Unit-7

Functions

Introduction

A function is a self contained sub program that is meant to be some specific, well defined task. A C-program consists of one or more functions. If a program has only one function then it must be the main( ) function.

Advantages of using function

1. Functions increases code reusability by avoiding rewriting of same code over and over.
2. If a program is divided into multiple functions, then each function can be independently developed. So program development will be easier.
3. Program development will be faster.
4. Program debugging will be easier.
5. Function reduces program complexity.
6. Easier to understand logic involved in the program.
7. Recursive call is possible through function.

C programs have two types of functions-

i. Library Functions
ii. User defined Functions

i. Library function

Library functions are supplied with every C compiler. The source code of the library functions is not given to the user. These functions are precompiled and the user gets only the object code. This object code is linked to the object code of your program by the linker. Different categories of library functions are grouped together in separate library files. When we call a library function in our program, the linker selects the code of that function from the library file and adds it to the program.

To use library function in our program we should know-

i. Name of the function and its purpose
ii. Type and number of arguments it accepts
iii. Type of the value it returns
iv. Name of the header file to be included

We can define any function of our own with the same name as that of any function in the C library. If we do so then the function that we have defined will take precedence over the library function with the same name.

Some examples are

printf(), scanf() are defined in header file stdio.h
getch(), clrscr() are defined in header file conio.h
strlen(), strupr() are defined in header file string.h
pow(), sqrt() are defined in header file math.h

ii. User defined function

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, we should know about these three things-

1. Function definition
2. Function declaration
3. Function call

Lets take an example of function to add two numbers.

```
#include<stdio.h>
#include<conio.h>
void sum(int,int);          //function declaration
void main()
{
int a,b;
clrscr();
printf("Enter two numbers\n");
scanf("%d%d",&a,&b);
sum(a,b);                   //function call
getch();
}

void sum(int x,int y)
{                           //function definition
int s;
s=x+y;
printf("Sum =%d",s);
}
```

1. **Function Definition**

The function definition consists of the whole description and code of a function. It tells what the function is doing and that are its inputs and outputs. A function definition consists of two parts - a function header and a function body. The general syntax of a function definition is-

```
return_type func_name(type1 arg1, type2 arg2, ……..)
{
      local variables declarations;
      statements;
      ……..
      return(expression);
}
```

The first line in the function definition is known as the function header and after this the body of the function is written enclosed in curly braces.

The ***return_type*** denotes the type of the value that will returned by the function. The return_type is optional and if omitted, it is assumed to be int by default. A function can return either one value or no value. If a function does not return any value then void should be written in place of return_type.

func_name specifies the name of the function and it can be any valid C identifier. After function name, the argument declarations are given in parentheses, which mention the type and name of the arguments. These are known as formal arguments and used to accept values. A function can take any number of left empty or void can be written inside the parentheses.

The body of function is a compound statement (or block), which consists of declarations variables, and C statements followed by an optional return statement. The variables declared inside the function are known as local variables, since they are local to that function only, i.e. they have existence only in the function in which they are declared, and they cannot be used anywhere else in the program. There can be any number of valid C statements inside a function body. The ***return*** statement is optional. It may be absent if the function does not return any value.

2. **Function Declaration/Function prototype:**

The calling function needs information about the called function. If definition of the called function is placed before the calling function, then declaration is not needed. The function declaration is also known as the function prototype, and it informs the compiler about the following three things-

1. Name of the function
2. Number and type of arguments received by the function
3. Type of value returned by the function

Function declaration tells the compiler that a function with these features will be defined and used later in the program. The general syntax of a function declaration is-

```
return_type func_name(type1, type2,……..);
```

3. **Function call**

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

   ```
   func_name(arg1, arg2, arg3,….);
   ```

   These arguments arg1,arg2,arg3,…are called actual arguments. Here ***func_name*** is known as the called function while the function in which this function call is placed is known as the calling function. When a function is called, the control passes to the called function, which is executed and after this control is transferred to the statement following the function call in the calling function. The code of a function is executed only when it is called by some other function. If the function is defined and not called even once then it's code will never be executed. A function can be called more than once, so the code is executed each time it is called. The execution of a function finishes either when the closing braces of the function body are reached or if return statement is encountered.

Return statement

The return statement is used in a function to return a value to the calling function. It may also be used for immediate exit from the called function to the calling function without returning a value. This statement can appear anywhere inside the body of the function. There are two ways in which it can be used-

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

Here return is a keyword. The first form of return statement is used to terminate the function without returning any value. The second form of return statement is used to terminate a function and return a value to the calling function.

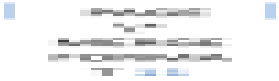
Function arguments

The calling function sends some values to the called function for communication; these values are called arguments or parameters.

- ***Actual argument***

The arguments which are mentioned in the function call are known as actual arguments, since these are the values which are actually sent to the called function. Actual arguments can be written in the form of variables, constants or expressions or any function call that returns a value.

- ***Formal argument***



The name of the arguments, which are mentioned in the function definition are called formal or dummy arguments since they are used just to hold the values that are sent by the calling function. These formal arguments are simply like other local variables of the function which are created when the function call starts and are destroyed when the function ends.

The order, number and type of actual arguments in the function call should match with the order, number and type of formal arguments in the function definition.

Types of user defined Functions:

The functions can be classified into four categories on the basis of the arguments and return value;

1. Function with no arguments and no return value.
2. Function with no argument and return value.
3. Function with argument and no return value.
4. Function with argument and return value.

1. **Function with no arguments and no return value.**

Example.

Program to add two numbers.

```
#include<stdio.h>
#include<conio.h>
void sum();
void main()
{
clrscr();
sum();
getch();
}

void sum()
{
int x,y,s;
printf("Enter two numbers\n");
scanf("%d%d",&x,&y);
s=x+y;
printf("Sum =%d",s);
}
```

2. Function with no argument and a return value.

Example.

Program to add two numbers.

```
#include<stdio.h>
#include<conio.h>
```

```
int sum();
void main()
{
int c;
clrscr();
c=sum();
printf("sum=%d",c);
getch();
}

int sum()
{
int x,y,s;
printf("Enter two numbers\n");
scanf("%d%d",&x,&y);
s=x+y;
return(s);
}
```

3. **Function with argument and no return value**

   Example

   Program to add two numbers.

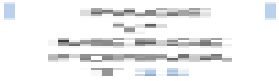
```
#include<stdio.h>
#include<conio.h>
void sum(int,int);
void main()
{
int a,b;
clrscr();
printf("Enter two numbers\n");
scanf("%d%d",&a,&b);
sum(a,b);
getch();
}

void sum(int x,int y)
{
int s;
s=x+y;
printf("Sum =%d",s);
}
```

4. **Function with argument and return value**

   Example

   Program to add two numbers.

```
#include<stdio.h>
```



```
#include<conio.h>
int sum(int,int);
void main()
{
int a,b,c;
clrscr();
printf("Enter two numbers\n");
scanf("%d%d",&a,&b);
c=sum(a,b);
printf("sum=%d",c);
getch();
}

int sum(int x,int y)
{
int s;
s=x+y;
return(s);
}
```

Difference between library function and user defined function

Library function	User defined function
1. Library function is a predefined function in a header file or preprocessor directive	1. User defined function is not a predefined function, it is defined by the programmer according to the need.
2. Programmer can simply use this function by including respective header file.	2. Programmer has to declare, define and use this function by themself.
3. The program using library function will be usually short as the programmer doesn't have to define the function.	3. The program using user defined function will be usually lengthy as the programmer has to define the function.
4. Program development time will be faster.	4. Program development time will be usually slower.
5. Program will be simple.	5. Program will be complex.
6. This function requires header file to use it.	6. This function requires function prototype to use it.

main ( )

Execution of every C program always begin with the function main( ). Each function is called directly or indirectly in main( ) and after all functions have done their operations, control returns back to main( ). There can be only one main( ) function in a program.

The main( ) function is a user defined function but the name, number and type of arguments are predefined in the language. The operating system calls the main function and main( ) returns a value of integer type to the operating system. If the value returned in zero, it implies that the function has terminated successively and any non zero returns value indicates an error. If no return value is specified in main( ) then any garbage value will be returned automatically. Calling the function exit( ) with an integer value is equivalent to returning that value from main( ). The function main( ) can also take arguments, which will be discussed in further chapters.

The definition, declaration and call of main( ) function-

function declaration - by the C compiler
function definition - by the programmer
function call - by the operating system

Recursion

Recursion is a powerful technique of writing a complicated algorithm in an easy way. According to this technique a problem is defined in terms of itself. The problem is solved by dividing it into smaller problems, which are similar in nature to the original problem. These smaller problems are solved and their solutions are applied to get the final solution of our original problem.

A function will be recursive, if it contain following features:

i. Function should call itself.

ii. Function should have a stopping condition (base criteria) and every time the function calls itself it must be closer to base criteria.

Example.

1. Write a program to calculate factorial of any given number using recursive function.

```
#include<stdio.h>
#include<conio.h>
long int factorial(int);
void main()
{
int num,fact;
clrscr();
printf("Enter a number\n");
scanf("%d",&num);
fact=factorial(num);
```

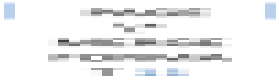
```
printf("Factorial of %d = %d",num,fact);
getch();
}

long int factorial(int n)
{
if(n==0)
return 1;
else
return (n*factorial(n-1));
}
```

2. Write a program to calculate the sum of N natural numbers using recursive function.

```
#include<stdio.h>
#include<conio.h>
int sum(int);
void main()
{
int n,s;
clrscr();
printf("Input a number\n");
scanf("%d",&n);
s=sum(n);
printf("Sum of natural numbers=%d",s);
getch();
}
int sum(int n)
{
if(n<=0)
return 0;
else
return(n+sum(n-1));
}
```

3. Write a recursive program to calculate x^y (x to the power y).

```
#include<stdio.h>
#include<conio.h>
int power(int,int);
void main()
{
int a,p,n;
clrscr();
printf("Enter value of a and n :");
scanf("%d%d",&a,&n);
p=power(a,n);
printf("%d raised to power %d is %d",a,n,p);
getch();
}
```



```
int power(int a,int n)
{
if(n==0)
return 1;
else
return(a*power(a,n-1));
}
```

Local variables

The variables that are defined within the body of a function or a block, are local variables. For example.

```
func()
{
int a,b;
…….
……
}
```

Here a and b are local variables which are defined within the body of the function ***func()***. Local variables can be used only in those functions or blocks, in which they are declared. The same variable name may be used in different functions. For example-

```
func1()
{
int a=2,b=4;
……….
……….
}
```

```
func2()
{
int a=15,b=20;
……….
……….
}
```

Here values of a=2 , b=4 are local to the function ***func1()*** and a=15, b=20 are local to the function ***func2().***

Global variables

The variables that are defined outside any function are called global variables. All functions in the program can access and modify global variables. It is useful to declare a variable global if it is to be used by many functions in the program. Global variables are automatically initialized to 0 at the time of declaration.

Example-

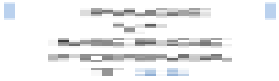
```
#include<stdio.h>
#include<conio.h>
int a,b=5;      //global variables
void func1();
void func2();
void main()
{
clrscr();
printf("Inside main() a=%d and b=%d\n",a,b);
func1();
func2();
getch();
}
void func1()
{
printf("Inside func1() a=%d and b=%d\n",a,b);
a=a+1;
b=b-1;
}
void func2()
{
printf("Inside func2() a=%d and b=%d",a,b);
}
```

Passing by value (call by value)

In pass by value, values of variables are passed to the function from the calling function. This method copies the value of actual parameters into formal parameters. In other words, when the function is called, a separate copy of the variables is created in the memory and the value of original variables is given to these variables. So if any changes are made in the value (of the called function) is not reflected to the original variable (of calling function). It can return only one value.

Example:-

```
#include<stdio.h>
#include<conio.h>
void change(int);
void main()
{
int a=15;
clrscr();
printf("Before calling function, a=%d\n",a);
change(a);
printf("After calling function, a=%d",a);
getch();
}
void change(int x)
{
x=x+5;
```



```
}
```

Output:

```
Before calling function, a=15
After calling function, a=15
```

Passing by reference (call by reference)

In passing by reference we pass the address or location of a variable to the function during function call. Pointers are used to call a function by reference. When a function is called by reference, then the formal argument becomes reference to the actual argument. This means that the called function doesn't create its own copy of values rather, it refers to the original values only by reference name. Thus function works with original data and the changes are made in the original data itself. It can return more than one value at a time.

Example:-

```
#include<stdio.h>
#include<conio.h>
void change(int*);
void main()
{
int a=15;
clrscr();
printf("Before calling function, a=%d\n",a);
change(&a);
printf("After calling function, a=%d",a);
getch();
}
void change(int *x)
{
*x=*x+5;
}
```

Output:

```
Before calling function, a=15
After calling function, a=20
```

Passing string to function

Use of Array in function

Passing single dimensional array to a function

Example:-

```
#include<stdio.h>
#include<conio.h>
void func(int[]);
void main()
```

```
{
int arr[5]={5,8,12,45,9};
clrscr();
func(arr);
getch();
}
void func(int a[])
{
int i;
for(i=0;i<5;i++)
{
printf("%d\n",a[i]);
}
}
```

Passing multidimensional array to a function

Example:-

```
#include<stdio.h>
#include<conio.h>
int sum(int[][2],int[][2]);
void main()
{
int mat1[2][2]={3,7,9,1};
int mat2[2][2]={4,2,6,8};
clrscr();
sum(mat1,mat2);
getch();
}

int sum(int a[][2],int b[][2])
{
int i,j,c[2][2];
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
c[i][j]=a[i][j]+b[i][j];
}
}
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}
```

```
}
```