Functions

Function is a self-contained block of statements that is assigned a particular task and returns a single value. A function is executed, only when it is called.

There are two types of functions:

Library function:

The function that is pre-defined in the header files is called library function. Example of some library functions are: printf(), scanf(), pow(), strcmp() and etc.

User-defined function:

The function that is defined by the programmer is known as user-defined functions.

What are the advantages of using function?

Defining a function:

```
returntype functionname(type parameters)
      {
      statements;
      }
```

```
int main()
      {
      printf("Welcome");
      show();
      }
void show()
           {
           printf("Hi Guest!")
           }
```

Facts about function

1. A program may consist of many functions. But program execution starts with main().
2. The statements of a function are executed, only when they are called.
3. A function cannot be defined inside another function.
4. A function can be called many times.

5. A function can be a calling function or a called function.

   Calling function/caller function: The function that invokes another function is known as called function for this function.

   Called function: The function that is called from a caller function is known as called function.

6. When a function is called, program control is transferred to the called function and when all the statement of the called function are executed, program control returns to the line of the calling function, from where the function is called.

7. If multiple functions are there in a program, they executed in the order, they are called.(Not in the order they are defined)

8. **Scope of variable:** Local and Global

   Local variable: The variable, defined within a function is called local variable. A local variable can be accessible only inside the function in which it is defined.

   Global variable: The variable, defined outside of functions is called global variable. A global variable can be accessible anywhere in the program.

```
int b=10;              //b is a global variable
int main()
    {
    int a=10;          // a is local to main()
    show();
    }
void show()
    {
    printf("%d",a);    //ERROR! id a is not defined
    printf("%d",b);        //Perfect
    }
```

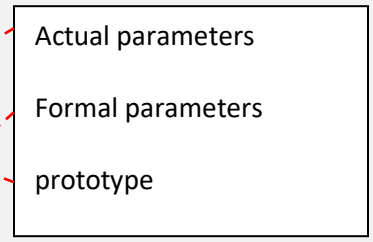
9. **Function Arguments/ Function parameters:** Arguments are the parameters that are used to convey the information from calling function to the called function. Arguments act like input to the function. A function can use more than arguments. It is two types:

   Actual Arguments: The arguments that are passed from the calling function are called actual arguments.

   Formal arguments: The arguments, which are used to receive the actual arguments in called function, are known as formal arguments.

Example-1: Write a program to Input a and b and using a function, compute a^b.

```
#include<stdio.h>
int mypower(int,int);
int main()
     {
     int a,b,p;
     printf("Enter number a and b:");
     scanf("%d%d",&a,&b);
     p = mypower(a,b);
     printf("\n%d^%d is : %d",a,b,p);
     return 0;
     }
int mypower(int a,int b)
     {
     int k =1;
     while(b!=0)
          {
          k = k* a;
          b--;
          }
     return k;
     }
```



10. **return statement :** return statement is used to return the control with or without a specific value from called function to calling function. In C, a function can return only one value.

```
return ;
return value;
```

11. Function prototype

The declaration of function is called function prototype. It specifies the function name, type of arguments a function receives and type of value it returns.

Function prototype is must, if a function is defined after the calling function.

Syntax:

```
returntype functionname(type argument,type argument,..);
```

12. Parameter Passing Techniques:

Call by value/Pass by value: When an argument is passed to the called function with its value, it is known as called by value.

In this case, the actual argument is copied to the formal argument. Any change in formal argument will not affect the actual argument.

Example : refer program Example-1.

Call by address/Pass by address: When address of an argument is passed to the formal parameter is called function, it is known as called by address.

In this case, the actual argument is referred by the formal argument. A pointer is used inside the function to access the actual argument. Any change in data in formal argument will affect the actual arguments.

Notes: Refer the concept of pointer for clear understanding

Program F2 : Using call by reference, use a function f(a) =5a

```
#include<stdio.h>
int fun(int *);
int main()
    {
    int a = 20,b;
    b = fun(&a);
    printf("5a=%d",b);
    return 0;
    }
```

```
int fun(int *p)
    {
    int result = *p * 5;
    return result;
    }
```

Programs: (Use Call by value and call by reference)

1. Input two numbers. Find the greater number.
2. Input a number. Find out its factorial.
3. Input a number. Count the digits.
4. Input a number. Check if it is a palindrome or not.
5. Input a number. Check if it is a prime or not.
6. Input two numbers. Find HCF.
7. Input two numbers. Multiply them.(Don't use multiplication sign)
8. Input two numbers and swap them.

Try this also….

1. Input two numbers. Find sum using no argument , no return,
2. Input two numbers. Find sum using call by value and no return.
3. Input two numbers. Find sum using no argument and return.
4. Input two numbers. Find sum using call by value and return.
5. Input two numbers. Find sum using call by reference and return.
6. Input two numbers. Find sum using call by value and function to pointer.
7. Input two numbers. Find sum using call by reference and function to pointer.
8. Input two numbers. Find sum using recursion.

Function returning a pointer / Function to pointer

A pointer or an address can also be returned from pointer. While returning a pointer, another pointer variable must be there in the caller function to store the return address from the called function.

```
#include<stdio.h>
int *fun(int *);
int main()
    {
    int a = 20,*b;
    b = fun(&a);
    printf("5a=%d",*b);
    return 0;
    }
```

```
int *fun(int *p)
    {
    int result = *p * 5;
    return &result;
    }
```

Passing array to function:

Like a primitive type, an array can be passed to the function. For this, the base address of the array has to be passed to called function. In called function, the array is received either as an array or as a pointer.

Note:	An array is always passed by address

Example-2: Program to input an array of n elements. Add all the elements.

```
#include<stdio.h>
int add(int [],int);
int main()
  {
  int a[]={10,20,30,40},s;
  s = add(a,4);
  printf("\n%d",s);
  return 0;
  }
```

```
//1. Received by array
int add(int p[],int n)
     {
     int i,s=0;
     for (i=0;i<n;i++)
               s= s+ p[i];
     return s;
     }
```

```
//2. received by pointer
int add(int *p,int n)
     {
     int i,s=0;
     for (i=0;i<n;i++)
               s= s+ *(p+i);
     return s;
     }
```

Programs Exercise:

1. Write a function that accept an array of n numbers and returns the sum of all the n umbers. Use a driver program to input the array of n numbers and to print the sum.
2. Write a function that accept an array of n numbers and a key and return the position of the key in array. Use a driver program to input data and display program
3. Input an array of n numbers. Sort the Array in ascending order.
4. Input a string. Using a function countlength() , compute the length of the string.

Passing 2D array to function:

A 2D array can be passed to the function by its base address or its name as actual argument. In receiving end, the function can set a pointer or a 2d array mentioning the column size only.

Example -3 : Passing a 2D array as argument.

```
#include<stdio.h>
void show(int [][3],int,int );
int main()
     {
     int a[2][3]={{1,2,3},
                  {4,5,6}};
     show(a,2,3);
     return 0;
     }

void show(int b[][3],int m,int n)
     {
     int i,j;
     for (i=0;i<m;i++)
          {
          for(j=0;j<n;j++)
               printf("%4d",b[i][j]) ;
          printf("\n");
          }
     }
```

Recursion

When a function calls itself, it is called recursion and the function is called recursive function.

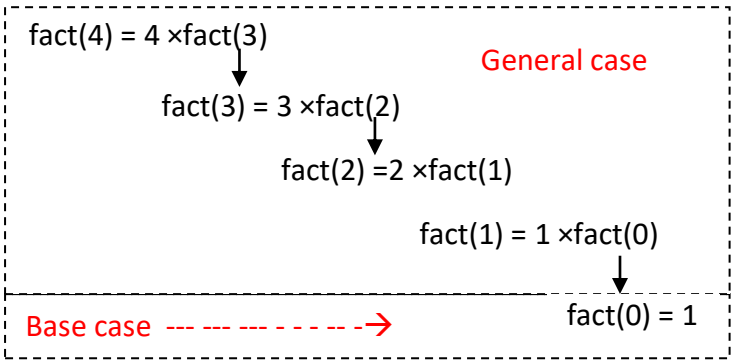
Recursion is applied to the problem that is recursive in nature. A problem is recursive in nature if the solution of the problem can be expressed in terms of same solution to the sub-problems.

Example: fact(4) = 4 ×fact(3)

Steps to solve a problem in recursion

1. Identify the base case/termination condition: Base case is the statement that solves the problem. There may be 1 or more base case.

2. Identify the general case: Here the size of the problem is reduced and executed repeatedly.



fact(0)	= 1
fact(1) = 1 ×fact(0)	= 1
fact(2) = 2 ×fact(1)	= 2
fact(3) = 3 ×fact(2)	= 6
fact(4) = 4 ×fact(3)	= 24

Stack

Find the output, if p = 4.

```
int fun(p)
     {
     if(p!=0)
          {
          fun(p-1);
          printf("%d",p);
          fun(p-1);
          }
     }
```

Example-4: Write a recursive function to that return accept n as argument and return nth number in Fibonacci series.

```
#include<stdio.h>
int fibonacci(int) ;
int main()
     {
          printf("%d",fibonacci(8));
     }
int fibonacci(int n)
     {
          if(n==1)
               return 0;
          else if(n==2)
               return 1;
          else
               return fibonacci(n-1) + fibonacci(n-2);
     }
```

Program Exercise (Use recursion)

1. Input a number. Find out its factorial.
2. Input two numbers. Multiply them.(Don't use multiplication sign)
3. Input a and b. find a^b.(Don't use pow () function)
4. Input two numbers. Add them.
5. Input two numbers. Find HCF.
6. Input an array of n numbers. Add all the elements.
7. Input an array of n numbers. Print all the elements
8. Input an array of n numbers. Print all the elements in reverse order.
9. Input a number. Count all the digits
10. Input a number. Reverse it.