Chapter 3 :- Functions

Concept Of Function:- C Functions

In c, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the C program.

In other words, we can say that the collection of functions creates a program. The function is also known as procedureor subroutinein other programming languages.

Advantages of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.
- However, Function calling is always a overhead in a C program.

User Defined Functions

User can define functions to do a task relevant to their programs. Such functions are called user-defined functions.

The main( ) function in which we write all our program is a user-defined function. Every pro0gram execution starts at the main( ) function.

Any function (library or user-defined) has 3 things

Function declaration

Function calling

Function defintion

In case of library functions, the function declaration is in header files. The function is library files and the calling is in the program.

But in case of user-defined functions all the 3 things are in your source program. Function declaration:

Syntax: return data_type function_name(arguments list);

For Example.

```
#include <stdio.h>
int addNumbers(int a, int b);         // function prototype
int main()
{
   int n1,n2,sum;
```

```
printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);
    sum = addNumbers(n1, n2);        // function call
    printf("sum = %d",sum);
    return 0;
}
int addNumbers(int a, int b)        // function definition
{
    int result;
    result = a+b;
    return result;                  // return statement
}
```

Recursive Function

In C programming, a function is allowed to call itself. A function which calls itself directly or indirectly again and again until some specified condition is satisfied is known as Recursive Function.

For Ex. :-

/*

Program: Factorial of a given number using recursive function.

Author: Codesansar

Date: Jan 22, 2018

*/

```
#include<stdio.h>
#include<conio.h>
int fact(int n); /* Function Definition */
void main()
{
 int num, res;
clrscr();
 printf("Enter positive integer: ");
 scanf("%d",&num);
 res = fact(num); /* Normal Function Call */
 printf("%d! = %d" ,num ,res);
```

```
getch();
}
int fact(int n) /* Function Definition */
{
 int f=1;
 if(n <= 0)
 {
  return(1);
 }
 else
 {
```

```
f = n * fact(n-1); /* Recursive Function Call as fact() calls itself */
  return(f);
 }
}
```

Output

```
Enter positive integer: 5
5! = 120

********** END OF PROGRAM **********
```

Scope of Variables

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language −

Inside a function or a block which is called local variables.

Outside of all functions which is called global variables.

In the definition of function parameters which are called formal parameters.

Local Variables

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

The following example shows how local variables are used. Here all the variables a, b, and c are local to main() function.

```
#include <stdio.h>
int main () {
  int a, b;  /* local variable declaration */
  int c;
  a = 10; /* actual initialization */
  b = 20;
  c = a + b;
  printf ("value of a = %d, b = %d and c = %d\n", a, b, c);
 return 0;
}
```

Global Variables

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. The following program show how global variables are used in a program.

```
#include <stdio.h>
/* global variable declaration */
int g;
int main () {
  /* local variable declaration */
  int a, b;
  /* actual initialization */
  a = 10;
  b = 20;
  g = a + b;
  printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
  return 0;
}
```

Formal Parameters

Formal parameters, are treated as local variables with-in a function and they take precedence over global variables.

Following is an example –

```
#include <stdio.h>
/* global variable declaration */
int a = 20;
int main () {
  /* local variable declaration in main function */
```

```
  int a = 10;
  int b = 20;
  int c = 0;
  printf ("value of a in main() = %d\n",  a);
  c = sum( a, b);
  printf ("value of c in main() = %d\n",  c);
  return 0;
}
/* function to add two integers */
```

```
int sum(int a, int b) {
   printf ("value of a in sum() = %d\n",  a);
   printf ("value of b in sum() = %d\n",  b);
   return a + b;
}
```

********** END OF PROGRAM **********