  #include <bitset>  using namespace std;  int main() {  int int1 = 16;  cout<<bitset<32> (int1)<<endl ;  int int2 = int1 >> 1;  cout << bitset<32>(int2) << endl;    int int3 = int1 >> 2;  cout << bitset<32>(int3) << endl;  cout << int1 << endl;  cout << int2 << endl;  cout << int3 << endl; } |

## Exercise:

## Replace value of example 2 from 4 to 1073741824.Answer following questions:

## What happens to int2 and int3 values.

## Write the mathematical formula for conversion of int1 to int2.

## Print value of all three variables at end of your program.

## Replace “unsigned int” datatype to “int”. What change did you observe in your program?

## The number 1073741824=2^30, then why in your program only bit 31 is ON. Shouldn’t be bit 30 be ON and rest of all should be OFF.

|  |
| --- |
| Answer: |

* 1. Make following changes in example 3 to understand bitwise right shift.

1. Replace value of int1 value from 16 to -16. What change did you observe in output?
2. How many minimum bitwise right shifts are required in above case, after which you obtain the same result (check it by shifting bits by different values.).
3. Replace value from 16 to 2147483648. Observe the output. After that, replaced all integer datatype with “unsigned int”. Mention the changes you observed in variables.
4. Now assign -2147483648 value to int1. Why error appears?

|  |
| --- |
| **Answer:** |

## *Implement following tasks without using bitset library.*

* + 1. Ask user to enter a number. Using bitwise operator, check number entered by user is even or odd. In main(), call only function.
    2. Swap two numbers without using ‘temp’ variable. Use XOR bitwise operator.
    3. Write a function that can find parity of a number entered by user. Parity of a number refers to whether it contains an odd or even number of 1-bits. Your program should print whether number has odd numbers of one’s or even.
    4. XOR is widely used for data encryption. Suppose you want to send a secret message to your friend living far away. You and your friend know a secret key, with which you both can encode and decode message. For this program your task is to encode a string and then decode it. Perform both tasks using functions.

\*Use a string datatype array to store the sentence and a char type variable as secret key.

**BONUS TASK:**

1. Given an array arr[], the task is to calculate the sum of the elements from the given array which has even parity i.e. the number of set bits is even using bitwise operator.

|  |
| --- |
| *Input: arr[] = {2, 4, 3, 5, 9} Output: 17 //Only 3(0011), 5(0101) and 9(1001) have even parity So 3 + 5 + 9 = 17*  *Input: arr[] = {1, 5, 4, 16, 10} Output: 15* |