SESSION-2

OPERATORS:

Operators are the basic components of C programming. They are symbols that represent some kind of operation, such as mathematical, relational, bitwise, conditional, or logical computations, which are to be performed on values or variables. The values and variables used with operators are called **operands**.

Example:

```
#include <stdio.h>
int main() {
// Expression for getting sum
 int sum = 10 + 20;
printf("%d", sum);
return 0;
}
```

Output:

30

Unary,Binary and Ternary Operators:

On the basis of the number of operands they work on, operators can be classified into three types :

1. **Unary Operators:** Operators that work on single operand.
   Example: Increment(++) , Decrement(--)

2. **Binary Operators:** Operators that work on two operands.
   Example: Addition (+), Subtraction(-) , Multiplication (*)

3. **Ternary Operators:** Operators that work on three operands.
   Example**:** Conditional Operator(? :)

Types of Operators in C:

C language provides a wide range of built in operators that can be classified into 6 types based on their functionality:

Arithmetic Operators:

The **arithmetic operators** are used to perform arithmetic/mathematical operations on operands. There are **9 arithmetic** operators in C language**:**

Code:

SESSION-2

```
#include <stdio.h>
int main() {
int a = 25, b = 5;
// using operators and printing results
printf("a + 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 %% b = %d\n", a % b);
printf("+a = %d\n", +a);
printf("-a = %d\n", -a);
printf("a++ = %d\n", a++);
printf("a-- = %d\n", a--);
return 0;
}
```

Output

```
a + b = 30
a - b = 20
a * b = 125
a / b = 5
a % b = 0
+a = 25
-a = -25
a++ = 25
a-- = 26
```

Relational Operators:

The **relational operators** in C are used for the comparison of the two operands. All these operators are binary operators that return true or false values as the result of comparison.

Code:

```
#include <stdio.h>
int main() {
int a = 25, b = 5;
// using operators and printing results
printf("a < 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 == b: %d\n", a == b);
printf("a != b : %d\n", a != b);
return 0;
}
```

Output

SESSION-2

```
a < b  : 0
a > b  : 1
a <= b: 0
a >= b: 1
a == b: 0
a != b : 1
```

Here, 0 means false and 1 means true.

Logical Operator:

Logical Operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a Boolean value either **true** or **false**.

Symbol	Operator	Description	Syntax
&&	Logical AND	Returns true if both the operands are true.	a && b
||	Logical OR	Returns true if both or any of the operand is true.	a || b
!	Logical NOT	Returns true if the operand is false.	!a

Code:

```
#include <stdio.h>
int main() {
int a = 25, b = 5;
// using operators and printing results
printf("a && b : %d\n", a && b);
printf("a || b : %d\n", a || b);
printf("!a: %d\n", !a);
return 0;
}
```

Output:

```
a && b : 1
a || b : 1
!a: 0
```

SESSION-2

Bitwise Operators:

The **Bitwise operators** are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands.
Note: Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing.

Symbol	Operator	Description	Syntax
&	Bitwise AND	Performs bit-by-bit AND operation and returns the result.	a & b
|	Bitwise OR	Performs bit-by-bit OR operation and returns the result.	a | b
^	Bitwise XOR	Performs bit-by-bit XOR operation and returns the result.	a ^ b
~	Bitwise First Complement	Flips all the set and unset bits on the number.	~a
<<	Bitwise Left shift	Shifts bits to the left by a given number of positions; multiplies the number by 2 for each shift.	a << b
>>	Bitwise Right shift	Shifts bits to the right by a given number of positions; divides the number by 2 for each shift.	a >> b

Code:

```
#include <stdio.h>
int main() {
int a = 25, b = 5;
```

SESSION-2

```
// using operators and printing results
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);
printf("a >> b: %d\n", a >> b);
printf("a << b: %d\n", a << b);
return 0;
}
```

Output

```
a & b: 1
a | b: 29
a ^ b: 28
~a: -26
a >> b: 0
a << b: 800
```

Assignment Operators:

Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and the right side operand of the assignment operator is a value. The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.

The assignment operators can be combined with some other operators in C to provide multiple operations using single operator. These operators are called compound operators.

Symbol	Operator	Description	Syntax
=	**Simple Assignment**	Assign the value of the right operand to the left operand.	**a = b**
+=	**Plus and assign**	Add the right operand and left operand and assign this value to the left operand.	**a += b**
-=	**Minus and assign**	Subtract the right operand and left operand and assign this value to the left operand.	**a -= b**
*=	**Multiply and assign**	Multiply the right operand	**a *= b**

SESSION-2

Symbol	Operator	Description	Syntax
		and left operand and assign this value to the left operand.	
/=	**Divide and assign**	Divide the left operand with the right operand and assign this value to the left operand.	**a /= b**
%=	**Modulus and assign**	Assign the remainder in the division of left operand with the right operand to the left operand.	**a %= b**
&=	**AND and assign**	Performs bitwise AND and assigns this value to the left operand.	**a &= b**
|=	**OR and assign**	Performs bitwise OR and assigns this value to the left operand.	**a |= b**
^=	**XOR and assign**	Performs bitwise XOR and assigns this value to the left operand.	**a ^= b**
>>=	**Rightshift and assign**	Performs bitwise Rightshift and assign this value to the left operand.	**a >>= b**
<<=	**Leftshift and assign**	Performs bitwise Leftshift and assign this value to the left operand.	**a <<= b**

SESSION-2

Code:

```
#include <stdio.h>
int main() {
int a = 25, b = 5;
// using operators and printing results
printf("a = 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 /= 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 ^= b: %d\n", a ^= b);
printf("a >>= b: %d\n", a >>= b);
printf("a <<= b: %d\n", a <<= b);
return 0;
}
```

Output

```
a = b: 5
a += b: 10
a -= b: 5
a *= b: 25
a /= b: 5
a %= b: 0
a &= b: 0
a |= b: 5
a ^= b: 0
a >>= b: 0
a <<= b: 0
```

SESSION-2

Input and Output in C:

In C, there are many input and output for different situations, but the most commonly used functions for Input/Output are scanf() and printf() respectively. These functions are part of the standard input/output library **<stdio.h>**. scanf() takes user inputs (typed using keyboard) and printf() displays output on the console or screen.

Basic Output in C

The **printf()** function is used to print formatted output to the standard output stdout (which is generally the console screen). It is one of the most commonly used functions in C.
The following examples demonstrate the use of printf for output:

```
#include <stdio.h>
int main() {
 // Prints some text
printf("First Print");
 return 0;
}
```

Output

```
First Print
```

Explanation: The text inside "" is called a string in C. It is used to represent textual information. We can directly pass strings to the printf() function to print them in console screen.

Basic Input in C

scanf() is used to read user input from the console. It takes the format string and the addresses of the variables where the input will be stored.
Syntax

```
scanf("formatted_string", address_of_variables);
```

Remember that this function takes the address of the arguments where the read value is to be stored.

Examples of Reading User Input

The following examples demonstrate how to use the scanf for different user input in C:

Reading an Integer

```
#include <stdio.h>
int main() {
```

SESSION-2

```
    int age;
    printf("Enter your age: ");
   // Reads an integer
    scanf("%d", &age);
    // Prints the age
    printf("Age is: %d\n", age);
    return 0;
}
```

Output

```
Enter your age:
25 (Entered by the user)
Age is: 25
```

Problems on Leetcode on Operators:

1. Sum of two Integers
2. Subtract the Product and Sum of Digits of an Integer
3. Add Digits
4. Number of 1 bits
5. Counting Bits
6. Divide Two Integers
7. Minimum Bit Flips to Convert Number