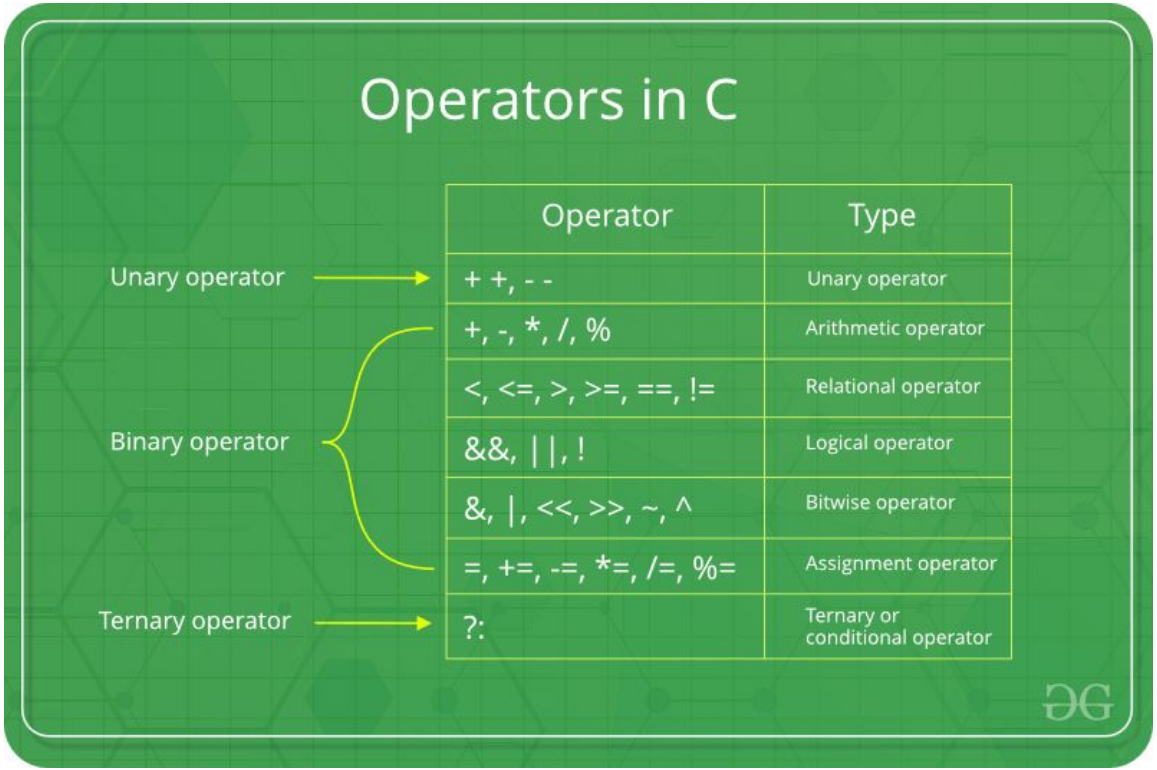
Lecture 22

Bitwise Operators

Operators in C



Bitwise Operators

1. The **& (bitwise AND)** in C takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
2. The **| (bitwise OR)** in C takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.
3. The **^ (bitwise XOR)** in C takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
4. The **<< (left shift)** in C takes two numbers, the left shifts the bits of the first operand, and the second operand decides the number of places to shift.
5. The **>> (right shift)** in C takes two numbers, right shifts the bits of the first operand, and the second operand decides the number of places to shift.
6. The **~ (bitwise NOT)** in C takes one number and inverts all bits of it.

Example : Lecture 22A - Bitwise Operators

Truth Table

X	Y	X & Y	X | Y	X ^ Y
0	**0**	0	0	0
0	**1**	0	1	1
1	**0**	0	1	1
1	**1**	1	1	0

The left-shift and right-shift operators should not be used for negative numbers.

- If the second operand(which decides the number of shifts) is a negative number, it results in undefined behavior in C.
- For example, results of both 1 <<- 1 and 1 >> -1 are undefined.

- Also, if the number is shifted more than the size of the integer, the behavior is undefined.
- For example, 1 << 33 is undefined if integers are stored using 32 bits.
- Another thing is NO shift operation is performed if the additive expression (operand that decides no of shifts) is 0.
  See this for more details.

The bitwise XOR operator is the most useful operator from a technical interview perspective.

It is used in many problems.

A simple example could be “Given a set of numbers where all elements occur an even number of times except one number, find the odd occurring number”.

This problem can be efficiently solved by doing XOR to all numbers.

The ~ operator should be used carefully.

The result of the ~ operator on a small number can be a big number if the result is stored in an unsigned variable.

The result may be a negative number if the result is stored in a signed variable (assuming that the negative numbers are stored in 2’s complement form where the leftmost bit is the sign bit).