Contents

Unit 6	Functions	Page No.
6.0.	Structure of the Unit	1
6.1.	Unit Outcomes	1
6.2.	Library functions	1
6.3.	User-defined Functions	3
6.4.	Methods to call the Functions	5
6.5.	Call by value and call by reference	7
6.6.	Function call stack and activation records	9
6.7.	Recursive functions	10
6.8.	Static and Dynamic Linking	13
6.9.	Self-Assessment Questions	13
6.10.	Self-Assessment Activities	13
6.11.	Multiple Choice Questions (MCQs)	14
6.12.	Keys to MCQs	16
6.13.	Summary of the Unit	16
6.14.	Keywords	16
6.15.	Recommended Learning Resources	16

Unit 6
Functions

6.0. Structure of the Unit

6.1. Unit Outcomes
6.2. Library functions
6.3. User-defined Functions
6.4. Methods for User-defined Functions
6.5. Call by value and call by reference
6.6. Function call stack and activation records
6.7. Recursion, recursive functions
6.8. Static and Dynamic Linking
6.9. Self-Assessment Questions
6.10. Self-Assessment Activities
6.11. Multiple Choice Questions
6.12. Keys to Multiple Choice Questions
6.13. Summary of the Unit
6.14. Keywords
6.15. Recommeded Learning Resources

6.1. Unit Outcomes

After the successful completion of this unit, the student will be able to:

1. Explain the library and user-defined Functions.
2. Describe the examples using function prototype, call, and arguments
3. Discuss the call-by-value and call-by-reference methods.
4. Discuss recursive and iterative functions
5. Develop programs using library and user-defined functions.

6.2. Library Functions

C programming is based on the divide and conquer rule, where a given problem is divided into subproblems and each subproblem is implemented separately. The implementation of each subproblem is known as a function in C programming language. There can be a set of functions in a C program. A function is a self-contained block of statements that perform a specific task. With the help of functions, there are several advantages such as the same set of statements need not be repeated multiple times. Functions help to understand a lengthy and complex code. During compilations, detecting and correcting errors in functions is quicker than a single large main program. Functions are classified as:

1. Library Functions
2. User Defined Functions

The striking features of the functions are reusability and modularity. Every C program contains at least one main function (). Functions are considered as basic building blocks of a C program. To support reusability a function can be called multiple times. A function is enclosed in a bracket {} and it provides abstraction in the functionality. Library functions are built-in for a C compiler. The data definition and prototype of functions are defined in header files. For example, `printf()` and `scanf()` functions are defined in a standard input/output file called `stdio. h.` These functions are included in the user's C program and are called from its respective header file. A list of commonly used header files is given in Table 6.1.

Table 6.1 Header files in C Compiler

#	Name of the File	Description
1	`<stdio.h>`	Supports I/O functions
2	`<math.h>`	Defines various mathematical functions
3	`<string.h>`	Supports string-related functions
4	`<time.h>`	Defines data and time-related functions
5	`<limits.h>`	This file defines whether the variable can have a value beyond some limits. An unsigned character can save up to the maximum value of 255.
6	`<float.h>`	This file supports specific features of the float number library.
7	`<ctype.h>`	This file is used for dynamic memory allocation.
8	`<time.h>`	This file is used for handling date and time.
9	`<complex.h>`	This file is used for manipulating complex numbers.

The above-mentioned files include several library functions as given in Table 6.2.

Table 6.2 Library functions in Header Files

#	Name of the Function	Description	Examples
1	`printf()`	Display output data and message	`printf(“welcome”);` `int x = 10;` `printf(“%d”, x);`
2	`scanf()`	Used to read the data.	`int y;` `printf(“%d”, &y);`
3	`pow()`	To find power value.	`pow(10,2)` will display 100
4	`sqrt()`	To find a square root of a number	`sqrt(16)` will result `in 4`
5	`sin(), cos(), tan()`	To find sine, cosine, and tangent of a number.	`sin(90) will result in 1` `cos(90) will result in 0`
6	`log()`	To find the logarithmic value	`log(5.6) = 1.72`
7	`FLT_MAX,` `FLT_MIN` `INT_MAX` `INT_MIN`	Returns the maximum and minimum values of floating numbers.	`maximum value of float = 3.4028234664e+38` `minimum value of float = 1.1754943508e-38`

	CHAR_MAX CHAR_MIN		minimum value of int = -2147483648 The maximum value of int = 2147483647 minimum value of char = -128 maximum value of char = 127
8	strcat(), strcmp(), strlen(), strcpy()	Used for concatenating, comparing, copying, and calculating the length of a text.	Illustrated in Program 1

9	malloc(), free	Allocating and deallocating memory	int *p;
			p = (int *)malloc(sizeof(int));
			free p

```
  #include <stdio.h>
  #include <string.h>
  int main()
  {
        char s1[100] = "happy",
        char s2[100] = " birthday";
        // concatenates str1 and str2
        strcat(s1, s2);
        // resultant string is stored in s1
        printf(“concatenated string is %s”,str1);
        strcpy(s1, s2); // copies string 2 in string 1
        printf(“copied string is %s”,s1);
        int l = strlen(s2); // Find length of s2
        printf(“the length of string is %d” l);
        return 0;
}
```

6.3. User-defined Functions

User-defined functions are written by the user for the execution of a subtask. User-defined functions avoid duplication and reduce time and effort. The three important factors in user-defined functions are:
Function Declaration or Function Prototype: A function prototype is an interface of the user-defined functions with the main(). Prototype includes a return type, function name, list of arguments, and semicolon at the end of the statement. A function prototype helps a compiler to understand the functions

and their connection to the main(). A function declaration is also a function prototype that allows the compiler to know the return type, function name, and the number of arguments. Consider the statement:
`int addition(int, int);`
In this example the return type is `int`, the name of the function is addition and there are two arguments of the type `int`. The details shown in the function declaration, function call, and the body of the function should match.

Function Definition: The function definition shows the body of the function as shown below. It includes a set of statements and the return value. If a function is not returning any value, it is considered a `void` data type.

```
int addition(int x, int y)
{
   int z = 0;
   z = x + y;
   return z;
}
```

Function Call: A function call is a statement that transfers the control to the function definition and executes a set of statements inside the body of a function. There can be multiple function calls in a program.

An example of a function call is shown below. The variable `Answer` saves the value returned by the function.

```
Answer = addition(10,20);
```

A complete program using a function call, definition, and declaration is given in Program 2.

```
#include <stdio.h>
int addition(int, int);
int addition(int x, int y)
{
   int z = 0;
   z = x + y;
   return z;
}
int main(void)
{
  int a, b;
  printf(“enter two numbers”);
    scanf(“%d %d”, &a, &b);
    int c = addition(a,b);
   printf(“the result of addition is %d”,c);
   return 0;
 }
```

Program 2: A C Program with Functions

Details about different variables in Program 2.

Formal Parameters: The parameters that are used in the function definition to carry the actual values of variables are known as formal parameters. In the above example, the variables `x` and `y` are formal parameters.

Actual Parameters/Variables: Actual variables are used in the `main().` In Program 2, the variables `a` and `b` are the actual variables that store the input/output values.

Local Variables: Variables that are declared inside a function are known as local. The local variable has a lifetime inside a function. Outside the function, the variables can be changed. In the above example Program 2, the variable `z` is called the local variable.

Global Variable: When a variable is called outside any function, it is known as a global variable. This variable remains fixed in its data type throughout the entire C program.

6.4. Methods to Call the Functions

There are four methods to call the user-defined functions as given below.

1) Function with no argument and no returning value
2) Function with argument and no returning value
3) Function with no argument but has a returning value
4) Function with argument and returning value

1) **Function with no argument and no return value**: In this method, no arguments are passed and there is no return value from the function. The variables are declared inside the function if necessary. This is shown in Program 3. Here there are no variables passed to the function or returned from the function.

```
#include<stdio.h>
void print_message();
void print_message()
{
  printf(“hello, welcome to user-defined functions”);
}
int main(void)
{
  print_message();
  return 0;
}
```

Program 3: Function with no argument and return value

2) **Function with arguments and no return value**: A function can accept the arguments from the main(), but the function's return type can be void. This is shown in Program 4. The program checks whether the entered number is a palindrome or not. A number is a palindrome if it remains the same after reversing. For example, 121, 111, 243, madam, mom, pop, etc. The function reverse_number() is a function that reads a number as an argument and there is no return value.

```
#include<stdio.h>

void reverse_number(int);

void reverse_number(int num)
{ int temp = 0, rem, rev;
  while(num!=0)
  {   rem = num%10;
      rev = rev*10 + rem;
```

```
        num = num/10;
    }
    if(rev == temp)
    printf("Number is palindrome");
    else
    printf("Number is not  palindrome");
    }



  int main(void)
  {
    reverse_number(10);
    return 0;
}
Sample Input/Output
Number is not a palindrome
```

Program 4: Function with argument but no return value

3) **Function with no argument but has a returning value**: A function can be without any arguments, but there is a processing of information and return value as given in Program 5.

```
  #include <stdio.h>
  float area_calculate(void);
  float area_calculate(void)
  {
        float r = 10, a;
        a = 3.14 * r * r;
        return a;
  }
  int main()
  {
        float a;
        a = area_calculate();
        printf("Area of a circle is %f", a);
        return 0;
}
Sample Input/Output
Area of a circle is 314.000000
```

Program 5 Function without argument but with return value

4) **Function with argument and returning value:** This is the most used function that has both the arguments and the return value. An example of this type of function is shown in Program 6.

```
      #include <stdio.h>
      float area_rectangle(float, float);
      float area_rectangle(float l, float b)
      {
        float a;
        a = l * b;
        return a;
      }
      int main()
      {
        float l,b, answer;
        printf("Enter the length and breadth of a rectangle");
        scanf("%f %f",&l,&b);
        answer = area_rectangle(l,b);
        printf("Area of a rectangle is %f", answer);
        return 0;}

Sample Input/Output
Enter the length and breadth of a rectangle 12.5 12.6
12.5 12.6
Area of a rectangle is 157.500000
```

Program 6 Function with argument and returning value

6.5. Call by Value and Call by Reference

Functions can be invoked in two ways: Call by Value or Call by Reference.
There are two ways to invoke the functions: Call by value or call by reference. These methods differ in the type of values that are passed as parameters. In call by value, the actual values are passed whereas in call by reference, the address of the variables is given.
Reference is used for the address of a variable. A function can be called using values or references. In call by value method, the values of variables are passed for different computations. The values that are passed are stored in a temporary memory. The functions that are shown so far are used by the value method. An example of checking whether the number is prime or not is shown in Program 6. Here the function is using call by value method.

```
#include <stdio.h>
int check_prime(int);

int check_prime(int num)
{
    int flag, i;
    flag = 0;
     if (num == 0 || num == 1)
        flag = 1;
     for (i = 2; i <= num / 2; i++)
     {
    //If the number is divisible by i, then it is not prime
    //Change the flag to 1 for non-prime number
    if (num % i == 0) {
      flag = 1;
      break;
    }
  }
  return flag;
}
int main() {

  int n, i, answer = 0;
  printf("Enter a positive integer: ");
  scanf("%d", &n);
  answer = check_prime(n);
```

```
   if (answer == 0)
     printf("%d is a prime number.", n);
   else
     printf("%d is not a prime number.", n);

   return 0;
}
Sample Input/Output
Enter a positive integer: 13
13 is a prime number.
```

Program 7 Call-by-Value Method

In call by reference method of parameter passing, the address of the actual parameters is passed to the function as the formal parameters. An address of a variable is given using a pointer data type. A pointer is declared as given below.

```
int *p;
```

Here the * operator indicates that `p` is the pointer variable of the integer type. It holds the address or memory location where an integer value is saved. The concept of pointer is as given in Program 8.

```
  #include <stdio.h>

  int main() {
      int x, *p;
      printf("Enter a number");
      scanf("%d",&x);
      p = &x;
      printf("%d is stored at %d address",*p,p);
      return 0;
}
Sample Input/Output
Enter a number 12
12 is stored at 1044068388 address
```

Program 8 C Program using Pointers

In Program 8, the variable *p gives the value saved at the memory location and the just p displays the address of a variable. This concept of using a pointer is utilized in call-by reference. In call by reference
method, the address of an argument is copied into the formal parameters. When the actual arguments are
used, if there are any changes made in formal parameters, they are reflected in actual arguments. A call-
by-reference concept is used in Program 9 for interchanging values of two variables.

```
  #include <stdio.h>
  int main () {
  /* local variable definition */
     int x = 10;
     int y = 20;
       printf("Before interchanging the values of number 1
        is  : %d\n", x );
     printf("Before interchanging the values of number 2 is :
       %d\n", y );

     /* calling a function to swap the values */
     swap(&x, &y);

     printf("After swap, value of Number 1 : %d\n", x );
     printf("After swap, value of Number 2 : %d\n", y );

     return 0;
  }
  void swap(int *p1, int *p2) {

     int temp;

     temp = *p1; /* save the value of p1*/
     *p1 = *p2;    /* put p2 into p1 */
     *p2 = temp; /* put temp into p2 */

     return;
}
```

Sample Input/Output

```
Before interchanging the value of number 1 is  : 10
Before interchanging the value of number 2 is : 20
After the swap, the value of Number 1 : 20
After the swap, the value of Number 2 : 10
```

Program 9 Call by Reference method to interchange values of two variables

In the above program, x and y are used as pointers in the function definition. The function call includes passing addresses of two variables. There is no need to return any value from the functions to the main program. A comparison of these two methods is given in Table 6.1.

Table 6.1 Call by Value and Call by Reference

Call by Value	Call by Reference
In this method, more than one copy of parameters is stored in different memory addresses.	The address of actual parameters is passed to formal parameters. This will maintain only one copy of the value and be stored in a single location.
The values of actual parameters will not change, and there are duplicates.	In this method, both formal and actual parameters are at the same location, thus changes are made in both the parameters.
Multiple copies are prepared, resulting in more memory consumption.	A single copy is created with better memory usage.

Advantages of Functions: Functions are commonly used in C programming with many advantages as listed below.

- With the use of functions, there can be code reusability.
- Code redundancy can be reduced.
- With the help of functions, a complex program can be designed in a modular form.

- The functionality of the program can be expressed at an abstract level. For example, a function print() in a college management program can be used for printing attendance, roll nos, marks, admissions, teacher details, different departments, etc.
- Functional C program is easy to understand and manage.
- Extensive and complex programs can be transformed into smaller, easy-to-understand pieces.

6.6. Function Stack and Activation Records

Stack is a data structure in which an element that is inserted at the end is removed first, also known as the Last In First Out (LIFO) order. An example of a stack is a set of books placed one above another. The book that is kept at the end is taken out first. Stack is used as a memory storage in several computer applications. Whenever a function is called, a contiguous block of storage is necessary for a single function. This block of memory is known as the activation record, and it saves the status of the correct activation functions.

Activation Record: An Activation Record is defined as a data structure that is activated or created when
a function is invoked. The activation record includes data about the function as given below.

- Number of arguments
- Actual parameters used in the function call
- Return value
- Return address
- Stack pointer
- Local variables holding data in a function

6.7. Recursion

Recursion is a method of making a function call itself. Recursion helps in the divide and conquer concept. It is used to give easy and smaller solutions to subproblems in modular programming. C programming supports recursive functions. For example, consider the problem of adding numbers in a range from 1 to n, where n is a positive integer. A recursive function for this problem is given in Program 10.

```
#include <stdio.h>
int add(int);
int main() {
  int num;
  printf("Enter a number ");
  scanf("%d",&num);
  int ans = add(num);
  printf("%d", ans);
  return 0;
}

int add(int n) {
  if (n > 0) {
    return n + add(n - 1);
  } else {
    return 0;
    }
}
Sample Input/Output
Enter a number 4
10
```

Program 10: Recursive Program to find Addition of numbers from 1 to n

The above example includes a base function if (n >0). The recursive function is called till this condition is true. When the value of n is zero, the function returns 0 and halts. The recursive process is given as follows.

```
  10 + add(9)
  10 + ( 9 + add(8) )
  10 + ( 9 + ( 8 + add(7) ) )
  ...
  10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + add(0)
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
```

A stack allocates memory to a function when it is called from the main() program. In recursion, when a function calls itself, the local variables are created in duplicates and the memory is allocated on top of the stack. This memory is de-allocated when the base condition is reached, and the base function returns the value.

The recursive function to find the factorial of a number is given in Program 11. Here the base condition is to check whether the value of the limit or ‘n’ value reaches base condition = 1.

```
  #include<stdio.h>
  long int facto(int num);
  int main() {
      int num;
      printf("Enter a positive integer: ");
      scanf("%d",&num);
      printf("Factorial of %d = %ld", num, facto(n)um);
      return 0;
  }

  long int facto(int num) {
      if (num>=1)
          return num*facto(num-1);
      else
          return 1;
}
Sample Input/Output:
Enter a positive integer: 5
Factorial of 5 = 120
```

Program 11: Recursive C Function to Find Factorial of a Number

In the above program, the factorial of a number is illustrated as shown in Figure 6.1.

```
return 5 * facto(4)
return 5 * 4 * facto(3)
return 5 * 4 * 3 * facto(2)
return 5 * 4 * 3 * 2 * facto(1) → here base condition is reached
return 5 * 4 * 3 * 2 * 1 → 120
```

Program 11: Recursive C Function to Find Factorial of a Number

This program can also be written using an iterative approach. This is given in Program 12. Here a loop is used where the starting value is 'n' and the ending state is 1.

```
  #include<stdio.h>
  long int facto(int num);
  int main() {
      int num;
      printf("Enter a positive integer: ");
      scanf("%d",&num);
      printf("Factorial of %d = %ld", num, facto(num));
      return 0;
  }

  long int facto(int num) {
      int i,f = 1;
      for(i =num;i>=1;i--)
         f = f* i;
      return f;
    }
Sample Input/Output
Enter a positive integer: 6
Factorial of 6 = 720
```

Program 12: Factorial Program Using Iterative Approach

Table 6.1 Iteration Veres Recursion

Iteration	Recursion
Iteration is a method of repeating a set of instructions	Recursion is a method where a function calls itself directly or indirectly.
For iteration, mainly loops are used	For recursion, functions are used.
Iteration includes initial value, condition, and increment values	Recursion includes base condition and a series of function calls.
In iteration, there is an infinite loop if the condition in the iteration is not met.	In case of recursion, if the base condition is not met, then the loop is infinite.
Iteration does not use stack; hence it is faster than recursion.	There is an overhead of maintaining records, allocation, and deallocation of memory in recursion, thus it is slower.
In comparison with recursion, the memory requirement of iteration is less.	Recursion requires more memory than iteration.
The size of the C program is larger if iteration is used.	Recursion reduces the size of the program.

A comparison of iterative and recursive approaches is given in Table 6.2.

The choice of iteration or recursion depends on the problem or the application. There are several advantages and drawbacks of recursion. Recursion makes the program compact and thus saves space and time. The code readability is better, and data processing is done efficiently using recursion. Though the recursion helps apply the divide and conquer approach for problem-solving, there are certain drawbacks. If the base condition is not met, then the recursion goes in an infinite loop, which results in stack overflow and a system crash. The recursive functions have a slower execution time, which makes it less attractive. Many users find recursive programs difficult to understand and debug. The recursive approach is not suitable for certain problem solving and it cannot be used in all the applications.

6.8. Static and Dynamic Linking

When the functions are called there is a linking of header files with the user's program. All the library functions are stored in header files. The linking can be static and dynamic. Static linking refers to the linking of a program after compilation. Dynamic linking refers to the linking of library files during runtime. Thus, linking is a method of combining external program files with the user's program. The static and dynamic linking mechanisms as given below.
The library functions are copied in a single final executable file in static linking. Static linking is performed by the linker by combining the relevant library files. If a program includes a reference to the external file, then it is resolved using static linking. Static linking creates a large executable file that contains calling programs and called programs. If there are any changes in the external files, then there is a need for relinking and re-compilation. In dynamic linking, the names of external files are stored in a single executable file. When a linking is necessary, the library files are brought into the memory and these files are dynamically linked to the program. Dynamically linked files are smaller in size and take less time in comparison with static files. In dynamic linking, there are compatibility issues, whereas static binding is linking to the fixed header files, thus there are no compatibility problems.

6.9. Self-Assessment Questions

Q1. Explain the necessity of functions in C programming. [5 Marks, L1]

Q2. Explain the following Terms with a suitable example. [6 Marks, L2]

1) Function Call
2) Function Declaration
3) Function Definition

Q3. Discuss the following categories of user-defined functions with examples. [10 Marks, L2]

1) Function with arguments and return value
2) Function without argument and return value

Q4. Compare library and user-defined functions with suitable examples. [8 Marks, L2]

Q5. Explain call by reference method for user-defined functions with an example. [5 marks, L2]

Q6. Compare call-by-value and call-by-reference parameter passing mechanisms. [8 Marks, L2]

Q7. What is a recursion? Explain with an example. [8 Marks, L2]

Q8. Discuss the advantages and drawbacks of recursive functions. [8 Marks, L2]

Q9. Differentiate between actual and formal parameters using examples. [8 Marks, L1]

Q10. Discuss how the function calls are done in a program with a recursive function to find the factorial of a number. [8 Marks, L1]

6.10. Self-Assessment Activities

Q1. Design and Develop a C program to find maximum and minimum numbers between two numbers. [5 Marks, L2]
Q2. Write a C program to find the power of any number using recursion [10 Marks, L3]
Q3. Design and develop a C program sum of all natural numbers between 1 and 100. [10 Marks, L2]
Q4. What is the base condition in a recursive function? Consider a Fibonacci numbers program using recursion. Discuss the base condition in this example. [8 Marks, L1]
Q5. Demonstrate the use of stack data structure in recursive function calls. [8 Marks, L3]

6.11. Multiple-Choice Questions

Q1. What is the output of the following program? [1 Mark, L2]

```
#include <stdio.h>
int sum (int, int);
int main (void)
{
    int total;
    total = sum (10, 20);
    printf ("%d\n", total);
    return 0;
}

int sum (int a, int b)
{
      int s;
      s=a+b;
      return s;
}
```

A. 30
B. 20
C. 10
D. 50

Q2. Any C program must contain_____ [1 Marks, L1]

A. At least one function
B. Need not contain any function
C. Variables and constants

D. None of the above

Q3. _________ of the following gives return value. [1 Mark, L1]

A. static
B. for
C. return
D. const

Q4. The default return type of a function is_______ [1 Mark, L2]

A. float
B. void
C. int
D. automatic

Q5. Default parameter passing mechanism is_______ [1 mark, L3]

A. Call by Value
B. Call by Reference
C. Any of the technique
D. None of the above

Q6. Recursion is handled using _____ data structure. [1 Mark, L2]

A. Queue
B. Arrays
C. Stack
D. Structure

Q7. The recursive function can be terminated using____ [1 Mark, L3]

A. Function Call
B. Loop
C. Function Prototype
D. Base Condition

Q8. Determine the output of the following program [1 Mark, L3]

```
void print(int n)
{
if (n == 0)
return;
printf("%d", n%2);
print(n/2);
}
int main()
{
print(12);
return 0;
}
```

A. 0000
B. 0001
C. 1000
D. 0100

Q9. When a function is called from the main() program the memory is allocated using___[1 Mark, L2]

A. Random Access Memory
B. Stack
C. Arrays
D. Compiler

Q10. What is the output of the following program? [1 Mark, L3]

```
int main()
{
    printf("Happy Birthday");
    main();
    return 0;
}
```

A. Happy Birthday
B. Infinite Loop
C. Compilation Error
D. Garbage Value

6.12. Keys to Multiple-Choice Questions

Q1.30 (a)
Q2.At least one function (a)
Q3.return (c)
Q4.int (c)
Q5.Call by value
Q6.stack (c)
Q7.base condition (d)
Q8.0001 (b)
Q9.Stack (b)
Q10. Infinite loop (b)

6.13. Summary of the Unit

Functions are useful in supporting the divide and conquer concept and modular programming. A set of repetitive statements can be replaced by a user-defined function. Functions can be library or user-defined.
Function declaration specifies the return type, number, and type of arguments/parameters. A function can call itself till a bae condition is satisfied. This concept is known as recursion. Recursive functions make the program compact and reduce time and space efficiency.

6.14. Keywords:

- Library Functions

- User-defined Functions
- Function prototype, call, return
- Recursive Functions

6.15. Recommended Learning Resources

[1] Herbert Schildt. (2017). *C Programming: The Complete Reference*. 4th ed. USA: McGraw-Hill.