• Chapter 13 : Functions in C

13.1 Overview

Q : 13-01-01 : Describe Unstructured and Structured Programming ? Define Function and differentiate between Unstructured and Structured Programming ?

Answer :

Overview : The idea of modular programming is the result of inspiration from the hardware manufacturing where replicable components of different items are available. If a component of an item gets out of order, it is replaced with a newer one. Many different components from different manufacturers can be combined together to form a hardware device such as computers, cars, and washing machines.

Functions : Functions are the building blocks of C programs. They encapsulate pieces of code to perform specific operations. Functions allow us to accomplish the similar kinds of tasks over and over again without being forced to keep adding the same code into the program. Functions perform tasks that may need to be repeated many times.

Unstructured Programming : When the whole program logic is contained in a single main function, the style of writing programs is known as unstructured programming.

Structured Programming : It is a modular way of writing programs. The whole program logic is divided into number of smaller modules or functions. The main function calls these functions where they are needed. A function is a self-contained piece of code with a specific purpose.

Difference between Unstructured and Structured Programming : In Unstructured Programming, the entire logic of the program is implemented in a single module (function), which causes the program error prone, difficult to understand, modify and debug. Whereas in Structured Programming, the entire logic of the program is divided into number of smaller modules, where each module (piece of code) implements a different functionality.

13.2 Importance of Functions

Q : 13-02-01 : Describe Importance / Benefits of Use of Functions ?

Answer :

Importance of Functions : A program may have repetition of a piece of code at various places. Without the ability to package a block of code into a single function, programs would end up being much larger. But the real reason to have functions is to break up a program into easily manageable chunks.

Benefits of Functions : The use of functions provides several benefits :

Easier to Understand : They make programs significantly easier to understand and maintain. The main program can consist of a series of function calls rather than countless lines of code.

Reusability of Code : Functions increase reusability of the code. Well written

functions may be reused in multiple programs. The C standard library is an example of the reuse of functions.

Parallel Development of The Software : Different programmers working on one large project can divide the workload by writing different functions, hence ensuring the parallel development of the software.

Repeated Execution : Functions can be executed as many times as necessary from different places in the program.

Easy Debugging : When an error arises, rather than examining the whole program, the infected function is debugged only.

13.3 Types of Functions

Q : 13-03-01 : Explain the TWO Types of Functions ?

Answer :

Types of Functions : There are two types of functions in C :

Built-In Functions : Built-in functions are predefined functions that provide us convenient ways to perform variety of tasks. These functions are packaged in libraries. Through these functions we can easily access complex programming functionality. We should not re-invent the wheel. All that we need to do is just making a function call and the rest of the task is performed by the called function. Some of the built-in functions, e.g., stdio library and some of the functions defined in it, printf and scanf, which are defined in the library of standard input / output, similarly we have getch and getche functions which are defined in the library of console input / output.

Important Note : To use a Built-In Function in C, we need to include the header file defining the function. To use printf() and scanf(), we have to include stdio.h file in our program.

User-Defined Functions : Built-In Functions are not sufficient for solving every kind of problem. A programmer may need to write own functions depending on the nature of problem being solved. Such functions are called user-defined functions.

13.4 Writing Functions In C

Q : 13-04-01 : Explain the procedure of writing a Function in C ?

Answer :

Writing Functions In C : Every function in C has almost the same basic structure as that of the main() function. A function in C consists of a function header which identifies the function followed by the body of the function between curly braces containing the executable code for the function. Every function in C is written according to the following general form :

```
returen_type FunctionName (parameter_list) // number, order and types of parameters
{
        Executable Statement (s) ;
```

```
        return expression ;
    }
```

Function Header : The first line of function definition is called the function header i.e.

```
    return_type FunctionName (parameter_list)
```

It consists of three parts : The type of the return value, the name of the function and the parameters of the function enclosed in parentheses.

The Return Type : The return_type can be any valid data type. If the function does not return a value, the return type is specified by the keyword void. A function that has no parameter specifies the keyword void as its parameter list. Hence, a function that has no parameter and does not return any value to the calling function, will have the header :

```
    void FunctionName (void)
```

However the keyword void is optional. The above function header for a function that has no argument can be re-written as follows :

```
    void FunctionName ()
```

The Function Body : Variables declaration and the program logic are implemented in the function body. Function body makes use of the arguments passed to the function. It is enclosed in curly braces. A function can be called in the body of another function.

The return Statement : The return statement is used to specify the value retuned by a function.

The general form of return statement is :

```
    return [expression] ;
```

When the return statement is executed, expression is evaluated and returned as the value of the function. Execution of the function stops when the return statement is executed, even if there are other statements still remaining in the function body. If the type of the return value has been specified as void in the function header then there is no need to use a return statement.

13.5 Function Prototype

Q : 13-05-01 : Explain the Function Prototype in C ?

Answer :

Writing Functions In C : The compiler must know functions used in the program. That's why we include corresponding header files in the source program before using built-in functions such as stdio.h and conio.h etc. A header file contains the prototypes of the functions provided by the library. The compiler actually needs enough information to be able to identify the function that we are using. A function prototype is a statement that provides the basic information that the compiler needs to check and use a function correctly. It specifies the parameters to be passed to the function, the function name, and the type of the return value. The general form of the function prototype is as follows :

```
    Return_type FunctionName (parameter_list) ;
```

The prototype for a function which is called from another function must appear before the function call statement. Functions prototypes are usually placed at the beginning of the source file just before the function header of the main function.
Important Note : Function Prototype looks like Function Header but with a Semi Colon (;) at the end.

13.6 Calling A Function

Q : 13-06-01 : Explain the Function Call in C ?

Answer :
Function Call In C : Function call is a mechanism that is used to invoke a function to perform a specific task. A function call can be invoked at any point in the program. In C the function name, the arguments required and the statement terminator (;) are specified to invoke a function call.
When function call statement is executed, it transfers control to the function that is called. The memory is allocated to variables declared in the function and then the statements in the function body are executed. After the last statement in the function is executed, control returns to the calling function.

13.7 Local Variables And Their Scope

Q : 13-07-01 : Explain Local Variables in C, their Life Time, Memory Allocation / De-allocation and The Scope of a Local Variable with an example program ?

Answer :
Local Variables And Their Scope In C :
Life Time of The Variable : When the program executes, all variables are created in memory for a limited period. They come into existence from the place where they are declared and then they are destroyed. The duration in which a variable exists in memory is called life time of the variable.
Memory Allocation / De-allocation : Operating System (Memory Management Module) manages the allocation (reserve space in RAM) and de-allocation (free space in RAM) of memory for all variables in our program. Destroying a variable means returning the memory allocated to a variable back to the operating system for other programs. The value stored in such a variable is lost for ever.
The Scope of a Variable : The scope of a variable refers to the region of a program in which it is accessible. The name of a variable is only valid within its scope. So a variable can not be referred outside its scope. Any attempt to do so will cause a compiler error.
Local Variables : All variables declared within a block (within the extent of a pair of curly braces) are called local variables and have local scope. The scope of a local variable is from the point in the program where it is declared until the end of the block containing its declaration.
Example :

```
#include<stdio.h>
void main (void)
{
        int nCount = 0 ;
        if (nCount == 0)
        {
                int chk ;
                chk = 10 ;
        }
        printf("%d", chk) ;
}
```

Explanation : Two variables are used in this program; these are nCount, and chk. Both of these are local variables. But, they have different scope. The scope of the variable nCount is the block of main( ) function i.e., from its point of declaration to the end of the main( ) function. Whereas the scope of the variable chk is the block of if statement i.e., from its point of declaration until the end of the block of if statement.

These variables can only be referenced within their respective scopes. Any reference made to them outside of their scopes would be illegal, thus the program causes the following compiler error :

'chk' : undeclared identifier

This is because in the last printf( ) statement of the program, the variable chk is referenced outside of if block i.e., out of its scope, which is illegal. The lifetime of local variables is the duration in which the program control remains in the block in which they are declared. As soon as the control moves outside of their scope, these variables are destroyed.

13.8 Global Variables And Their Scopes

Q : 13-08-01 : Explain Global Variables in C, their Life Time, Memory Allocation / De-allocation and The Scope of a Local Variable with an example program ?

Answer :

Global Variables : The variables which are declared outside all blocks i.e. outside / above the main( ) and all other functions are called global variables and have global scope.

Global Scope : The global variables are accessible from the point where they are declared until end of the file containing them. It means they are visible throughout all the functions in the file, following their point of declaration.

Life Time of Global Variables : The lifetime of global variables is from the start of the program till its termination. They exist in memory from the start to the end of the program.

Example :

```
#include<stdio.h>
#include<conio.h>
void main (void)
```

```
{
        int nCount = 1 ;
        clrscr( ) ; // a function of conio.h used to clear screen
        while (nCount <= 10)
        {
                int chk = 10 ;
                printf("%d\t", chk) ;
                chk = chk + 1 ;
                nCount++ ;
        }
}
```

Important Note : Every time the block of statements containing a declaration for a local variable is executed, the variable **is** created anew, and if we specify an initial value for the local variable, it will be re-initialized each time it is created. Thus the previous value will be lost for ever.

Explanation : In the example program above, each repetition of the loop prints the same value of the variable. The addition to the value of chk will have no effect, because at the end of execution the body of the loop, the control moves outside the loop body (which is also the scope of chk variable) and returns to the while statement, this causes the chk variable to be destroyed it each repetition The chk variable is again created in the next repetition and gets destroyed at the end of the repetition This process continues until the loop condition is true.

Example :

```
#include<stdio.h>
#include<conio.h>
void Counter (void) ;
int nCount = 0 ;
void main (void)
{
        for (int n = 0 ; n <= 10 ; n += 2)
                Counter () ;
        printf("nCount = %d", nCount) ;
}
void Counter (void)
{
        nCount++ ;
}
```

Explanation : This is a simple program which demonstrates the use of global variable. Here, we have declared a global variable i.e., nCount outside the main and the Counter functions. This is not contained in any block. The global variable nCount, the function main, and the function Counter all are defined in the same file. Because, the variable nCount is declared on top of the two functions, therefore it is visible within them. The function Counter, increments the value of nCount by one each time it is called. The main( ) executes a loop six times and call the function Counter to increment the value of nCount. The value of the variable nCount is printed as the final output of the program i.e.,

Output :

nCount = 6

Important Note : The variable nCount is declared outside the functions main( ) and Counter( ), but these two functions can access and manipulate it as if it was declared within these. The nCount is created in the memory before the start of execution of main( ) and exists until the execution of the program ends.

13.9 Functions Without Arguments

Q : 13-09-01 : Explain Functions Without Arguments with an example program ?

Answer :

Functions Without Arguments : The simplest type of function is one that returns no value and no arguments are passed to them. The return type of such functions is void and the Parameter_List may either be empty or containing the keyword void.

Example : Write a function named Print_Asterisks that will print asterisks (*) according to the pattern shown and invoke a function call from the function main to print the asterisks.

```
******
*****
****
***
**
*
#include<stdio.h>
void Print_Asterisks (void) ; // function prototype
void main(void)
{
        // Function call
        Print_Asterisks ( ) ;
}
void Print_Asterisks (void) // function header
{
        // Function Body Starts
        int inner ;
        for (int outer = 7 ; outer >= 1 ; outer--)
        {
                inner = 1 ;
                while ( inner <= outer)
                {
                        printf(“*”) ;
                        inner++ ;
                }
                printf (“\n”) ;
        }
        // Function Body Ends
}
```

Explanation : The next line to the #include<stdio.h> directive is the prototype for the function Pririt_Asterisks( ). It tells the compiler about the function, its return_type and number of parameters (void in this case). Our main function consists of just one line of code i.e.,

Print_Asterisks ( ) ;

It represents a function call to the function Print_Asterisks( ). We can think of a function as a worker who takes necessary steps to accomplish the task assigned to him. The function Print_Asterisks ( ) is capable of printing asterisks in a specific order. When the function call statement is executed, the control is immediately transferred to the Print_Asterisks( ) function. Memory is allocated to the variables inner and outer. Then comes the for and while loops, which print asterisks. When the task is completed, the control is transferred to the function main from the function Print_Asterisks( ), and the memory allocated to the variables inner and outer is returned to the operating system again. Then, the control is transferred to the next statement to the function call statement in the calling function i.e., main( ). As there is no statement in the main function other than the function call, so the program will terminate.

13.10 Functions that Return a Value and Accept Arguments / Parameters

Q : 13-10-01 : Explain Functions that Return a Value and Accept Arguments / Parameters with example program(s) ?

Answer :

Functions that Return a Value and Accept Arguments / Parameters : We may need a function that could return a value and arguments could be passed to it. We have seen a number of such built-in library functions e.g., sqrt( ), toupper( ), tolower( ) etc. Let's consider the general form of function header :

return_type FunctionName (parameter_list)

The return_type specifies the data type of the value that the function returns. Parameter_list is a comma separated list which specifies the data type and the name of each parameter in the list.

Example :

```
#include<stdio.h>
#include<conio.h>
int Add (int num1, int num2) ;
void main (void)
{
        int a, b ;
        int sum ;
        clrscr( ) ; // clears the screen
        printf ("Enter Values for 'a' and 'b' => ") ;
        scanf ("%d %d", &a, &b) ;
        sum = Add(a, b) ;
        printf("%d + %d = %d", a, b, sum) ;
}
int Add(int num1, int num2)
```

```
{
        return num1 + num2 ;
}
```

Explanation : If we enter 12 and 15 for a and b respectively then the output of the of the program will be :

12 + 15 = 27

The 6th line of code in the main function is a function call to the Add( ) function. The Add( ) requires two parameters of type int to be passed to it. In the function call, we have passed two variables i.e., a and b of type int to the function. These arguments (i.e., variables a and b) are called actual arguments or actual parameters of the function. These are local variables and their scope is the body of main( ) function. Whereas the parameters specified in the function header (i.e., int num1 and int num2) are called formal arguments or formal parameters of the function and their scope is the body of Add( ) function. These are also called dummy arguments. When parameters are passed to a function, the value of actual parameters is copied in the formal parameters of the function. The function uses its formal parameters for processing data passed to it. Any change made to the value of formal parameters does not affect the value of actual parameters. Here, the values of a and b are copied in num1 and num2 respectively. The function Add( ) returns the sum of the two values to the main function which is then assigned to the variable sum.

Example :

```
#include<stdio.h>
#include<conio.h>
float Area_of_Triangle (int base, int altitude) ; // Function
Prototype
void main (void)
{
        int a, b ;
        float area ;
        clrscr( ) ; // clears the screen
        printf (“Enter Value for Altitude => ”) ;
        scanf (“%d”, &a) ;
        printf (“Enter Value for Base => ”) ;
        scanf (“%d”, &b) ;
        area = Area_of_Triangle (a, b) ;
        printf(“Area of Triangle = %.2f”, area) ;
}
float Area_of_Triangle (int base, int altitude) // Function
Header
{
        return (0.5 * base * altitude) ;
}
```

Explanation : Suppose we enter 25 and 45 for altitude and base respectively, then the program output will be :

Area of Triangle = 562.50

Exercise 13

Q-8. Write a program that calls two functions Draw_Horizontal and Draw_Vertical to construct a rectangle. Also write functions Draw_Horizontal to draw two parallel horizontal lines, and the function Draw_Vertical to draw two parallel vertical lines.

Answer :

```
#include<math.h>
#include<stdio.h>
void Draw_Horizontal( ) ; // Function Prototype
void Draw_Vertical( ) ; // Function Prototype
void main (void)
{
        clrscr( ) ; / clears screen
        Draw_Horizontal( ) ; // draws upper horizontal line
        Draw_Vertical( ) ; // draws two vertical lines
        Draw_Horizontal( ) ; // draws lower horizontal line
}
void Draw_Horizontal( ) // Function Header
{       // Go to new line and display 25 stars
        printf("\n") ;
        for (int i = 0 ; i < 25 ; i++)
                printf("*") ;
}
void Draw_Vertical( ) // Function Header
{
        for (int i = 0 ; i < 15 ; i++)
                printf("\n*                       *") ; // 23 Spaces
                between stars
        }
        // We can write Draw_Horizontal( ) function like this also –
have only one
void Draw_Horizontal( ) // Function Header
{       // Go to new line and display 25 stars
        printf("\n*************************") ;
}
```

Q-9. Write a program that prompts the user for the Cartesian coordinates of two points $P_1 (x_1, y_1)$ and $P_2 (x_2, y_2)$ and displays the distance between them. To compute the distance, write a function named Distance( ) with four input parameters. The function Distance( ) uses the following distance formula to compute the distance and return the result to the calling function :

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Answer :

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

```
float Distance(int x1, int y1, int x2, int y2) ;
void main (void)
{
        int x1, y1, x2, y2 ;
        float distance ;

        clrscr( ) ; // clears screen

        printf("\nEnter x-Coord of Point 1 => ") ;
        scanf("%f", &x1) ;
        printf("\nEnter y-Coord of Point 1 => ") ;
        scanf("%f", &y1) ;
        printf("\nEnter x-Coord of Point 2 => ") ;
        scanf("%f", &x2) ;
        printf("\nEnter y-Coord of Point 2 => ") ;
        scanf("%f", &y2) ;

        distance = Distance(x1, y1, x2, y2) ;
        printf("\nDistance Between P1 and P2 = %.2f",
distance) ;
}
float Distance(int x1, int y1, int x2, int y2)
{
        return (sqrt(((x2 – x1) * (x2 – x1)) + ((y2 – y1) * (y2 –
        y1)))) ;
}
```

Q-10. Write a program that prompts the user to enter a number and then reverses it. Write a function Reverse to reverse the number. For example, if the user enters 2765, the function should reverse it so that it becomes 5672. The function should accept the number as an input parameter and return the reversed number.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int Reverse(int n) ;
void main (void)
{
        int a, b ;

        clrscr( ) ; // clears screen

        printf("\nEnter The Number To Reverse => ") ;
        scanf("%f", &a) ;
        b = Reverse(a) ;
        printf("\n%d Reversed = %d", a, b) ;
}
int Reverse(int n)
{
```

```
int negative = 0 ;
int first = 0, second = 0, third = 0, fourth = 0, fifth = 0
;
if (n < 0)
{
        negative = 1 ;
        n *= -1 ;
}
first           =       n % 10 ;
n               =       n / 10 ;
second          =       n % 10 ;
n               =       n / 10 ;
third           =       n % 10 ;
n               =       n / 10 ;
fourth          =       n % 10 ;
n               =       n / 10 ;
fifth           =       n % 10 ;
n               =       n / 10 ;

b       = fifth * 1 ;
b       += fourth * 10 ;
b       += third * 100 ;
b       += second * 1000 ;
b       += first * 10000 ;

if (negative = 1)
                b = b * -1 ;
        return  b ;
}
```

Q-11. Write a function named Draw_Asterisks that will print asterisks (*) according to the pattern shown and make a function call from the function main to print the asterisks pattern.

```
*********
*******
*****
***
*
```

```
#include<stdio.h>
void Draw_Asterisks (void) ; // function prototype
void main(void)
{
        // Function call
        Draw_Asterisks ( ) ;
}
void Draw_Asterisks (void) // function header
{
        // Function Body Starts
```

```
int inner ;
for (int outer = 9 ; outer >= 1 ; outer -= 2)
{
        inner = 1 ;
        while (inner <= outer)
        {
                printf("*") ;
                inner++ ;
        }
        printf ("\n") ;
}
        // Function Body Ends
}
```

Q-12. Write a function Is_Prime that has an input parameter i.e num, and returns a value of 1 if num is prime, otherwise returns a value of 0.

Answer :

```
#include<math.h>
#include<stdio.h>
int Is_Prime(int num) ;
void main (void)
{
        unsigned int a ;
        printf("Enter a Positive Number > 2 To Check
        Primality => ") ;
        scanf("%f", &a) ;
        if (a < 3)
                printf("In-appropriate Number %d to Check
                Primality", a) ;
        else
        {
                int result = Is_Prime(a) ;
                if (result == 0)
                        printf("The Number %d is NOT Prime",
                        a);
                else
                        printf("The Number %d is Prime", a);
                }
}
int Is_Prime(int num)
{
        if (num % 2 == 0)
                return 0 ;
        else
        {
                for (int i = 3 ; i <= sqrt(num) ; i += 2)
                        if (num % i == 0)
```

```
                return 0 ;
            return 1 ;
        }
    }
```

Q-13. Write a complete C program that inputs two integers and then prompts the user to enter his / her choice. If the user enters 1 the numbers are added, for the choice of 2 the numbers are divided, for the choice of 3 the numbers are multiplied, for the choice of 4 the numbers are divided (divide the larger number by the smaller number, if the denominator is zero display an error message), and for the choice of 5 the program should EXIT (terminate). Write four functions Add( ), Subtract( ), Multiply( ) and Divide( ) to complete the task.

Answer :

```
#include<stdio.h>
#include<conio.h>
void Add(int x, int y) ;
void Subtract(int x, int y) ;
void Multiply(int x, int y) ;
void Divide(int x, int y) ;
void main (void)
{
        int a, b ;
        char choice ;

        clrscr( ) ; // clears screen

        printf("\nEnter First Integer => ") ;
        scanf("%f", &a) ;
        printf("\nEnter Second Integer => ") ;
        scanf("%f", &b) ;
        printf("\nEnter Your Choice 1 .. 4 => ") ;
        scanf("%f", &choice) ;
        switch(choice)
        {
                case '1' : Add(a, b) ; break ;
                case '2' : Subtract(a, b) ; break ;
                case '3' : Multiply(a, b) ; break ;
                case '4' : Divide(a, b) ; break ;
                default   : printf("\n%c is a Bad Choice",
                choice) ; break ;
        }
}
void Add(int x, int y) ;
{
        printf("%d + %d = %d", x+y) ;
}
void Subtract(int x, int y) ;
{
```

```
                printf("%d - %d = %d", x-y) ;
}
void Multiply(int x, int y) ;
{
        printf("%d * %d = %d", x*y) ;
}
void Divide(int x, int y) ;
{
        if (x > y)
        {
                if (y == 0)
                        printf("Error – Divisor Can't be ZERO")
                        ;
                else
                        printf("%d / %d = %.2f", x / y) ;
        }
        else if (y > x)
        {
                if (x == 0)
                        printf("Error – Divisor Can't be ZERO")
                        ;
                else
                        printf("%d / %d = %.2f", y / x) ;
        }
        else
                printf("Both Numbers are EQUAL thus result =
                1")
}
```

Q-14. Write a program that prompts the user to enter a number and calls a function Factorial( ) to compute its factorial. Write the function Factorial() that has one input parameter and returns the factorial of the number passed to it.

Answer :

```
#include<conio.h>
#include<stdio.h>
long int Factorial(int n) ;
void main (void)
{
        unsigned int a ;
        long int factorial ;
        printf("Enter a Valid Number > 1 To Calculate
        Factorial => ") ;
        scanf("%f", &a) ;
        if (a < 2)
                printf("In-Valid Number < 2 To Calculate
                Factorial") ;
        else
```

```
        {
                factorial = Factorial(a) ;
                printf("%d Factorial = %ld", a, factorial) ;
        }
}
long int Factorial(int n)
{
        long int fact = 1 ;
        if (n == 1 || n == 0)
                fact = 1 ;
        else
        {
                for (int i = n ; i >= 2 ; i--)
                        fact *= i ;
        }
        return fact ;
}
```

Q-15. Write a function GCD that has two input parameters and returns the greatest common divisor of the two numbers passed to it. Write a complete C program that inputs two numbers and calls the function GCD to compute the greatest common divisor of the numbers entered.

Answer :

```
#include<math.h>
#include<stdio.h>
int GCD(int m, int n) ;
void main (void)
{
        int a, b ;
        int gcd ;
        printf("Enter TWO integers To Calculate GCD => ") ;
        scanf("%d %d", &a, &b) ;
        if (a < 1 || b < 1)
                printf("In-Valid Numbers To Calculate GCD") ;
        else
        {
                gcd = GCD(a, b) ;
                printf("GCD(%d, %d) = %d", a, b, gcd) ;
        }
}
int GCD(int m, int n)
{
        int temp ;
        if (m > n)
        {
                if (m % n == 0)
                        return n ;
```

```
            temp = n ;
            n = m % n ;
            m = temp ;
      }
      else if (n > m)
      {
            if (n % m == 0)
                  return m ;
            temp = m ;
            m = n % m ;
            n = temp ;
            }
            else
                  return n ;
      }
```