FUNCTIONS

Definition : A function is self contained block of statements that performs a particular task.

C -functions are divided into two parts

1) Pre-defined functions(library functions)
1) User-defined functions

library functions OR pre-defined functions are not required to be written by us where as user define function has to be developed by the user at the time of writing program. Ex: printf(), scanf(), sqrt(), pow() etc.,,,

To make use of user defined function, we need to write three elements that are related to function.

1) Function declaration
1) Function definition
1) Function call

Function declaration / signature / prototype :

Just like variables , functions are also to be declared before we use them in the program. A function declaration consists of four parts.

1) Function type(return type)
1) Function name
1) Parameter List
1) Terminating semicolon.

The general format of declaring a function is :

return_type function_name(parameter list);

Example: **void display( );**
int square(int m);
int mul(int m, int n);

A prototype declaration may be placed in two places in a program

1) Above all the functions (including main)
2) Inside a function definition.

- When we place the declaration above all the functions (in the global declaration section), the prototype is referred to as a global prototype. Such declarations are available for all the functions in the program.
- When we place it in a function definition (in the local declaration section), this prototype is called local prototype.

Function call : is done inside main(). Whenever function call is made control from main() will be transfered to the definition of that function. After executing all the statements in tha function control comes back to main().

General form of function call : **function_name( ) ;**

If arguments are being passed then : **function_name(arguments);**

Function definition : (also known as FUNCTION IMPLEMENTATION)

format of function definition is :-

```
return_type  fuction_name(arguments)
    {
             Local variable declarations;
             stmt 1;
             stmt 2;
             --------
             ----------
              return statement;
      }
```

return type **specifies the type of value that the function is going to return. Various return types are ………void, int, char, float, double, long int etc.,,**

function name **is the name of function. It is given the user.**

- A FUNCTION IN WHICH FUNCTION CALL IS MADE IS KNOWN AS CALLING FUNCTION.
- A FUNCION WHICH IS CALLED BY OTHER FUNCTION IS KNOWN AS CALLED FUNCTION.

Example :

```
main ()
{
  printf("hello");
  display();
}
```

In above sample code **display( ) is called function** and **main( ) is calling function.**

main ( ) is a **specially recognized user-defined function**. That is ,

- **Declaration** of main( ) is **pre-defined**
- **Definition** of main( ) is **user-defined**

Arguments :

The mechanism of passing values between calling function and called function is known as ARGUMENT or PARAMETER.

There are 2 types of arguments or parameters :

1. Actual arguments
2. Formal arguments

- ➔ Variables in function call are actual arguments.
- ➔ Variables in function definition are formal arguments.

The return statement can take one of the following forms

```
return;
  or
return(expression);
```

The first one does not return any value; it acts much as closing brace of the function. When a return is encountered, the control is immediately passed back to the calling function.

An example of the use of a simple return is as follows

```
if(error)
return;
```

The second form of the return with an expression returns the value of the expression. For example the function

```
int mul(int x,int y)
{
      int p;
      p=x*y;
      return(p);
}
```

returns the value of the p which is the product of the values of x and y. The last two statements can be combined into one statement as follows.

```
return(x*y);
```

A function may have more than one return statement this situation arises when the value returned is based on certain conditions. For example

```
if(x<=0)
  return(0);
```

```
                else
                   return(1);
```

TYPES OF FUNCTIONS BASED ON ARGUMENTS AND RETURN VALUES :

1. Functions without arguments and without return values
2. Functions with arguments and without return values
3. Functions with arguments and with return values
4. Functions without arguments and with return values

Functions without arguments and without return values

- No values will be passed from calling function to called function.
- No values will be returned to calling function from called function.

Example program :

```
#include<stdio.h>
void add( ); // function declaration
main()  //  calling function
{
  add();  // function call
}

void add( )  // function definition and called function
{
 int x,y,z;
 printf("enter any two integers");
 scanf("%d %d", &x,&y);
 z=x+y;
 printf("sum = %d",z);
}
```

Functions with arguments and without return values

- (arguments) values will be passed from calling function to called function.
- No values will be returned to calling function from called function.

Example program :

```
#include<stdio.h>

void add( int a, int b ); // function declaration ;  ‘a’ and ‘b’ are formal arguments

main()  // calling function

{

  int x,y ;

  printf("enter any two integers");

  scanf("%d %d", &x,&y);

  add(x,y);  // function call ;  ‘x’ and ‘y’ are actual arguments

}

void add( int a, int b) // function definition  and called function

{

 int  c;

 c = a + b;

 printf("sum = %d",c );

}
```

Functions with arguments and with return values

- (arguments) values will be passed from calling function to called function.
- values will be returned to calling function from called function.

Example program :

```
#include<stdio.h>

int add(int a, int b ); // function declaration ;  'a' and 'b' are formal arguments

main() // calling function

{

  int x,y,z ;

  printf("enter any two integers");

  scanf("%d %d", &x,&y);

  z = add(x,y);  // function call ;  'x' and 'y' are actual arguments

  printf("sum = %d",z);

}

int  add( int x, int y) // function definition  and called function

{

 int c;

 c = x+y;

 return c;
}
```

Functions without arguments and with return values

- No values will be passed from calling function to called function.
- values will be returned to calling function from called function.

Example program :

```
#include<stdio.h>
int  add( ); // function declaration
main() // calling function
{
  int c;
  c = add( ); // function call
  printf("sum = %d",c);
}
int add( ) // function definition  and called function
{
  int x,y,z ;
  printf("enter any two integers");
  scanf("%d %d", &x,&y);
  z = x+y;
```

```
        return z;
      }
```

SCOPE OF A VARIABLE : is the extent to which it can be used in a program.

Based on this, variables are classified as LOCAL VARIABLES & GLOBAL VARIABLES.

- **Local variable** is restricted to a particular block in the program.
- **Global variable** is used in the entire program that is in all the functions present in the program.

Example :

```
#include<stdio.h>

int k=20; // global variable

void add ( );

main()

{

  int c =30; // 'c' is local variable to main( )

  add ( );

  printf("sum = %d", ( c+k ) ); // output is 50

}
```

```
void add( )

{

  int h = 40; // 'h' is local variable to add( )

  printf(" addition = %d", ( h+k ) ); // output is 60

}
```