Functions

Function is a logically grouped set of statements that perform a specific task. In C program, a function is created to achieve something. Every C program has at least one function i.e. *main()* where the execution of the program starts. It is a mandatory function in C.

Advantages of Function

The advantages of using functions are:

- Avoid repetition of codes.
- Increases program readability.
- Divide a complex problem into simpler ones.
- Reduces chances of error.
- Modifying a program becomes easier by using function.

Components of Function

A function usually has three components. They are:

1. Function Prototype/Declaration
2. Function Definition
3. Function Call

1. Function Prototype/Declaration

Function declaration is a statement that informs the compiler about

- Name of the function
- Type of arguments
- Number of arguments
- Type of Return value

Syntax for function declaration

returnType function_name ([arguments type]);

For example,

```
void sort(int []);          /*function name = sort, receives an array
                              of integer as argument and returns nothing*/
int product(int,int);       /*function name = product, receives two integers
                              as argument and returns an integer*/
```

A function declaration doesn't require name of arguments to be provided, only type of the arguments can be specified.

2. Function Definition

Function definition consists of the body of function. The body consists of block of statements that specify what task is to be performed. When a function is called, the control is transferred to the function definition.

Syntax for function definition

```
returnType function_name ([arguments])
{
   statement(s);
   ... ... ...
}
```

Return Statement

A return statement is used to return values to the invoking function by the invoked function. The data type of value a function can return is specified during function declaration. A function with void as return type don't return any value. Beside basic data type, it can return object and pointers too. A return statement is usually place at the end of function definition or inside a branching statement.

Syntax of return statement

```
return value;
```

For example,

```
int product (int x, int y)
{
   int p = x*y;
   return p;
}
```

In this function, the return type of *product()* is int. So it returns an integer value p to the invoking function.

3. Function call

A function call can be made by using a call statement. A function call statement consists of function name and required argument enclosed in round brackets.

Syntax for function call

function_name ([actual arguments]);

For example,

sort(a);

p = product(x,y);

A function can be called by two ways. They are:

- Call by value
- Call by reference

Call by value

When a function is called by value, a copy of actual argument is passed to the called function. The copied arguments occupy separate memory location than the actual argument. If any changes done to those values inside the function, it is only visible inside the function. Their values remain unchanged outside it.

Example: **C program to find the sum of two numbers using function (call by value)**

```
#include<stdio.h>
int sum(int,int);     // function prototype

int main()
{
   int a,b,s;
   printf("Enter two integers:");
   scanf("%d%d",&a,&b);
   s=sum(a,b);      // function call
   printf("Sum = %d",s);
   return 0;
}

int sum(int x,int y)  // function definition
{
   return (x+y);       // return statement
}
```

In this program, two integer values are entered by user. A user-defined function *sum()* is defined which takes two integer as arguments and return the sum of these values. This function is called from main() and the returned value is printed.

Output

```
Enter two integers:11 6
Sum = 17
```

Call by reference

In this method of passing parameter, the address of argument is copied instead of value. Inside the function, the address of argument is used to access the actual argument. If any changes is done to those values inside the function, it is visible both inside and outside the function.

Example: C program to swap two values using function (call by reference).

```
#include <stdio.h>
void swap(int *, int *); // function prototype

int main()
{
   int a,b;
   printf("Enter two numbers: ");
   scanf("%d%d",&a,&b);
   printf("Before swapping \n");
   printf("a = %d \n",a);
   printf("b = %d \n",b);
   swap(&a,&b); // function call by reference
   printf("After swapping \n");
   printf("a = %d \n",a);
   printf("b = %d \n",b);
   return 0;
}

void swap(int *x, int *y) // function definition
{
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
}
```

This program swaps the value of two integer variables. Two integer values are entered by user which is passed by reference to a function swap() which swaps the value of two variables. After swapping these values, the result is printed. In this program, the arguments must be passed by value because we are changing the values of two variables inside a function. If they are passed by value then the change won't be seen outside the function.

Output

Enter two numbers: 12 35
Before swapping
a = 12
b = 35
After swapping
a = 35
b = 12

Types of function

There are two kinds of function. They are:

1. Library functions
2. User-defined functions

1. Library functions

Library functions are the built in function that are already defined in the C library. The prototype of these functions are written in header files. So we need to include respective header files before using a library function. For example, the prototype of math functions like *pow(), sqrt()*, etc is present in math.h, the prototype of *exit(), malloc(), calloc()* etc is in stdlib.h and so on.

2. User-defined functions

Those functions that are defined by user to use them when required are called user-defined function. Function definition is written by user. *main()* is an example of user-defined function.