FUNCTIONS

Functions are programs .There are two types of functions- library functions and programmer written functions. We are familiarised with library functions and how they are accessed in a C program.

The advantage of function programs are many

1) A large program can be broken into a number of smaller modules.
2) If a set of instruction is frequently used in program and written as function program, it can be used in any program as library function.

Defining a function.

Generally a function is an independent program that carries out some specific well defined task. It is written after or before the main function. A function has two components-definition of the function and body of the function.
Generally it looks like

```
datatype   function name(list of arguments with type)
{
     statements
     return;
 }
```

If the function does not return any value to the calling point (where the function is accessed) .The syntax looks like

```
function name(list of arguments with type)
          {
            statements
             return;
           }
```

If a value is returned to the calling point, usually the return statement looks like return(value).In that case data type of the function is executed.
Note that if a function returns no value the keyword **void** can be used before the function name

Example:

(1)

```
writecaption(char x[] );
      {
         printf("%s",x);
          return;
        }
```

(2)

```
int maximum(int x, int y)
{
    int z ;
    z=(x>=y)? x : y ;
    return(z);
 }
```

(3)

```
maximum( int x,int y)
{
       int z;
       z=(x>=y) ? x : y ;
       printf("\n maximum =%d",z);
       return ;
 }
```

Note: In example (1) and (2) the function does not return anything.

Advantages of functions

1. It appeared in the main program several times, such that by making it a function, it can be written just once, and the several places where it used to appear can be replaced with calls to the new function.
2. The main program was getting too big, so it could be made (presumably) smaller and more manageable by lopping part of it off and making it a function.
3. It does just one well-defined task, and does it well.
4. Its interface to the rest of the program is clean and narrow
5. Compilation of the program can be made easier.

Accessing a function

A function is accessed in the program (known as calling program)by specifying its name with optional list of arguments enclosed in parenthesis. If arguments are not required then only with empty parenthesis.
The arguments should be of the same data type defined in the function definition.

Example:

```
1)     int a,b,y;
              y=maximum(a,b);

2)     char name[50] ;
            writecaption(name);

3)      arrange();
```

If a function is to be accessed in the main program it is to be defined and written before the main function after the preprocessor statements.

Example:

```
#include<stdio.h>
int maximum (int x,int y)
{
 int z ;
 z=(x>=y) ? x : y ;
 return (z);
}
main( )
{
 int a,b,c;
 scanf("%d%d",&a,&b);
 c=maximum(a,b);
 printf("\n maximum number=%d",c);
}
```

Function prototype

It is a common practice that all the function programs are written after the main( ) function .when they are accessed in the main program, an error of prototype function is shown by the compiler. It means the computer has no reference about the programmer defined functions, as they are accessed before the definition .To overcome this, i.e to make the compiler aware that the declerations of the function referred at the calling point follow, a decleration is done in the beginning of the program immediately after the preprocessor statements. Such a decleration of function is called prototype decleration and the corresponding functions are called function prototypes.

Example 1:

1)

```
#include<stdio.h>
int maximum(int x,int y);
main( )
{
  int a,b,c;
 scanf("%d%d",&a,&b);
 c=maximum(a,b);
 printf("\n maximum number is : %d",c);
}
int maximum(int x, int y)
{
  int z;
  z=(x>=y) ? x : y ;
  return(z);
}
```

Example 2:

```
#include<stdio.h>
void int factorial(int m);
main( )
{
  int n;
  scanf("%d",&n);
  factorial(n);
}
void int factorial(int m)
{
  int i,p=1;
 for(i=1;i<=m;++i)
 p*=i;
 printf("\n factorial of  %d  is  %d ",m,p);
 return( );
}
```

Note: In the prototype decleration of function, if it return no value, in the place of data-type we use void.
Eg: void maximum(int x,int y);

Passing arguments to a function

The values are passed to the function program through the arguments. When a value is passed to a function via an argument in the calling statement, the value is copied into the formal argument of the function (may have the same name of the actual argument of the calling function).This procedure of passing the value is called passing by value. Even if formal argument changes in the function program, the value of the actual argument does not change.

Example:

:

```
#include<stdio.h>
void square (int x);
main( )
{
 int x;
 scanf("%d",&x);
 square(x):
}
void square(int x)
{
  x*=x ;
 printf("\n the square is %d",x);
 return;
}
```

In this program the value of x in the program is unaltered.

Recursion

It is the process of calling a function by itself ,until some specified condition is satisfied. It is used for repetitive computation (like finding factorial of a number) in which each action is stated in term of previous result

Example:

```
#include<stdio.h>
long int factorial(int n);
main( )
{
  int n;
  long int m;
  scanf("%d",&n);
  m=factorial(n);
  printf("\n factorial is : %d", m);
}
long int factorial(int n)
{
           if (n<=1)
   return(1);
 else
   return(n*factorial(n-1));
}
```

In the program when n is passed the function, it repeatedly executes calling the same function for n, n-1, n-2,……..1.