Chapter: 6
Functions

Advanced College of Engineering And Management.

Topics to be covered

Introduction

Concept Associated with function

- Function Declaration or Prototype
- Function Definition
- Passing Arguments
- Return Statement
- Function Call
- Elimination of Function Declaration

Categories of Function

Categories of user defined function

- Function with arguments and return value
- Function with arguments and no return value
- Function with no arguments and return value
- Function with no arguments and no return value

Topics to be covered

Ways of Passing arguments to function

- Pass By value
- Pass By Reference

Function main()

Scope of Variables(Local and Global Variables)

Extent of variables(life time,longevity)

Stack

- Recursion
- Recursive Function

Comparing Loops with Recursive Function

Towers of Hanoi Problem

Introduction

- In c, we can divide a large program into the basic building blocks known as function
- A function is a group of statements that performs **some specific**, **well-defined** task.
- The function is also known as *procedures* or *subroutines* in other programming languages.
- Every C program has at least one function, which is main() and Additional functions will be subroutine to main.

Advantages of Function

- Redundancy and Reusability: If functionality is performed at multiple places in software, then rather than writing the same code, again and again, we create a function and call it everywhere.
- Modularity and Readability :It reduces the complexity of a program and gives it a modular structure. And enhances the readability by breaking the code in smaller functions .
- Maintenance and Debugging :In case of errors ,functions can be individually tested and maintained instead of debugging same thing in whole program.

Concept Associated with function

- Function Declaration or Prototype
- Function Definition
- Passing Arguments
- Return Statement
- Function Call
- Elimination of Function Declaration

Example

```
// Program to add two numbers
#include<stdio.h>
#include<conio.h>
int sum(int,int);
void main()
{
      int a,b;
      a=2;
      b=3;
      s=sum(a,b);    //Function Call
      printf("Sum is %d",s);
      getch();
}
int sum(int x, int y)      //Function Definition
{
      return(x+y);
}
```

Function Declaration

Main Function Section

User Defined Function

Function Declaration or Prototype

- Provides information to the compiler
  - Name of the function
  - Type of the value to be returned(optional, default return type is integer)
  - Number of arguments that must be supplied in a call to the function
  - Type of Arguments
- When a function call is encountered, the compiler checks the function call with its declaration so that correct arguments types are used.
- **Syntax:**
  - Return_type function_name(type 1, type 2, type 3, …, type n);
- **Examples :**

float add(int,int);

Or,

float add(int a, float b);

Function Definition

- **A function definition has two principal components:**
  - **Function Declarator**
  - **Function Body/Body of the Function**
- **Function Declarator**: We must declare as same as the function declaration[function name, no of arguments and their types, return type] must be same.
- **Syntax:**
  - Return_type function_name(type1 arg1, type2 arg2, type3 arg3, …, type n argn);

- **Here,**
  arg1 agr2 arg3 … argn n are called **Formal Arguments** or **formal parameters.**
- **Function Body**: Function declarator is followed by function body started and ended with curly brackets and contains the tasks to be done by that function

Passing Arguments

- Providing values to the **formal arguments** of called function through **the actual arguments** of calling function is called passing arguments

Example:

```
void add( int, int);
void main(){
    int a=5,b=6;
    add(a , b);
}
void add( int x, int y)  {
    printf("Sum =%d", x+y );
}
```

Actual Arguments

Formal Arguments

Note: **main()** is Calling function because it calls add function and **add()** function is Called function because it is being called by main()

Return Statement

- A Function may or may not send back any value to the calling function. If it does, it is through **return** statement
- The called function can **only return one value** per call, at the most.
- **Syntax** :

return ; or **return(**expression**);**

- Plain return does not return any value and acts as a closing bracket of the function.
- Example: void main(){ return; }

Function call

- Function call is an inactive part of the program which comes in to life when a call is made to the function
- A function call is specified by function name followed by the values of parameters enclosed within parenthesis and terminated by a semicolon.
- **Example :**

add(a , b);

- **Note:** the **number**, **type** and **order of arguments** along with **function name** should be same in ***function declaration*** , ***function call*** and ***function declarator***

Elimination of Function Declaration

- If functions are **defined** before **they are called** , then declaration is **unnecessary**.
- **Example:**

```
#include<stdio.h>
#include<conio.h>
void add( int x, int y)   {
      printf("Sum =%d", x+y );
}
void main(){
      int a=5,b=6;
      add(a , b);
}
```

Categories of Function

- Library Functions :

These functions **are already written**, **compiled** and **placed** in libraries. i.e, the prototype and data definitions of these functions are present in their respective header files. To use these functions we need to include **the header file** in our program.

Examples: printf(),scanf(),sqrt(),pow(),getch(),clrscr(),etc

- User Defined Functions : These are defined and used by programmers according to user requirements. They are categorized into 4 types on the basis of **arguments present or not** and **a value is returned or not**.
  - Function with arguments and return value
  - Function with arguments and no return value
  - Function with no arguments and return value
  - Function with no arguments and no return value

Function with arguments and return value

```
#include<stdio.h>
#include<conio.h>
float add(int, float);
void main()
{
int a;
float b, sum;
printf("Enter values of a and b : ");
scanf("%d%f", &a, &b);
sum=add(a , b);
printf("\nThe sum = %f",sum);
getch();
}
```

```
float add(int p, int q)
{
float s;
s=p+q;
return s;
}
```

Function with arguments and no return value

```
include<stdio.h>
#include<conio.h>
void add(int, float);
void main()
{
int a;
float b;
printf("Enter values of a and b : ");
scanf("%d%f", &a, &b);
add(a , b);
getch();
}
```

```
void add(int p, int q)
{
float s;
s=p+q;
printf(“Sum=%f”,s);
}
```

Function with no arguments and no return value

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

```
void add()
{
        int a;
        float b, sum;
        printf("Enter values of a and b
        : ");
        scanf("%d%f", &a, &b);
        sum=a+b;
        printf("\nThe sum = %f", sum);
}
```

Function with no arguments and return value

```
include<stdio.h>
#include<conio.h>
float add();
void main(){
float x;
x=add();
printf("\nThe sum = %f", x);
getch();
}
```

```
float add()
{
    int a;
    float b;
    printf("Enter values of a and b : ");
    scanf("%d%f", &a, &b);
    return a+b;
}
```

Ways of Passing Arguments to functions

- C provides two different mechanisms to pass arguments
  - Pass By Value
  - Pass By Reference

- **Pass By Value:** The process of passing actual value of variables is know as passing by value.In this mechanism, the values of actual arguments are copied to the formal arguments of the function definition. So the value of arguments in the calling function are not even changed even they are changed.

Example: WAP to swap two numbers using pass by value.

```
void  swap(int,int);
void main()
{
int a,b;
printf(“Enter values of a and b");
scanf("%d %d", &a,&b);
swap(a,b);
printf(“In Calling Function”);
printf(“a= %d, b=%d”, a , b);
}
```

```
void swap(int a, int b){
int temp;
temp =a;
a=b;
b=temp;
printf(“In called function”);
printf(“a= %d, b=%d”, a, b);
}
```

Pass by Reference

- Reference means address
- The process of calling a function giving address of variables as arguments is called passing by reference. Pointer variables are used for this purpose.
- Here, address of variables(&) are passed and pointer variables(*) to receive the address.

Example: WAP to swap two numbers using pass by reference.

```
void swap(int*,int*);
void main()
{
int a,b;
printf("Enter values of a and b");
scanf("%d %d", &a,&b);
swap(&a, &b);
printf("In Calling Function");
printf("a= %d, b=%d", a , b);
}
```

```
void swap(int *a, int *b){
int temp;
temp =*a;
*a=*b;
*b=temp;
printf("In called function");
printf("a= %d, b=%d", a, b);
}
```

Function main()

- It is also a user defined function except for its name, number and type of arguments.
- Starting point of a program execution
- Must have a main function
- The general form of main is

```
main()
{

}
```

- The above statement is sufficient for the execution of program though it does not do any useful operation.

Scope of Variables

On the basis of declaration, variables are categorized as **local** and **global**.

Local:.

They are generally implemented using a stack.

They only exists inside the specific function that creates them.

They are unknown to other functions

They are executed each time a function is executed or called.

These variables only exist inside the specific function that creates them.

Global Variables:

- A global variable is declared outside all the functions present in a program
- Can be accessed by any function in the program
- remains in existence till the whole program get executed completely
- helpful in situations where multiple functions are accessing the same data.

Extent of Variables

- THe time period during which memory is associated with a variable is called extent of variable.
- The lifetime of a variable is categorized by its storage classes
- The storage classes in C are :
  - ▫ auto variables
  - ▫ register variables
  - ▫ static variables
  - ▫ extern variables

auto variables :

- All variables declared within functions are auto by default
- can only be accessed within the function or nested block within they are declared.
- created when the control is entered into the functions and destroyed when the control is exited from the function
- Example: auto int a;
  auto float b
- register variables :
- Declared by using prefix register and can be declared only limited times
- are stored in the registers of the microprocessor.
- executes faster
- Loop indices, to be accessed more frequently, can be declared as register variables.
- Example : register int i;

static variables :

- value of static variables persists until the end of the program.
- can be initialized only once
- Static variable may be internal or external type, depending on the place of declaration
- Example: static int a;

- Example:

```
include<stdio.h>
#include<conio.h>
void staticdemo();
void main()
{       int i;
for(i=1;i<=5;i++) staticdemo();
getch();
}
```

```
void staticdemo()
{
static int p=150;
p=p+10;
printf("p=%d\n",p);
}
```

Output

p=160
p=170
p=180
p=190
p=200

extern variables :

- variables that are active and alive throughout the entire program are known as external variable
- Having a program distributed in multiple files uses external variables
- The keyword extern makes it possible to use a global variable in multiple files
- Example:

```
include<stdio.h>
#include<conio.h>
void staticdemo();
void main()
{       int i;
for(i=1;i<=5;i++) staticdemo();
getch();
}
```

```
void staticdemo()
{
static int p=150;
p=p+10;
printf("p=%d\n",p);
}
```

Output

p=160
p=170
p=180
p=190
p=200

Summary

Summary

Storage Classes	Place	Default Value	Scope	Lifetime
Auto	RAM	garbage value	Local	Till the control remains within the block in which the variable is defined
Extern	RAM	zero	global	Value of the variable persists between different function calls
Static	RAM	zero	local	Till the control remains within the block in which the variable is defined. If it is defined globally it is visible to all the functions.
register	Register	garbage value	local	As long as the program execution does not come to the end

Stack

- It is a data structure that works on the principle of last in first out (LIFO).
- the last item put on the stack is the first item that can be taken out.
- process of storing data in stack is called push and retrieving data from stack is called pop