Quadrant-1: Text

Introduction:

A function is a group of statements that together perform a specific, well defined task. Every C program has at least one function, which is main() and can define additional functions. When a C program is executed, the operating system calls the main() function of the program which marks the entry point for the execution. After execution, main() returns some integer value to the operating system. If the value returned is zero, it implies that the function has terminated successfully and any nonzero return value indicates an error.

Advantages of using Functions:

1. A C program can be divided into functions, each of which performs some specific tasks. So, the use of C functions modularizes and divides the work of a program.

2. When some specific code is to be used more than once and at different places in the program the use of functions avoids repetition of that code.

3. Error checking, debugging and modification of the program becomes easy if it is divided into functions.

C programs have two types of functions: Library functions, User defined functions.

Library functions:

All the libraries in C contain a set of predefined built-in functions that the programmers are free to use in their programs. The functions scanf() and printf() are input output library functions. Similarly, strlen(), strcmp() are used for string manipulation. To use a library function, corresponding header file must be included using the preprocessor directive #include. For example to use input output functions like printf(), scanf(), stdio.h is to be included. For mathematical library functions math.h and for string manipulation string.h should be included. The following program illustrates the use of library functions printf(), scanf() and sqrt() to find square root of an integer:

```
#include<stdio.h>

#include<math.h>

int main()

{

        int num,sqrnum;

        printf("Enter an integer:\n");

        scanf("%d",&num);

        sqrnum=sqrt(num);

        printf("\nThe square root of %d is %d",num,sqrnum);

        return 0;

}
```

Output:

```
Enter an integer:

100

The square root of 100 is 10
```

User defined functions:

Users can create their own functions for performing any specific task of the program. These types of functions are called user-defined functions. To create and use these functions, three aspects of C functions must be understood:

1. Function declaration

2. Function call

3. Function definition

Before using a function, the compiler must know about the number and type of parameters (or arguments) that the function expects to receive and the data type of the value that it will return to the calling program. A function declaration (also known as the function prototype) tells the compiler about a function's name, return type, and parameters. Functions can be declared globally before main() or locally (within another function). General syntax of function declaration is:

```
return_type func_name(typel, type2, ........);
```

A function is called by simply writing its name followed by the argument list inside the parentheses:

```
func_name(argl, arg2, arg3 ...);
```

These arguments arg 1, arg2,… are called actual arguments.

Same number and type of arguments must be passed during function call as it is declared in the function declaration. Globally declared functions can be called from anywhere in the program. Locally declared functions are required to be called from their calling functions (in which they are declared) only.

The function definition consists of the whole description and code of a function. A function definition consists of two parts - a function header and a function body. The general syntax of function definition:

```
return_type func_name(typel argl, type2 arg2,…)

{

        local variables declarations;

        statement;

        …………….

        return(expression);

}
```

Arguments arg 1, arg2… in the function header are called formal arguments.

The following program illustrates the declaration, calling and definition of a user defined function to add two integers:

```
#include<stdio.h>
int main()
{
        int a,b,sum;
        int add(int,int);//Function declaration
        printf("Enter two integers:\n");
        scanf("%d%d",&a,&b);
        sum=add(a,b);//Function call-here a and b are actual arguments
        printf("%d + %d = %d",a,b,sum);
        return 0;
}
//Function Definition
int add(int x,int y)//Function header-here a and b are formal arguments
{
        int result; //it is a variable local to this function
        result=x+y;
        return result;
}
```

Output:

```
Enter two integers:
12
14
12 + 14 = 26
```

Some points to remember:

1. The variables declared within the function are local to that function. So, their scope and lifetime are limited to that function. Any change made to these variables is visible only in that function. They are automatically called when the function is created and they cease to exit at the end of the function.

2. The argument names in the function declaration and function definition need not be the same. However, the data types of the arguments must match with that specified in function declaration as well as function definition.

3. The return statement is used to terminate the execution of a function and returns control to the calling function. When this statement is encountered, the program execution resumes in the calling function immediately following the function call.

4. In the previous example to add two integers, sum() function takes two integer type arguments and returns an integer (the result) to main(). However, usage of arguments and return statement in a function is optional. For example, a slightly modified function sum() do not have any return statement is shown below:

```
void add(int x,int y)//void keyword denotes this function doesnot return any value
{
        int result;
        result=x+y;
        printf("%d + %d = %d",x,y,result);
}
```

Passing arguments to the function:

1. **Call by value** copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

2. **Call by reference** copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, C uses call by value to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function. However, call by reference can be implemented using help of pointers.

References:

1. Programming with C, Byron S. Gottfried, McGraw Hill.

2. The C Programming Language, Kernighan and Dennis, PHI.

3. The Complete reference C, Herbert Schildt, McGraw Hill.

4. Programming in ANSI C, Balaguruswamy, McGraw Hill.