PROGRAMMING

IT 112: Introduction to Programming

Dr. Manish Khare

Dr. Bakul Gohel

Lecture 5,6&7

Literals

➢ value that is expressed as itself ; used to assign value to variable or represent in a statement or expression

➢ In some literatures it is also called *constant*

```
int a = 85;
int aa = -420;
float b = 0.00314;
float bb = 3.14e-2;
char x = 'Y';
float c;
c = 1.33+a + b*5;
```

⬅ literals

Integer Literals

➢ An integer literals must have at least one digit.

➢ It must not have a decimal point.

➢ It can be any of zero, positive or negative.

➢ If no sign precedes an integer literal, it is assumed to be positive.

➢ No commas or blanks are allowed within an integer literal.

➢ Example

- 426

- +782

- -8000

- -7605

Integer Literals

➢ Different integer data type literals

➢ It is important at different application and need to take into consideration

```
85        /* decimal */
0213       /* octal */
0x4b       /* hexadecimal */
30        /* int */
30u        /* unsigned int */
30l        /* long */
30ul        /* unsigned long */
```

We will see more about it whenever need it during the course...

Real or float Literals

➢Real literals could be written in two forms

- ▪ Fractional form

- ▪ Exponential form

Real or float Literals

➢ Fractional form

- A real literal must have at least one digit.

- It must have a decimal point.

- It could be either positive or negative.

- Default sign is positive.

- No commas or blanks are allowed within a real constant.

- Example
    - +325.34
    - 426.0
    - -32.76
    - -48.5792

Real or float Literals

➢ Exponential form

- The exponential form is usually used if the value is either too large or too small.

- In exponential form of representation, the real value is represented in two parts. The part appearing before 'e' is called **mantissa**, whereas the part following 'e' is called **exponent**

- For ex, 0.000342 can be written in exponential form of 3.42e-4 (which is equivalent to 3.42×10^{-4})

Exponential form

- The mantissa part and the exponential part should be separated by a letter e or E.

- The mantissa part may have a positive or negative sign.

- Default sign of mantissa part is positive.

- The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.

- Example
    - +3.2e-5
    - 4.1e8
    - -0.2e+3
    - -3.2e-5

Character literals

➢ A character literal is a single alphabet, a single digit or a single special symbol enclosed between single quotes.

➢ The maximum length of a character literals can be 1 character.

- ■ 'A'
- ■ 'I'
- ■ '5'
- ■ '='
- ■ '$'

Integer Literals

```
85       /* decimal */
0213      /* octal */
0x4b      /* hexadecimal */
30       /* int */
30u       /* unsigned int */
30l       /* long */
30ul      /* unsigned long */
```

C - Input / Output

➢ Output operation
  ▪ an instruction that displays information stored in memory e.g. printf
➢ Input operation
  ▪ an instruction that copies data from an input device into memory e.g. scanf
➢ Input/output function
  ▪ A C function that performs an input or output operation e. g scanf, printf

➢ These functions is available in "stdio.h" standard library file

➢ Therefore, we need to include it as **"#include<stdio.h>"** when ever we use printf or scanf
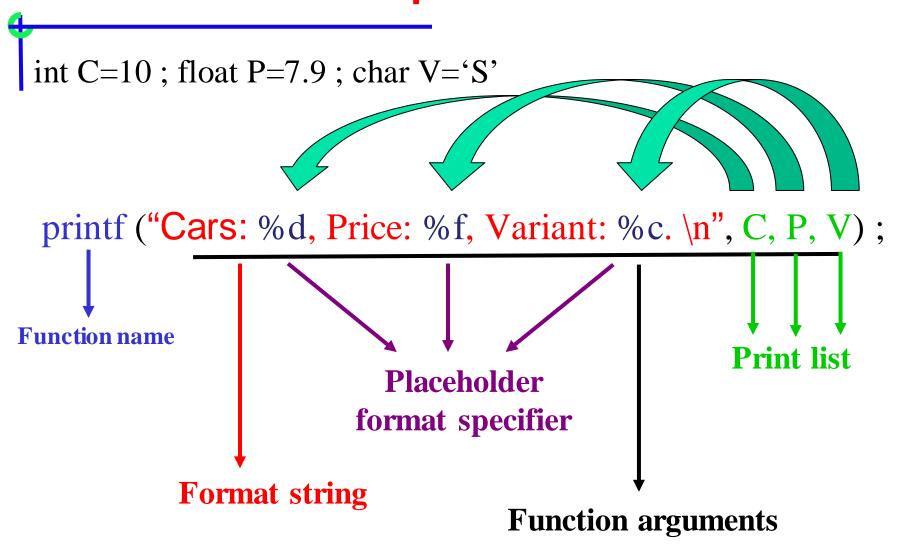
printf and scanf

```c
int main()
{
    int C ; float P; char V;

    scanf("%d %f %c",&C, &P, &V);
    printf("Cars: %d, Price: %f, Variant: %c. \n", C, P, V) ;

    return 0;
}
```
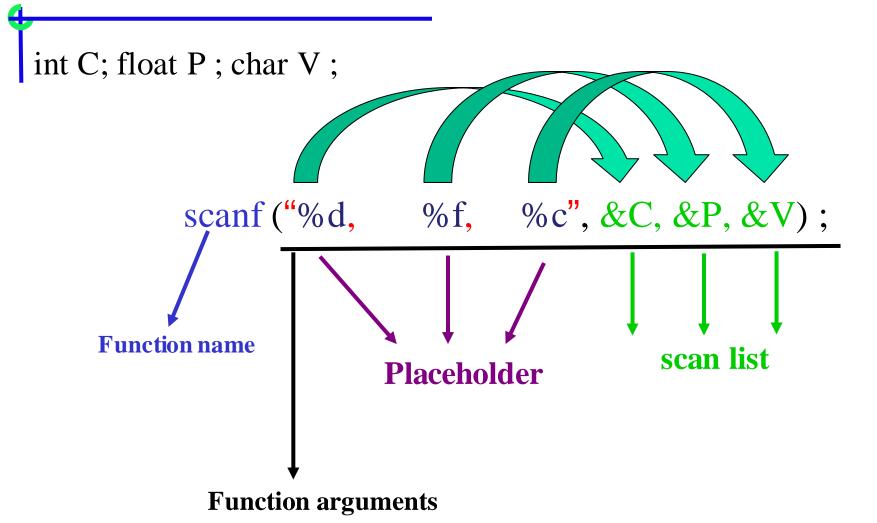
printf

int C=10 ; float P=7.9 ; char V='S'

printf ("Cars: %d, Price: %f, Variant: %c. \n", C, P, V) ;

Function name

Placeholder format specifier

Print list

Format string

Function arguments

Output -> Cars: 10, Price: 7.900000, Variant: S.

scanf

int C; float P ; char V ;

scanf ("%d,   %f,   %c", &C, &P, &V) ;

Function name

Placeholder

scan list

Function arguments

Input -> 10 7.9 S

printf

Placeholder	Variable Type
%c	char
%d	int
%f	double
%lf	Long double

Declaration of variable

➢ A value can be assigned to a variable at the time the variable is declared.

- int speed = 30;

- char flag = 'y';

➢ Several variables can be assigned the same value using multiple assignment operators.

- a = b = c = 5;

- flag1 = flag2 = 'y';

- speed = flow = 0.0;

Input / Output

- ➢ printf
    - Performs output to the standard output device (typically defined to be the screen).
    - It requires a format string in which we can specify:
- ➢ The text to be printed out.
- ➢ Specifications on how to print the values.
    - printf ("The number is %d.\n", num) ;
- ➢ The format specification %d causes the value listed after the format string to be embedded in the output as a decimal number in place of %d.
- ➢ Output will appear as: The number is 125

Form of a C Program

➢ Form of a C program indicated how it has to be written.

➢ There are certain rules about the form of C program that are applicable to all C programs.

- Each instruction in a C program is written as a separate statement.

- The statements in a program must appear in the same order in which we wish them to executed.

- Blank space may be inserted between two words to improve the readability of the statement.

- All statements should be in lower case letters.

- C has no specific rules for the position at which a statement is to be written in a given line.

- Every C statement must be end with a semicolon (;). Thus ; acts as a statement terminator.

➢ scanf

- Performs input from the standard input device, which is the keyboard by default.

- It requires a format string and a list of variables into which the value received from the input device will be stored.

- It is required to put an ampersand (&) before the names of the variables.

  - scanf ("%d", &size) ;

  - scanf ("%c", &nextchar) ;

  - scanf ("%f", &length) ;

  - scanf ("%d %d", &a, &b);

Comments in a C program

➢ Comments in C language are used to provide information about lines of code. It is widely used for documenting code.

➢ There are 2 types of comments in the C language.

- Single Line Comments
- Multi-Line Comments

➢ It is a good practice to begin a program with comment indicating the purpose of the program

➢ Comments should be enclosed within /*   */.

Escape Sequence in C

➢ An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character.

➢ It is composed of two or more characters starting with backslash \.

➢ For example: \n represents new line.

List of Popular Escape Sequences in C

Escape Sequence	Meaning
\n	New Line
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote

What is main()?

- main() forms a crucial part of any c program.
  - main() is a function
    - A function is nothing but a container for a set of statements.
- All statements that belong to main() are enclosed within a pair of braces { } as shown below.

    int main()

    {

    statement 1;

    statement 2;

    statement 3;

    }

main() is also a function

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

        a = 10;
        b = 20;
        c = a + b;
        printf ("\n The sum of %d and %d is %d\n",
                  a,b,c);
    }
```

https://ideone.com/XQas3R   https://ideone.com/t0LFLT

Sample C program 1

Header file includes functions for input/output

```
#include <stdio.h>

main()
    {

        printf ("\n Our first look at a C program \n");

    }
```

Main function is executed when you run the program. (Later we will see how to pass its parameters)

Curly braces within which statements are executed one after another.

Statement for printing the sentence within double quotes ("..."). '\n' denotes end of line.

Our first look at a C program

Sample C program 2

```c
#include <stdio.h>
main()
    {
        int    a, b, c;
        a = 10;
        b = 20;
        c = a + b;
        printf ("\n The sum of %d and %d is %d\n",
                a,b,c);
    }
```

Integers variables declared before their usage.

Control character for printing value of a in decimal digits.

The sum of 10 and 20 is 30

https://ideone.com/HHGLzd

Desirable Programming Style

- ➢ Clarity
    - ■ The program should be clearly written.
    - ■ It should be easy to follow the program logic.
- ➢ Meaningful variable names
    - ■ Make variable/constant names meaningful to enhance program clarity.
        - ■ 'area' instead of 'a'
        - ■ 'radius' instead of 'r'
- ➢ Program documentation
    - ■ Insert comments in the program to make it easy to understand.
    - ■ Never use too many comments.
- ➢ Program indentation
    - ■ Use proper indentation.
    - ■ Structure of the program should be immediately visible.

Indentation Example #1 :: Bad Style

```c
#include <stdio.h>
#define    PI    3.1415926
/* Compute the area of a circle */
main()
{
float   radius, area;
float   myfunc (float radius);
scanf ("%f", &radius);
area = myfunc (radius);
printf ("\n Area is %f \n", area);
}
```

```c
float   myfunc (float r)
{
float   a;
a = PI * r * r;
return (a);     /* return result */
}
```

Indentation Example #1 :: Good Style

```
#include <stdio.h>
#define   PI   3.1415926
/* Compute the area of a circle */

main()
   {
      float   radius, area;
      float   myfunc (float radius);

      scanf ("%f", &radius);
      area = myfunc (radius);
      printf ("\n Area is %f \n", area);
   }
```

```
float   myfunc (float r)
   {
      float   a;
      a = PI * r * r;
      return (a);      /* return result */
   }
```

Indentation Example #2 :: Bad Style

```c
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
int   a, b, c;
scanf ("%d %d %d", &a, &b, &c);
if ((a>b) && (a>c))   /* Composite condition check */
printf ("\n Largest is %d", a);
else
if (b>c)    /* Simple condition check */
printf ("\n Largest is %d", b);
else
printf ("\n Largest is %d", c);
}
```

Indentation Example #2 :: Good Style

```c
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
  {
    int   a, b, c;
    scanf ("%d %d %d", &a, &b, &c);
    if  ((a>b) && (a>c))                    /* Composite condition check */
       printf ("\n Largest is %d", a);
    else
       if  (b>c)                            /* Simple condition check */
          printf ("\n Largest is %d", b);
       else
          printf ("\n Largest is %d", c);
  }
```

C instruction

➢ I hope before continuing to this lecture you must have written the basic printf() and scanf() programs. If you didn't then I strongly recommend you to do it.

➢ A program is nothing but a set of instruction. The program behaves as per the instructions that we give in it.

➢ Different instructions help us to achieve different task in a program

Type of Instruction

➢ There are basically three types of instruction in C

- **Type declaration instruction** – This instruction is used to declare the type of variables used in a C program.
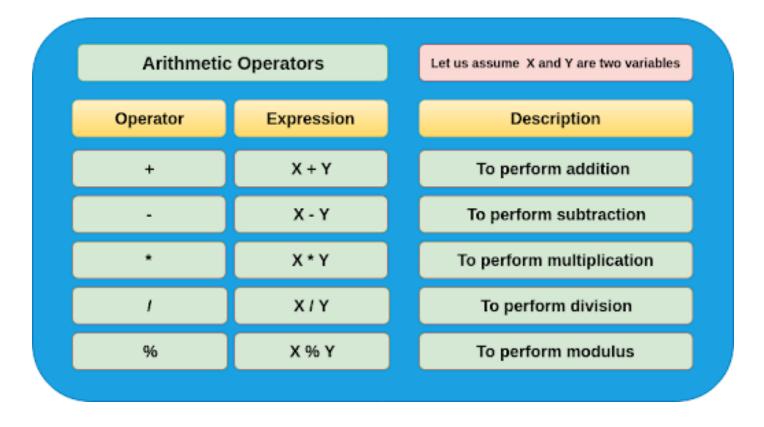
- **Arithmetic instruction** – This instruction is used to perform arithmetic operations on constants and variables.

- **Control instruction** – This instruction is used to control the sequence of execution of various statements in a C program

Type Declaration instruction

➢ This instruction is used to declare the type of variables being used in the program.

➢ Any variable used in the program must be declared before using it in any statement.

➢ The type declaration statement is written at the beginning of main( ) function.

➢ Ex.:

- int bas ;

- float rs, grosssal, P ;

- char name, code ;

Arithmetic operations

Arithmetic Operators		Let us assume X and Y are two variables
Operator	**Expression**	**Description**
+	X + Y	To perform addition
-	X - Y	To perform subtraction
*	X * Y	To perform multiplication
/	X / Y	To perform division
%	X % Y	To perform modulus

➢ There are several subtle variations of the type declaration instruction

- While declaring the type of variable we can also initialize it as shown below.

  - int i = 10, j = 25 ;

  - float a = 1.5, b = 1.99 + 2.4 * 1.44 ;

- The order in which we define the variables is sometimes important sometimes not.

  - For example,

    - int i = 10, j = 25 ;

    - is same as int j = 25, i = 10 ;

    - However, float a = 1.5, b = a + 3.1 ; is alright,

    - but float b = a + 3.1, a = 1.5 is not;

    - This is because here we are trying to use **a** even before defining it.

➢ Another interesting point while declaring variables is

➢ Below instruction will work

- int a,b,c,d;

- a=b=c=10;

➢ However, the following statement would not work.

- int a=b=c=d=10 ;

➢ We are trying to use b (to assign a) before defining it.

Arithmetic Instruction

➢ C arithmetic instruction consists of a variable name on the left hand side of =, and variable names, literals on the right hand side of =

➢ The variables and constants appearing on the right hand side of = are connected by arithmetic operators like +, -, *, /and %

➢ Ex.:

- int ad ;

- float kot, deta, alpha, beta, gamma ;

- ad = 3200 ;

- kot = 0.0056 ;

- **deta = alpha * beta / gamma + 3.2 * 2 / 5**

➢ Here,

- ■ *, /, -, + are the arithmetic operators.

- ■ = is the assignment operator.

- ■ 2, 5 and 3200 are integer literals.

- ■ 3.2 and 0.0056 are real literals.

- ■ ad is an integer variable.

- ■ kot, deta, alpha, beta, gamma are real variables.

➢ The variables and constants together are called '**operands**' that are operated upon by the '**arithmetic operators**' and the result is assigned, using the assignment operator, to the variable on left hand side.

➢ arithmetic statement could be of three types. These are as follows:

- Integer mode arithmetic statement

- Real mode arithmetic statement

- Mixed mode arithmetic statement

➢Integer mode arithmetic statement –

- This is an arithmetic statement in which all operands are either integer variables or integer constants. Ex.:

  - int i, king, issac, noteit ;

  - i = i + 1 ;

  - king = issac * 234 + noteit - 7689 ;

➢ Real mode arithmetic statement –

- This is an arithmetic statement in which all operands are either real constants or real variables. Ex.:

    - float qbee, antink, si, prin, anoy, roi ;

    - qbee = antink + 23.123 / 4.5 * 0.3442 ;

    - si = prin * anoy * roi / 100.0 ;

➢ Mixed mode arithmetic statement –

- This is an arithmetic statement in which some of the operands are integers and some of the operands are real. Ex.:

    - float si, prin ; float anoy, roi, avg ;

    - int a, b ; float c, num ;

    - si = prin * anoy * roi / 100.0 ;

    - avg = (a + b + c + num) / 4 ;

➢ It is very important to understand how the execution of an arithmetic statement takes place.

➢ Firstly, the right hand side is evaluated using literals and the numerical values stored in the variable names. This value is then assigned to the variable on the left-hand side.

➢ Though Arithmetic instructions look simple to use one often commits mistakes in writing them.

➢ Let us take a closer look at these statements. Note the these points carefully.

➢ C allows only one variable on left-hand side of =. That is, **z = k * l** is legal, whereas **k * l = z** is illegal.

➢ In addition to the division operator, C also provides a modular division operator.

  ▪ This operator returns the remainder on dividing one integer with another.

    ▪ Thus the expression 10 / 2 yields 5, whereas, 10 % 2 yields 0.

  ▪ Note that the modulus operator (%) cannot be applied on a float.

  ▪ Also note that on using % the sign of the remainder is always same as the sign of the numerator.

    ▪ Thus –5 % 2 yields –1, whereas, 5 % -2 yields 1.

➢ No operator is assumed to be present. It must be written explicitly.

➢ In the following example, the multiplication operator after b must be explicitly written.

- a = c.d.b(xy)   usual arithmetic statement

- b = c * d * b * (x * y)   C statement

➢ Unlike other high level languages, there is no operator for performing exponentiation operation. Thus following statements are invalid.

- a = 3 ** 2 ;

- b = 3 ^ 2 ;

➢ If we want to do the exponentiation we can get it done this way:

```
#include <stdio.h>
#include <math.h>
main( )
{
int a ;
a = pow ( 3, 2 ) ;
printf ( "%d", a ) ;
return 0;
}
```

➢ Here pow( ) function is a standard library function.

➢ It is being used to raise 3 to the power of 2.

➢ #include is a preprocessor directive.

➢ It is being used here to ensure that the pow( ) function works correctly.

➢ We would learn more about standard library functions

Integer and Float Conversions

➤ In order to effectively develop C programs, it will be necessary to understand the rules that are used for the implicit conversion of floating point and integer values in C.

➤ These are mentioned below

- An arithmetic operation between an integer and integer always yields an integer result.

- An operation between a real and real always yields a real result.

- An operation between an integer and real always yields a real result.

- In this operation the integer is first promoted to a real and then the operation is performed. Hence the result is real.

Operation	Result	Operation	Result
5 / 2	2	2 / 5	0
5.0 / 2	2.5	2.0 / 5	0.4
5 / 2.0	2.5	2 / 5.0	0.4
5.0 / 2.0	2.5	2.0 / 5.0	0.4

Type Conversion in Assignments

➢ It may so happen that the type of the expression and the type of the variable on the left-hand side of the assignment operator may not be same.

➢ In such a case the value of the expression is promoted or demoted depending on the type of the variable on left-hand side of =.

➢ Consider the following assignment statements.

- int i ;

- float b ;

- i = 3.5 ;

- b = 30 ;

➢ Here in the first assignment statement though the expression's value is a float (3.5) it cannot be stored in i since it is an int.

➢ In such a case the float is demoted to an int and then its value is stored. Hence what gets stored in i is 3.

➢ Exactly opposite happens in the next statement. Here, 30 is promoted to 30.000000 and then stored in b, since b being a float variable cannot hold anything except a float value.

- if a complex expression occurs, still the same rules apply.
- For example, consider the following program fragment.

  - float a, b, c ;

  - int s ;

  - s = a * b * c / 100 + 32 / 4 - 3 * 1.1 ;

- Here, in the assignment statement some operands are ints whereas others are floats.
- As we know, during evaluation of the expression the ints would be promoted to floats and the result of the expression would be a float.
- But when this float value is assigned to s it is again demoted to an int and then stored in s.

➤ Observe the results of the arithmetic statements shown in below. It has been assumed that **k** is an integer variable and **a** is a real variable.
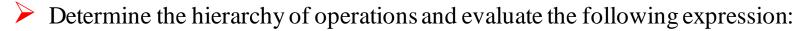
Arithmetic Instruction	Result	Arithmetic Instruction	Result
k = 2 / 9	0	a = 2 / 9	0.0
k = 2.0 / 9	0	a = 2.0 / 9	0.2222
k = 2 / 9.0	0	a = 2 / 9.0	0.2222
k = 2.0 / 9.0	0	a = 2.0 / 9.0	0.2222
k = 9 / 2	4	a = 9 / 2	4.0
k = 9.0 / 2	4	a = 9.0 / 2	4.5
k = 9 / 2.0	4	a = 9 / 2.0	4.5
k = 9.0 / 2.0	4	a = 9.0 / 2.0	4.5

Hierarchy of Operations

➢ While executing an arithmetic statement, which has two or more operators, we may have some problems as to how exactly does it get executed.

➢ For example, does the expression 2 * x - 3 * y correspond to (2x)-(3y) or to 2(x-3y)?

➢ Similarly, does A / B + C correspond to A / (B + C) or to (A / B) + C?

➢ To answer these questions satisfactorily one has to understand the 'hierarchy' of operations.

➢ The priority or precedence in which the operations in an arithmetic statement are performed is called the hierarchy of operations

Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

➢ Within parentheses the same hierarchy as mentioned in figure is operative.

➢ If there are more than one set of parentheses, the operations within the innermost parentheses would be performed first, followed by the operations within the second innermost pair and so on

➢ Determine the hierarchy of operations and evaluate the following expression:

$$i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8$$

Stepwise evaluation of this expression is shown below:

$i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8$	
$i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8$	operation: *
$i = 1 + 4 / 4 + 8 - 2 + 5 / 8$	operation: /
$i = 1 + 1 + 8 - 2 + 5 / 8$	operation: /
$i = 1 + 1 + 8 - 2 + 0$	operation: /
$i = 2 + 8 - 2 + 0$	operation: +
$i = 10 - 2 + 0$	operation: +
$i = 8 + 0$	operation : -
$i = 8$	operation: +

➢ Determine the hierarchy of operations and evaluate the following expression:

$$kk = 3 / 2 * 4 + 3 / 8 + 3$$

Stepwise evaluation of this expression is shown below:

kk = 3 / 2 * 4 + 3 / 8 + 3

kk = 1 * 4 + 3 / 8 + 3   operation: /

kk = 4 + 3 / 8 + 3   operation: *

kk = 4 + 0 + 3   operation: /

kk = 4 + 3   operation: +

kk = 7   operation: +

➢ So far we have seen how the computer evaluates an arithmetic statement written in C.

➢ But our knowledge would be incomplete unless we know how to convert a general arithmetic statement to a C statement.

➢ C can handle any complex expression with ease.

Algebric Expression	C Expression
$a \times b - c \times d$	a * b – c * d
$(m + n)(a + b)$	(m + n) * (a + b)
$3x2 + 2x + 5$	3 * x * x + 2 * x + 5
$\dfrac{a + b + c}{d + e}$	(a + b + c) / (d + e)
$\left[\dfrac{2BY}{d+1} - \dfrac{x}{3(z+y)} \right]$	2 * b * y / (d + 1) – x / 3 * (z + y)

Associativity of Operators

➢ When an expression contains two operators of equal priority the tie between them is settled using the associativity of the operators.

➢ Associativity can be of two types—Left to Right or Right to Left.

➢ Lets understand this with the help of example. Consider the expression: a = 3 / 2 * 5 ;

➢ Here there is a tie between operators of same priority, that is between / and *. This tie is settled using the associativity of / and *. But both enjoy Left to Right associativity. Therefore firstly / operation is done followed by *.