**Unschool**

**C & C++ Assignment**

Name:- Vaidehi Dayaram Dahare

E-mail:- [Vaidehidahare06@gmail.com](mailto:Vaidehidahare06@gmail.com)

Q1) Make a PDF file containing description (around 3-4 lines) syntax, and example of

1. Bitwise Operators:-

In computer programming, a bitwise operation operates on one or more bit patterns or binary numerals at the level of their individual bits. It is a fast and simple action, directly supported by the processor, and is used to manipulate values for comparisons and calculations.

On simple low-cost processors, typically, bitwise operations are substantially faster than division, several times faster than multiplication, and sometimes significantly faster than addition. While modern processors usually perform addition and multiplication just as fast as bitwise operations due to their longer instruction pipelines and other architectural design choices, bitwise operations do commonly use less power because of the reduced use of resources.

* There are four types of Bitwise operators:-

1.Bitwise And

2. Bitwise Or

3. Bitwise Xor

4. Bitwise Not

* **Syntax:-**

1. **Bitwise And:-**

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Let us suppose the bitwise AND operation of two integers 12 and 25.

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bit Operation of 12 and 25

00001100

& 00011001

\_\_\_\_\_\_\_\_

00001000 = 8 (In decimal)

* **Example #1: Bitwise AND**

#include <stdio.h>

int main()

{

int a = 12, b = 25;

printf("Output = %d", a&b);

return 0;

}

Output=8

**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 |.

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25

00001100

| 00011001

\_\_\_\_\_\_\_\_

00011101 = 29 (In decimal)

* **Example #2: Bitwise OR**

#include <stdio.h>

int main()

{

int a = 12, b = 25;

printf("Output = %d", a|b);

return 0;

}

Output =29

* **3.Bitwise Xor:-**

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

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25

00001100

^ 00011001

\_\_\_\_\_\_\_\_

00010101 = 21 (In decimal)

* **Example #3: Bitwise XOR**

#include <stdio.h>

int main()

{

int a = 12, b = 25;

printf("Output = %d", a^b);

return 0;

}

Output= 21

**4. Bitwise Not:-**

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

35 = 00100011 (In Binary)

Bitwise complement Operation of 35

~ 00100011

\_\_\_\_\_\_\_\_

11011100 = 220 (In decimal)

Twist in bitwise complement operator in C Programming

The bitwise complement of 35 (~35) is -36 instead of 220, but why?

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

**2's Complement**

Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1. For example:

Decimal Binary 2's complement

0 00000000 -(11111111+1) = -00000000 = -0(decimal)

1 00000001 -(11111110+1) = -11111111 = -256(decimal)

12 00001100 -(11110011+1) = -11110100 = -244(decimal)

220 11011100 -(00100011+1) = -00100100 = -36(decimal)

**Note:** Overflow is ignored while computing 2's complement.

The bitwise complement of 35 is 220 (in decimal). The 2's complement of 220 is -36. Hence, the output is -36 instead of 220.

Bitwise complement of any number N is -(N+1). Here's how:

bitwise complement of N = ~N (represented in 2's complement form)

2'complement of ~N= -(~(~N)+1) = -(N+1)

* **Example #4: Bitwise complement**

#include <stdio.h>

int main()

{

printf("Output = %d\n",~35);

printf("Output = %d\n",~-12);

return 0;

}

Output = - 36

Output **=** 11

1. **Ternary Operator:-**

In computer programming, ?: is a ternary operator that is part of the syntax for basic conditional expressions in several programming languages. It is commonly referred to as the conditional operator, inline if (iif), or ternary if. An expression a ? b : c evaluates to b if the value of a is true, and otherwise to c. One can read it aloud as "if a then b otherwise c".

It originally comes from CPL, in which equivalent syntax for e1 ? e2 : e3 was e1 → e2, e3.[1][2]

Although many ternary operators are possible, the conditional operator is so common, and other ternary operators so rare, that the conditional operator is commonly referred to as the ternary operator.

* **Syntax for Ternary Operator:-**

The conditional operator is of the form

variable = Expression1 ? Expression2 : Expression3

It can be visualized into if-else statement as:

if(Expression1)

{ variable = Expression2;}

else

{

variable = Expression3;

}

Since the Conditional Operator ‘?:’ takes three operands to work, hence they are also called ternary operators.

* **Example of Ternary Operator:-**

#include <stdio.h>

main()

{

int a , b;

a = 10;

printf( "Value of b is %d\n", (a == 1) ? 20: 30 );

printf( "Value of b is %d\n", (a == 10) ? 20: 30 );

}

**Output:-**

Value of b is 30

Value of b is 20