Lecture 11

Functions

Functions

A number of statements grouped into a single logical unit are called a function. The use of function makes programming easier since repeated statements can be grouped into functions. Splitting the program into separate function make the program more readable and maintainable.

It is necessary to have a single function ‘main’ in every C program, along with other functions used/defined by the programmer.

Functions

Contents :

Functions in C :

The function is a self contained block of statements which performs a coherent task of a same kind.

C program does not execute the functions directly. It is required to invoke or call that functions. When a function is called in a program then program control goes to the function body. Then, it executes the statements which are involved in a function body. Therefore, it is possible to call function whenever we want to process that functions statements.

Types of functions :

There are 2(two) types of functions as:

1. **Built in Functions**
2. **User Defined Functions**

Functions in C :

Advantages of functions:

It is easy to use.

Debugging is more suitable for programs.

It reduces the size of a program.

It is easy to understand the actual logic of a program.

Highly suited in case of large programs.

By using functions in a program, it is possible to construct modular and structured programs.

Functions in C :

1. **Built in Functions :**These functions are also called as 'library functions'. These functions are provided by system. These functions are stored in library files. e.g.

- ✓**scanf()**
- ✓**printf()**
- ✓**strcpy**
- ✓**strlwr**
- ✓**strcmp**
- ✓**strlen**
- ✓**strcat**

Functions in C :

1. User Defined Functions :The functions which are created by user for program are known as 'User defined functions'.

Syntax:

```
void main()
{
                    // Function prototype
                    <return_type><function_name>([<argu_list>]);

                    // Function Call
                    <function_name>([<arguments>]);
}
// Function definition
 <return_type><function_name>([<argu_list>]);
 {
           <function_body>;
 }
```

To display any text using a function

```
#include<stdio.h>
#include<conio.h>
Void main()
{
void text (void);
text();
getch();
}
void text (void)
{
Printf (“this is a demo program of function\n”);
Printf (“ Showing function calling in main program’);
}
```

Functions in C :

Program :

```
/* Program to demonstrate function.
#include <stdio.h>
 #include <conio.h>
 void add()
{
		int a, b, c;
		clrscr();
		printf("\n Enter Any 2 Numbers : ");
		scanf("%d %d",&a,&b);
		c = a + b;
		printf("\n Addition is : %d",c);
 }

void main()
 {
		void add();
		add();
		 getch();
 }
```

Output :

```
Enter Any 2 Numbers : 23 6
Addition is : 29_
```

A function definition has two principal components:

The function header

and

body of the function.

The function header is the data type of return value followed by function name and (optionally) a set of arguments separated by commas and enclosed in parenthesis. Associated type to which function accepts precedes each argument.

In general terms function header statement can be written as

return_type function_name (type1 arg1,type2 arg2,..,typen argn)

where

- return_type represents the data type of the item that is returned by the function,
- function_name represents the name of the function,
- type1,type2,…,typen represents the data type of the arguments arg1,arg2,..,argn.

Example: Following function returns the sum of two integers.

```
int add(int p,int q)
{
return p+q; //Body of the function
}
```

Here p and q are arguments. The arguments are called formal arguments or formal parameters, because they represent the name of the data item that is transferred into the function from the calling portion of the program.
The corresponding arguments in the function call are called actual arguments or actual parameters, since they define the data items that are actually transferred.

A function can be invoked whenever it is needed.
It can be accessed by specifying its name followed by a list of arguments enclosed in parenthesis and separated by commas.
e.g., add(5,10);