Functions

Introduction

The most of the general problems cannot be solved by using small programs, for that one has to write a big computer program. Large programs are difficult to handle and manage. Large programs can be made manageable by dividing it into smaller programs or modules is called **modularization**. Programming with such an approach is called modular programming. These subprograms are called as **functions**.

C functions

C functions are divided into following types:

- ✓ Built-in Library functions.
- ✓ User defined functions.

Library functions are available through libraries that are provided by the C compiler developers.

Example:

sqrt(), pow(), strcmp(), strcat(), strcpy(), strlwr(), strupr(), strlen(), abs(), fabs(), cos()…etc.

C allows users to create their own functions or subprograms called user defined functions.

Need for user defined functions (purposes)

- ✓ **Repetition** If a portion of the program is to be repeated in a number of places in the program, then a function may be used to avoid rewriting the sequence of code in two or more locations in the program.
- ✓ **Universal use** Same task might be needed in more than one program or a large group of programmers may need the same task. By writing a function and making it available to others the duplication of effort can be avoided.
- ✓ **Modularity** A program for complex problems can be developed in a modular fashion by dividing the program into modules or functions, each function performing definite tasks. Since these functions are independent of each other they can be tested independently without affecting other module.
- ✓ **Teamwork** If a program is divided into subprograms, a number of members form a team and each person in the team can develop a separate subprogram, rather than having a single person work on the complete program.
- ✓ **Debugging easy** Debugging becomes easier as the program is divided into a number of functions.

Function Definition

A function is a self contained block of statements that perform a specific task. After processing a function returns a single value or null value.

Advantages of functions

- ✓ Updating and modification can be made very easy.
- ✓ Coding is reduced.
- ✓ Looping is reduced.
- ✓ Debugging is easy.
- ✓ Subprogram can be executed many times as per requirement.

Structure of C Function

Structure of C Function is as follows:

return_type_specifier function_name(argument_list)

{

body of the function;

return(expression);

}

where,

return_type_specifier

It specifes the type of value which will be return by the function after execution. It may be of type int, float, double and char. The data type void is used if the function does not return any value.

function_name

A name of the function. It is declared according the rules of declaring a variable.

argument_list

the argument list shows the pair of data-type and variable_name which are passed to the function. If more then variable are passed they are separated by commas.

Body_of_the_function

It will be having the declaration of Local_variables, and statements (input, ouput, computations)

return_type(expression)

The expression will yield a single value which will be returned to function, which as invoked it. It will be not present if the function is not returning any value.

Write a C program to find the sum of given two numbers using function.

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

```
c=sum(a,b);
printf("\n the sum is %d  ",c);
getch();
}

int sum(int m,int n)
{
int res;
res=m+n;
return(res);
}
```

Output:

```
enter two numbers
5
10
the sum is 15
```

Function Call

To execute a function we will have to call the function. When the function is called values are passed to the functions through the use of actual parameters or arguments.

The general form is:

variable=function_name(arg1,arg2,……);

OR

Function_name(arg1,arg2,……);

Actual and Formal Arguments (Parameters)

Actual arguments: The arguments appearing in the function call are known as actual arguments.

Formal arguments: The arguments appearing in the function header are known as formal arguments.

Example:

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

```
c=sum(a,b);
printf("\n the sum is %d  ",c);
getch();
}

int sum(int m,int n)
{
int res;
res=m+n;
return(res);
}
```

Here a & b are actual arguments.
m & n are formal arguments.

Local & Global Variables

Variables declared inside a block or function are said to belong only to that block and are referred as **Local Variables**. They can be accessed in the block but not outside of the block.

Variables declared before main function block are referred as **Global Variables**. They can be accessed throughout the program. i.e in all the blocks of the program.

Example:

```
sub(int p, int q)
{
Int a,b;
-----------------------------------
-----------------------------------
}
```

Here a & b are local variables.

```
int p, q;
main( )
{
int a,b;
-----------------------------------
-----------------------------------
}
```

Here p & q are global variables.
Here a & b are local variables.

Types of Functions (categories of functions)

- ✓ Function with no arguments and no return values.
- ✓ Function with arguments and no return values.
- ✓ Function with arguments and with return values.
- ✓ Recursive Functions.

Function with no arguments and no return values.

In this type of function the first function calls the second functions, but no arguments are passed to second function. The second function performs its calculations but no value is sent back to the first function.

The syntax is:

Write a C program to illustrate function with no arguments and no return value.

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

void sum( )
{
int a,b,c;
printf("\n enter two numbers  \n");
scanf("%d%d",&a,&b);
c=a+b;
printf("\n the sum is %d  ",c);
}
```

Output:

```
enter two numbers
5
10
the sum is 15
```

Function with arguments and no return values.

In this type of function the first function calls the second functions, by passing one or more arguments to second function. The second function performs its calculations but no value is sent back to the first function.

The syntax is:

Write a C program to illustrate function with arguments and no return value.

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

void sum(int m, int n )
{
int c;
c=m+n;
printf("\n the sum is %d  ",c);
}
```

Output:

```
enter two numbers
5
10
the sum is 15
```

Function with arguments and return values

In this type of function the first function calls the second functions, by passing one or more arguments to second function. The second function performs its calculations returns answer or a value to the first function.

The syntax is:

Write a C program to illustrate function with arguments and return value.

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

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

int sum(int m, int n )
{
int res;
res=m+n;
return(res);
}
```

Output:

```
enter two numbers
5
10
the sum is 15
```

Recursive Function

In many situations it is possible for us to have a function that call itself directly or indirectly again and again such function are called recursive functions. It is also called recursion.

General form is:

```
function1(a)
{
--------------------
----------------------
y=function1(a);
}
```

Write a C program to find factorial of a given number using recursive.

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

```
int fact(int );
void main( )
{
int n,f;
clrscr( );
printf("\n enter a numbers  \n");
scanf("%d",&n);
f=fact(n);
printf("\n the factorial of %d is %d  ",n,f);
getch();
}

int fact(int m )
{
If(m=0)
return(1);
else
return(m*fact(m-1));
}
```

Output:

```
enter a number
5
the factorial of 5 is 120
```

Nesting of Functions

Function within another function is called nested function.

The syntax is:

```
function1(a)
{
------------------
------------------
------------------
Y=function2();
}
```

Write a C program to illustrate nesting of function.

```
#include<stdio.h>
#include<conio.h>
int sum(int, int );
void read();
```

```
void main( )
{
clrscr( );
read();
getch();
}

void read()
{
int a,b,c
printf("\n enter two numbers  \n");
scanf("%d%d",&a,&b);
c=sum(a, b);
printf("\n the sum is %d  ",c);
}

int sum(int m, int n )
{
int res;
res=m+n;
return(res);
}
```

Arrays and Functions

We can pass an entire array from one function to another. To pass an entire array to a function we will have to just pass the name of the array as the actual argument.

When we pass normal variable other than array, C makes a copy of the data and places it in a memory location associated with the receiving variable.