Chapter 3

OPERATORS AND EXPERSSIONS

OPERATORS:-

- **Arithmetic Operator:-**

  Ex: +, - , /, *, % , are used to perform the basic arithmetic operator such as addition, subtraction, division, multiplication, etc.

(*) Note:

Modulus operator can't be used on floats.

- **Equality, Relational and Logical Operators:-**

  These operators are basically used to compare the operands. These are binary operators. The result of these operators is of type boolean I-e true or false. They are,

  < (less than), > (greater than), == (equality), <= (less than or equal to), >= (greater than or equal to), != (not equal to).

- **Logical Operators:-**

  These operators are used to make decision. They are also used to connect two or more relational expressions. They are,

  ! (logical NOT), && (logical AND), || (logical OR).

- **Assignment Operator:-**

```
Syntax :-     identifier = expression;
Ex:-          x = 1234;
                  x = a + b;
```

- **Increment and Decrement Operators:-**

  Increment operators are represented by ++ symbol and decrement operators are represented by -- symbol.

  ++ increments the value by 1.

  -- decrements the value by 1.

  Ex:- x++, y—

Note:

++ and – can be placed before or after the operand. If it appears before the operand it is called preincrement (or predecrement). If it is placed after the operand it is called the post increment (or postdecrement).

```
Ex:-  int x, y;
          x++;      ➔postincrement
              ++x;      ➔preincrement
```

y--; ➔postdecrement
--y; ➔predecrement

Ex:- In a program,

int x, y;

y = x++; ➔ Assigns 'x' value to 'y' first, then increment 'x' value.
y = ++x; ➔ First increments 'x' value and then assigns.
y = x--; ➔ Assign first and then decrement.
y = --x; ➔ First decrement and then assign.

- **<u>Conditional Operator:-</u>**

  ? ➔ is used . It is also called as ternary operator as it takes three operands.
  Syntax:-

  (exp) ? (Operand1) : (Operand2);

  here exp is tested, if it results is true value, then it returns operand 1 otherwise operand 2.

  Ex:-

  Int x = 10, y = 20, z;
  z = (x > y) ? x : y;

- **<u>Sizeof Operator:-</u>**

  ***) returns size (in bytes) of an object or datatype;**
  ***) sizeof() operator should be written in lower-case letters.**
  ***) it should precede its operand.**

  Ex:-

  Int x;
  Sizeof(x); ➔ returns 2 bytes.

- **<u>Bitwise Operator:-</u>**

  All data items are stored in computer's memory as a sequence of 0's and 1's. there are many applications which require the manipulatio of these 0's and 1's directly. So these are used.
  Bitwise operators are,

  ~ ➔ bitwise not
  << ➔ bitwise leftshift
  >> ➔ bitwise rightshift
  & ➔ bitwise AND
  ! ➔ bitwise OR
  ^ ➔ bitwise XOR

EXPRESSIONS

An expression is a combination of operators , constants & variables arrayed as per the rule of the language. It may also include function calls which return values.
An expression may consist of one or more operands, and zero or more operators to produce a value.

Types:

1) Constant Expression	**2) Integral Expression**
3) Float Expression	**4) Pointer Expression**
5) Relational Expression	**6) Logical Expression**
7) Bitwise Expression	

1) Constant Expression:-

Consists of only constant values like
Ex:- 15, 20 +5 / 2.0

2) Integral Expression:-

Are those which produce integer results after implementing all the automatic and explicit type conversions

```
Ex:-        m * n - 5;
           5 + int (2.0);
```

3) Float Expression:-

Which, after all conversions produce floating point results.

```
Ex:-         x + y          // where x and y are floats
```

4) Pointer Expression:-

Produce address values
Ex:- &m, ptr

5) Relational Expression:-

Yields results of type true or false.
Ex:- x <= y, a + b == c + d

6) Logical Expression:-

Combines two or more relational expressions and produces Boolean type results I-e true or false.

```
Ex:-    (a > b)  && (x==10)
        (x==10)  || ( y==5)
```