C.P. PATEL & F.H. SHAH COMMERCE COLLEGE
(MANAGED BY SARDAR PATEL EDUCATION TRUST)
BCA, BBA (ITM) & PGDCA PROGRAMME
PGDCA SEM I
PS01CDCA22: C AND DATA STRUCTURE
UNIT 3 Structured Programming and Advanced Computing

- **Loop statements :-**
- When there is the need to execute same type of statements multiple times the looping statements or iterative statements are used.
- 'A loop' is a part of code of a program which is executed repeatedly.
- A loop is used using condition. The repetition is done until condition becomes true.
- A loop declaration and execution can be done in following ways:
    - Check condition to start a loop
    - Initialize loop with declaring a variable.
    - Executing statements inside loop.
    - Increment or decrement of value of a variable.

Types of looping statements:

- Basically, the types of looping statements depend on the condition checking mode.
- Condition checking can be made in two ways as: Before loop and after loop. So, there are 2(two) types of looping statements:
    - **Entry controlled loop**
    - **Exit controlled loop**

1. Entry controlled loop:

- In such type of loop, the test condition is checked first before the loop is executed.
- Some common examples of this looping statements are :
    - **while loop**
    - **for loop**

2. Exit controlled loop:

- In such type of loop, the loop is executed first.
- Then condition is checked after block of statements are executed.
- The loop ***executed at least one time*** compulsorily.

Some common example of this looping statement is:

- **do-while loop**

1) For loop:

- This is an entry controlled looping statement.
- In this loop structure, more than one variable can be initialized.
- One of the most important features of this loop is that the three actions can be taken at a time like:
    - Variable initialization,
    - Condition checking and
    - Increment/ Decrement.

- The for loop can be more concise and flexible than while and do-while loops.

Syntax:

```
for(initialization; test-condition; incre/decre)
{
        statements;
}
Stat -X
```

- In above syntax, the given three expressions are separated by ';' (Semicolon).
- Syntax of for loop 3 parts will be include.
  1) Initialization
  2) Condition
  3) Increment /decrement
- In initialization part the counter value will be initialize with the use of the assignment operator
  (=).
  Ex: = int i=1
- In condition part the condition will be checked with the use of the relational operator. And

base on the condition the body of the loop will execute. If the condition is true then the body

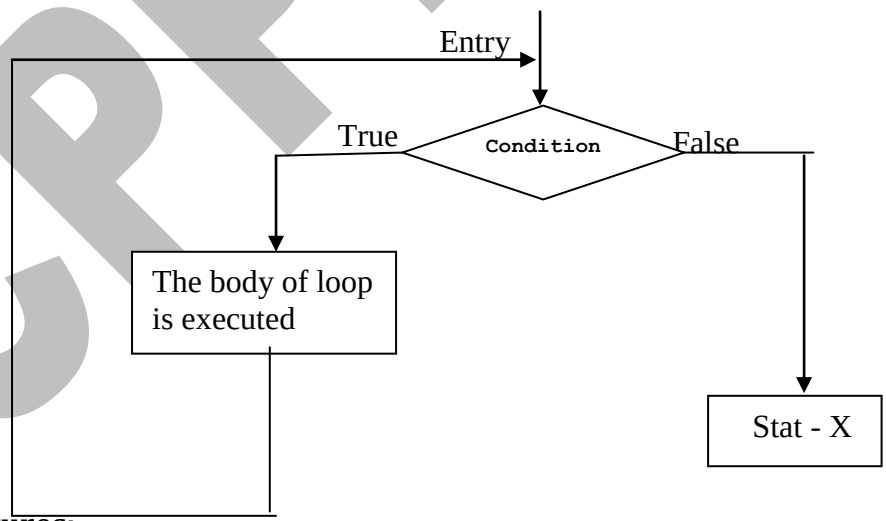
of the loop will be execute and again the condition is check this process is continue until

the condition is false.

Ex: = i<10

- In increment or decrement part the counter value either increment or decrement base on the
  condition.

Ex: = i++



Features:

- It is a more precise loop
- Its structure is easier to use.
- We can initialize more than one variable.
- We can increment more than once.

- More than two conditions can be used.
- **The example of for loop is given below:**

```
#include<stdio.h>
#include<conio.h>
Void main()
{
        int i;
        clrscr();
        for(i=1;i<=10;i++)
        {
        printf("%d",i);
        }
        getch();
}
```

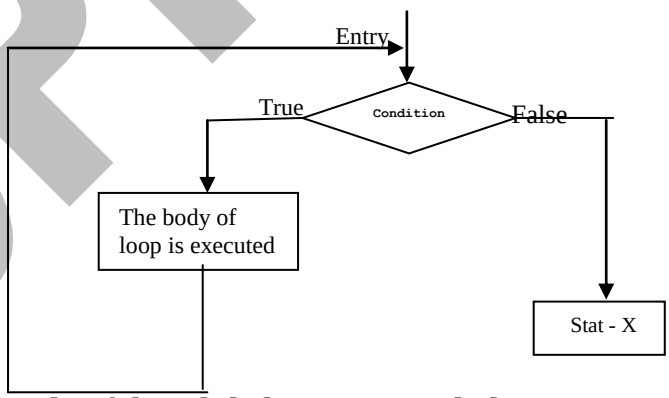
While loop:

- This is an entry controlled looping statement.
- It is used to repeat a block of statements until condition becomes true.

Syntax:

```
while (condition)
{
        statements;
        increment/decrement;
}
```

- In above syntax, the condition is checked first.
- If it is true, then the program control flow goes inside the loop and executes the block of statements associated with it.
- At the end of loop increment or decrement is done to change in variable value.
- This process continues until test condition satisfies.



- **The example of the while loop is given below:**

```
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
        int count=1,fact=1,n;
        clrscr();
        printf("input any number ");
        scanf("%d",&n);
        while(count<=n)
        {
        fact=fact*count;
        count++;
        }
        printf("facorial of %d is:%d",n,fact);
        getch();
}
```

Do-While loop:

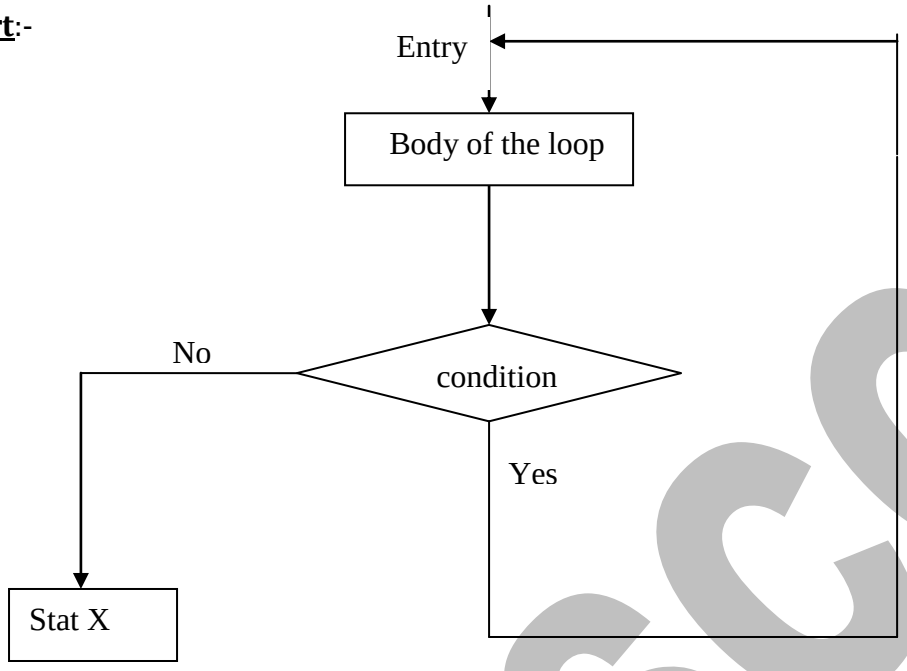
- This is an ***exit controlled*** looping statement.
- Sometimes, there is need to execute a block of statements first then to check condition.
- At that time such type of a loop is used.
- In this, block of statements are executed first and then condition is checked.

Syntax:

```
do
{
        statements;
        (increment/decrement);
} while (condition);
```

- In above syntax, the first the block of statements are executed.
- While statement is executed at the end of loop.
- If the resultant condition is true then program control goes to evaluate the body of a loop once again.
- This process continues till condition becomes true.
- When it becomes false, then the loop terminates.

Flowchart:-



Note: The while statement should be terminated with ; (semicolon). ***And do statement will be executed at least one time.***

Break Statement:

- Sometimes, it is necessary to exit immediately from a loop as soon as the condition is satisfied.
- When break statement is used inside a loop, then it can cause to terminate from a loop.
- *The* statements *after break statement are skipped.*

Syntax: The syntax for using break statement can be as below:

```
while (condition)
{
   _ _ _
   _ _ _ _
     break ;
   _ _ _ _
   _ _ _
}
```

Arrays

Def :- " An array is a fixed sized sequential collection of element of using the same Data type. "

Or

"An array is a group of related data items of the same data types"

- An Array may be defined as the collection of the same type of elements that are stored in the contiguous memory locations.
- For example, if we have to store 5 integer values, we don't have to declare 5 variables with different identifiers but we can define an array of integer type and can store the values in array directly with a unique identifier.
  C language can support the three typed of array.
  1) One – dimensional array
  2) Two – dimensional array

❖ One – dimensional array :-

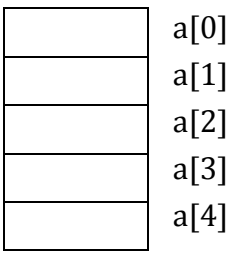
Def: - "The array using only one subscript is known as the one dimensional array "

➢ Declaration of One dimensional array :-

The one dimensional array can be declare with the use of the following syntax

```
Data_type array_name [size];
```

- The data_type may be int, float or character type.
- Array_name is the valid user define name.
- The size indicates maximum number to be stored in the array.
- Ex : - int a [5];
  float a [5];
  char name [10];
- Computer can reserve the five storage location as shown in the below fig.

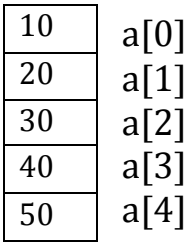


- The value to the array element can be assigned as follow.

a[0] = 10;
a[1] = 20;
a[2] = 30;

a[4] = 40;
a[5] = 50;

- This would cause the array **a** to store the value as shown in the below fig.



- The subscript of the any array will be start form the 0
- So the first element of array a will be stored at a[0].
- The last element of any array will be stored at **n-1** where n is the size of the array.

➢ **Initialization of 1-Dimensional Array**:-

The one dimensional array can be initialized at following stages:
1) Compile Time
2) Run -Time

❖ **Compile Time Initialization:-**

The general form of compile time initialization is given below.

Syntax: Data_type Array_name [size] = {list of array};

The data type may be int, float or char and the values in the list are separated by **(,)**.

For e.g.:-

```
int a[5]={10,20,30,40,50};
float a[3]={1.1,2.1,3.1,};
char a[4]={'a', 'b', 'c', '\0'};
int a[ ]={10, 20, 30};
```

Write a program to find the sum of five elements using the compile time initialization of One dimensional array.

```
#include<conio.h>
#include<stdio.h>
void main()
{
 int a[5] = { 10,20,30,40,50 },sum=0;
 clrscr();
 sum=a[0]+a[1]+a[2]+a[3]+a[4];
```

```
 printf("The sum of the 5 element is :%d",sum);
 getch();
}
```

Output :-
The sum of the 5 element is :150

- **Run-Time Initialization**:-

The general form of compile time initialization is given below

Syntax: `Data_type Array_name [size];`

The data type may be int, float or char
For access the array element using the run – time initialization use the loop.

Ex: - Write a program to find the sum of five elements using the run time initialization of One dimensional array.

```
#include<conio.h>
#include<stdio.h>
void main()
{
  int a[5],i,sum=0;
 clrscr();
 for(i=0;i<5;i++)
 {
 printf("\nEnter the array element :");
 scanf("%d",&a[i]);
 }
 for(i=0;i<5;i++)
 {
 printf("\nThe %d array element is:%d",i+1,a[i]);
 sum = sum+a[i];
 }
 printf("\nThe sum of array element is %d",sum);
 getch();

}
```

Output :-

Enter the array element: 10
Enter the array element: 20
Enter the array element: 30
Enter the array element: 40
Enter the array element: 50

The 1 array element is: 10
The 2 array element is: 20
The 3 array element is: 30
The 4 array element is: 40
The 5 array element is: 50

The sum of array element is 150

❖ Two Dimensional Array:-

Def :- "The array using only two subscript is know as the two dimensional array".

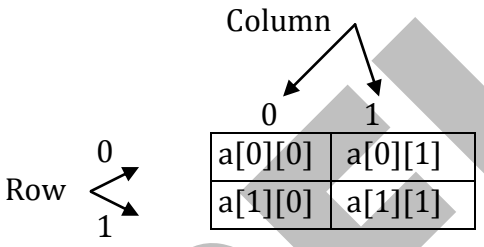
➢ Declaration of two dimensional array :-

The two dimensional array can be declare with the use of the following syntax

Syntax: – `Data_type array_name [row- size][column-size] ;`

- The data_type may be int, float or character type.
- Array_name is the valid user define name.
- The size indicates maximum number to be stored in the array.
- Ex : - int a [2][2];
  float a [2][2];

Computer can reserve 4 storage locations as shown below:



The value to the array element is assigned as follows:

a[0] [0]=10;
a[0] [1]=20;
a[1] [0]=30;
a[1] [1]=40;

This would cause the array **a** to store the value as shown in the below fig.

	0	1
0	10	20
1	30	40

- The subscript of the any array will be start form the 0
- So the first element of array a will be stored at a[0][0].

➢ Initialization of 2-Dimensional Array:-

The one dimensional array can be initialized at following stages:

1) Compile Time
2) Run -Time

❖ **Compile Time Initialization:-**

The general form of compile time initialization is given below.

Syntax:

```
Data_type Array_name [row- size][column-size] = {list of
array};
```

The data type may be int, float or char and the values in the list are separated by **(,)**.

For e.g.:-
```
int a[2] [2]= {10, 20, 30, 40};
int a[2] [2]= {{10,20},{30,40}};
int a[ ] [ ]={10, 20, 30, 40};
float a[2] [2]={1.1, 2.1, 3.1, 4.1};
```

❖ **Run-Time Initialization**:-

The general form of compile time initialization is given below

Syntax: `Data_type Array_name [row- size][column-size];`

The data type may be int, float or char
For access the array element using the run – time initialization use the loop.

❖ **Multi-Dimensional Array**:-

The general form of multi dimensional array is:

Syntax:-Data_type array_name[s1] [s2] [s3]….s[n];

Where s1, s2…..sn, is the subscript use in the multi-dimensional array.

For e.g. : –
```
int a [2] [2] [2];
float a [2] [3] [4];
char a [5] [6] [7];
```

What are the advantages of using an Array?

Following are main advantages of using an array:

1. Arrays are the most convenient way of storing the fixed amount of data that can be accessed in an unpredictable fashion.
2. Arrays are also one of the most compact data structures as in for storing the 50 elements in an array of integer type it will take the memory equivalent to the amount of memory required to store 50 elements in addition to that a few overhead bytes for whole array.

3. Another advantage of array is that iterating through an array is more faster than compared to other data structure types like linked list because of its good locality of reference.

What are the disadvantages of using an Array?

Following are the main disadvantages of using an Array:

1. Performing various operations like insertion and deletion is quite a stiff task in arrays as after inserting or deleting an item from an array every other element in array have to move forward or backwards prior to your operation, which is a costly affair.
2. In case of the static arrays the size of the array should be know up priori i.e. before the compile time.
3. C language does not have a checking mechanism for the array sizes.

- **String handling.**

String:- **A sequence of characters is called string**. It can be used for storing and manipulating text such as words, names and sentences. A string is an array of characters.

Declaring string variable:

C does not support the string as a data type.

The general form of declaration of a string variable is

```
char   StringName [ size ] ;
```

StringName is valid variable name and size indicates the maximum number of elements the string can hold.

Initializing string variable:

The string is initialized when they are declared as follow.

```
char name[20] = {'T', 'u', 's', 'h', 'a', 'r' } ;
        or
char name[20] = "Tushar" ;
```

When the compiler assigns a character string to a character array, it automatically appends a **null character ('\0')** at the end of the string. So the size should be equal to the maximum number of the character.

The size may be omitted. The following is also valid initialization.

```
char name[] = {'T', 'u', 's', 'h', 'a', 'r' } ;
              or
char name[] = "Tushar" ;
```

Reading the string form the screen .

The scanf ( ) and gets ( ) function are used to read the string from the screen.

Example:

```
char name[20];
printf(" \n Enter the name of the student :");
scanf( "%s" , name);
```

- **The ampersand (&) is not use to read the string value form user.**

- The problem with the scanf function is that its stop reading on first blank space.
- There for, if the input text is “hi hello” Then the string can contain only “hi”
- So we use the gets ( ) function to remove the problem.

Example:

```
char name[20];
printf(“ \n Enter the name of the student :”);
gets(name);
```

Display string on screen.

The printf ( ) and puts function is used to printf string on the screen.

Example:

```
char name[20];
printf(“ \n Enter the name of the student :”);
scanf( “%s” , name);
printf( “ The name of the of student is %s”,name);
```

Or

Example:

```
char name[20];
printf(“ \n Enter the name of the student :”);
scanf( “%s” , name);
puts(name);
```

Operation on a string ::

There are following operation on a string can be carried out:

1. Length :: To count number of charcters.
2. Copy :: To copy contents of one string to another.
3. Compare :: To compare two string
4. Concatenation :: Combine two string.
5. Upper :: Convert given string into upper case letter.
6. Lower :: Covert given string into lower case letter.
7. Reverse :: Reverse the contents of the given string.

String Manipulation:

C++ has several built-in functions such as strlen(), strcat(), strlwr(), etc. for string manipulation. To use this function the header file string.h must be included in the program using the statement.

```
#include<string.h>
```

1) String length:

The string function strlen() returns the length of a given string. A string constant or an array of characters can be passed as an argument.The length of the string excludes the end-of-string character(NULL).

Syntax: - **Var_name = strlen (string);**

Example:

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

```
void main()
{
  char name[20];
  int n;
  clrscr();

  printf("\n Enter the name of the student :");
  scanf("%s",name);
  n=strlen(name);
  printf("\nThe length of the string is :%d",n);
  getch();
}
```

Output :-

```
Enter the name of the student :jatan
The length of the string is :5
```

2) <u>String Copy</u>:

The string function strcpy() copies the contents of one string to another. It takes two arguments the first argument is the destination string array & the second argument is the source string array.The source string is copied into the destination string.

Syntex :- **strcpy(string1,string2);**

Example:-

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
  char name[20],city[20];
  int n;
  clrscr();
  printf("\n Enter the name of City     :");
  scanf("%s",name);
  strcpy(city,name);
  printf("\nThe name of the city is :%s",city);
  getch();
}
```

Output:-

```
 Enter the name of City  :anand
 The name of the city is :anand
```

3) <u>String Concatenation</u>:

The string function strcat() concatenates two strings resulting in a single string. It takes two arguments which are the destination & source strings. The destination & source strings are concatenated & the resultant string is stored in the destination (first) string.

Syntex :- strcat(string1,string2)

Example:-

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
  char str1[20],str2[20];
  int n;
  clrscr();
  printf("\n Enter First string:");
  scanf("%s",str1);
  printf("\n Enter Second string:");
  scanf("%s",str2);

  printf("\nThe Concatnation String is :%s",strcat(str1,str2));
  getch();
}
```

Output:-

```
 Enter First string: hello
 Enter Second string:hi
The Concatenation String is :hellohi
```

4) <u>String Comparison</u>:

The string function strcmp() compares two strings, character by character. It accepts two strings as parameters & returns an integer whose value is

- < 0 if the first string is less than the second
- ==0 if both are identical
- >0 if the first string is greater than the second

Syntax :- Strcmp(string1,string2);

Example:-

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

void main()
{
  char str[20],str1[20];
  int i;
  clrscr();
  printf("Enter first string");
  scanf("%s",str);
  printf("Enter second string");
  scanf("%s",str1);
  i=strcmp(str,str1);
```

```
  printf("The out put of comparision is==>%d",i);
  getch();
}
```

Output:-

```
Enter first string hello
Enter second string hillo
The out put of comparision is==>-4
```

5) <u>String to Upper/Lower Case</u>:

The function strlwr() & struper() convert a string to lower case & upper case respectively & return the address of the converted string.

Syntax:- strupr(string name);
strlwr(string name);

Example:-

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
  char name[20];
  clrscr();
  printf("\n Enter any string in lowercase:");
  scanf("%s",name);
  printf("\nThe string is:%s",strupr(name));
  getch();
}
```

Output:-

```
Enter any string in lowercase:hello
The string is:HELLO
```

6) <u>String to Reverse Case</u>

The function strrev () covert the string in to reverse order.

Syntax: - strrev (String name);

Example:-

```
 #include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
  char name[20];
  clrscr();
  printf("\n Enter any string:");
  scanf("%s",name);
  printf("\nThe Reverse string is:%s",strrev(name));
```

```
    getch();
}
```

Output:-

```
Enter any string:hello
The Reverse string is:olleh
```

- **Function :-**

1. It is difficult to implement a large program.
2. To implement such a program easily, it should be split into a number of independent tasks, which can be easily designed, implemented and managed.
3. This process of splitting a large program into small manageable tasks and designing them independently is called **modular programming OR divide-and conquer technique.**

Function: "A function is a set of program statements that can be processed independently."

OR

Function: "A function is a self contain block of code that performed particular task "

Generally C Programming Language provides two types of functions:

1. **Standard Library Functions:** The functions specified into C Library are known as the library functions ex.: clrscr(), getch(), printf(), scanf(), etc.
2. **User Defined Functions:** A function that is designed by the user using the executable statements as well as the standard library functions, to achieve some task.
   ex.: void main ()

Function Element:-

Every function has the following elements

1. **Function declaration or prototype**
2. **Function definition**
3. **Function call**

1. Function declaration or prototype:

- The declaration of a function before main() function is a function prototype.
- In above program statement – ex. *void max(int x, int y);* is a function prototype.
- The elements of the functions declarations or prototype are:
  1) return type
  2) Function name
  3) Parameter List
  4) Semicolon (at the end)
- The syntax of the function declarations is as below:
  type function-name (**parameter-list**);
- For example:

```
int max(int x, int y);
void sum(void);
void sum( );
```

2. Function definition:

- Function definition includes function declaration (or prototype) and function body.
- The elements of the function definition are:

1. Return type
2. Function name
3. Parameter list

Function Header

4. Local variable declaration
5. Executable statements
6. return statement

Function Body

```
function_type function_name(parameter_list)
{
        local variable declaration;
        executable statements;
        ----------------------------------
        ----------------------------------
        return statement;
}
```

Function Header: Function header is the combination of the first three elements of the function elements, Return type, function name and parameter list.

1. **Function Type (Return Type):**
   The function generally returns some value to the calling function. int, float, etc.
   When the return type is not specified C considers the default type as int.
   When any function does not return any value *void* keyword is used for that.
2. **Function name:**
   The name of the function is any identifier that is valid according to the naming conventions of the variables in C.
3. **Function parameters:**
   The parameters specified in the function call are known as **actual parameter**.
   The parameters specified in the calling function are known as **formal parameter.**
   The parameters are used to pass the values to the functions

Example: Write a C program to find maximum of two integer numbers.

```
#include<stdio.h>
#include<conio.h>
void max(int x, int y); //function declaration or prototype
void main()
{
   int  a, b, max;
    clrscr();
    printf("Enter a…");
    scanf("%d",&a);
```

```
        printf("Enter b...");
        sacnf(%d", &b);
        max(a, b);  //Function Call
    getch();
}
void max( int x, int y) // function definition
{
    if ( x > y )
            printf("Maximum number is %d", x);
    else
            printf("Maximum number is %d", y);
}
```

The parameters a and b are actual parameter. The parameters x and y are formal parameter. The scope of formal parameter is limited to its function only.

Function Body: Function body is the combination of the last three elements of the function elements, local variables, executable statements, return statement.

a) **Local Variables:** The variables declared inside the function are known as the local variable of that function.
b) **Executable statements:** Executable statements specify the task of the functions.
c) **return statement:** This statement can return the value.

3. Function call: A function call is specified by the function name followed by the arguments enclosed in parentheses and terminated by a semicolon.

Syntax: - function_name (parameter list);

In above program, function call statement is max(a, b);
Callee: A function which is called. It is also known as called function.
Caller: A function which calls. It is also known as calling function.

Advantages of the function are
- Modular programming
- Reduction in the amount of work and development time
- Program and function debugging is easier.
- Division of work is simplified due to the use of divide-and-conquer principle.
- Reduction in size of the program due to code reusability
- Function can be access repeatedly without redevelopment, which in turn promotes reuse of code.
- Library of function can be implemented by combining well designed, tested and proven function.

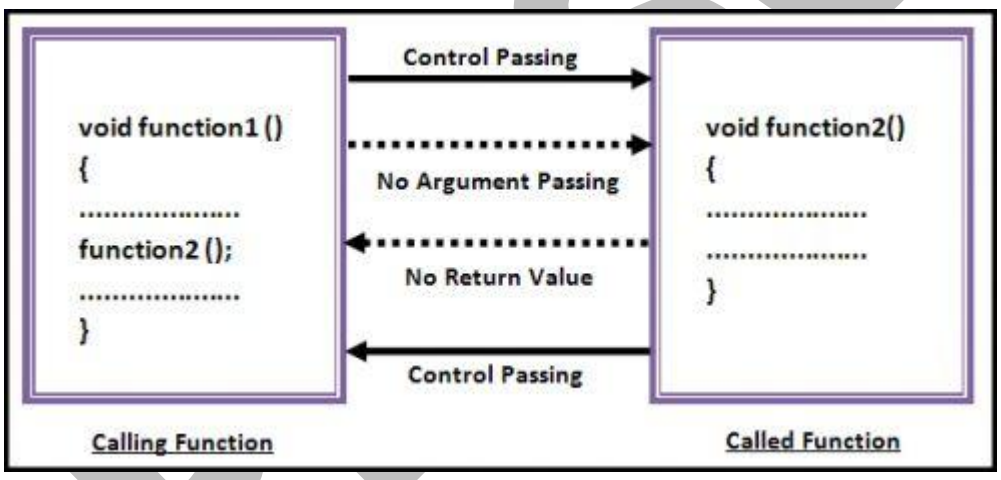
❖ **Categories Of Function :-**

A function, depending on the argument and return value the function can be classified in
to following category

1) Function with no argument and no return value
2) Function with argument and no return value
3) Function with no argument and with return value
4) Function with argument and with return value

1) Function with no argument and no return value

A C function without any arguments means you cannot pass data (values like int, char etc) to the called function. Similarly, function with no return type does not pass back data to the calling function. It is one of the simplest types of function in C. This type of function which does not return any value cannot be used in an expression it can be used only as independent statement. The following fig indicates that there is only a transfer the control but not data.



Example:-

```
#include<conio.h>
#include<stdio.h>
void sum();          // Function declaration
void main()
{
 clrscr();
 sum();              // Function calling
 getch();
}
void sum()                // Function definition
{
 int a,b,c;
 printf("\nEnter the value of a:");
 scanf("%d",&a);
 printf("\nEnter the value of b:");
 scanf("%d",&b);
```

```
 c=a+b;
 printf("\nThe sum of the number is =%d",c);
}
```

Output:-

```
Enter the value of a:10
Enter the value of b:20
The sum of the number is =30
```

2) Function with argument and no return value

In our previous example what we have noticed that "main()" function has no control over the UDF "sum ()", it cannot control its output. Whenever "main()" calls "sum()", it simply prints line every time. So the result remains the same.

A C function with arguments can perform much better than previous function type. This type of function can accept data from calling function. In other words, you send data to the called function from calling function but you cannot send result data back to the calling function. Rather, it displays the result on the terminal. But we can control the output of function by providing various values as arguments.

Actual parameter:-

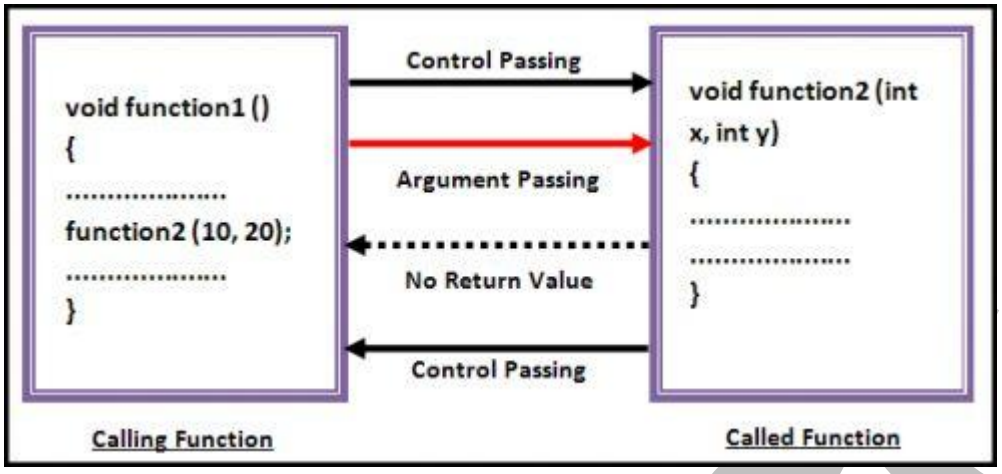
Actual parameter is use in the function calling.
These are the parameters, transferred from the Calling Function/Program (main( ) program) to the Called Function/Program (Function) is called the actual parameter.

Ex:- sum(a,b);

Formal parameter:-

Formal parameter is use in the function definition .
These are the parameters, transferred into the Calling Function/Program (main( ) program) from the Called Function/Program (Function) is called the Formal parameter.

Ex;- void sum(int x, int y)



Example:-

```
#include<conio.h>
#include<stdio.h>
void sum(int,int);    //Function declaration
void main()
{
 int a,b;
 clrscr();
 printf("\nEnter the value of a:");
 scanf("%d",&a);
 printf("\nEnter the value of b:");
 scanf("%d",&b);
 sum(a,b);          //Function call
 getch();
}
void sum(int x,int y)          // Function  definition
{
 int c;
 c=x+y;
 printf("\nThe sum of the number is =%d",c);
}
```

Output:-

```
Enter the value of a:10
Enter the value of b:20
The sum of the number is =30
```

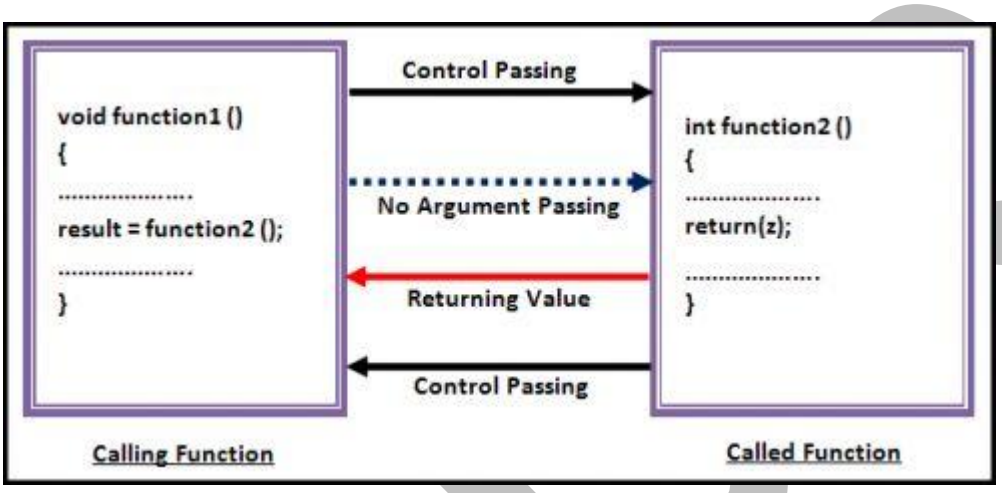
For the above example the a & b is called the **Actual parameter** and x & y is called **Formal parameter.**

3) Function with no argument and with return value

We may need a function which does not take any argument but only returns values to the calling function then this type of function is useful.

When the UDF function is return the value to the main function the return type may be
int,char,or float .

When the UDF function is return the int value then the return type may be int.
Same for the float and char.



Example:-

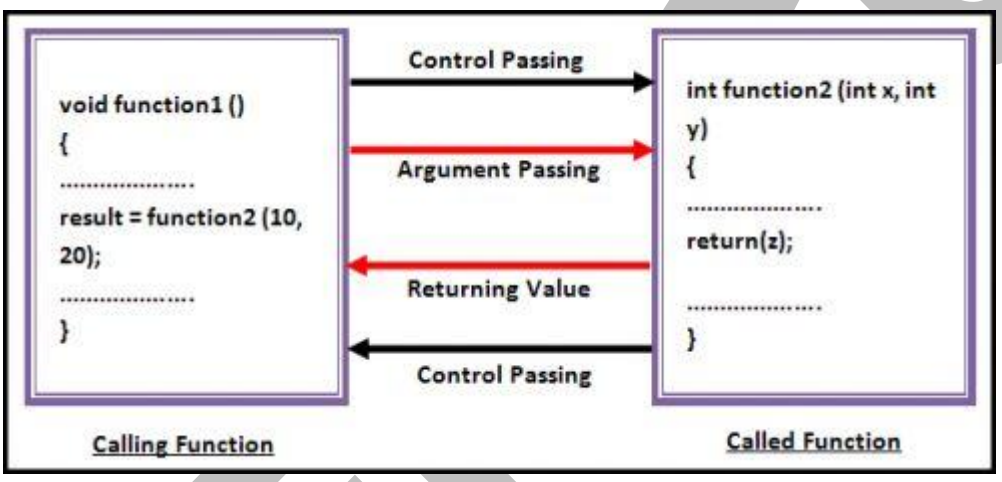
```
#include<conio.h>
#include<stdio.h>
int sum();
void main()
{
 int n;
 clrscr();
 n=sum();
 printf("\nThe sum of the number is =%d",n);
 getch();
}
int sum()
{
 int a,b,c;
 printf("\nEnter the value of a:");
 scanf("%d",&a);
 printf("\nEnter the value of b:");
 scanf("%d",&b);
 c=a+b;
 return(c);
}
```

Output:-
Enter the value of a:10
Enter the value of b:20

The sum of the number is =30

4) Function with no argument and with return value

This type of function can send arguments (data) from the calling function to the called function and wait for the result to be returned back from the called function back to the
calling function. And this type of function is mostly used in programming world because
it can do two way communications; it can accept data as arguments as well as can send back data as return value. The data returned by the function can be used later in our program for further calculations.



Example:-

```
#include<conio.h>
#include<stdio.h>
int sum(int,int);
void main()
{
 int n,a,b;
 clrscr();
 printf("\nEnter the value of a:");
 scanf("%d",&a);
 printf("\nEnter the value of b:");
 scanf("%d",&b);
 n=sum(a,b);
 printf("\nThe sum of the number is =%d",n);
 getch();
}
int sum(int x,int y)
{
 int c;
 c=x+y;
 return(c);
}
```

Output:-
Enter the value of a:10
Enter the value of b:20
The sum of the number is =30

Note:-
Creating the program with the use of the above categories remember the following table.

Categories	In main function	In user define function
1	Call the UDF function	Read the value, Calculation and Print the value
2	Read the value, Call the UDF function	Calculation and Print the value
3	Call the UDF function, Print the value	Read the value, Calculation
4	Read the value Call the UDF function, Print the value	Calculation

Command Line Arguments:

It is possible to pass arguments to C programs when they are executed. The brackets which follow main are used for this purpose.

The syntax for this is:

```
main( int argc, char *argv[] )
{
}
```

argc refers to the number of arguments passed, and *argv[]* is a pointer array which points to each argument which is passed to main. A simple example follows, which checks to see if a single argument is supplied on the command line when the program is invoked.

```
#include <stdio.h>
main( int argc, char *argv[] )
{
        if( argc == 2 )
                printf("The argument supplied is %s\n", argv[1]);
        else if( argc > 2 )
                printf("Too many arguments supplied.\n");
        else

                printf("One argument expected.\n");
}
```

Note that **argv[0]* is the name of the program invoked, which means that **argv[1]* is a the first argument supplied, and **argv[n]* is the last argument. If no arguments are supplied, *argc* will be one. Thus for n arguments, *argc* will be equal to n + 1. The program is called
by the command line,

myprog x_file y_file

In the above example values of:

Argv[0]-> myprog
Argv[1]-> x_file
Argv[2]-> y_file