ASSIGNMENT: OPERATORS

1.Bitwise Operators: Bitwise operators are operators (just like +, *, &&, etc.) that operate on int and uint at the binary level. This means they look directly at the binary digits or bits of an integer.

Some of the bitwise operators are listed below:

1. & (bitwise AND)
2. | (bitwise OR)
3. ~ (bitwise NOT)
4. ^ (bitwise XOR)
5. << (bitwise left shift)
6. >> (bitwise right shift)

1. & (bitwise AND): The & operator compares each binary digit of two integers and returns a new integer, with a 1 wherever both numbers have a 1 and 1 on both side.

   Syntax: a & b = c

   Example: 37 & 23 = 5

2. | (bitwise OR): The | operator compares each binary digit of two integers and returns a new integer, with a 1 wherever either or both the side of binary numbers have a 1.

   Syntax: a | b = c

   Example: 37 | 23 = 55

3. ~ (bitwise NOT): the ~ operator reverses each binary digit in an integer: from 0 to 1 and 1 to 0:

   Example: let us consider

   a: 0 1 0 1 0 0 1 0

   ~a: 1 0 1 0 1 1 0 1

4. ^ (bitwise XOR): The ^ operator is similar to the & and |. If one or the other is a 1, it will insert a 1 in to the result, otherwise it will insert a 0. This is where the name XOR, or "exclusive or" comes from.

   Syntax: a ^ b = c

   Example: 37 ^ 23 = 50

5. << (bitwise left shift): Instead of comparing two integers like &, |, and ^ did, these operators shift an integer. On the left side of the operator is the integer that is being shifted, and on the right is how much to shift by.

   Example: 37 << 3 is shifting the number 37 to the left by 3 places. And working with the binary representation of 37. Replacing the empty blocks by 0. We get the answer as 40 in a binary form.

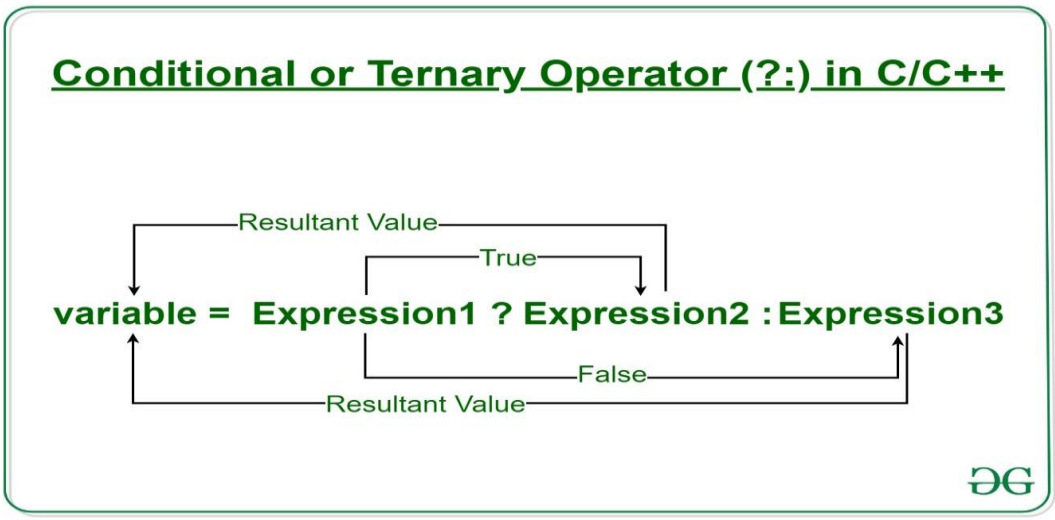
   Therefore 37 << 3 is 40.

6. >> (bitwise right shift): Now that we understand the left bit shift, the next one, the right bit shift, will be easy. Everything slides to the right the amount we specify. The only slight difference is what the empty bits get filled with.

   If we're starting with a negative number (a binary number where the leftmost bit is a 1), all the empty spaces are filled with a 1. If we're starting with a positive number (where the leftmost bit, or most significant bit, is a 0), then all the empty spaces are filled with a 0.

2.Conditional or Ternary Operator (? :):

The conditional operator is kind of similar to the if-else statement as it does follow the same algorithm as of if-else statement but the conditional operator takes less space and helps to write the if-else statements in the shortest way possible.



Syntax:

The conditional operator is of the form

```
variable = Expression1 ? Expression2 : Expression3
```

It can be visualized into if-else statement as:

```
1)if(Expression1)

1){
2)    variable = Expression2;
3)}
4)else
5){
6)    variable = Expression3;
7)}
```

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

3.Program to demonstrate arithmetic operations

```
#include<stdio.h>
int main()
{
int a=5, b=9, c, d, e, f, g;
c=a+b;
printf("Sum of two numbers=%d\n",c);
d=a-b;
printf("difference between two numbers=%d\n",d);
e=a*b;
```

```
printf("Multiplication of two numbers=%d\n",e);
f=a/b;
printf("division of two numbers=%f\n",f);
g=a%b;
printf("Sum of two numbers=%d\n",g);
return 0;
}
```

.