INPUT AND OUTPUT FUNCTIONS:

C doesn't have any built-in input/output statements as part of its syntax, instead all I/O operations are carried out through functions like scanf and printf which are collectively known as standard I/O library functions.

Reading a Characters:

Reading a single character can be done by using the function **getchar( )**. The general form of getchar( ) is:

Variable_name = getchar( );

Where Variable_name is a valid C name declared as char type. When this statement is encountered, the computer wait until a key is pressed and then assign this character as a value to getchar( ) functions.

Example

```
Char CSE
CSE=getchar();
```

} will assign the value if key pressed to CSE.

Note: the getchar( ) function accepts any character keyed in including enter key tab i.e. when a single character input is keyed in the new line character is waiting in the input queue after getchar() returns.

Writing a character:

To write character one at a time to the output terminal it can be done by using the function **putchar**(). The general from of putchar() is

```
putchar(variable_name);
```

Where variable_name is a type char variable containing a character, and displays the character contained at the output terminal.

Example

```
ON = 'y';
putchar(ON);
```

} will display the character y on the screen

Formatted Input:

Input function: The function scanf() is used to read data into variables from the standard input. The general format of an input statements is:

Scanf("control string", arg1, arg2, ….. argn);

The control string specifies the field format in which the data is to be entered and the arguments arg1, arg2, … argn specify the address of locations where the data is stored.

The control string contains field specifications, consisting of the **conversion character %,** a data type character (or type specifier) and an optional number, specifying the filed width. The **data type** character in the control string indicates the type of data that is to be assigned to the variable associated with the corresponding argument.

INTEGER I/P
Ex:1

% w d

Where **%** indicates the conversion specification follow up
w is an integer number specifying field width of the of the number to be read
d is a data type character indicating the number to be read is of integer

Example scanf("%2d %4d", &n1, &n2);

If the input data assigned was 25 and 12345 then they will be assigned to the variables n1 and n2 respectively, instead if the data assigned was 12345 and 50 then the variable n1 will be assigned 12 and n2 will be assigned 345 (unread part of 12345) and 50 will be assigned to variable in next scanf call. If no field specifiers are used then this type of errors can be eliminated.

Example scanf("%d %d", &n1, &n2); will take 12345 for n and 50 for n2.
If the number to be read is of long int then the specification should be %ld instead of %d.

Real I/P
Ex:2

The filed specifications for reading a real number is **%f**. %f can be used for both decimal and exponential notation
Example scanf("%f %f %f", &x, &y, &z);

If the input is 123.45 12.34E-1 123 will assign the value 123.45 to x, 1.234 to y and 123.0 to z
If the number to be read is of double type, then the specification should be %lf instead of %f.

Character I/P
Ex:3

If more than one character is to be read then instead of getchar() we can use scanf function. The filed specification for reading character string is **%ws** or **%wc**

The corresponding argument should be a pointer to a character array. However %c may be used to read a single character when the argument is a pointer to a char variable.

Mixed Mode I/P Expression3:

The mixed mode data can be read with the scanf(), however care should be taken such that the input data items match the control specifications in order and type

Example:
Scanf ("%d %c %f %s", &sum, &id, &avg, name)' will take the data like 100 s 60.6 xyz correctly.

scanf() format codes

%c read a single character

%d	read a decimal integer
%f, %e, %g	read floating point value
%i	read a decimal, hexadecimal or octal integer
%o	read a octal integer
%s	read a string
%x	read a hexadecimal integer
%[]	read a string of word

While using the scanf( ) function for input statement the following points are to be considered.

1. All function arguments, except the control string, must be pointers to variables.
2. Input data items must be separated by spaces and must match the variables receiving the input in the same order.
3. Format specifications contained in the control string should match the arguments in order.
4. When the field width specifier w is used, it should be large enough to contain the input data size.
5. The reading will be terminated, when scanf() encounters an 'invalid mismatch' of data or a character than is not valid for the value being read.
6. Any unread data items is a line will be considered as a part of the data input line to the next scanf() call.

Formatted Output

Output Function: the function printf() can be used to write output data on to the terminals. The general form of an output statement is:
Printf("control string", arg1, arg2, … argn);

The control string consists of the following items.

1. Characters that will be printed on the screen as they appear.
2. Format specifications that define the output format for display of each item.
3. Escape sequence characters such as \n, \t, and \b

The control string indicates how many arguments will follow and what is their data type. The arguments used must match in number, order and type with the format specification mentioned in control string.
A simple format specification has the following form:
%w.p type_speicifer

where **w** is an integer specifying the total no. of columns for O/P value and
p is an integer number specifying the no. of digit after decimal point.
Both w and p are optional.

Integer Output Example: The format specification for printing an integer number is **%wd**
Where **w** specifier the minimum field width for the output. However if a number is greater than the specified filed width, if will be printed in fully, overriding the minimum specification. The number is written right-justified in the given filed width.
Example

printf ("%d", 1234) ⟹ | 1 | 2 | 3 | 4 |

printf ("6%d", 1234) ⟹ | 1 | 2 | 3 | 4 | 5 |

printf ("2%d", 1234) ⟹

1	2	4	5

Real Output Expression

The output of a real number may be displayed in decimal notation using the following format specification.

%w.p.f where **w** indicates the min. no. of positions that are to be used for the display of value and
p indicates the no. of digits to be displayed after the decimal point

Example If y = 12.3456

printf ("%7.4f",y) ⟹

1	2	.	3	4	5	6

printf ("%7.2f", y) ⟹

		1	2	.	3	4

printf ("%f", y) ⟹

1	2	.	3	4	5	6

Exponential form

A real number is exponential can be specified by the following format specifications: **%w.p.e**

Example

printf ("%10.2e",y) ⟹

	1	.	2	3	e	+	0	1

printf ("%11.4e", -y) ⟹

-	1	.	2	3	4	5	e	+	0	1

printf ("%e", y) ⟹

	.	2	3	4	5	6	0	+	0	1

Single Character Output

A Single character can be displayed in a desired position using the format.

%wc The character will be displayed right – justified in the filed and w columns.

String

The format specification for outputting string is similar to that of real numbers. the general form is

%w.p.s where **w** specifies the field width for display and p instructs that only the first **p** characters of the string are to be displayed

Mixed Mode output:

A mixed data types can be permitted is a single printf statement.

Example printf("%d %f %s %c", a,b,c,d);

printf format codes

%c	prints a single character
%d	prints a decimal integer
%f, %e, %g	prints a floating point value
%i	prints a signed decimal integer

%s	prints a string
%x	print a hexadecimal integer

To improve the clarity, readability and understandability of o/p s we can

1. Provide blank space between two numbers
2. Print appropriate headings and suitable messages in output.
3. Introduce blank lines between the important sections of the output.

OPERATORS IN C

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations on the data and variables in a program.

These operators in C can be classified as

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators.
4. Assignment Operators.
5. Increment & Decrement Operators.
6. Conditional Operators.
7. Bitwise Operators.
8. Special Operators.

DATA TYPES

ANSI C supports four classes of *data types*

1. Primary data types.
2. User-defined data types.
3. Derived data types.
4. Empty data set.

1. Primary Data types

The four fundamental (Primary) data types supported by all C compliers are namely

1. integer (int).
2. character (char) .
3. floating point (float) .
4. Double-precision / floating point (double).

Range of data type

1. char - 128 to 127

2. int - -32,768 to 32,767
3. float - 3.4e-38 to 3.4e+38
4. double - 1.7e-308 to 1.7e+308

Data type	Range of values (machine dependent)	
int	-2^{15} to $2^{15}-1$	⟹ for 16 bit
char	-128 to 127	⟹ 8 bit for storage (0-255 for unsigned)& (-128 to 127 signed)
float	$3.4e^{-38}$ to $3.4e^{38}$	⟹ it uses 32 bits with 6 digits of precision
double	$1.7e^{-308}$ to $1.7e^{308}$	⟹ uses 64 bits with 14 digits of precision

Primary type declaration (A variable can be used to store a value of any data type) The general syntax for a primary type declaration is data-type variable list or/i.e. data-type identifier list

Example

```
int   x,y,z;
int   sum;
float average;
```

user–defined type declaration:

a) C supports typedef that allows the user to define an identifier that would represent an existing data type and can be later used to declare a general variable.

```
Syntax      typedef  type  identifier;
Example     typedef  int  rollno;
            Typedef float  marks;
```

The rollno and marks can be used to declare variables as

```
Rollno batch1, batch2;  /  batch1 & batch2 are declared as int
Marks  m. p. c;         / m, p, c are declared as float
```

b) *Enumerated data types*: it is a form of user-defined data type supported by ANSI C. It is defined

Enum identifier (value1, value2, …… value N} enum identifier (value list}

Where identifier is an user-defined enum data type that can be used to declare variables that can have one of the values enclosed within braces.

The general syntax is enum identifier v1, v2, v3, …. VN;

where v1, v2, … vN takes one of value1, value2, …. Value N

Example

```
enum day {Monday, Tuesday, …….. Sunday}
enum day week_str, week_end
```

Then week_str will take Monday

week_end will take Sunday

the compiler assigns integer digits beginning with 0 to all values in value list i.e. value1-0, value2-1, …..

Arithmetic Operators.

C provides all the basic arithmetic operators like

Unary minus -
Division /
Multiplication *
Addition +
Subtraction -

Integer Arithmetic: When both the operands in an expression are integers, the expression is referred as integer expression and the operation is called integer operation
Eg. If a, b, c, d are defined an integers then
1) a+b, a-b, -a*b, c-d*a, b/c are all valid integer expressions
2) a+b=c, a-4.0, a**b, a(b+c) are invalid

Floating Point Arithmetic: An expression containing real operands either in the form of decimal or exponential notation is referred as floating point expression
Eg. If x, y, z are defined as floats then
1) x=4.0/5.0: y=x/z; z=x*y; -x+y, (x/y-z)*5.0 are valid
2) x-y+p; (x+y)(x-y); x**y; (x+(y*z); xy. Are invalid

Mixed-mode Arithmetic: When one of the operands is real and the other is integer, the expression is called a mixed-mode arithmetic expression. If either operand is of real type then the resultant is always a real number
Example 15/10.0=1.5, 15/10=1

OPERATOR PRECEDENCE

The order in which the arithmetic operations are executed in an expression is based on the under of precedence of operators. The precedence of operators is

Unary	-	First Level	Addition	+	Second Level
Multiplication	*		Subtraction	-	
Division	/				
Modulus	%				

All the expressions are evaluated from Left to Right with the order of operator precedence.

Example –a * b/c + d%k - m/d * k+c is evaluated as –db/c + d%k – m/d * k +c

Parenthesis can be used if the order of operations governed by precedence values is to be overridden. The expressions within the parenthesis (Governed by precedence rules) are evaluated first and proceed from Left to Right while evaluating a expression

Example a * b/c (c+d * k/m + k) + a $\Longrightarrow$ a * b / (c + dk/m + k) +a

$$\Longrightarrow \frac{ab}{c + dk/m + k} + a$$

If an expression possess more than one pair of parenthesis then the expression from the innermost parenthesis is evaluated first and then outermost and the rest of the expression is evaluated from Left to Right following the precedence rules.

Increment and Decrement Operators.

C supports two useful operators for incrementing and decrementing the values stored in variable names is of the form of ++ and --
The operator ++ adds 1 to the operand while -- subtract 1

Example
++m is equivalent to m=m+1 or m+ =1; (A short hand notation for m=m+1)
--m is equivalent to m=m-1 or m- =1;
Though ++m and m++ are same when used independently but when used in an expression it gives different values

Example
M=10 The value of m=11 and value of y=11
Y=++m

But when m=10; y=m++ then the value of m=11 and value of y=10

A prefix operator first adds 1 to the operand and then the result is assigned to variable m while a postfix operators first assigns the value to the variable on Left and increment the operand.

Example
Int x=6, y=8, z=10 then
Y=x++; = 6;
Z=++x; = 7+1=8;

The ? : Operator

The C language supports a two-way decision making operator known as conditional operator.
The general form of the use of conditional operator is

Conditional expression ? expression1 : expression2

If the result of conditional expression in non zero then expression1 is evaluated and returns a value to conditional expression else expression2 is evaluated and it returns a value to conditional expression.

Example1

Let y=1.5x+2 for x<=2
y=2x+4 for x>2

It can be written as

y=(x>2) ? (2*x+4) : (1.5x+2);

Example2

4x+100 for x<50

y= 500 for x=50
5x+300 for x>50

can be written as

y=(x!=50)?((x<50)?(4*x+100):5*x+300);500

the same statement can be written by using if..else statement

```
if(x<=50)
if(x<50)
y=4*x+100;
else
y=500;
else
y=5.0*x+300;
```

Advantages: the code becomes more concise and more efficient

Disadvantages: readability is poor.

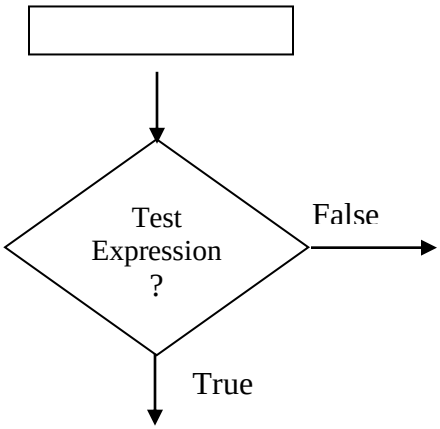
CONTROL STATEMENT:

Decision Making & Branching

C language supports the following control/decision making statements

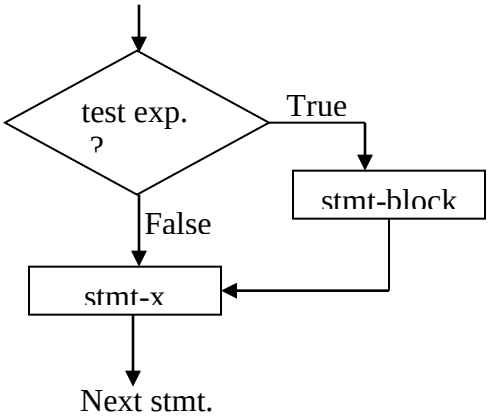
1. if statement.
2. switch statement
3. conditional operator statement
4. goto statement.

<u>if Statement</u>: The if statement is a two-way decision making statement used in conjunction with an expression which is used to control the flow of execution of statements. The General form is **if(test expression)**



The general form of simple if statement is

```
if(test expression)
{
statement_block;
}
Statement-x;
```

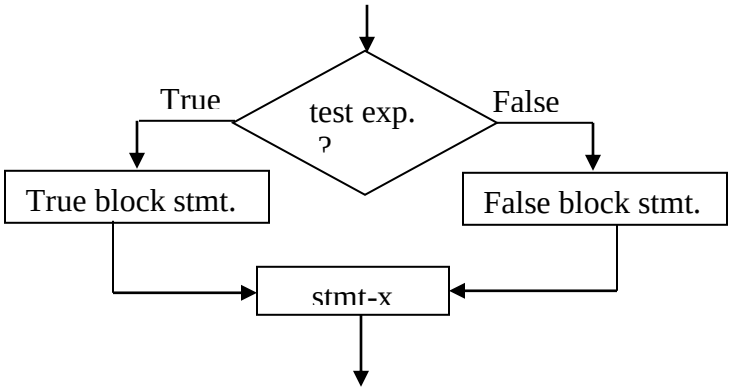


The statement-block may be a single or group of statements
if test expression is true then it executes statement block before executing statement-x otherwise it skips statement block and executes statement-x directly.

Example : If (a>=70)
{ dsum=dsum+1;
}

The if…else statement

```
if(test expression)
{
true-block statement;
}
else
{
false-block statement
Statement-x;
```



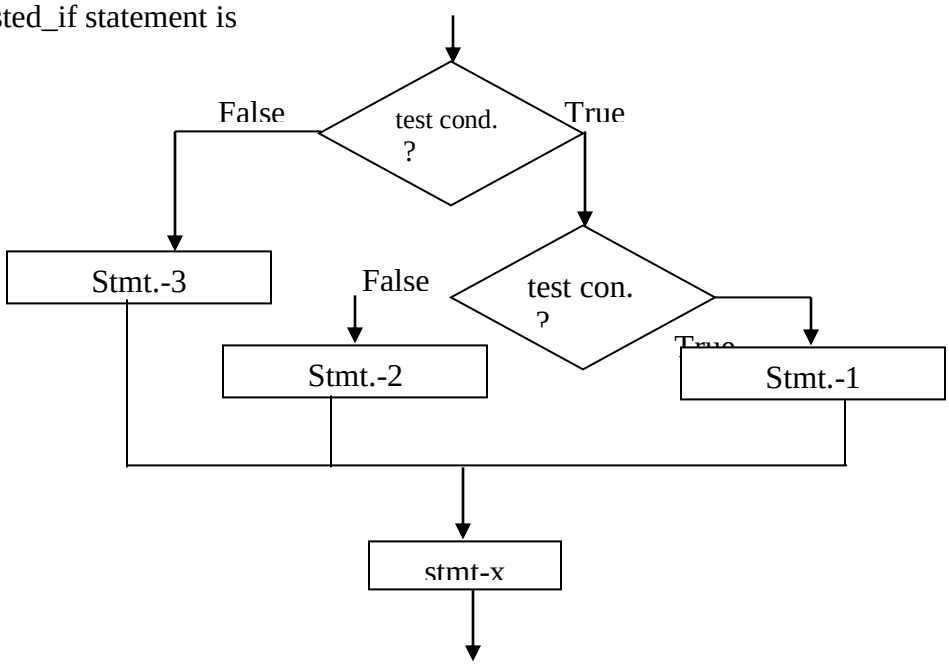
Example

```
If(a>b)
{
printf("a is largest");
}
else
{
printf("b is largest");
}
```

Nested if…else statement

The general form of nested_if statement is

```
if(test condition1)
   {
if (test condition2)
   {
     statement-1;
   }
     else
   {
     statement-2;
   }
   }
    else
{
   statement-3;
 }
   statement-x;
```



Example1: Write a program to find the Largest of 3 numbers

```
#include<stdio.h>
main()
        {       int a, b, c;
                scanf("%d %d %d", &a, &b, &c);
        if(a>b)
       {
        if(a>c)
                printf("a=%4d\n", a);
                else
                printf("c=%4d\n", c);
                }
                else
                {
        if(b>c)
                printf("b=%4d\n", b);
                else
                printf("c=%4d\n", c);
                }
                }
```

Example2
Write a C program to generate and print first 'n' FIBONACCI numbers.

```
#include<stdio.h>
#include<conio.h>

main()
{
int  i, prev=0,curr=1,next,n;
clrscr();
printf("\n Enter a number");
scanf("%d",&n);
if(n==1)
printf("%d",prev);
else
if(n>=2)
{
printf("\n%d\t%d",prev,curr);
for(i=3;i<=n;i++)
{
next=prev+curr;
printf("\t%d",next);
prev=curr;
curr=next;
}}
getch();
}
```

Example 3: Write a C program to reverse a given integer number and check whether it is a palindrome or not. Output the given number with suitable message.

```
#include <stdio.h>
#include<conio.h>

main()
{
int n,n1,rev=0,i,lastdigit;
clrscr();
printf("\n Enter a number");
scanf("%d",&n);
n1=n;
while(n>0)
{
lastdigit=n%10;
rev=rev*10+lastdigit;
n=n/10;
}
printf("\n The reverse of %d is %d",n1,rev);
if(n1==rev)
printf("\n the given number %d is a Palindrome",n1);
else
printf("\n the given number %d is not a Palindrome",n1);
getch();
}
```

Example 4: Write a C program to find the value of sin (x) using the series x-x3/3!+x5/5!... upto N terms accuracy(without using user defined functions). Also print sin(x) using library functions.

```
#include<stdio.h>
#include<math.h>

main()
{
int n,i,j=1,k;
long fact;
float sum=0;deg,rad;
scanf("%f",&deg);
rad=(22.0/7 * deg)/180;
printf("Enter the no of terms");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
k=1,fact=1;
while(k<=j)
{
fact=fact*k;
```

```
k++;
}
sum=sum+(pow(-1,i-1)*pow(rad,j))/fact;
j=j+2;
}
printf("\n Value of sine series upto %d terms is %f",n,sum);
printf("\n value of sine series using library function is %f",sin(rad));
getch();
}
```

The switch statement:

C is supported by built-in multiway decision statement known as switch. The switch statement tests the value of a given variable (expression) against a list of case values and when a match is found, a block of statements associated with that case is executed

The general form of the switch statement is

```
Switch(expression)
        {
                case value1:    Block-1
                                break;
                case value2:    block-2
                                break;
                default:        default-block
                                break;
        }
        statement-x;
```

The break statement after a block indicates the control to exit from the case and jump to statement-x; following the switch and the default is an optional case if exist the control will jump to it when no other case value matches to that of expression

Example

```
Percent=marks/10
Switch(percent)
{
        case 10:
        case 9:
        case 8:
        case 7: grade="FCD";
                break;
        case 6: grade="FC";
                break;
        case 5: grade="SC";
                break;
        case 4: grade="Fail";
                break;
}
        printf("%s\n", grade);
```

Example 1: Write a C program to simulate a simple calculator that performs arithmetic operations like addition, subtraction, multiplication and division only on integers. Error message should be reported, if any attempt is made to divide by zero. Use switch statement.

```
#include <stdio.h>
#include <math.h>
#include <conio.h>

main()

{

float a,b,result;
int ch;
clrscr();
printf("\n1.Addition");
printf("\n 2. Subtraction");
printf("\n 3. Multiplication");
printf("\n 4. Divison");
printf("\n 5. Exit");

printf("Enter your choice");
scanf("%d",&ch);

switch(ch)

{

case 1: printf("\n Enter 2 numbes");
        scanf("%f%f", &a,&b);
        result=a+b;
        break;

case 2: printf("\n Enter 2 numbes");
        scanf("%f%f", &a,&b);
        result=a-b;
        break;

case 3: printf("\n Enter 2 numbes");
        scanf("%f%f", &a,&b);
        result=a * b;
        break;

case 4: printf("\n Enter 2 numbes");
        scanf("%f%f", &a,&b);
        if (b==0)
        {
        printf("\n Cannot divide by zero");
```

```
        getch();
        exit(0);
        }
        result=a/b;
        break;

case 5: exit(0);

default:
        printf("Wrong choice\n");
        getch();
}

Printf("The result is %f", result);
Getch();
}
```

GOTO Statement:

For any unconditional jump of control from one statement to other go to can be used. The go to is followed by a label to identify the location to which it has to jump. The label is any valid variable name and must be followed by condition.

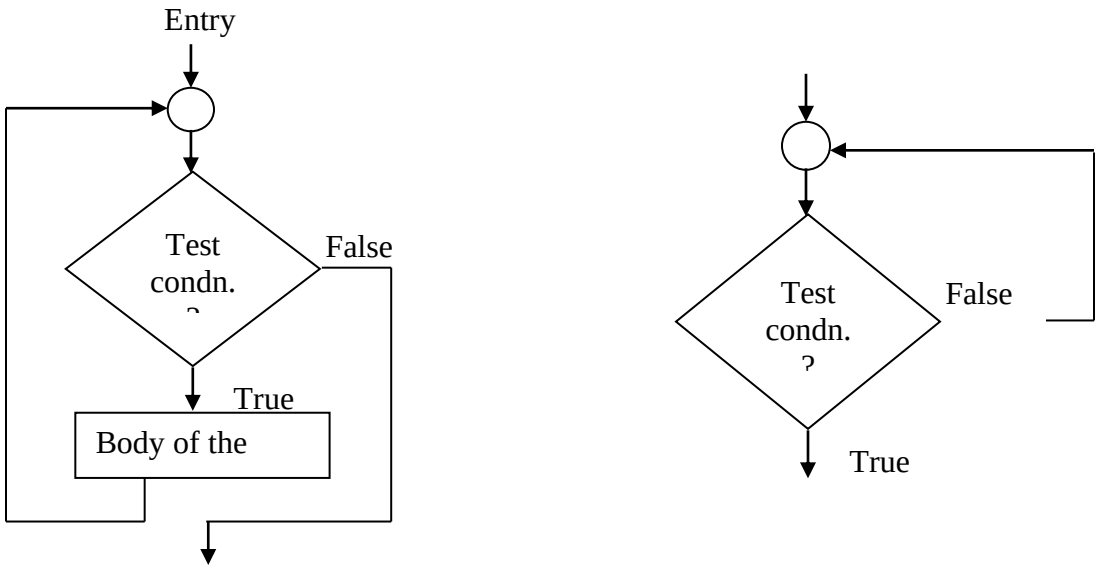
LOOP CONTROL STRUCTURE:

A sequence of statements executed till the termination condition is satisfied is called as a loop. A program loop consists of two segments

1. Body of the loop
2. Control statement.

The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.

Two types of loop: Depending on the position of the control statement in the loop, a control structure may be classified as **entry-controlled** for or the **exit-controlled** loop



a) Entry Control b) Exit Control

In the entry-controlled loop, the control conditions are tested before the start of the loop execution upon the if executes body of loop.
In the exit-controlled loop, the test is performed at the end of the body of the loop. Body of loop is executed at least once before testing condition.

The C language provides the three loop constructs for performing loop operations they are

1. The while statement
2. The do statement
3. The for statement

The for statement:

The for loop is an entry-controlled loop providing a more concise loop control structure.
The general form of the for loop is

```
For(initialization; test-condition; increment)
{
        body of the loop
}
```

Where the control variable is initialized first example c=0 or c=1 etc,. and using test-condition the control variable is tested resulting the true or false and executing body of loop till test condition result false.

Once the body of loop is executed the current value is incremented by increment step and the new value is assigned to control variable (current value) resulting in for test condition and execution of body of loop

Features of for loop:

1. Initialization, testing and incrementing are placed in a single statement
2. More than one variable can be initialized at a time in the for statement
   Ex. i=1 } for(i=1; j=0; j<10; ++j)
   For(k=0; j<10; j++) }
3. Expressions are allowed in the assignment statement of initialization and increment section
   Ex. For(x=(i+j)/2; x>0; x=x\2)

The While statement

The general format of the while statement is

```
While(test_condition)
{
        body of the loop
}
```

The while is an entry-controlled loop statement. The test condition is evaluated and if the condition is true, then the body of the loop is executed until the test condition becomes false on exit the program continues with the statements immediately after the body of the loop

Example

```
while(n<10)
{
        sum=sum+n;
        n=n+1;
}
printf("sum=%d\n", sum);
```

The do statement

The general form of the do statement is

```
do
 {
        body of the loop
 }
        while(test_condition);
```

The body of the loop is executed at least once before checking test condition hence it is an exit-controlled loop

Example

```
do
{
printf("input the number\n")
number=getnum();
}
while(number>0);
```

Example 1:Write a C program to find whether the given number is prime or not and output the given number with suitable message.

```
#include<stdio.h>
#include<math.h>
main()
{
int n,i, flag=0;
printf("enter the number\n");
scanf("%d", &n);
if(n<=1)
{
printf("%d is not a prime number\n", n);
exit(1);
}
for(i=2; i<=n/2; i++)
{
if((n%i)==0)
{
flag=1;
break;
}
}
if(flag==0)
printf("%d is a prime number\n");
else
printf("%d is not a prime\n", n);
}
```

ARRAYS

An array is a group of related data items that share a common name. It can also be defined as an ordered collection of homogenous data items where each item is of type int, float, char, etc and is referred to by its position in the group in the form of subscript or index

Declaration of Array: must be declared before using it in the program statement

General form: type specifier var_name[size];

Initialization of array:

The array variables can also be initialized similar to an ordinary variable. The general form of initialization of array is

Static type array_name[size] = {list of value};

Example static int num[3]={0,1,2}; will declare variable num of type array assigned with values 0,1,2 as elements.

Example static int counter[]={1,1,1,1}

Static char name[]={'G', 'N', 'D', 'E', 'C'}

Disadvantages:

1. Selected items in an array cannot be initialized
2. It is difficult to initialize large no. of array elements.

One-Dimensional array:

The items in an array can be given one variable name using only one subscript and such a variable is called a single-subscripted variable or a one-dimensional array.

Ex. int num[10];

Example:

int class[10]; declares class as int containing 10 integer constants.

Example:

char name[10]; declares the name as a character array (string) that can hold a max. of 10 characters.

Example 1: Write a C program to read a set of elements in an array and print them.

```
#include<stdio.h>

main()
{
int a, i, a[10];
printf("enter the size of array\n");
scanf("%d", &n);
printf("enter the elements of array\n");
for(i=0; i<n; i++)
{
scanf("%d",&a[i])'
}
printf("elements of array are\n");
for(i=0; i<n; i++)
}
printf("no.[%d]=%d\n", i, a[i]);
```

Example 2: Write a program to read N integer (0, +ve, -ve) into an array A and

i. Find sum of Negative numbers.

ii. Find sum of positive numbers.

iii. **Find avg. of all input numbers**.

Output the various results computed with proper headings.

```
#include<stdio.h>
main()
{
int a[25], n, i, psum=0, nsum=0, sum=0;
float avg;
clrscr();
printf("enter the no. of elements in array\n");
scanf("%d", &n);
printf("enter the elements of array\n");
for(i=0; i<n; i++)
{
scanf("%d", &a[i]);
}
/* computation*/
for(i=0; i<n; i++)
{
if(a[i]>0) psum=psum+a[i];
if(a[I]<0) nsum=nsum+a[i];
sum=sum+a[i];
}
avg=sum/(float)n;
}
printf("\n the sum of all positive integers is %d", psum);
printf("\n the sum of all negative integers is %d, nsum);
```

```
printf("\n the avg of all integers is %d"m avg);
getch();   }
```

Code for Searching a number

```
for(i=0; i<n; i++)
{
if(a[i]==key) flag=1;
}
if(flag==0)
printf("element not found");
else
printf("element found");
```

Code for Sorting a number

```
for(i=0; i<n; i++)
{
for(j=0; j<n-I-1; j++)
{
if(a[j]<a[j+1])
{
temp=a[j];
a[j]=a[j+1])
a[j+1]=temp;
}
}
}
```

Example 3:Write a C program to accept N numbers in ascending order and to search for a given number using Binary Search. Report success or failure in the form of suitable messages.Binary search

```
#include<stdio.h>
#include<math.h>
void main()
{
int n, i, a[20], key, low, high, mid, flag=0;
clrscr();
printf("enter the number of elements");
scanf("%d", &n);
printf("enter the elements is ascending order in array\n");
for(i=0; i<n; i++)
{
scanf("%d", &a[i]);
}
printf("enter the element to be searched");
scanf("%d", &key);
```

```
low=0;
high=n-1;
while(low<high)
{
mid=(low+high)/2;
if(key>a[mid])
low=mid+1;
else if(key<a[mid])
high=mid-1;
else
{
flag=1;
break;
}
}
if(flag==1)
printf("\n element is found");
else
printf("element not found");
getch();
}
```

Example 4: Write a C Program to input N real Numbers and to find Mean, Variance and Standard Deviation,

where mean = (∑x[i]/N, Variance=(∑x[i]-mean)2)/N, Standard Deviation=√variance, for I<= i<=N.

Output the computed results on different lines.

```
#include<stdio.h>

main()
{
int n,i,a[20], sum=0, sumv=0;
float mean, sd, var;
clrscr();
printf("enter the numbers of elements\n");
scanf("%d", &n);
printf("enter the elements of the array\n");
for(i=0; i<n; i++)
{
scanf("%d",&a[i]);
}
for(i=0; i<n; i++)
{
sumv=sumv+pow((a[i]-mean)2);
}

var=sumv/(float)n;
sd=sqrt(var);
printf("\n mean=%f", mean);
```

```
printf("\n variance=%f", var);
printf("\n standard deviation="%f",sd);
getch();
}
```

Example 5: Write a C program to sort N numbers in ascending order using bubble sort and print both the given array and the sorted array with suitable headings:

```
#include<stdio.h>

main()
{
int i,j,temp,n, a[10];
clrscr();
printf("enter the size of array\n");
scanf("%d", &n);
printf("enter the elements into array\n");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
/*sorting*/
for(i=0; i<n; i++)
for(j=0; j<n; j++)
if[j]<a[j+1]
{
temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
}
printf("sorted elements are\n");
for(i-0; i<n; i++)
printf("%d",a[i])
getch();
}
```