LECTURE NOTES

ON

Programming in C language

Course code: DSE-IV
(6th Semester)

By

Prof. Indubaran Mandal

Department of Mathematics

Bidhan Chandra college , Asansol-04

Content

Module – I :

Introduction to C language :
Character Set , Variables and Identifiers, keywords , Built-in Data types , Variable definition ,Arithmetic Operators and Expression ,Precedence of Operators ,Constants and Literals ,Simple Assignment Statement, Basic Input/ Output statements ,Simple C programs.

Lecture-1:

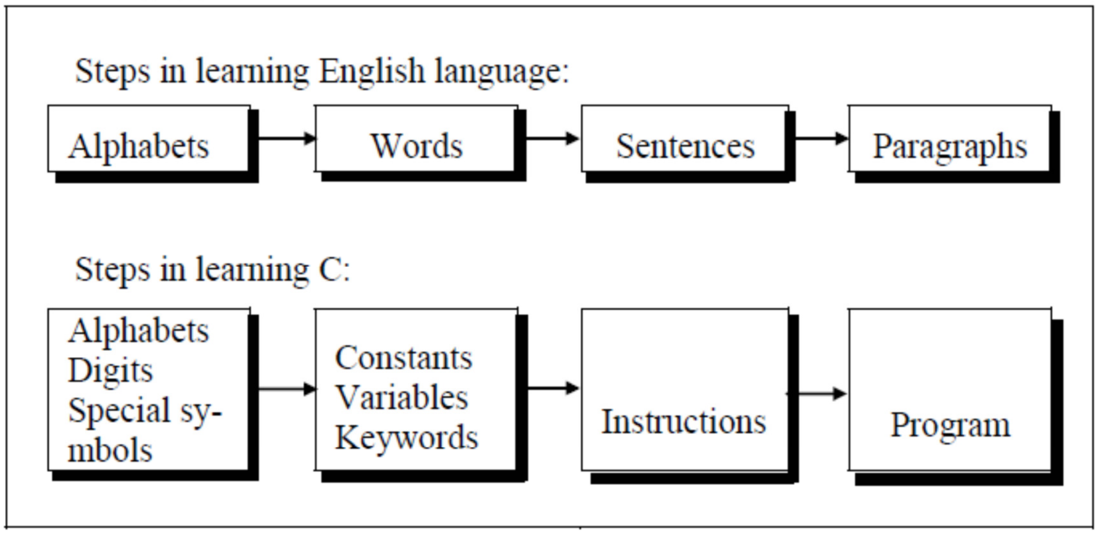
Introduction to C:

C evolved from a language called B, written by Ken Thompson at Bell Labs in 1970. Ken used B to write one of the first implementations of UNIX. B in turn was a descendant of the language BCPL (developed at Cambridge (UK) in 1967), with most of its instructions removed.

So many instructions were removed in going from BCPL to B, that Dennis Ritchie of Bell Labs put some back in (in 1972), and called the language C. The famous book The C Programming Language was written by Kernighan and Ritchie in 1978, and was the definitive reference book on C for almost a decade.

The original C was still too limiting, and not standardized, and so in 1983 an ANSI committee was established to formalise the language definition.It has taken until now (ten years later) for the **ANSI (American National Standard Institute)** standard to become well accepted and almost universally supported by Compilers.

There is a close analogy between learning English language and learning C language. The classical method of learning English is to first learn the alphabets used in the language, then learn to combine these alphabets to form words, which in turn are combined to form sentences and sentences are combined to form paragraphs. Learning C is similar and easier. Instead of straight-away learning how to write programs, we must first know what alphabets, numbers and special symbols are used in C, then how using them constants, variables and keywords are constructed, and finally how are these combined to form an instruction. A group of instructions would be combined later on to form a program. So a computer program is just a collection of the instructions necessary to solve a specific problem. The basic operations of a computer system form what is known as the computer's instruction set. And the approach or method that is used to solve the problem is known as an algorithm.


Steps in learning English language:
Alphabets
Words
Sentences
Paragraphs
Steps in learning C:
Alphabets Digits Special sy-mbols
Constants Variables Keywords
Instructions
Program

Structure of a C program:

1. Comment line

2. Include header file section

3. Global declaration section

4. main( )

```
{
      Declaration part;

      Executable part;
}
```

User-defined function

```
{
        Statements;
}
```

Comment line:

It indicates the purpose of the program. It is represented as

/*…………………………….*/

Comment line is used for increasing the readability of the program. It is useful in explaining the program and generally used for documentation. It is enclosed within the decimeters. Comment line can be single or multiple line but should not be nested. It can be anywhere in the program except inside string constant & character constant.

Include header file section :

C program depends upon some header files for function definition that are used in program. Each header file by default is extended with .h. The header file should be included using # include directive as given here.

#include<stdio.h> tells the compiler to include information about the standard input/output library. It is also used in symbolic constant such as #define PI 3.14(value). The stdio.h (standard input output header file) contains definition & declaration of system defined function such as printf( ), scanf( ), pow( ) etc. Generally printf() function used to display and scanf() function used to read value.

Global declaration section:

This section declares some variables that are used in more than one function. These variables are known as global variables. This section must be declared outside of all the functions.

Main function :

Every program written in C language must contain main ( ) function. The function main( ) is a starting point of every C program. The execution of the program always begins with the function main ( ).

It is the user defined function and every function has one main() function from where actually program is started and it is encloses within the pair of curly braces. The main( ) function can be anywhere in the program but in general practice it is placed in the first position.

Syntax :

```
main( )

{
   ……..

   ……..

   ……..

}
```

The program execution start with opening braces and end with closing brace. And in between the two braces declaration part as well as executable part is mentioned. And at the end of each line, the semi-colon is given which indicates statement termination.

The main function

The main is a part of every program C permits different form of main statement . The following forms are allowed :

- main ( )
- int main ( )
- void main ( )
- main (void)
- void main (void)
- int main (void)

The empty pair of parentheses indicates that the function has no argument . This may be explicitly indicated by using the keyword void inside the parentheses . We may also specify the keyword int or void before the word main . The keyword void means that the function does not return any information to the operating system and int means that the function returns an integer value to the operating system . When int is specified , the last statement in the program must be " return 0" . For the sake of simplicity we use the first form of main in our programs.

Declaration part:

The declaration part declares the entire variables that are used in executable part. The initializations of variables are also done in this section. Initialization means providing initial value to the variables.

Executable part:

This part contains the statements following the declaration of the variables. This part contains a set of statements or a single statement. These statements are enclosed between the braces.

User-defined function:

The functions defined by the user are called user-defined functions. These functions are generally defined after the main ( ) function.

Example of some simple C-Program:

1)
```
/* First C- Programming */

 #include<stdio.h>

 main( )

      {

          Printf(" Welcome to C- Programming.\n");

       }
```

Output: Welcome to C-Programming.

2)
```
/* C program to find sum of two integer */

  #inlcude<stdio.h>

   main( )

      {
         int a ,b, c;

         printf(" Enter any two integer \n");

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

         c= a+b;

         printf(" The sum is = % d",c);
```

```
            }
```

Output:

```
                Enter any two integer

                        4 , 8

                The sum is = 12
```

3) /* C program to find the circumference of a circle */

```
    #include<stdio.h>

    main( )

      {

          float r , cir ;

          printf (" Enter the value of the radius \n");

          scanf (" %f ", & r);

           cir = 2* 3.14 * r ;

           printf (" Circumference of the circle = %f " , cir);

        }
```

Output:

```
             Enter the value of the radius

                         2.5

             Circumference of the circle = 15.70
```

Steps for executing the program:

Creation of program:

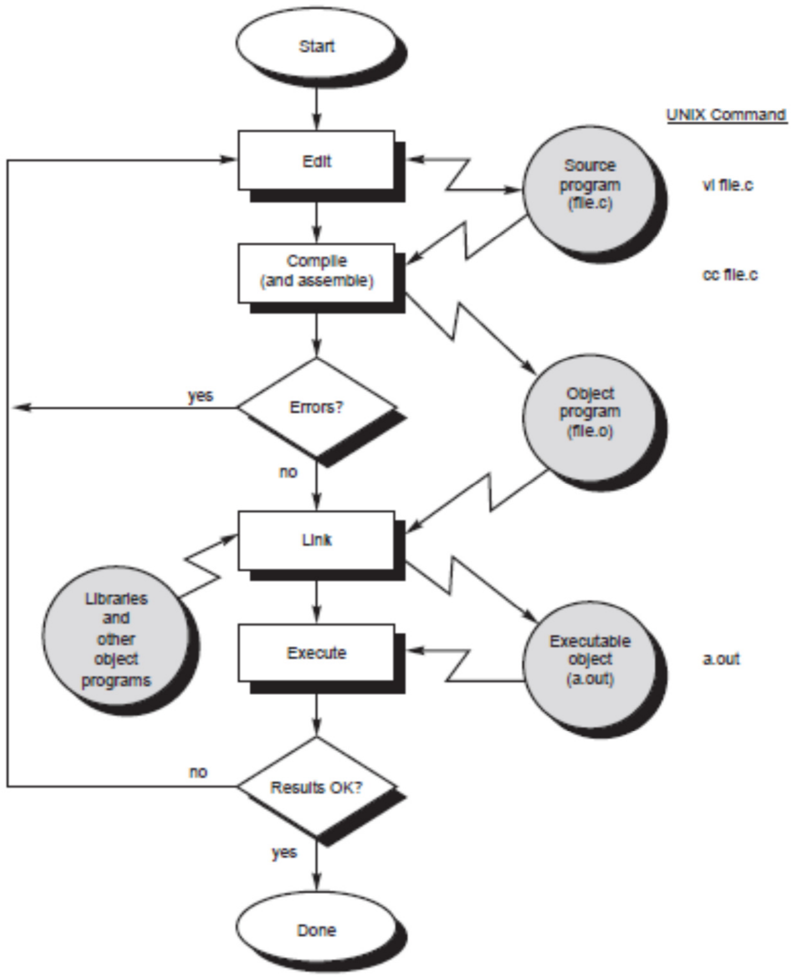
Programs should be written in C editor. The file name does not necessarily include extension C. The default extension is C.

Compilation of a program:

The source program statements should be translated into object programs which is suitable for execution by the computer. The translation is done after correcting each statement. If there is no error, compilation proceeds and translated program are stored in another file with the same file name with extension ".obj".

Execution of the program:

After the compilation the executable object code will be loaded in the computers main memory and the program is executed.



Example of a simple C program:

```
/* C-Program to find the area of a circle */

#include<stdio.h>

#include<math.h>

 main( )

     {
        float r , a ;

        printf (" Enter the radius of the circle \n");

        scanf ( "%f  ", & r);

        a= 3.14*r*r ;

        printf ("Area = % f ", a);

      }
```

Output:

```
            Enter the radius of the circle

                        2.5

             Area = 19.625
```

Character Set:

A character denotes any alphabet, digit or special symbol used to represent information. Valid alphabets, numbers and special symbols allowed in C are

Letters	**Digits**	**White Spaces**
Capital A to Z	All digits from 0 to 9	Blank space
Small a to z		Horizontal tab
		Vertical tab
		New line
		Form feed

Special Characters:

,	Comma	&	Ampersand
.	Dot	^	Caret
;	Semicolon	*	Asterisk
:	Colon	-	Minus
‘	Apostrophe	+	Plus
“	Quotation mark	<	Less than
!	Exclamation mark	>	Greater than
|	Vertical bar	( )	Parenthesis left/right
/	Slash	{ }	Braces left/right
\	Back Slash	[ ]	Braces left/right
~	Tilde	%	Percent
_	Underscore	#	Hash
?	Question mark	=	Equal to
$	Dollar	@	At the rate

Delimiters :

Delimiters	Use
: Colon	Useful for label
; Semicolon	Terminates the statement
( ) Parenthesis	Used in expression and function
[ ] Square Bracket	Used for array declaration
{ } Curly Brace	Scope of the statement
# hash	Pre-processor directive
, Comma	Variable separator

Identifires :

Identifiers are user defined word used to name of entities like variables, arrays, functions, structures etc. Rules for naming identifiers are:

1) Name should only consists of alphabets (both upper and lower case), digits and underscore (_) sign.

2) First characters should be alphabet or underscore .

3) Name should not be a keyword .

4) Since C is a case sensitive, the upper case and lower case considered differently, for example code, Code, CODE etc. are different identifiers.

5) Identifiers are generally given in some meaningful name such as value, net_salary, age, data etc. An identifier name may be long, some implementation recognizes only first eight characters, most recognize 31 characters. ANSI standard compiler recognize 31 characters. Some valid identifiers are area, average, x12 , name_of_place etc and some invalid identifiers are 5cb, int, res#, avg no etc.

Keywords:

There are certain words reserved for doing specific task, these words are known as reserved word or keywords. These words are predefined and always written in lower case or small letter. These keywords can't be used as a variable name as it assigned with fixed meaning. Some examples are-

Auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	void	signed
default	goto	sizeof	volatile
do	if	static	while

Data type :

Data types refer to an extensive system used for declaring variables or functions of different types before its use. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted. The value of a variable can be changed any time.

C has the following 4 types of **data types –**

basic built-in data types: int, float, double, char

Enumeration data type: enum

Derived data type: pointer, array, structure, union

Void data type: void

A variable declared to be of type int can be used to contain integral values only—that is, values that do not contain decimal places. A variable declared to be of type float can be used for storing floating- point numbers (values containing decimal places). The double type is the same as type float, only with roughly twice the precision. The char data type can be used to store a single character, such as the letter a, the digit character 6, or a semicolon similarly A variable declared char can only store character type value.

There are two types of type **qualifier** in c

Size qualifier: short, long

Sign qualifier: signed, unsigned

When the qualifier unsigned is used the number is always positive, and when signed is used number may be positive or negative. If the sign qualifier is not mentioned, then by default sign qualifier is assumed. The range of values for signed data types is less than that of unsigned data type. Because in signed type, the left most bit is used to represent sign, while in unsigned type this bit is also used to represent the value. The size and range of the different data types on a 16 bit machine is given below:

Data type	Range	Format Specifiers
char or signed char	-128 to 127	%c
Unsigned char	0 to 255	%c
Short int or signed short int	-128 to 127	%i
Unsigned short int	0 to 255	%i
int or signed int	-32768 to 32767	%d
Unsigned int	0 to 65535	%u
long int or signed long int	-2147483648 to 2147483647	%ld
Unsigned long int	0 to 4294967295	%lu
float	-3.4E-38 to 3.4E+38	%f
double	1.7E-308 to 1.7E+308	%lf
Long double	3.4E-4932 to 1.1E+4932	%lf

For a for *w* bit machine range are as following :-

Integer : -2^{w-1} to $+2^{(w-1)}-1$

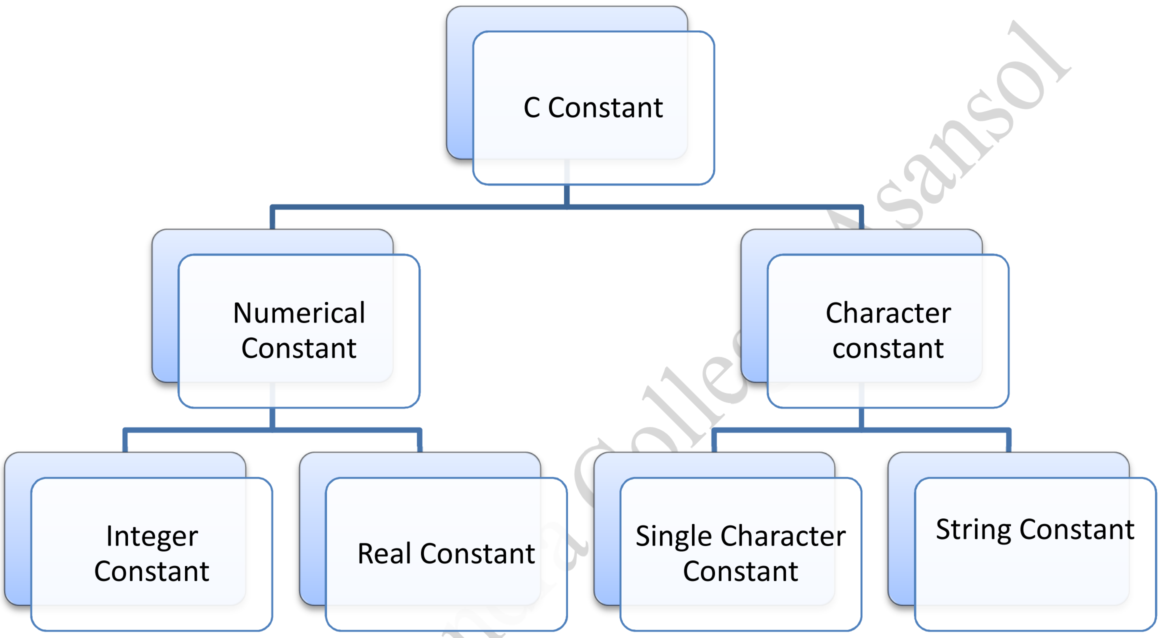
Unsigned integer : 0 to 2^{w-1}

Short integer : $-2^{\frac{w}{2}-1}$ to $+2^{\frac{w}{2}-1}-1$

Long integer : -2^{2w-1} to $+2^{(2w-1)}-1$

Constant :

Constant is a any value that cannot be changed during program execution. In C, any number, single character, or character string is known as a constant. A constant is an entity that doesn't change whereas a variable is an entity that may change. For example, the number 50 represents a constant integer value. The character string "Programming in C is fun.\n" is an example of a constant character string. C constants can be divided into two major categories-



Numerical Constant:

Numeric constant consists of digits. It may be positive or negative but by default sign is always positive. No comma or space is allowed within the numeric constant and it must have at least 1 digit. It is categorized into integer constant and real constant.

Integer Constant: These are the sequence of numbers from 0 to 9 without decimal points or fractional part or any other symbols. Types of integer constants are:

Decimal integer (base 10): It consists of 0,1,2,….. ,9 . In decimal constant first digit should not be zero.

Examples: 15 ,+ 245 , - 14256 etc are valid and
5,784, 39,98, 2-5,09 etc are not valid integer constants.

Octal integer (base 8) : An octal integer constant consists of digits 0,1,…,7 with 1st digit 0 to indicate that it is an octal integer.

Examples: 0, 01, 0756, 032, etc are valid and
32, 083, 07.6 etc are not valid octal integers.

Hexadecimal Constant (base 16): A hexadecimal integer constant consists of 0,1,…,9,A, B, C, D, E, F. Its first two digit should be 0x/ 0X .

Examples: 0x7AA2, 0xAB, are valid and
0x8.3, 0AF2, 0xG etc are not valid hexadecimal constants.

By default type of integer constant is integer but if the value of integer constant is exceeds range then value represented by integer type is taken to be unsigned integer or long integer. It can also be explicitly mention integer and unsigned integer type by suffix l/L and u/U.

Real Constant or floating constant:

It is a decimal number (ie: base 10) with a decimal point or an exponent or both. To construct real constant we must **follow these rule** : real constant 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 .

Examples : 32.65, 0.654, 0.2E-3, 2.65E10 etc.

To express small/large real constant exponent(scientific) form is used where number is written in mantissa and exponent form separated by e/E. Exponent can be positive or negative integer but mantissa can be real/integer type, for example 3.6*105=3.6e+5. By default type of floating point constant is double, it can also be explicitly defined it by suffix of f/F.

Character Constant :

Single Character constant : A character constant is a single character. Characters are also represented with a single digit or a single special symbol or white space enclosed within a pair of single quote marks.

Examples: ‘ A’ , ‘0’ , ‘.’ , ‘ ’ etc .

String Constant : String constants are sequence of characters enclosed within double quote marks.

Examples : “ Hello” , “1542” , “ C proram” etc

Escape sequences:

An escape sequence is used to express non printing character like a new line, tab etc. it begin with the backslash (\ ) followed by letter like a, n, b, t, v, r, etc. the commonly used escape sequence are –

\a : for bell sound	\n : new line	\0 : null
\b : backspace	\f : form feed	\? : question mark
\f : horizontal tab	\r : carriage return	\’ : single quote
\v : vertical tab	\” : quotation mark	\\ : to print backslash

Variables :

Variable is a data name which is used to store some data value or symbolic names for storing program computations and results. The value of the variable can be change during the execution. **The rule for naming the variables is same as the naming identifier**. Before used in the program it must be declared. Declaration of variables specify its name, data types and range of the value that variables can store depends upon its data types.

Examples :

a , r , area , avg_marks , b12 these are some valid variable names and

3n ,1+b ,goto ,printf ,a@b these are some invalid variable names.

Variable initialization :

When we assign any initial value to variable during the declaration, is called initialization of variables. When variable is declared but contain undefined value then it is called garbage value. The variable is initialized with the assignment operator (means with '=' operator) such as

Data type variable name=constant;

Example : int a = 20 ;

or int a;
a = 20;

DECLARATIONS :

This is for specifying data type. All the variables, functions etc must be declared before they are used. A declaration tells the compiler the name and type of a variable you'll be using in your program. In its simplest form, a declaration consists of the type, the name of the variable, and a terminating semicolon:

Example : int a ,b, c ;

float r, f ;

char c ;

According to the size of the number we can declare it as short or long or unsigned or double or long double as shown below –

Short int a , area ;

long int b , sum ;

Unsigned int sum ,area ;

double f , side;

long double f, side;

String variables :

String variables can be declared in the following format –

Char var1[n] ,var2[m] ;

where n ,m etc are integer constants which represent the maximum size of the string. A string is stored as an array of characters. Suppose we want to store " pepsi" to the variable drink , it is done in the following manner –

```
drink[0]  = 'p'
drink[1]  = 'e'
drink[2]  = 'p'
drink[3]  = 's'
drink[4]  = 'i'
drink[5]  = '\0'
```

	[0]	[1]	[2]	[3]	[4]	[5]
drink	p	e	p	s	i	\0

The following are some examples –

```
char drink [5];
char drink[10];
char drink[  ] = "pepsi" ;
```

The number in the square bracket represents the maximum number of characters allowed in each string variables. In the third example the string variable drink assigned a word of 5 letters .But no number is mentioned in the square bracket. In such cases the variables is assigned the of the string +1 .

For example consider the declaration :

```
char  drink [  ] = "pepsi"
```

Here the length of the word is 5 character . Automatically 5+1=6 is assigned as the length of the string variable. The sixth character is a null character ,which remains invisible . This character marks the end of the string . this end-mark is utilised in several string functions and operations.

You may wonder why variables must be declared before use. There are two reasons:

1. It makes things somewhat easier on the compiler; it knows right away what kind of storage to allocate and what code to emit to store and manipulate each variable; it doesn't have to try to intuit the programmer's intentions.

2. It forces a bit of useful discipline on the programmer: you cannot introduce variables willy-nilly; you must think about them enough to pick appropriate types for them. (The compiler's error messages to you, telling you that you apparently forgot to declare a variable, are as often helpful as they are a nuisance: they're helpful when they tell you that you misspelled a variable, or forgot to think about exactly how you were going to use it.)

Expression :

An expression is a combination of variables, constants, operators and function call. It can be arithmetic, logical and relational for example:-

```
int z= x+y        // arithmatic expression

 a>b              //relational

 a= =b            // logical

func(a, b)        // function call
```

Expressions consisting entirely of constant values are called constant expressions. So, the expression 121 + 17 – 110 is a constant expression because each of the terms of the expression is a constant value.

Statements :

Statements are the ``steps'' of a program. Most statements compute and assign values or call functions, but we will eventually meet several other kinds of statements as well. By default, statements are executed in sequence, one after another .A statement causes the compiler to carry out some action. There are 3 different types of statements – expression statements ,compound statements and control statements. Every statement ends with a semicolon.

Examples : 1) c = a + b ;   (expression statement)

```
2) {

      a = 5;

      b = 7;                              (Compound expression)

      c = a + b ;

   }


3)  if (a < b )

     {

       Printf (" a is less than b \n");          (control statement)

     }
```

Operators

Arithmetic Operator :

Symbol	Operator
+	Addition
-	Subtraction
*	Multiplication
\	Divison
%	Modulo (Remainder)

If a and b are two variables the following table illustrate the arithmetic operations :-

Expression	Meaning
a + b	a and b are added
a - b	b subtracted from a
a * b	a is multiplied with b
a \ b	a is divided by b
a % b	remainder of a/b

The modulus operator ' %' can not be applied with floating numbers , valid only for integer numbers. Rules for arithmetic expression are –

Rule -1 : More than one arithmetic operator should not be used successively .

Rule-2 : Every arithmetic operator must be explicitly given by its symbol.

Suppose we want to multiply a with –b . Then the expression a * - b is not valid ,we must write a * (-b) . Consider the expression $\frac{(a+b)(c+d)}{(a-b)(c-d)}$. This can be written as ((a+b)*(c+d))\((a-b)*(c-d)) .

Integer division :

Division of an integer quantity by another is referred to integer division. This operation results in truncation. i.e.When applied to integers, the division operator / discards any remainder, so 1 / 2 is 0 and 7 / 4 is 1. But when either operand is a floating-point quantity (type float or double), the division operator yields a floating-point result, with a potentially nonzero fractional part. So 1 / 2.0 is 0.5, and 7.0 / 4.0 is 1.75.

Example :
```
int a, b, c;
a=5;
b=2;
c=a/b;
```

Here the value of c will be 2

Actual value will be resulted only if a or b or a and b are declared floating type. The value of an arithmetic expression can be converted to different data type by the statement (data type) expression.

Example :

```
int a, b;
float c;
a=5;b=2;
c=(float) a/b
```

Here the value will be , c=2.5 .

Order of Precedence :

The order of precedence of these operators is % / * + - . it can be overruled by parenthesis. Multiplication, division, and modulus all have higher precedence than addition and subtraction. The term ``precedence'' refers to how ``tightly'' operators bind to their operands (that is, to the things they operate on). In mathematics, multiplication has higher precedence than addition, so 1 + 2 * 3 is 7, not 9. In other words, 1 + 2 * 3 is equivalent to 1 + (2 * 3). C is the same way.

Unary Operator :

Unary operators are the operators which operate on single operand. The arithmetic operators eg. + , * etc which operate on two operands are binary operators . Three important unary operators are **increment operator** (++) , **decrement operators** (--) and logical not operator (!) .

++ is the operator used to increment a variable. The statement ++a is equivalent to a = a+1 . Similarly the statement --b is equivalent to b = b-1 . Unary operator can preceed or follow the operand. The position of the unary operator decides the value alternation. Consider the following two cases :-

Case -1 : Consider the following statement

```
a = 1 ;
b = a++ ;
```

The statement b = a++ causes the value to be assigned to b and then increments the value of a by 1. So , the final value of b is 1 and value of a is 2 . So b = a++ is equivalent to

```
b = a;
a = a+1;
```

Case- 2 : Consider the following statement

```
a = 1;
b = ++ a ;
```

In this case , ++ preceeds the variable a in the second statement . In this case the value of a first incremented by 1 and then assigned to b. So, the final value of a as well as b are 2 . So the statement b = ++ a is equivalent to

```
a = a+1 ;
b = a ;
```

Logical Not (!) operator: The operator (!) changes the value of logical expression .

x	!x
True	False
False	True

For example suppose x = 2 and y = 4 . Here x < y is true , so !(x<y) is false .

Sizeof operator : sizeof is another unary operator . Size of operator is a Unary operator, which gives size of operand in terms of byte that occupied in the memory. An operand may be variable, constant or data type qualifier. Consider the following statement,

```
int  x, y;
y = sizeof (x);
```

The value of y is 2 . The sizeof of an integer type data is 2 , that of float is 4, that of double is 8 and that of char is1.

<u>**Relation Operators :**</u>

The Relation operators used in C program are < , <= , > ,>= , = = (equal to) and != (not equal to) . It is use to compared value of two expressions depending on their relation . Logical expression are formed using relation operators and variables . If the expression is true it will return integer value 1 otherwise integer value zero will be returned. Let a = 1 , b = 2.5 and c = 100 , then

Expression	**Logical value**	**Value returned**
a > 3	Flase	0
(a + b)>30	False	0
b = = c	False	0
c < 150	True	1
b + c > a	True	1

<u>**Logical Operators**</u> : The three logical operators in C are :

Operator	**Symbol**
AND	&&
OR	!!
NOT	!

The **AND** operator returns a true value only when both the operands are true otherwise returns false value .

Let x = 2 , y = 4 . Consider the expression (x < 6) && (y >= 4) . Since here both the operands are true ,so the expression returns an integer value 1 .
Truth table for AND operator :

a	b	a && b
True	True	true
True	False	false
False	True	False
False	False	False

Truth table for **OR** operators :

a	b	a !! b
True	True	True
True	False	True
False	True	True
False	False	False

For example suppose x = 3.5 and y = 7 . Consider the expression (x > 3)!! (y = = 8) . Here the first operand is true but the second operand is false . Since one operand is true , the expression returns an integer value 1 .

<u>**Assignment Operator :**</u>

These operators are used for assigning a value of expression to another identifier . In C language =, + =, - = , * =, /= and %= are assignment operators. For example the following are some examples of assignment operator

```
x = 21 ;
y = z = 3 ;
k = x + y –z – 5.2 ;
```

Note : If a floating point number is assigned to a integer type data variable, the value will be truncated.

```
Example :   float a=5.36;
             int b;
             b=a ;
```

In this case 5 will be stored in b. Similarly if an integer value is a assigned to a float type ,

```
float x=3 ;
```

the value of x stored is 3.0 .

The other assignment operator available in C and their usage the given below –

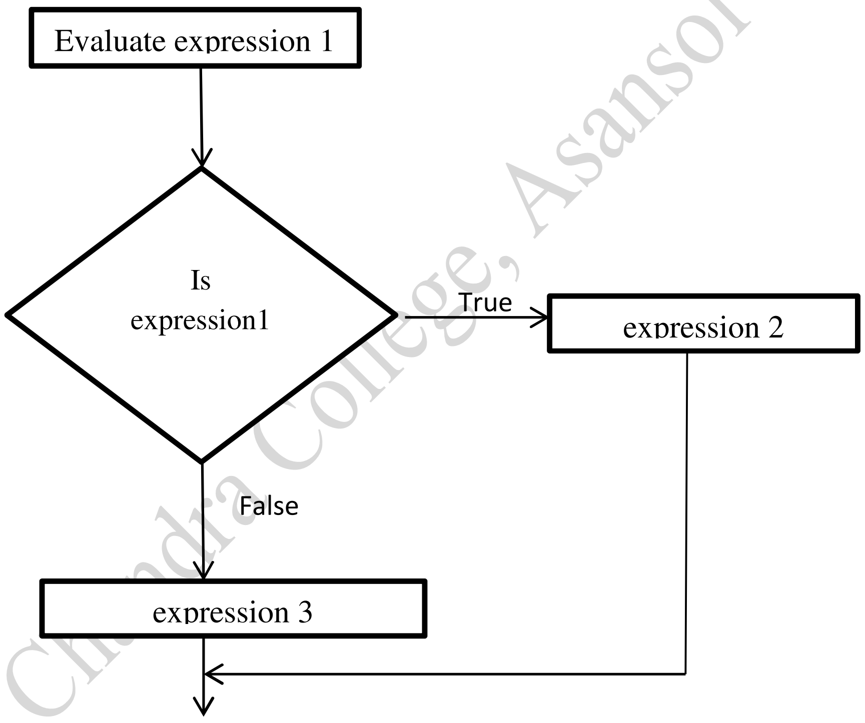
Operator	Example	Meaning
+=	a += 3	a = a + 3
-=	b -= 6	b = b - 6
*=	a *= 2	a= a * 2
\=	b \= 3	b = b \ 3
%=	c %= 5	c = c % 5

Conditional Operator :

The conditional operator used in C language is (? :). The general form for using this operator is as follows :

expression 1 ? expression 2 : expression 3

First the expression 1 is evaluated . If the expression 1 is true ,then expression 2 is evaluated . If the expression 1 is false then the expression 3 is evaluated . This is explained by the flow chart shown below :



For example suppose m = 5 , n = 20 and p = 5.5 . Now consider the conditional expression

(m = = 4) ? n =1 : n = 0 ;

Here m is not equal to 4 . Since the first expression is false , so the 3rd expression is evaluated and the value of n becomes 0 .

There is another form of usage of conditional expression

Variable = (condition) ? value 1 : value 2 ;

In this case the condition evaluated first .If it is true , value 1 is assigned to the variable otherwise value 2 is assigned .

```
Example :           int a ,b ,c ,d ,e ;
                     a = 3 ,b = 5 , c = 8 ;
                    d = (a < b) ? a : b ;
                     e = ( b > c) ? b : c ;
```

Output : d = 3 and e = 8 .

Precedence or Hierarchy of operators :

Each operator in C has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated . There are distinct level of precedence and an operator may belong to one of these levels. The operators at the higher level of precedence are evaluated first.

Hierarchy Table

Level of precedence	**Operator type**	**Operator**
1	Unary operator	++ , -- , ! , sizeof
2	Multiplication, Division, Modulus	* , \ , %
3	Addition , Subtraction	+ , -
4	less than, less than or equals to, greater than ,greater than or equals to	< , <= , > , >=
5	Equals to , Not equals to	== , !=
6	logical operator	&& , !!
7	conditional operator	?:
8	assignment operator	= , += ,-= , *= ,\= ,%=

Examples of some simple C program :

1) **Program to convert temperature from degree centigrade to Fahrenheit.**

```
#include<stdio.h>

 main( )

  {
      float c ,f ;

      printf (“Enter the temperature in centigrade : \n”);
```

```
                scanf (“ %f ”, & c);
                f = (1.8 * c) + 32 ;

                printf ( “ The temperature in Fahrenheit is : %f  \n ”, f);

             }
```

Output :

```
Enter the temperature in centigrade : 32

The temperature in Fahrenheit is : 89.5998
```

2) **Program to find the simple interest.**

```
    #include<stdio.h>

     main ( )

       {

          int p , r , t , si ;

          printf (“ Enter principle ,rate of interest and time to find simple interest \n”);

          scanf (“ %d , %d ,%d ”, &p , &r , &t );

          si = (p*r*t)\100 ;

           printf ( “Simple interest = %d ”,si);

        }
```

Output :

```
Enter principle ,rate of interest and time to find simple interest

           500       5        2


 Simple interest = 50
```
