Operator

Operator:- There are the foundation of any programming language. Functionality of C programming language is incomplete without the use of operators. We can define operators as symbols that help us to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands. For example, consider the below statement.

c = a + b;

Here, '+' is the operator known as addition operator and 'a' and 'b' are operands. The addition operator tells the compiler to add both of the operands 'a' and 'b'.

We will discuss some type of operators.

1. Arithmetic operator:- An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Example 1:- **Write a program for arithmetic operators.**

Solution:-

```
#include <stdio.h>
int main()
{
    int a,b,c;
    printf("Enter any two number");
    scanf("%d%d",&a,&b);
    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);
    return 0;
}
```

Output:-

```
Enter any two number 9
                                        4
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

In normal calculation, 9/4 = 2.25. However, the output is 2 in the program.

It is because both the variables a and b are integers. Hence, the output is also an integer. The compiler neglects the term after the decimal point and shows answer 2 instead of 2.25.

The modulo operator % computes the remainder. When a=9 is divided by b=4, the remainder is 1. The % operator can only be used with integers.

Increment and Decrement Operators:- Two operators increment (++) and decrement (--) to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

Example 2:-

```
#include <stdio.h>

int main()
```

```
{
     int a,b;

printf("Enter two number");
scanf("%d%d",&a,&b);
  printf("++a = %d \n", ++a);
  printf("--b = %d \n", --b);
  float c,d;
printf("Enter two float value");
scanf("%f%f",&c,&d);
  printf("++c = %f \n", ++c);
  printf("--d = %f \n", --d);
  return 0;
}
```

Output:-

```
Enter two number 10
                    100
++a = 11
--b = 99
Enter two float value 10.5
                       100.5
++c = 11.500000
++d = 99.500000
```

The operators ++ and -- are used as prefixes. These two operators can also be used as postfixes like a++ and a--.

Assignment Operators:- An assignment operator is used for assigning a value to a variable. The most common assignment operator is (=).

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Example 3:-

```
#include<stdio.h>

int main()

{

int a,c;
printf("Enter any one number");
scanf("%d",&a);
   c = a;      // c is 5
   printf("c = %d\n", c);
   c += a;     // c is 10
   printf("c = %d\n", c);
   c -= a;     // c is 5
   printf("c = %d\n", c);
   c *= a;     // c is 25
   printf("c = %d\n", c);
   c /= a;     // c is 5
   printf("c = %d\n", c);
   c %= a;     // c = 0
   printf("c = %d\n", c);
   return 0;
}
```

Output:-

```
Enter any one number 5
        c = 5
        c = 10
        c = 5
        c = 25
        c = 5
        c = 0
```

Relational Operators:- **A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.**

Relational operators are used in decision making and loops.

Operator	*Meaning of Operator*	*Example*
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

Example 4:-

```
#include <stdio.h>
int main()
{
   int a = 5, b = 5, c = 10;
   printf("%d == %d is %d \n", a, b, a == b);
   printf("%d == %d is %d \n", a, c, a == c);
   printf("%d > %d is %d \n", a, b, a > b);
```

```
        printf("%d > %d is %d \n", a, c, a > c);
        printf("%d < %d is %d \n", a, b, a < b);
        printf("%d < %d is %d \n", a, c, a < c);
        printf("%d != %d is %d \n", a, b, a != b);
        printf("%d != %d is %d \n", a, c, a != c);
        printf("%d >= %d is %d \n", a, b, a >= b);
        printf("%d >= %d is %d \n", a, c, a >= c);
        printf("%d <= %d is %d \n", a, b, a <= b);
        printf("%d <= %d is %d \n", a, c, a <= c);
        return 0;
    }
```

Output:-

```
5 == 5 is 1
5 == 10 is 0
5 > 5 is 0
5 > 10 is 0
5 < 5 is 0
5 < 10 is 1
5 != 5 is 0
5 != 10 is 1
5 >= 5 is 1
5 >= 10 is 0
5 <= 5 is 1
5 <= 10 is 1
```

Logical Operators:- **An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.**

Operator	*Meaning*	*Example*
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

Example 5:-

```
#include <stdio.h>
int main()
{
  int a = 5, b = 5, c = 10, result;

  result = (a == b) && (c > b);
  printf("(a == b) && (c > b) is %d \n", result);
  result = (a == b) && (c < b);
```

```
        printf("(a == b) && (c < b) is %d \n", result);
        result = (a == b) || (c < b);
        printf("(a == b) || (c < b) is %d \n", result);
        result = (a != b) || (c < b);
        printf("(a != b) || (c < b) is %d \n", result);
        result = !(a != b);
        printf("!(a == b) is %d \n", result);
        result = !(a == b);
        printf("!(a == b) is %d \n", result);

        return 0;
    }
```

Output:-

```
(a == b) && (c > b) is 1
(a == b) && (c < b) is 0
(a == b) || (c < b) is 1
(a != b) || (c < b) is 0
!(a != b) is 1
!(a == b) is 0
```

Explanation of logical operator program

(a == b) && (c > 5) evaluates to 1 because both operands (a == b) and (c > b) is 1 (true).

(a == b) && (c < b) evaluates to 0 because operand (c < b) is 0 (false).

(a == b) || (c < b) evaluates to 1 because (a = b) is 1 (true).

(a != b) || (c < b) evaluates to 0 because both operand (a != b) and (c < b) are 0 (false).

!(a != b) evaluates to 1 because operand (a != b) is 0 (false). Hence, !(a != b) is 1 (true).

!(a == b) evaluates to 0 because (a == b) is 1 (true). Hence, !(a == b) is 0 (false)

Bitwise Operators:- **During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.**

Bitwise operators are used in C programming to perform bit-level operations.

Operators	***Meaning of operators***
&	*Bitwise AND*
|	*Bitwise OR*
^	*Bitwise exclusive OR*
~	*Bitwise complement*
<<	*Shift left*
>>	*Shift right*

Comma Operator:- **Comma operators are used to link related expressions together for example.**

int a, c = 5, d;

The sizeof operator

The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

Example 6:-

```
#include <stdio.h>
int main()
{
   int a;
   float b;
   double c;
   char d;
   printf("Size of int=%lu bytes\n",sizeof(a));
   printf("Size of float=%lu bytes\n",sizeof(b));
   printf("Size of double=%lu bytes\n",sizeof(c));
   printf("Size of char=%lu byte\n",sizeof(d));
   return 0;
}
```

Output:-

```
Size of int = 4 bytes

Size of float = 4 bytes

Size of double = 8 bytes

Size of char = 1 byte
```