I/II SEM Engineering.

MODULE - 1.

# Introduction to C Language.

## Pseudo Code Solution to Problem:-

The Pseudo Code is the first Step in writing a Program. Pseudo Code is used by most Programmers to help translate from an English Language description of a Problem to a Program which is written in 'C'.

"Pseudo Code is a Compact and informal high-level description of an algorithm that uses the Structural Conventions of a Programming Language"

Since the Pseudo Code is not an executable Program, no Standards are defined for writing it.



Example Problem:- Print the numbers from 4 to 9 and their Squares.

Solution:-

Start with the number 4
Compute its Square.
Print the number and its Square.
do the Same for each of the other numbers from 5 to 9.

An Algorithm is the Collection of Processes which give a Precise method to Solve a Problem. It is a Step by Step Procedure followed to Solve a given Problem.

Basic Concepts that Provides an unambigous and Precise Steps to Solve a Problem.

Example:-

Step 1 : [Initialize]
Start

Step 2 : [Input the number starting with 4]
Read number

Step 3 : [Compute the Square of number]
Sqr ← number * number

Step 4 : [Display the number and its Square]
Print number, Sqr.

Step 5 : [Finished]
Stop.



## Basic Concepts of a C Program:-

The Basic Concepts of C Program includes Comments used in Program, The main Program headers and the General Outline of a "C" Program.

These are the Various features that must be included in an every "C" Program.

## Comments and Programming Style.

The line that is not in the Pseudocode and written at the begining of each Program with the following Symbols Called "/* */" Comment. The Comments are not Strictly required because a Program without the Comment is also technically correct. But at Some Part it is Considered as a good Programming Style.

The Comment which describes the Short description that the Problem to be Solved.

The Comments begins with "/*" and ends with */. The Symbols /* and */ are Called Comment delimiters because they mark the begining and end of the Comment.



Example:- /* Program: Print numbers from 4 to 9 */

## The Program Header:

The next two lines of the C Program after the Comment line will be.

```
#include <Stdio.h>
main()
```

Any line in a Program that Start with Symbol # is an instruction to the Compiler.

The line #include tells the compiler to allow the Program to Perform Standard Input and Output Operations and will be using Part of the Standard function library.

A function is a Building block of a Program that Performs one Particular task.

The header file Stdio.h will be included in the Program as,

Stdio → Standard input / Output

.h → Says this file is header file.



The second line after the Comment, main() is the main Program header. Every Program must have a main function since this is where the Computer begins to execute a Program.

The Parentheses ( ) are necessary for every function in a C Program.

## The Body or Action of the Program:-

In a C Program, after the the header or topline, is a set of braces { } Containing a Series of C Statements which Comprise the body Called the action of the Program.

## The Structure of C Program.

```
/* Program Pr.C: Print numbers from 4 to 9 */
#include<Stdio.h>
main()
{
    /* action Portion of the Program */
    ----
}
```

## Declaration, Assignment and Print Statements:-

The C-Statements that are used to declare Variables, Store Values in these Variables and Print the information.



### Variables and Declaration Statements:

Data type - Int. (integer)

The first C Statement introduced called as a declaration Statement, it is a very Crucial Part of every C-Program.

The declaration describes the Computer as Storage locations or Variables to use in Program.

A Variable is a Name for a Physical location within the Computer that Can hold only one Value at a time. As the name itself indicates the Variable Can change its Value.

As the variable number which holds an integer, So, declare its data type as int, that is a simplest integer datatype in C. And also the sqnumber holds an integer, the same integer data type can be used to declare these two variables as shown below.

```
int number, sqnumber;.   equivalent to  ⇒  int number;
↓                                          int sqnumber;
Datatype
        Variables.
```

The (;) Punctuation at the end of declaration. A semicolon terminates the every complete statement in C.

A comment line and #include compiler directives does not need a semicolon since these are not a C-Statements. And also the main Program header does not need. because it is not a complete C statement.



The Pseudo code modified by Variables.

```
Start with number=4
Compute Sqnumber
Print number, sqnumber
do the same for number= 5, 6, 7, 8, 9.
```

In the above Pseudo code the modified variables are number and sqnumber.

## Assignment Statements :-

The assignment Statement which computes a value and places it in a given storage location. The actual Processing of assignment is by giving values to variables.

Example :- `int number;`
`number = 4; /* Assignment Statement */`

The above assignment Statement puts 4 into the storage location associated with the variable number. In an assignment statement we use the symbol = which is assignment operator in C.



### General Form of Assignment Statement.

```
Varname = expr ;
```

↑ Name of variable ↑ Value of expression.

The above statement in C is interpreted as Evaluate the expression expr on the right hand side of the assignment statement and Stores the value in the variable name Varname on the left hand side of the assignment statement.

Example :- `Cost = 100;`

↑ Variablename ↑ Value.

## Arithmetic Operations in Assignment Statements:-

The right-hand Side of an assignment Statement must be a Valid C expression. An arithmetic expression is one of the following, a Constant [eg:- 3], a Variable [eg: cost] or a larger formula built from Simpler expressions by an arithmetic operations

Example:- The assignment Statement which gives the Value of the expression cost+3 to the variable price.

```
Price = cost + 3;
```



## Print Statements:-

The PSeudoCode, Print number, Sqnumber Can be revise it to the following, which is not quite C but very close as:

```
Print(number, Sqnumber)
```

The very Simple way to translate the above Statement into C using Printf. The basic form is shown below.

```
Printf("%d %d", number, Sqnumber);
```

The above Printf will display the Values of number and Sqnumber on the output Screen of a Computer.

## The different forms of Printf.

* The Printf has just a literal String
* The Printf has Values of one or more expressions to be Printed.

General form of a Printf with just a literal String.

```
Printf (" ..... ");
        ↑
   Literal String.
```

Example :-

```
Printf (" Welcome to C Program\n");
```

The literal String will be Placed inside the Parentheses, the Symbols within that String are Printed



General form of Printf with Expressions to be Printed :

```
Printf (" ..... " , ..... );
        ↑              ↑
Control String, including    List of Variables,
Conversion Characters        Expressions to be Printed.
```

Example :-

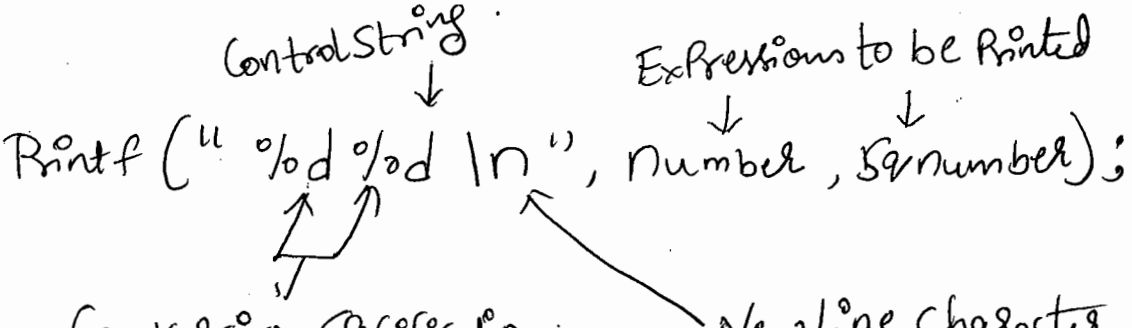
```
Printf (" %d %d \n" , Sum, avg);
```

The above Printf displays the values for the Variables which are Sum and avg will be displayed on output Screen.

## The Various Guidelines for Printf :-

* A Printf always Contains a Control String or format String in Quation marks.
* Each Value to be Printed needs a Conversion Specification like %d to hold it space in Control String.
* The Symbol \n in the Control String tell the machine to SKiP to a new line.
* If there are Variables or expressions to be Printed, Commas are used to seperate them from the Control String and each other
* The Control String for Printf is "%d %d \n" and two items to be Printed are the Values of the Variables number and Sqnumber. Since these two Variables hold integers the Control String must Contain Conversion Specifications for two integer Values Printing. %d (d Stands for decimal or digit).





Types, Operators and Expressions:

There are few Basic datatypes in C as:

* Char - a single byte, it holds only one Character in the local Character set.
* int - an integer, it holds 16-bit (2 bytes).
* float - Single Precision floating Point, it holds (4bytes)
* double - double Precesion floating Point.

Here, the Various Qualifiers that Can be applied to these basic data types as.

-> Integers - Short int a;(16 bit)
long int x;(32 bit) The int Can be omitted in the above declarations, and intent is that Short and long Can be Provided with different lengths of integers.



-> The Qualifier Signed or Unsigned may be applied to Char or any integer.

* Unsigned numbers are always Positive Zero, and it has the laws of arithmetic modulo $2^n$, where n is number of bit in that type.

* Signed numbers are either Positive or Negative numbers.

Example:- If characters are 8-bits

-> Unsigned char variables have values between 0 and 255

-> Signed char variables have values between -128 and 127.

Declarations of different datatypes:-

```
int lower, upper;
Char c;
Char str[1000];
```

We can initialize variables through declaration also

```
int i=0;
float eps=1.0;
Char c='1*';
double x=1.5;
```

The qualifier Constant [Const] can be used for declaration of any variable to specify that its value will be unchanged.

Example:-

```
Const double P= 2.156782341116;
Const char c[] = "welcome";
```

The Const declaration can also be used with array arguments as

```
int Strlen (Const char[]);
```

## The Various Types of Operators:

Types of Operators
- → Arithmetic Operators Ex: +, -, *, etc.
- → Assignment Operators Ex: =, +=
- → Increment/DeCrement Ex: ++, --
- → Relational Operators Ex: <, >, <=, ...
- → Logical Operators Ex: &&, ||, !
- → Conditional Operators Ex: ? :
- → Bitwise Operators Ex: &, ^, etc.



## Arithmetic Operators:

The operators that are used to Perform arithmetic Operations Such as addition, Subtraction, etc are Called arithmetic Operators.

The Various arithmetic Operators are Shown below.

Arithmetic Operators
- → Addition +
- → Subtraction -
- → Multiplication *
- → Division /
- → Modulus %

Program: C-Program Show Usage of Arithmetic Operators.

```
#include<stdio.h>
Void main()
{
    int a, b, Sum, Sub, mul, div, modu;
    a=15;
    b= 5;
    Sum = a+b;  /* Perform addition */
    Sub = a-b;  /* Perform Subtraction */
    mul = a*b;  /* Multiplication */
    div = a/b;  /* Division */
    modu = a%b;  /* Modulus */
    Printf(" Sum = %d", Sum);
    Printf(" Subtraction= %d", Sub);
    Printf(" Multiplication = %d", mul);
    Printf(" Division = %d", div);
    Printf(" Modulus = %d", modu);
    getche();
}
```



Output:

```
Sum = 20
Subtraction = 10
Multiplication = 75
Division = 3
```

## Assignment Operator:

An Operator which is used to assign the data or result of an expression into a Variable called an assignment Operator.

An assignment operator is denoted by '='

Types of Assignment Statements
- → Simple assignment Statement
- → Shorthand assignment Statement
- → Multiple assignment Statement.



### → Simple assignment Statement:

Syntax:

```
Variable = expression;
```

```
a = 10 ; //Constant
a = b ; //Variable.
a = x+y ; // expression.
```

Here, the expression can be a constant, Variable or any expressions.

### → Short hand assignment Statement.

The Operators Such as +=, -=, *=, /= and %= are called Short hand assignment Operators. The assignment Statements that use these Operators are called assignment Statements.

Example: a = a+10; //Simple assignment

a+=10; //Short hand assignment Statement.

-> Multiple Assignment Statement.

Assigning a Value or a Set of Values to different Variables in one Statement Called Multiple Assignment Statement.

Example: i=10;
j=10;
K=10;
} Can be written as i=j=K=10;

Multiple assignment Statement.



## Increment and Decrement Operators:

### Increment Operator:

++ is an increment operator. This is a Unary Operator. It increments the value of a variable by one. There are two ways of increment operators shown below.

Increment Operator
- -> Post increment: a++
- -> Pre increment: ++a

} Increments the value of a by 1

## Post increment:

The increment operator ++ is placed immediately after the operand, is called Post increment.

Example:

```
Void main()
{
    int a=20;
    a++;
    Printf("a=%d", a);
}
```

```
/* Initialization */
a = 20
a = 21
/* Output */
a = 21
```

## Pre increment:

The increment operator ++ is placed before the operand is called Pre increment.



Example:

```
Void main()
{
    int a=10;
    ++a;
    Printf("%d", a);
}
```

```
Initialization
a = 10
a = 11
Output
11
```

## The difference between Post and Pre increment

Let a = 20

Post increment

b = a++;

- Current value of a used — b = a — b = 20
- a is incremented by 1 — a = a + 1 — a = 21

The current value of the variable is unchanged even it increments the variable.

Pre increment

Let a = 20;

b = ++a;

→ a is incremented by 1 a = a+1 a = 21

↓ Incremented Value of a is used b = a b = 21

From the above example even though the final values of a's remain same, the values of b are different.

## Decrement Operator :

-- is a decrement operator. This is a unary operator. It decrements the value of variable by one.

The two ways of decrement operators are.



Decrement Operator →
- Post decrement: a-- } Decrements value of a by 1
- Pre decrement: --a } Decrements value of a by 1

Example:

Post decrement

```
Void main()
{ int x=5;
  x--;
  Printf("%d", x);
}
```

Output

4

Pre decrement:

```
Void main()
{
  int x=5
  --x;
  Printf("%d", x);
}
```

Output

4.

## Relational Operators:

The operators that are used to find the relationship between two operands are called relational operators.

The relationship between two operand values results in true (always 1) or false (always 0).

Relational Operators

| Description | Operator | Priority | Associativity |
|---|---|---|---|
| Less than | < | 1 | Left to right |
| Lesser/equal | <= | 1 | Left to right |
| Greater | > | 1 | Left to right |
| Greater/Equal | >= | 1 | Left to right |
| Equal | == | 2 | Left to right |
| Not Equal | != | 2 | Left to right |



## Relational Expressions:

An expression that contains relational operators called relational expression.

```
a+b>c    /* Valid */

a=<b     /* Invalid operators not in order */
```

Precedence of operators:

-> Evaluate the expression within Parentheses

-> Evaluate Unary Operators.

→ Evaluate Arithmetic Operations/expressions.

→ Evaluate Relational expressions.

Example: 100/20 <= 10-5+ 50%10

NOTE:

Arithmetic Operation and Then Relational Operators

5 <= 10-5+ 50%.10

5 <= 10-5+ 0

5 <= 5 + 0

5 <= 5 [Here 5 = 5]

So result = 1.



## Logical Operators:

C Language has no logical data type and hence other data types such as int and char are used to represent logical data.

→ data Value zero considered for False

→ data Value one considered for True.

The operators that are used to combine two or more relational expressions are called logical operators.

In other way, the logical operators are used to combine two or more relational expressions.

| Description | | Operator | Priority | Associativity |
|---|---|---|---|---|
| Logical Operators | not (unary operator) | ! | 1 | Left to right |
| | and (binary operator) | && | 2 | Left to right |
| | OR (binary operator) | \|\| | 3 | Left to right |

## Logical NOT:

The logical not operation denoted by operator "!" can be true or false. The result is true if the operand is false and result is false if operand is true.

| Operand | !Operand |
|---|---|
| True (1) | False (0) |
| False (0) | True (1) |



## Logical AND

The logical AND operation denoted by operator "&&" is true if and only if both the operands are evaluated to true. If any one of operand is evaluated false, the result is false.

| Operand1 | AND | Operand2 | Result |
|---|---|---|---|
| True (1) | && | True (1) | True (1) |
| True (1) | && | False (0) | False (0) |
| False (0) | && | True (1) | False (0) |
| False (0) | && | False (0) | False (0) |

## Logical OR:

The logical OR operation is denoted by operator || is true if and only if at least one of the operands is evaluated to true. If both the operands are evaluated to false, the result is false.

| Operand 1 | OR | Operand 2. | Result |
|---|---|---|---|
| False (0) | \|\| | False (0) | False (0) |
| False (0) | \|\| | True (1) | True (1) |
| True (1) | \|\| | False (0) | True (1) |
| True (1) | \|\| | True (1) | True (1). |



## Evaluating Logical Expressions:-

The Precedence rule to be followed while evaluating the expression.

* Paranthe Ses
* Unary expression.
* Arithmetic expression
* Relational expression
* Logical expression.

Example: Evaluate the expression

$a+2>b \;\&\&\; !c \;||\; a != d \;\&\&\; a-2 <= e.$

Where $a=11$ $b=6$ $c=0$ $d=7$ $e=5$

After Substituting the Values.

$$11+2>6\ \&\&\ !0\ ||\ 11!=7\ \&\&\ 11-2<=5$$

$$11+2>6\ \&\&\ 1\ ||\ 11!=7\ \&\&\ 11-2<=5$$

$$13>6\ \&\&\ 1\ ||\ 11!=7\ \&\&\ 11-2<=5$$

$$13>6\ \&\&\ 1\ ||\ 11!=7\ \&\&\ 9<=5$$

$$1\ \&\&\ 1\ ||\ 11!=7\ \&\&\ 9<=5$$

$$1\ \&\&\ 1\ ||\ 11!=7\ \&\&\ 0$$

$$1\ \&\&\ 1\ ||\ 1\ \&\&\ 0$$

$$1\ ||\ 1\ \&\&\ 0$$

$$1\ ||\ 0$$

Result = 1



## Conditional Operator:

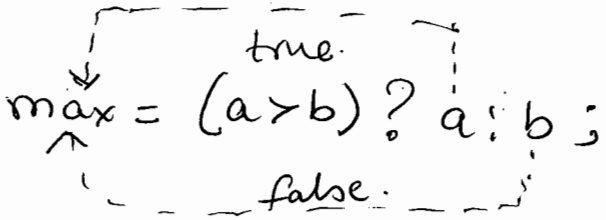
The conditional operator is also called ternary operator. As the name indicates, an operator that operates on three operands is called ternary operator

The ternary operators are ? and :

Syntax: (exp1) ? exp2 : exp3;

- -> exp1 is an expression evaluated to true/false
- -> if exp1 is evaluated to true, exp2 is executed
- -> if exp1 is evaluated to false, exp3 is executed.

Example:




Program:

```
#include <stdio.h>
Void main()
{
  int a, b, big ;
  Printf(" entir value of a,b \n");
  Scanf (" %d %d", &a, &b);

  big = (a>b) ? a:b;
  Printf (" Big %d \n", big);
}
```



Output:

Entir Value of a, b

15 45

a = 15 b = 45 /* (a>b) ? a : b */

big = 45 /* big = ? */

Output

45

## Bitwise Operators:-

The operators that are used to manipulate the bit of given data are called bitwise operators. The various bitwise operators are.

| Description | Operator | Precedence | Associativity |
|---|---|---|---|
| Bitwise and | & | 3 | L → R |
| Bitwise or | \| | 5 | L → R |
| Bitwise Xor | ^ | 4 | L → R |
| Bitwise Negate | ~ | 1 | L → R |
| Left Shift | << | 2 | L → R |
| right Shift | >> | 2 | L → R. |



### Bitwise AND:

If the corresponding bit positions in both the operands are 1, then AND operation result in 1 otherwise results in "0". The AND operator denoted by Symbol '&'

Example :-

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

Equivalent ⟹

0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1

Example Program:-

```
#include<stdio.h>
Void main()
{
  int a,b,c;
  a=10;
  b=6;
  c=a&b;
  Printf("%d %d=%d\n",a,b,c);
}
```

Initialization

| | |
|---|---|
| a=10 | 0 0 0 0 1 0 1 0 |
| b=6 | 0 0 0 0 0 1 1 0 |
| AND Bit | 0 0 0 0 0 0 1 0 |
| C=2 | |



Output

10 & 6 = 2

Bit wise OR :-

If the corresponding bit positions in both the operands are 0, then OR operations results in 0, otherwise, OR operation results in 1.

The bit wise OR operator resulted or denoted by symbol "|".

| | |
|---|---|
| 0 OR 0 = 0 | 0\|0 = 0 |
| 0 OR 1 = 1 | 0\|1 = 1 |
| 1 OR 0 = 1 | 1\|0 = 1 |
| 1 OR 1 = 1 | |

Example Program:-

```
#include <stdio.h>
Void main()
{
  int a,b,c;
  a=10;
  b=6;
  c=a|b;
  Printf("%d|%d=%d",a,b,c);
}
```

Initialization:

| | Binary |
|---|---|
| a=10 | 00001010 |
| b=6 | 00000110 |
| ORbit | -------- |
| c=14 | 00001110 |

Output
10|6 = 14.

## Bit wise XOR: (^)

If the corresponding bit positions in both the operands are different then ex-OR operation results in 1 otherwise Zero (0).

| | |
|---|---|
| 0 X-OR 0 = 0 | 0^0=0 |
| 0 X-OR 1 = 1 | 0^1=1 |
| 1 X-OR 0 = 1 | 1^0=1 |
| 1 XOR 1 = 0 | 1^1=0. |

Example Program:-

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

```
    c = a ^ b;
    Printf("%d ^ %d = %d", a, b, c);
}
```

Initialization

a = 10 — 0 0 0 0 1 0 1 0

b = 6 — 0 0 0 0 0 1 1 0

X-OR bits — 0 0 0 0 1 1 0 0

c = 12

Output: 10 ^ 6 = 12.

## Negate or One's complement (~)

The Operator that is used to change every bit from 0 to 1 and 1 to 0 in Specified Operand. Called ones complement.



Example Program:

```
#include <stdio.h>
Void main()
{
  unsigned char a, b;
  a = 10;
  b = ~a;
Printf("\n%d = %d\n", a, b);
}
```

Initialization

a = 10

0 0 0 0 1 0 1 0

Change 0 to 1, 1 to 0 — 1 1 1 1 0 1 0 1.

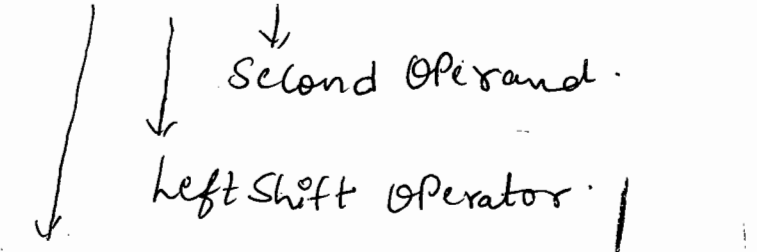
Output

~10 = 245.

## Left Shift Operator! (<<)

The operator that is used to Shift the data by a Specified number of a bit Positions towards left called Left Shift operator.

Syntax & Example: b = a << num.




where,
→ first operand is the Value to be Shifted.
→ Second operand is Specied number of bit to be Shifted.



Example Program:

```
#include <Stdio.h>
Void main()
{
  int P, q;
  P = 5;
  q = p<<1;
Printf("%d<<1=%d\n", P, q);
}
```

Initialization

P=5 00000101

Leftshift 00001010

Output:

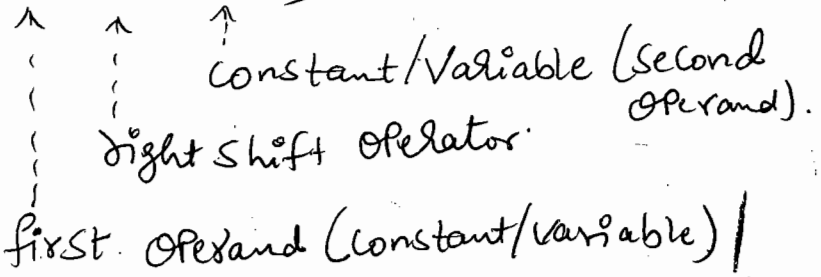
5<<1 = 10

Note: Left Shift by one bit is multiply by 2.

## Right Shift Operator (>>):-

The operator that is used to Shift the data by a Specified number of bit Positions towards right called right Shift operator.

Syntax & Example:- `b = a >> num ;`




→ The first operand is value to be shifted

→ The second operand Specifies number of bit to be shifted.



Program:-

```
#include<Stdio.h>
Void main()
{
  int a, b;
  a= 10;
  b = a>>1;
Printf("%d>>1=%d", a, b);
}
```

Initialization:

Binary.

a=10 0 0 0 0 1 0 1 0

b=5 0 0 0 0 0 1 0 1

Output

10 >> 1 = 5

Note: Right Shift by 1-bit is divide by 2.

# Precedence Of all the Operators!

## [Hierarchy of Operators]

| Operator Category | Order Of Precedence | Associativity |
|---|---|---|
| ( )<br>[ ] | Inner most Brackets / function calls / Array elements | Left → right<br>L → R |
| Unary operators | ++, --, !, ~, +, -, &, * | Right → left<br>R → L. |
| Arithmetic operators | *, /, % | L → R |
| Arithmetic operators | -, + | L → R |
| Shift Operator | <<, >> | L → R. |
| Relational Operators | <, <=, >, >= | L → R |
| Equality Operators | ==, != | L → R |
| Bitwise AND | & | L → R |
| Bitwise XOR | ^ | L → R. |
| Bitwise OR | \| | L → R. |
| Logical AND | && | L → R |
| Logical OR | \|\| | L → R |
| Conditional Operator | ?: | R → L |
| assignment operators | =, +=, -=, /=, *=, %= | R → L. |

Example:-

$X_1 = (-b + \text{Sqrt}(b*b - 4*a*c))/(2*a)$

Assume $a=1 \quad b=-5 \quad c=6.$,

$X_1 = (-(-5) + \text{Sqrt}(-5)*(-5) - 4*1*6))/(2*1)$

$(5 + \text{Sqrt}(-5)*(-5) - 4*1*6))/(2*1)$

$(5 + \text{Sqrt}(25 - 4*1*6))/(2*1)$

$(5 + \text{Sqrt}(25 - 4*6))/(2*1)$

$(5 + \text{Sqrt}(25 - 24))/(2*1)$

$(5 + \text{Sqrt}(1))/(2*1)$

$(5 + 1.0)/(2*1)$

$(5 + 1.0)/2.$

$6.0/2.$

Result = 3.0