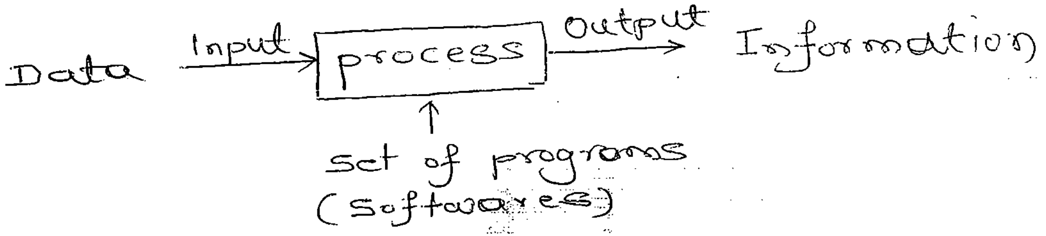
Introduction to Computers & Programming

Computer is advanced electronic computing device. It receives raw data as an input from users to process and generates desired result as an output.



Module-I : PCD by C.S. Kusur

The major four functions of a computer system are as follow.

1) Input: The written computer program receives raw data as an input from end users through the input devices like keyboard.

In C programming, the programmer makes the use of input functions like scanf(), getch(), and gets(), etc to read raw data from keyboard.

Example: To read two integer values.

```
scanf("%d%d", &n1, &n2);
```

2) Process: The raw data will be processed by CPU i.e. the programmer makes the use of operators & expressions to get desired result.

Example: To find addition of two integer values.

```
ans=n1+n2;
```

3) Output: The processed data (information) can be displayed on monitor screen. For this purpose, C programmer makes the use of output functions like printf(), putch(), and puts(), etc

Example: To display result on screen.

```
printf("\n The addition is %d", ans);
```

(cont..)

4) Store: The input and output data can be stored in computers secondary memory like hard disks for future use with suitable file name. For this purpose, the C programmer makes the use of File Handling Functions like fopen(), fscanf(), fprintf(), fgetc(), fputc(), fgetw(), fputw(), etc. (File Management in C.)

Computer Program: The set of instructions (statements or commands) given to a computer system to do specific task is called computer program.

Example: A simple C program to display "Hello World" message on screen.

```
/* A simple C program */
#include <stdio.h>
#include <conio.h>
void main()
{
 clrscr();
 printf("Hello World!");
 getch();
}
```

<u>Shortcut Keys</u>

Save → F2
Compile → Alt + F9
Run → ctrl + F9
close → Alt + F3
Maximize window → F5
Delete line → ctrl + Y
Home and End.

Output:

```
Hello world!.
```

Ex (ii). C program to find addition of two unknown whole numbers.

```
/* To find addition of two nos */
#
#
void main()
→ { int n1, n2, ans;
  clrscr();
  printf("Enter two numbers");
  scanf("%d %d", &n1, &n2);
  ans=n1+n2;
  printf("\n Addition is %d", ans);
  getch();
→ }
```

<u>output:</u>

```
Enter two numbers
 7 3 ↲
Addition is 10
```

③ C program to find addition, substraction, multiplica and division of entered two unknown whole nos.

```
/* To perform arithmetical operations */
#include <stdio.h>
#include <conio.h>
void main()
→{
    int n1, n2, ans1, ans2, ans3, ans4;
    clrscr();
    printf("Enter any two numbers:");
    scanf("%d %d", &n1, &n2);
    ans1= n1+n2;
    ans2= n1-n2;
    ans3= n1*n2;
    ans4= n1/n2;
    printf("\n Addition=%d Substraction=%d
              Multiplication=%d Division=%d", ans1, ans2,
                                               ans3, ans4);
    getch();
→}
```

Output:

```
Enter any two numbers:
 10 5 ↵
Addition=15 Substraction=10 Multiplication=50 Division=2
```

④ C program to find area and circumference of a circle for unknown radius r.

```
/* To find area and circumference of circle */
#include<stdio.h>
#include <conio.h>
void main()
→{ float r, area, cir;
  clrscr();
  printf("Enter radius of circle:");
  scanf("%f", &r);
  area= 3.142*r*r;
  cir= 2*3.142*r;
  printf("\n Area of circle=%f", area);
  printf("\n Circumference of circle=%f", cir);
  getch();
→}
```

(5) C program to calculate area of triangle by reading 3 sides of it.

```
/* To find area of triangle */
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
→{
  float a, b, c, s, area;
  clrscr();
  printf("\n Enter 3 sides of triangle:");
  scanf("%f %f %f", &a, &b, &c);
  s= (a+b+c)/2;
  area= sqrt(S*(S-a)*(S-b)*(S-c));
  printf("\n Area of triangle=%f", area);
  getch();
→}
```

O/P:

```
Enter 3 sides of triangle
1 2 3 ↵
Area of triangle= ______
```

(6) C program to convert temperature reading from Celsius to Fahrenheit.

```
/* To convert temperature reading from ~~C~~ F to ~~F~~ C */
#include <stdio.h>
#include <conio.h>
void main()
→{
  float f, c;
  clrscr();
  printf("\n Enter temperature reading in Celsius");
  scanf("%f", &f);
  f=(9/5)*C+32;
  printf("\n Temperature in Fahrenheit=%f", f);
  getch();
→}
```

To Remember

(1) F to C conversion
$c=(5/9)*(f-32)$

(2) C to F conversion
$f=(9/5)*C+32$

O/P:

```
Enter temperature reading in Celsius
5 ↵
Temperature in Fahrenheit= ______
```

Problem Solving with C:

In order to solve problems by using a computer the programmer needs to follow systematic approach. This approach includes 6 phases. as follow.

(1) Problem Definition

(2) Analysis

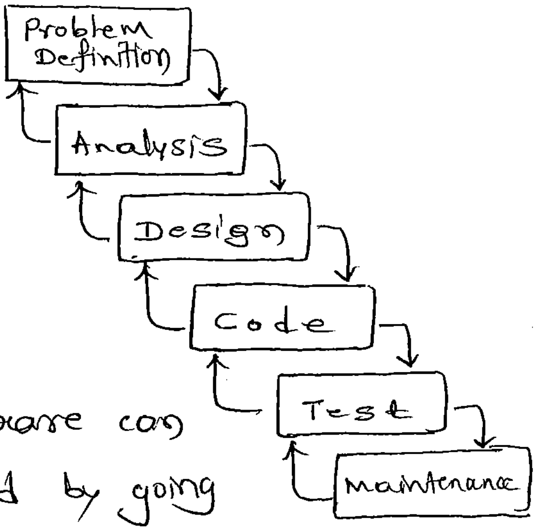
(3) Design

- Algorithm, Flowchart, Pseudocode

(4) Code

(5) Test

(6) Maintenance



The program or software can be designed and developed by going through above listed 6-phases. This cycle can be called as program development life cycle or software development life cycle

(1) Problem Definition: In this phase, the system analyst defines the problem precisely and check for its completeness. The defined problem should be unambiguous and precise.

Ex:(i) Design and develop a C program to find the area of triangle by reading 3 sides of it.

(ii) Design and develop a C program to find roots of quadratic equation for three non-zero co-efficients

(2) Analysis: In this phase, the system analyst looks at different alternatives from a system point of view to solve the problem.

(3) Design: In this phase, the system designers develop the solution to solve a problem by writing an algorithm, drawing a flowchart and writing pseudocode.

Algorithm:- "An algorithm can be defined as a sequence of instructions to be carried out to solve a given problem with finite number of steps in hierarchical order."

"An algorithm can be defined as ~~a~~ unambigua step by step procedure to solve a given problem with its solution to get the desired result"

Examples:

(1) An algorithm to find addition of two unknown numbers

```
step1: start
step2: Read two numbers n1 and n2.
step3: Calculate the sum.
           sum=n1+n2
step4: Display sum
step5: Stop
```

(2) An algorithm to find area ~~of circle.~~ and circumference of circle.

```
step1: start
step2: Read radius of a circle i.e. r.
step3: Calculate area and circumference.
         area= 3.142*r*r;
         cir= 2*3.142*r.
step4: Output area, cir
step5: stop
```

(3) An algorithm to calculate square and cube of a number.

```
step1: start
step2: Read a number n.
step3: Calculate
         sq=n*n, cu=n*n*n
step4: Output sq, cu
step5: stop
```

(4) An algorithm to find largest of two numbers.

```
step1: start
step2: Read two numbers a,b
step3: Assume large=a
step4: If (b>large) Then large=b
step5: Output large
Step6: Stop
```

(*) Algorithm to check a number for even or odd number.

```
step1: Start
step2: Read a number n.
step3: If (n%2==0) Then
          print "Even Number"; Otherwise,
          print "Odd Number"
       End If
step4: Stop
```

(*) Algorithm to generate and display 1 to 10 numbers

```
step1: Start
step2: Initialize the counter i=1
step3: while (i<=10) repeatedly do
          a) print i
          b) increment i by 1
             i=i+1
step4: Stop
```

(*) Algorithm to generate and display 1 to n nos.

```
step1: Start
step2: Read a number n
step3: Initialize the counter i=1
step4: while (i<=n) repeatedly do
          a. print i
          b. increment i by 1
             i=i+1
step5: Stop
```

(*) Algorithm to print odd numbers bet'n 1 to n

(*) " " " even " " " " "

(*) " " find sum of 1 to 10 numbers.

(P.T.O)

Properties of algorithm:-

*) Algorithm should have finite number of steps to get the desired result.

*) Algorithm should be unambiguous

*) Algorithm should produce correct result

*) Every step must be complete and error free

*) It should occupy less computer's memory

*) It should work with high speed.

*) It should not have any uncertainity steps.

Advantages of writing algorithms:

*) Effective communication bet'n analysts.

*) Effective analysis of written algorithm.

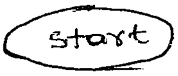
*) Proper documentation for programming.

*) Easy and efficient coding

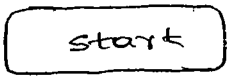
*) Program maintenance & debugging is easier.

Flowchart: Flowchart can be defined as a graphical representation of written algorithm to solve a given problem. It represents the sequence of instructions to be carried out to get the desired result. ~~to draw a flowchart~~ The flowcharts can be drawn by using the following predefined symbols.

*) Oval or Rounded Rectangle:

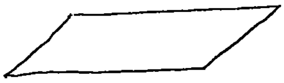


or



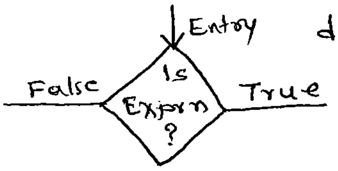
It helps to represent the beginning or end of a program.

(*) parallelogram:- It helps to represent input and output functions of a program

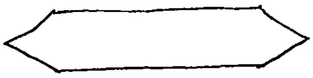


(*) rectangle:- It helps to represent the processing or calculational steps.

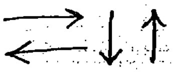
(*) diamond / rhombous:- It helps to represent the decision making steps.



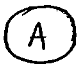
(*) Hexagon :- It helps to represent iterative or repeatative steps of a program.



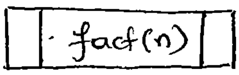
(*) Arrows:- Arrows indicates the direction of progra execution flow between the start and end of a program.



(*) small circle:- It acts as a connector i.e. it represents the continuation of flowchart in the same or next page.

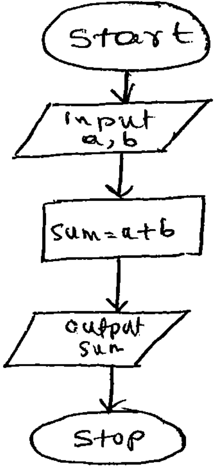


(*) double lined rectangle:- It represents the sub-programs or user defined functions

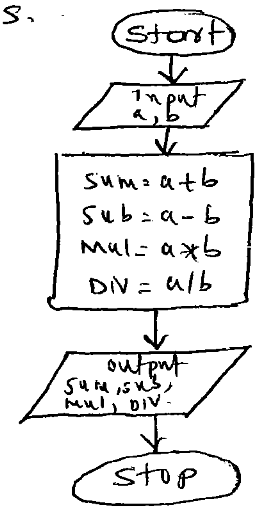


Examples: (*) Draw a flowchart to find addition of two integer values.

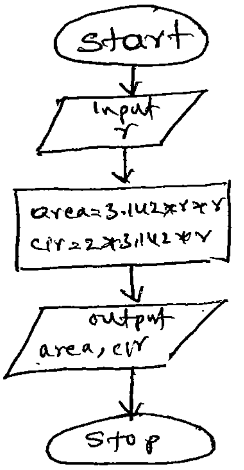
(i)



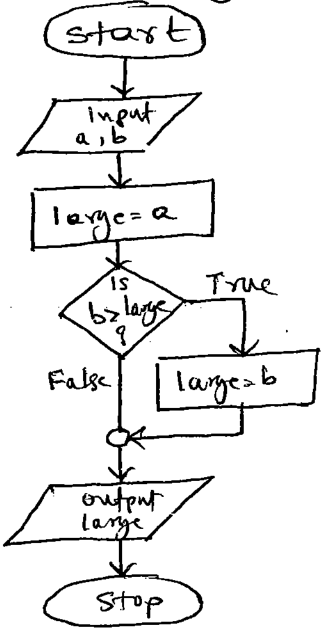
(ii) To find addition, substraction, multiplication & division of 2 nos.



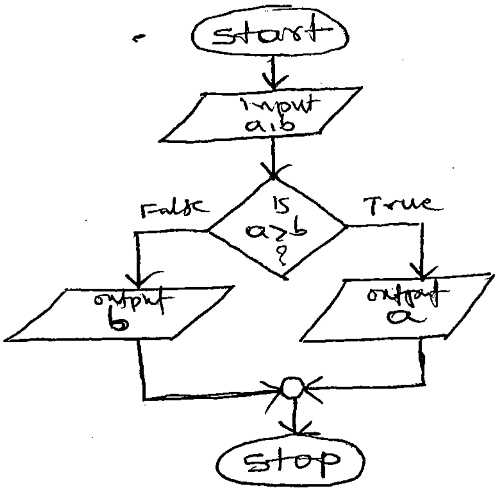
(*) To find area & circumference of a circle.



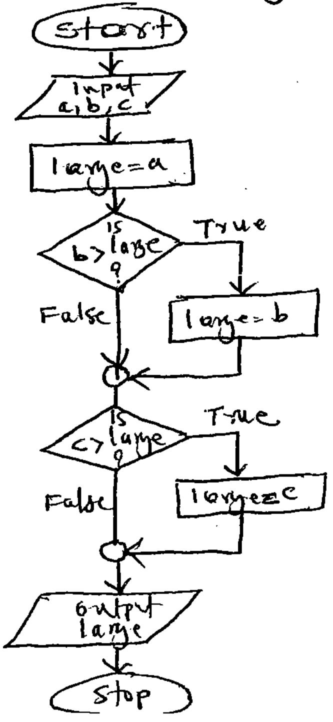
(*) To find largest of two unknown numbers.



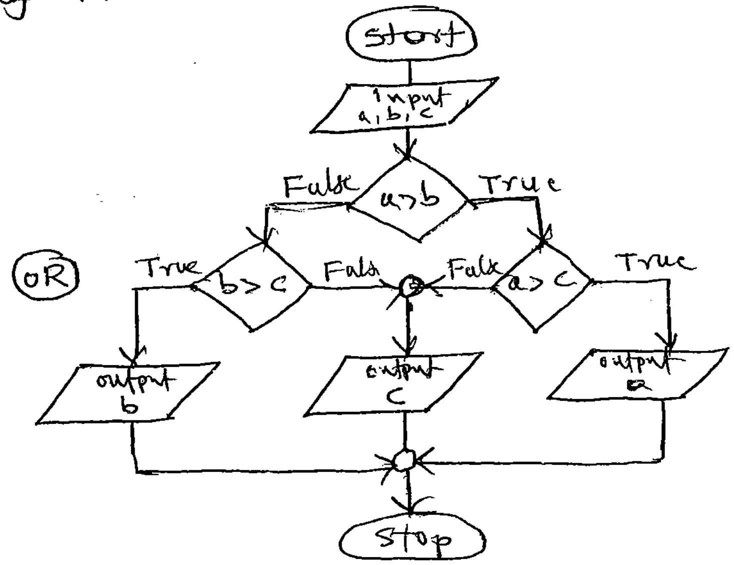
OR



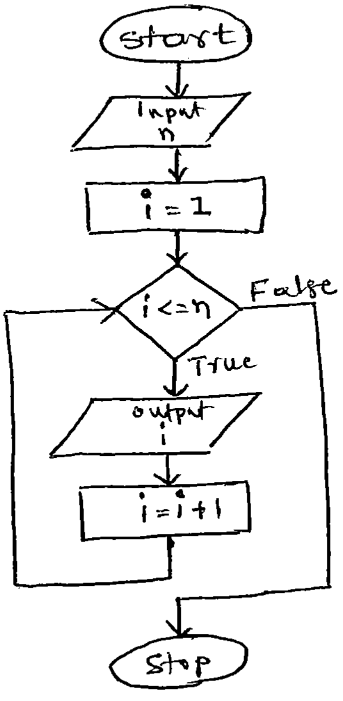
(*) To find largest of three numbers.



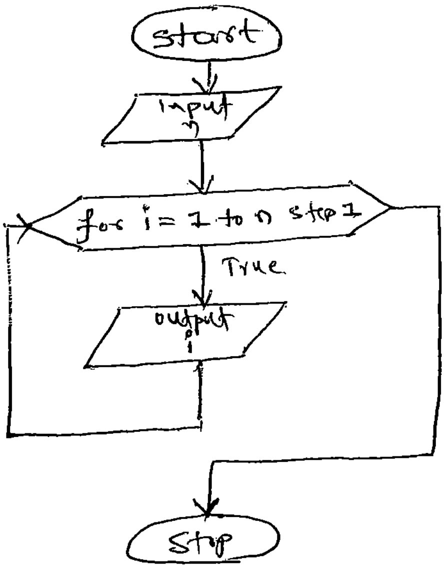
OR



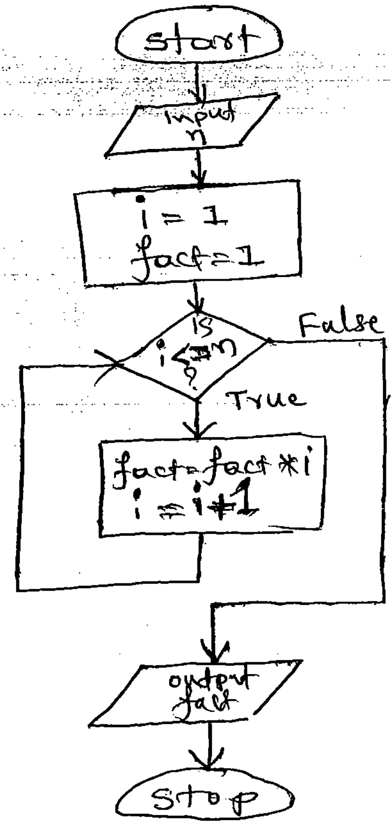
(*) To generate and display 1 to n numbers



or



(*) To find factorial of "n".



(*) To find sum and average of 1 to n numbers.

(*) To check, whether the entered number is PALINDROME or NOT PALINDROME.

(*) Draw a flowchart to find roots of a quadratic equation.

(imp) Pseudocode: The pseudocode can be defined as outline of a source code (program). It is written before writing the program. It can be written by using generic syntax and normal English words like BEGIN, END, DEFINE, INPUT, SET and OUTPUT, etc. It helps the programmer to understand the logic (solution) to solve a given problem to get desired result.

Keywords used in Pseudocode:

BEGIN, END → represents the beginning and end of pseudocode

DEFINE → represents memory allocation steps

INPUT, OUTPUT → represents input and output operations.

SET → represents the calculational steps.

IF... END IF → represents one-way decision making steps.

IF THEN ELSE END IF → represents the two-way decision making steps.

WHILE ... END WHILE, REPEAT... UNTILL, FOR... END FOR → represents repeatative steps.

Examples:-

(1) pseudocode to find addition of two integer values.

```
BEGIN
     DEFINE: integer n1, n2, ans
     INPUT: integer n1, n2
     SET  : ans = n1+n2
     OUTPUT : ans
END
```

(2) Pseudocode to find area & circumference of circle.

```
BEGIN
     DEFINE: float r, area, cir
     INPUT: float r
     SET: area = 3.142*r*r, cir = 2*3.142*r
     OUTPUT: area, cir
ENDIF
```

(*) Pseudocode to find largest of two numbers.

```
BEGIN
      DEFINE: integer a,b
      INPUT : integer a,b
              IF (a>b) THEN
              OUTPUT: a
              ELSE
              OUTPUT: b
              END IF
END
```

(*) Pseudocode to generate and display 1 to n nos.

```
BEGIN
      DEFINE: integer n, i
      INPUT  : integer n
      SET    : i=1
               WHILE (i<=n)
               OUTPUT: i
               SET: i=i+1
               END WHILE
END
```

(*) Pseudocode to find factorial of 'n'.

```
BEGIN
      DEFINE: integer n, i, fact
      INPUT  : integer n
      SET    : i=1, fact=1
               WHILE (i<=n)
               SET: fact=fact*i
               SET: i=i+1
               END WHILE
      OUTPUT: fact
END
```

(*) Pseudocode to find sum of even and odd nos bet'n 1 to n.

```
BEGIN
      DEFINE : integer n, i, esum, osum
      INPUT  : integer n
      SET    : i=1, esum=0, osum=0
              WHILE(i<=n)
                    IF (i%2==0)
                     esum=esum+i
                    ELSE
                    osum=osum+i
      OUTPUT: esum,   SET:  END IF
```

(4) Coding a program: In this phase, the programmer writes the source code (program) by using different programming languages.

Examples:

(*) C program to find Largest of two nos.

```
/* largest of two nos */
#include<stdio.h>
#include <conio.h>
void main()
{
  int a, b;
  clrscr();
  printf ("Enter any two numbers");
  scanf ("%d %d", &a, &b);
         if (a>b)
         printf ("\n Largest number is %d", a);
         else
         printf ("\n Largest number is %d", b);
  getch();
}
```

OR

```
#include<stdio.h>
#include< conio.h>
void main()
{
  int a, b, large;
  clrscr();
  printf ("Enter any two numbers");
  scanf ("%d %d", &a, &b);
  large=a;
         if (b>large)
         large=b;
 printf ("\n Largest number is %d", large);
 getch();
}
```

o/p:

```
Enter any two numbers
7  3 ↵
Largest number is 7
```

(5) Testing : In this phase, the written program will be tested for its correctness and accuracy by giving different set of inputs.

(6) Maintenance: In this phase, the improvement or correction of the written program takes place.

Introduction to C Programming Language:-

C programming language was developed by Dennis Ritchie in 1972 at Bell Telephone Lab., U.S.A. This is one of the High Level Language or structured language or procedure oriented programming language.

structured programming:- The programming approach, in which the programmer makes the use of set of rules (syntax) and predefined functions to solve a given problem to get desired result. For this purpose, the programmer makes use of different programming techniques like sequence, selection and iteration. This programming approach is also called as procedure oriented programming.

(imp) Structure of C program:- Each programming language makes the use of different set of rules and formats to write computer programs. To write C programs, the programmer needs to provide basic elements such as comment lines, preprocessors, global declarations, implementation of the main() and sub-programs. The arrangement of these basic elements from top to bottom in C is called structure of C program.

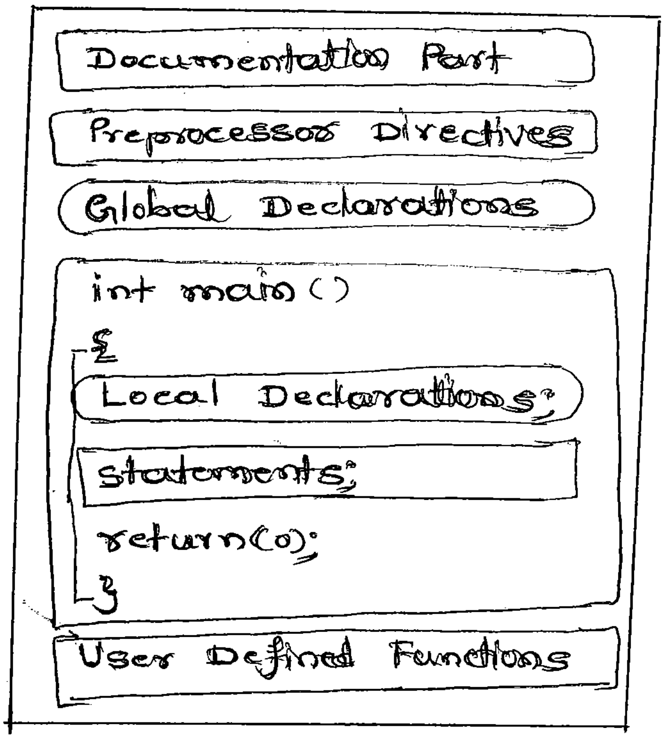


Figure 1. structure of a C program

Documentation Part:- In this part, the programmer writes the comment lines. It includes the main objective of the program to be write and other details like programmer name, date, etc.

e.g. /* C program to find simple interest */

/* C program to find the roots of a quadratic equation for three non-zero co-efficients
Author: X.Y.Z date: _/_/2015 */

Preprocessor Directives:- In this part, the programmer includes the required header files and defines the symbolic constants by using the directives include and define respectively.

e.g.
```
#include <stdio.h>        /* to include header files */
#include <conio.h>
#include <math.h>
# include < string.h>
#include < alloc.h>
# define PI 3.142         /* defines symbolic constants */
# define MAX 100
# define MIN 0
# define TRUE 1
# define FALSE 0
# define RATE_INT 0.05
```

Global Declarations:- In this part, the programmer defines global (public) variables and user defined functions. The global variables can be accessed inside the main() and sub-programs also.

e.g.

```
 int total;          /* global variable */
 void f1 (void);
 void f2 (void);
 void main()
 {
  ...
  ...
  f1();
  f2();
  ...
  ...
 }
void f1 (void)
{ int amt;    /* local variable */
  ...
  ...
  total=total+amt;
  ...
}
void f2 (void)
{ int amt;   /* local variable */
  ...
  total=total+amt;
  ...
}
```

The main() function:- Every C program must have one main function. The execution of the written program begin with main() only. It contains two parts namely declaration part followed by executable part.

The declaration part helps the programmer to allocate computer's memory to solve the problem. whereas, the executable part contains set of instructions/ statements. All these statements must end with the semicolon.

e.g.

```
int main()
{
 int n, sq;    /* declaration part */
 clrscr();
 printf("\nEnter n");              printf("\n Square is %d", sq);
 scanf("%d", &n);                  getch();
 sq=n*n;
```

User Defined Functions:- In this part, the programmer writes the sub programs to do specific task. These sub-programs are also called as user defined functions.

e.g.

```
int max (int x, int y);              /*global Declration*/
void main()
{
 int n1, n2, ans;                    /*local Declaration*/
 clrscr();
 printf(" Enter two nos");
 scanf("%d%d", &n1, &n2);            /*Executable Part*/
 ans= max(n1,n2);
 printf("\n Largest no is %d", ans);
 getch();
}
int max (int x, int y)               /*Subprogram or UDF*/
{
 if (x>y)
   return(x);
 else
   return(y);
}
```

o/p:

```
Enter two nos
7 3 ↵
Largest no is 7
```

(imp) **C-tokens:-** C tokens are the smallest individual units of a c program. These are the basic building of blocks of every C program that can't be further divided into elements. C program contains six types of tokens as follow.

(1) Keywords

(2) Identifiers

(3) Constants

(4) Strings

(5) Operators and

(6) Special Symbols.

Keywords:- Keywords are the predefined or reserved words used by developers of C-compiler. Keywords have fixed meanings to compiler and can not be changed by programmers. Keywords can not be used as identifiers or variable names. Keywords are written in lowercase. ANSI C has only 32 keywords. The keywords are also called as reserved words.

Examples: int, float, char, void, double, signed, unsigned, short, long, if, else, switch, case, default, while, do, for, static, register, auto, break, continue, goto, etc.

Identifiers:- Identifiers are the names given to the elements of a program such as variables, arrays, functions, structure and files by the programmer. Identifiers consist of sequence of letters and digits with letter as a first character. The under_score character can be used to create link between two words in long identifiers.

Rules for defining identifiers:-

(*) First character of an identifier must be letter or underscore.

(*) Keywords can not be used as identifiers.

(*) Identifiers are case sensitive
e.g. salary, Salary and SALARY are three different identifiers.

(*) Only first 31 characters are significant.

(*) Good identifiers are descriptive but they should be short.
e.g. student can be abbrevated as stdnt

Examples:

(i) Identifiers are names of variables.

```
int n1, n2, ans;
float r, area, cir;
```

(ii) Identifiers are names of arrays

```
int rno[65];
float avg[65];
```

(iii) Identifires are names of functions.

```
int max (int x, int y)
{
  if (x>y)
    return (x);
  else
    return(y))
}
```

(iv) Identifires are names of structure.

```
struct student
{
  int rno;
  char name[25];
  float avg;
};
```

Valid Identifires	Invalid Identifires
n1	char
number1	$price
Average	1n
roll_no	roll no
	123

③ **Constants:** The constants are the fixed values. These can not be changed during the program execution. The programmer can make the use of literal constants or symbolic constants or constant variable, while solving the problems.

Examples:

(i) literal constants like 125, 0.05, 'A', 'c', "VTU", etc

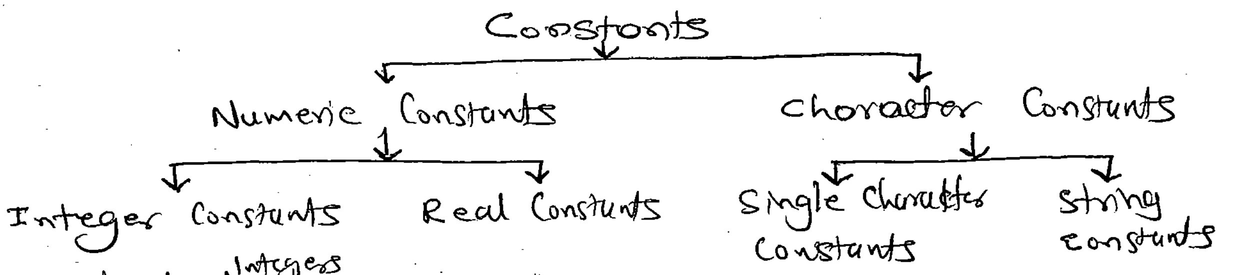
(ii) symbolic constants:

```
#define  PI  3.142
#define  N   100
```

(iii) Variable constants can be defined by using the keyword const.

e.g.
```
const int n=100;
const int max=500;
```

Types of Constants: The constants can be classified into following types.



(a) Decimal ~~Numbers~~ Integers
e.g. 170, 507, 157, 1018, etc

(b) Octal Integers
e.g. 01, 05, 07, 010, 014, etc

(c) Hexadecimal Integers
e.g. 0X9, 0XA, 0XF, 0X27, etc

Single Character Constants e.g. 'A', 'c', '7', etc

String constants e.g. "VTU", "2015", etc

④ **Strings:**- The string can be defined as sequence of characters enclosed within double quotes.

Examples:
"Enter any two numbers", "John", "2015", "@34", etc.

To design and develop user friendly programs, the programmer makes the use of strings. (i.e. to display appropriate messages)

⑤ Operators:- The operator can be defined as a symbol that specifies about the operation to be perform on operands to get desired result. The proper sequence of operonds & operators creates an expression. The expression helps the programmer to process the input data to get result.

C supports several operators, some of the important are as follow.

(1) Unary Operators: The operators that can be used with only one operond are called unary operators.

Ex: increment operator (++) and decrement operator(--)

```
int n=5;                        int n=5;
n++;   /*n=n+1*/                n--;   /*n=n-1*/
∴ n=6                           ∴ n=4
```

The increment operator adds one to the given operond; whereas, the decrement operator substracts one from the given operond.

Exs logical Not (!), minus operator (-), ampersand (&) etc

(2) Binary operators: The operators that can be used with two operond are called binary operators.

~~Ex: Binary~~

Ex: Arithmetical operators like +, -, *, /, %.

Relational operators like <, <=, >, >=, ==, !=.

Logical operators like logical AND, logical OR.

(3) Ternary operator: C supports one ternary operator called conditional operator (?:). It requires three operands and two operators to make two-way decisions. It is similar to if...else statement in C.

Ex:
```
int x=7, y=3, ans;
ans=(x>y ? x : y);      ∴ ans=7
```

(Imp) **Data Types in C:** C is rich in supporting different varieties of data types to store and process different data like integers, floating point numbers, characters and strings, etc. The type of data, that can be stored in computer's memory can be defined by using different data types like int, float, double, char, etc.

C supports the following three classes of data types.

(*) Basic / Fundamental / Built-in / Primitive Data Types (e.g. char, int, float, double, void)

(*) Derived Data Types (e.g. array)

(*) User Defined Data Types (e.g. structure, union, etc)

Basic Data Types in C

Type	Description	Data Type	Format Specifier	size	Range of Values
) character	To store & process single character	char	%c	1 byte (8 bits)	-128 to +127 (signed) 0 to 255 (unsigned)
) integer	Numbers without fractional part	int	%d	2 bytes (16 bits)	-32,768 to +32,767 (signed) 0 to 65,535 (unsigned)
) Floating Point or Real Nos.	Numbers with fractional part. precision = 6	float	%f	4 bytes (32 bits)	3.4 e-38 to 3.4 e+38
Double precision	Numbers with fractional part precision = 15	double	%lf	8 bytes (64 bits)	1.7 e-308 to 1.7 e+308
Non-Specific	To represent no values	void	—	—	—

Data Type Modifiers:- C supports the following 4 types of data type modifies to increase or decrease the storage range of values and to work with positive and negative values.

(*) **signed** :- to work with both -ve & +ve values

(*) **long**: storage range increases

(*) short int → %hd → 1 byte (8 bits) → 0 to 255 (unsigned)
($2^8 = 256$) -128 to +127 (signed)

(*) long int → %ld → 4 bytes (32 bits) → -2,147,483,648 to +2,147,483,647 (signed)

Variables:- A quantity which may vary during program execution is called a variable. Each variable has a specific storage location in computer's memory with valid name (identifier). Each variable has its address in terms of integers, which can be accessed by using ampersand sign. (&).

Ex (i):- int n1=7, n2=3, ans=n1+n2;

Identifier →	n1	n2	ans
Contents →	7	3	10
Address →	8000	9000	10000

Declaration of variables: The variables must be declared in the declaration part of the main() before they are used in the executable part. The syntax for variable declaration is as follow.

Syntax:-

```
data-type v1,v2,v3,...,vn;
where,
data-type → The basic data type like int, float,
            char, double, etc
v1,v2,v3...vn → The valid identifiers
```

Examples:-

(i) int n1, n2, ans;

(ii) float radius, area, cir;

(iii) float p, t, r, si;

(iv) float a, b, c, d, r1, r2;

(v) double result;

(vi) char ch1, ch2;

Initialization of variables:- The process of assigning the values to the defined variable is called initialization of variable. The values can be assigned to variables at the program compile time or at run time. The syntax and examples are as follow.

(1) At compile time:- If the values are known by the programmer, then programmer makes the use of compile time initialization.

Syntax:

```
data-type variable-name = value;
```

Example:

```
(i) int n1=7, n2=3, ans=n1+n2;

(ii) float r=2.5, area=3.142*r*r;
(iii) char ch1='A', ch2='Z';
```

(2) At run-time:- The values can be read from end users of program from a keyboard by using input functions and is possible by run-time initialization.

Examples

```
(i) int n1, n2, ans;
    printf("Enter two nos");
    scanf("%d %d", &n1, &n2);

(ii) float r, area;
     printf("Enter radius of circle");
     scanf("%f", &r);
```

Assignments:-

(1) What are data types? Explain in bref. (10 M)

(2) What is a variable? Explain its declaration and initialization with syntax and examples. (6 M)

(3) Read text books and solve question papers.

(4) Read the given notes frequently & recall the topics.

Managing input/output operations by using Input/ output functions in C:- The major four functions of a computer system are input, process, output and store. The input and output (I/O) operations can be performed by C programmers by making use of Input and output (I/O) functions predefined by developers of C compiler. These I/O functions are stored in standard input and output library named stdio.h.

Types of I/O functions:- The C language supports the following two types of I/O functions to perform I/O operations.

1. Formatted I/O Functions
   e.g. scanf(), printf()
2. Unformatted I/O Functions
   e.g. getch(), putch(), gets(), puts().

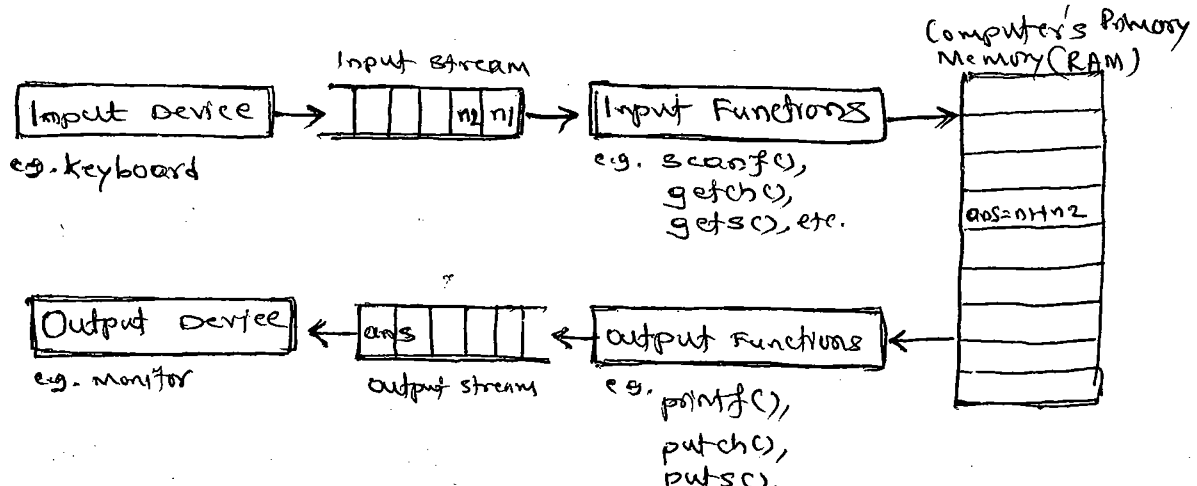


Figure 2. I/O operations in C

The input functions like scaf(), getch(), and gets() are used by programmers to read input data from keyboard's. ~~buffer~~ i.e. the input data will be transmitted from keyboard's buffer (temporary storage) to the computer's primary memory RAM. (Random Access Memory).
The output functions transmit processed data from RAM to output device.

(i) Formatted I/O Functions:- The formatted I/O functions are used by C programmers to read and display different variety of data like integers, floating point numbers, characters and strings. The most commonly used formatted I/O functions are scanf() and printf().

scanf() : The scanf() is one of the formatted input function. It helps the programmer to read different variety of data like integers, characters, floating point numbers, double type of value and strings from a keyboard. This function transmits the input data from keyboard's buffer to computer's internal memory RAM.

The general form or syntax of the scanf() is as follow.

Syntax

```
scanf("format specifiers", &V1, &V2, ..., &Vn);
```

where,

format specifiers → The data specifiers like %d, %f, %c, %s, %u, %ld, etc.

&V1, &V2, ... &Vn → List of valid variables with ampersand (&) sign.

Examples:

(i) To read two integer values

```
int n1, n2, ans;
scanf("%d %d", &n1, &n2);
```

(ii) To read three floating point numbers.

```
float a, b, c;
scanf("%f %f %f", &a, &b &c);
```

(iii) To read two characters.

```
~~se~~ char ch1, ch2;
scanf("%c%c", &ch1, &ch2);
```

(iv) To read three values of the type integer, float and char.

```
int a;
float b;
char c;
scanf("%d %f %c", &a, &b, &c);
I/P: 10 20.5 A ↵
∴ a=10, b=20.5, c='A'.
```

(v) To read three integers with the delimeter comma.

```
int a, b, c;
scanf("%d, %d, %d", &a, &b, &c);
I/P: 10, 20, 30 ↵
∴ a=10, b=20, c=30
```

(vi) To read specified number of digits from a keyboard (%wd)

```
int a, b, c;
scanf("%3d %3d %3d", &a, &b, &c);
I/P: 123456789 ↵
∴ a=123, b=456, c=789.
```

② printf() : This is one of the formatted output function. It helps the programmer to display different variety of data on the screen. It transmits the processed data from computer's internal memory RAM to output device. The syntax of printf() is as follow.

Syntax:

```
printf("text with format specifiers", variables_list);
```

Examples:

```
(i) int n1=7, n2=3, ans=n1+n2;
→ printf("%d", ans);        → printf("Addition is %d", ans);
  O/P: 10                     O/P: Addition is 10
                            → printf("Addition of %d and %d is %d", n1, n2,
                                                                   ans);
```

Examples

(i)
```
int n1=7, n2=3, ans=n1+n2;
printf("%d", ans);
o/p: 10
printf("Addition is %d", ans);
o/p: Addition is 10
printf("The addition of %d and %d is %d", n1,n2,ans);
o/p The addition of 7 and 3 is 10.
```

(ii)
```
printf("\n Root1=%f \t Root2=%f", r1,r2);
o/p: Root1=-1.000000    Root2=-1.000000
```

(iii)
```
char ch='y';
printf("The charoeter is %c", ch);
o/p: The chorector is Y
```

(iv)
```
printf("The reverse of %d is %d", dup,rev);
o/p: The reverse of 1234 is 4321
```

More about printf()

% <flag> <width> <precision> <size> conversion code

size: %hd → represents short integer.

h ↑ size

d ↑ conversion code

%ld → represents long integer

l ↑ size

d ↑ conversion code

Conversion codes

%d, %i → signed integers

%u → unsigned integers

%ld → long   %hd → short

%f → float   %lf → double

%e → floating point in exponential form

%c → character   %s → string

width (%wd):-

e.g Value	%d
12	12
123	123
1234	1234
12345	12345

%4d

		1	2	
	1	2	3	
1	2	3	4	
1	2	3	4	5

precision :

e.g. float ans = 98.3214;

printf ("Result = %5.2f", ans);

O/P: Result = 98.32

flag: The flag value may be minus (-) or zero (0).

e.g int n=123;

printf ("%5d", n);

O/P:

		1	2	3

printf ("%-5d", n); /* left aligned within specified width */

O/P:

1	2	3		

printf ("%05d", n); /* value is preceded by leading zero's */

O/P:

0	0	1	2	3

② Unformatted I/o Functions:- These are used to read and display only characters & strings without any format i.e. only values will be read and displayed. C supports the following unformatted I/o functions.

(i) getch() and putch()

(ii) gets() and puts()

(i) getch() and putch():- These are used to read and display only single character.

Syntax

character-variable = getch();   putch(character_variable);

Example for getch() and putch():-

```
(i) char ch;
    clrscr();
    printf ("Enter a character");
    ch=getch();
```

o/p: Enter a character

```
 y ↵
∴ ch='y'
```

```
printf("\n The character is");
putch(ch);
```

o/p: The character is y

To remember:

(i) ch=getch();

Input is not visible

No need to press ENTER KEY

(ii) ch=getche();

Input is visible

No need to press ENTER KEY

(iii) ch=getchar();

Input is visible

Press ENTER KEY

(2) gets() and puts():- These two are the unformatted I/O functions. These are used by programmers to read and display only the strings.

Syntax:

```
gets(string-variable);          puts(string_variable);
```

Example

```
char name[25];  /*string variable */
clrscr();
printf("Enter your name");
gets(name);
printf("\n Hello ! ");
puts(name);
getch();
```

o/p: Enter your name

```
Thomas ↵
Hello !, Thomas.
```

Operators and Expressions:

C is very rich in supporting different types of mathematical operators and expressions to solve a given problem. The operators and expressions are used by programmers to process the input raw data to get desired result.

Operator:- An operator can be defined as a symbol that specifies about the operation to be perform on operands. The operand may be the constant or variable.

Examples: Arithmetical operators (+, -, *, /, %),
Relational operators (<, <=, ==, >, >=, !=). &
Logical operators (&&, ||, !), etc

Expression:- An expression can be defined as a proper sequence of operands and operators that can be reduced to a single value.

Examples:

i) Arithmetical expressions like ans=a+b, d=(b*b)-(4*a*
r1=(-b+sqrt(d))/(2*a); c=(5/9)*(f-32), etc

ii) Relational expressions like (num!=0), (a!=0), (a>b), etc

iii) Logical expressions like (a>b && a>c),
(((year%4==0) && (year%100!=100)) || (year%400==0)) etc

iv) Ternary expression: large=(a>b?a:b);

v) Unary expressions: n++, i++, n--, i--, etc

vi) prefix expression: ++n, --n, etc.

vii) postfix expression: n++, n--, etc

viii) infix expression: a+b, a-b, a*b, a/c, etc

(V.imp) Types of operators:- C supports rich set of mathematical operators to process the input data to get desired result. Some of the important are as follow.

i) Arithmetical Operators
ii) Relational operators
iii) Logical Operators
iv) Assignment Operators
v) Increment and Decrement Operators
vi) Conditional Operator
vii) Bitwise Operators and
viii) Special Operators.

(i) Arithmetical Operators:- These are used to perform arithmetic operations on two operands. The operands may be constants or variables. The operators include addition (+), subtraction (-), multiplication (*), division (/) & modulus (%). The arithmetic operators are also called as binary operators because they require two operands to perform an operation.

Operator	Meaning	Example	Precedence	Associativity
*	Multiplication	4*2=8	1	Left to Right
/	Division	4/2=2	1	"
%	Modulus	4%2=0	1	"
+	Addition	4+2=6	2	"
–	Subtraction	4-2=2	2	"

(V.imp) Precedence of Operator:- The order or priority value ~~used by machine to evaluate~~ of an operator used by machine to evaluate a complex expression is called precedence of operator or hierarchy of operator.

(vimp) Associativity of operators:- If two or more operators with the same precedence value are adjacent to each other in an expression then the associativity rule is used to evaluate it from left to right or right to left.

Examples:

(i) precedence and associativity of arithmetic operators.

operator	Meaning	Precedence	Associativity
*	multiplication	1	L→R
/	Division	1	"
%	modulus	1	"
+	Addition	2	"
−	subtraction	2	"

Evaluation of arithmetic expressions:-

$x = \boxed{8/4} * 16$ $\qquad$ $x = \boxed{8 * 4}/16$

$x = 2 * 16$ $\qquad$ $x = 32/16$

$\therefore x = 32$ $\qquad$ $\therefore x = 2$

In the above examples, the division (/) and multiplication (*) are having same priority value 1. Hence, associativity rule L→R is followed.

Ex (ii) Evaluate the following expression.

$5 * 2 \% 3 + 7 * 10/5$

step1: $\boxed{5 * 2} \% 3 + 7 * 10 / 5$ $\qquad$ /* multiplication */

step2: $\boxed{10 \% 3} + 7 * 10/5$ $\qquad$ /* Division */

step3: $1 + \boxed{7 * 10}/5$ $\qquad$ /* multiplication */

step4: $1 + \boxed{70/5}$ $\qquad$ /* Division */

step5: $\boxed{1 + 14}$ $\qquad$ /* Addition */

step 6: 15 $\qquad$ ∴ Result = 15

Ex (iii) Evaluate the following expressions

(i) $x = a + c * d / e \% f - b$ if $a=1, b=2, c=3, d=4, e=5, f=6$

(ii) $ans = 10 + 5 \% 10 - 1 * 5$

(3) Relational Operators:- The relational operators are used to ~~[illegible]~~ find the relationship between two operands to take certain decisions The operands may be the variables or constants. The result of relational expression may be either true or false. They are as follow.

Operator	Meaning	Example	Precedence	Associativity
<	less than	5<5 (F)	1	L→R
<=	less than or equals to	5<=5 (T)	1	"
>	greater than	5>5 (F)	1	"
>=	greater than or equals to	5>=5 (T)	1	"
==	equals to	5==5 (T)	2	"
!=	Not equals to	5!=5 (F)	2	"

Relational Expressions:
(num!=0), (d==0), (d>0), (d<0), (a>b), etc.

Evaluate the following expressions:-

(i) 10==5<3!=0>=7

Step1 :- $10 == \boxed{5<3} != 0 >= 7$

Step2 :- $10 == 0 != \boxed{0>=7}$

Step3 :- $\boxed{10==0} != 0$

Step4 :- $0 != 0$

Step5 0 Result=0 (False)

(ii) 5>=1 !=0<5==7>0

(iii) a>b+c !=d<=a==b-1
if a=10, b=20, c=30, d=40,

④ Logical operators:- These are used to combine two or more relational expressions into a single expression to take certain decisions. The result of the logical expression is also either <u>true</u> or <u>false</u>. They are as follow.

Operator	Meaning	Example	Precedence	Associativity
!	logical NOT	!5 (False)	1	L→R
&&	logical AND	5&&5 (True)	2	"
||	logical OR	5||5 (True)	3	"

Truth Table of logical AND (&&); logical OR(||) and logical NOT

x	y	x&&y	x||y	!x	!y
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

T → True $\xrightarrow{\text{means}}$ Non-zero Value
e.g. 1, -1, 5, 10, "VTU", etc

F → False $\xrightarrow{\text{means}}$ zero

Example for logical expression:-

(i) 5||0&&7||!5

step1: 5||0&&7||$\boxed{!5}$

step2: 5||$\boxed{0\&\&7}$||0

step3: $\boxed{5||7}$||0

step4: 1||0

step5: 1 Result=1 (True)

(ii) Evaluate the given expression a&&b||!c||a
(if a=10, b=20, c=30)

(iii) a+b%c<a==c!=b&&!a-b (if a=10, b=20, c=30)

Hierarchy of operators | Brackets
Unary operators
Arithmetic
Relational

④ **Assignment Operators**:- The assignment operator (=) is used by C programmers to assign a new value to a given variable at the left side of assignment operator.

Example:
```
int n1=7, n2=3, ans=n1+n2;
int a=10, b=20, c=30;
int i,j,k;
i=j=k=5;
```

Shorthand Assignment Operators:- C support the shorthand assignment operators to write assignment expressions into a compact.

Shorthand Assignment operator	Example	simple Assignment expression
+=	n+=5	n=n+5
-=	n-=5	n=n-5
*=	n*=10	n=n*10
/=	n/=10	n=n/10
%=	n%=5	n=n%5

Note: Assignment operators are having Right → Left associativity.

(imp) ⑤ **Increment and Decrement Operators** (++, --):→

C supports two unary operators as increment (++) and decrement (--) operator. The increment operator adds only one to the given operand, whereas, the decrement operator subtracts only one from given operand.

Examples:

(i)
```
int n=5;
n++;  /* n=n+1 */
∴ n=6
```

(ii)
```
int n=5;
n--;  /* n=n-1 */
∴ n=4
```

The increment and decrement operators can be used as <u>prefix</u> and <u>postfix</u>. The examples are as follow.

Prefix	Postfix
int n=5, ans;	int n=5, ans;
ans = ++n;	ans=n++;
∴ ans=6, n=6	∴ ans=5, n=6

A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left.

A postfix operator first assigns the value to the variable on left and then increments the operand.

Note:- The increment/decrement operators are having Right → Left associativity.

⑥ Conditional Operator (?:):- This is one of the ternary operator that makes the use of two operators and three operands to take two-way decisions. The use of conditional operator is equal to if... else statement available in C.

Syntax (expr1 ? expr2 : expr3)

The given expression-1 will be evaluated for true or false; if it is true, then expression-2 executes; otherwise, expression3.

Examples:-(i) int a=10, b=20, large;

large= (a>b ? a: b); $\xrightarrow{\text{is equals to}}$ if (a>b)
large=a;
else
large=b;

∴ large=20

(ii) int a=10, b=20, c=30, large;

large= (a>b ? a: b);

large= (large>c ? large: c);

∴ large= 30

(vimp) **Bitwise Operators:-** C supports the bitwise operators to manipulate the data at bit level or low level. The bit stands for binary digit i.e. 0 or 1. The bitwise operators are used by programmers to set, reset, shift or complement the binary digits to get the desired result. These are used by programmers while writing system programs for low level activities. These operators are used with only integer operands. These are having left to right associativity. The bitwise operators are as follow.

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwse exclusive OR
<<	bitwise LEFT SHIFT
>>	bitwise RIGHT SHIFT
~	bitwise Compliment (Negate)

Examples:

(i) bitwise AND(&):- It is often used to mask OFF or turn OFF the binary digits.

e.g. int a=4, b=3;

∴ a&b = ?

Assume 8 bits m/c

```
    a = 0000 0100
    b = 0000 0011
   ---------------
a&b = 0000 0000
```

(ii) bitwise OR (|):- It is often used to turn ON some set of binary digits (bits).

```
    a = 0000 0100
    b = 0000 0011
   ---------------
a|b = 0000 0111
```

(iii) bitwise exclusive OR:- It is used to set 1 (one) in each bit position where its operonds have different bits and 0 (zero) whether they are same.

Truth Table of XOR:

x	Y	x^R
0	0	0
0	1	1
1	0	1
1	1	0

e.g

a= 0000 0100

b= 0000 0011

a^b = 0000 0111

(iv) bitwise complement (NOT):- It is unary operator. It converts each 1 (one) into 0 (zero) and vice-versa.

e.g. Let a=10

Decimal Binary

$(10)_{10} = (1010)_2$

b= ~(a);

b= ~(1010);

∴ b= 0101

(v) bitwise LEFT SHIFT & RIGHT SHIFT operators:- (<<, >>)

These are used by programmers to shift the binary digits to left or right by specified number of positions. The dropped bits will be filled with 0's.

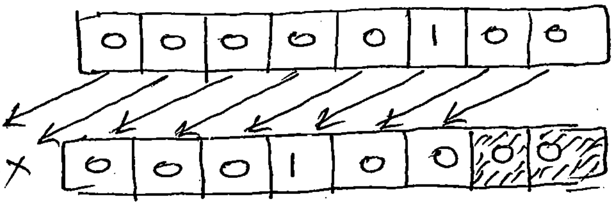
e.g. int a=4; $(4)_{10} = (100)_2$

Assume 8 bits m/c.

(i) before left shift

a << 2

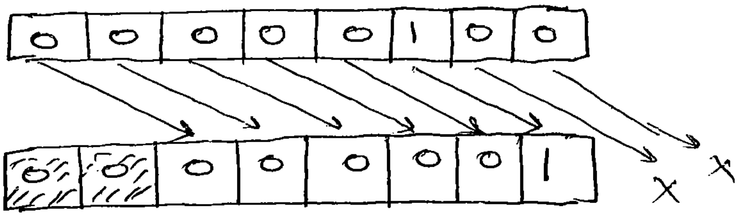
after left shift



(ii) before right shift

a>>2

after right shift



Special Operators:- C supports the following two special operators.

(i) The Comma Operator

(ii) The sizeof() operator

(i) The Comma operator:- It helps to combine multiple expressions into a single expression. It has the least precedence among all the C operator.

Examples:

(i) To exchange (swap) the contents of two variables by using temporary variable.

```
int a=7, b=3, temp;
temp=a;   }
a=b;      }   temp=a, a=b, b=temp;
b=temp;   }
```

(ii) To swap the values without using temporary variable.

```
a=a+b;    }
b=a-b;    }   a=a+b, b=a-b, a=a-b;
a=a-b;    }
```

(ii) The sizeof() operator:- It helps the programmer to find the size (memory) occupied by the given operand. The operand may be the variable or data type.

Syntax,

```
sizeof(operand);
```

where,

operand → may be variable or data-type

Examples:

(i)
```
int a, temp;
temp = sizeof (a);
∴ temp = 2   /*bytes*/
```

(iii)
```
double result;
int temp;
temp = sizeof (result);
∴ temp = 8   /*bytes*/
```

(ii)
```
float b;
int temp;
temp = sizeof (b);
∴ temp = 4   /*bytes*/
```

(iv)
```
char ch; int temp;
temp = sizeof (ch);
∴ temp = 1   /*bytes*/
```

(v)
```
temp = sizeof(int);
∴ temp = 2
```

Ex1 C program to check whether the m/c is 16 bits or 32 bits m/c.

```
{int a; int temp;
 clrscr();
 temp= sizeof (a);
       if (temp==2)
       printf ("16 bits machine");
       else
       printf ("32 bits machine");
 getch();
}
```

(vimp) **Type Conversion or Type Cast:-** The process of converting the data from one data type to another data type to get accurate result, while working with the similor <u>or</u> mixed type of c-expressions is called type conversion. or type cast.

Types of Type Conversions:- The data type conversion is done by C programmers ~~to~~ in the following two ways to get accurate results.

(1) Implicit type conversion

(2) Explicit type conversion

① **Implicit type conversion:-** In this type, the data type conversion takes place by the C compiler automatically while working with the mixed type expression to get accurate result. The data type conversion takes place from lower rank data type to higher rank data type.

(short int → int → long int → float → double → long double)
↑ lower rank — higher rank.

Example:

```
float ans;
int k= 2;
ans= 1/k;
∴ ans= 0.000000 /* incorrect result */
```

```
float ans;
ans= 1/2.0;  /* implicit conversion */
∴ ans= 0.500000
     /* correct result */
```

EX(2):-

```
float ans;
ans= 1.0/2;          or     ans= 1/2.0;    </* implicit type conversion */
∴ ans= 0.500000             ∴ ans= 0.500000
```

Advantages of implicit type conversion:-

(1) Automatic type conversion from lower rank to higher rank data-type by C compiler. in mixed type expression.

Disadvantages:-

(*) If both the operands are similar data type then type conversion not possible.

(*) Conversion from higher rank to lower rank data type is not possible.

(2) Explicit Type Conversion:- In this type, the data type conversion from one type to another type is done by programmers explicitly i.e. not by C-compiler. This conversion is useful while working with the similar type of operands, to get accurate results.

Syntax

```
data-type (expression);
```

Examples:

```
(i)    float ans=1.5;
       int a;
       a = int (ans);   /* Explicit type conversion */
       ∴ a=1

(ii)  float ans;
      int k=2;
      ans= 1/(float)k;        or     ans= (float)1/k;

      ∴ ans= 0.500000               ∴ ans= 0.500000

(iii) int a=3, b=3, c=3;
      float avg;
      avg= (float) (a+b+c)/3;   /* Explicit type Conversion */
```

Advantages of explicit type conversion:-

(*) higher rank data type to lower rank data type conversion is possible.

mathematical Expressions	C - Expressions
$\frac{x}{a}$	x/a
b^2-4ac	(b*b)-(4*a*c)
$r1=\frac{-b}{2a}$	r1= (-b)/(2*a);
$avg=\frac{a+b+c}{3}$	avg=(a+b+c)/3
$\sqrt{s*(s-a)(s-b)(s-c)}$	sqrt(s*(s-a)*(s-b)*(s-c))
x^n	pow(x,n)
$r1=\frac{-b+\sqrt{disc}}{2a}$	r1=((-b)+sqrt(disc))/(2*a)
$D=x^{25}+y^{35}$	D = pow(x,25)+pow(y,35)
$x=\frac{e^{\sqrt{x}}+e^{\sqrt{y}}}{x\sin\sqrt{y}}$	~~x=(pow(e,sqrt(x))+pow(e,sqrt(y))~~

x= (pow(e,sqrt(x)) + pow(e,sqrt(y)))/(x*sin*sqrt(y))

(Read, Understand, Recall) (Read books) vimp

Dear Student,

You are hereby informed that, this notes helps you to understand the topics but, it acts as a reference notes. Hence, it is necessary for you to read recommended text books of PCD Subject to add some more points to this notes to get good marks in exam. Hence, read books, understand, recall the imp points. —