- Every function in the program is supposed to perform a well defined task. The moment the compiler encounters a function call, the control jumps to the statements that are a part of the called function. After the called function is executed, the control is returned back to the calling program.
- All the libraries in C contain a set of functions that have been prewritten and pre-tested, so the programmers use them without worrying about the code details. This speeds up program development.
- While function declaration statement identifies a function with its name, a list of arguments that it accepts and the type of data it returns, the function definition, on the other hand, consists of a function header that identifies the function, followed by the body of the function containing the executable code for that function
- `main()` is the function that is called by the operating system and therefore, it is supposed to return the result of its processing to the operating system.
- Placing the function declaration statement prior to its use enables the compiler to make a check on the arguments used while calling that function.
- A function having void as its return type cannot return any value. Similarly, a function having void as its parameter list cannot accept any value.

SUMMARY

- When a function is defined, space is allocated for that function in the memory. A function definition comprises of two parts: function header and function body.
- Call-by-value method passes values of the variables to the called function. Therefore, the called function uses a copy of the actual arguments to perform its intended task. This method is used when the function does not need to modify the values of the original variables in the calling program.
- In call-by-reference method, address of the variables are passed by the calling function to the called function. Hence, in this method, a function receives an implicit reference to the argument, rather than a copy of its value. This allows the function to modify the value of the variable and that change will be reflected in the calling function as well.
- Scope means the accessibility and visibility of the variables at different points in the program. A variable or a constant in C has four types of scope: block, function, file, and program scope.
- The storage class of a variable defines the scope (visibility) and life time of variables and/or functions declared within a C program.
- The auto storage class is the default storage class for variables declared inside a block. The scope of the variable is local to the block in which it is declared. When a variable is declared using register as its storage class, it is stored in a CPU register instead of RAM. Extern is used to give a reference of a global variable that is visible to all the program files. Static is the default storage class for global variables.
- A recursive function is defined as a function that calls itself to solve a smaller version of its task until a final call is made which does not require a call to itself. They are implemented using system stack.

GLOSSARY

Argument A value passed to the called function by the calling function

Block A sequence of definitions, declarations, and statements, enclosed within curly brackets.

Divide and conquer Solving a problem by dividing it into two or more smaller instances. Each of these smaller instances is recursively solved, and the solutions are combined to produce a solution for the original problem.

Iteration Solving a problem by repeatedly working on successive parts of the problem.

Recursion An algorithmic technique where a function calls itself with a smaller of the task in order to solve that task.

Recursion termination The point at which base condition is met and a recursive algorithm stops calling itself and begins to return values.

Recursion tree Technique to analyse the complexity of an algorithm by diagramming the recursive function calls.

Tail recursion A form of recursion in which the last operation of a function is a recursive call.

Tower of hanoi Given three towers or poles and *n* disks of decreasing sizes, move the disks from one pole to another one by one without putting a larger disk on a smaller one.

EXERCISES

Fill in the Blanks

1. ________ provides an interface to use the function.
2. After the function is executed, the control passes back to the ________.
3. A function that uses another function is known as the ________.
4. The inputs that the function takes are known as ________.

5. `main()` is called by the ________.
6. Function definition consists of ________ and ________.
7. In ________ method, address of the variable is passed by the calling function to the called function.
8. Function scope is applicable only with in ________.
9. Function that calls itself is known as a ________ function.
10. Recursive functions are implemented using ________.
11. ________ function is defined as a function that calls itself.
12. The function `int func();` takes ________ arguments.
13. ________ variables can be accessed from all functions in the program.
14. The execution of a program starts at ________.
15. By default, the return type of a function is ________.
16. Parameters used in function call are called ________.
17. Variable declared inside a function is known as ________.

Multiple Choice Questions

1. The function that is invoked is known as
   (a) calling function (b) caller function
   (c) called function (d) invoking function
2. Function declaration statement identifies a function with its
   (a) name
   (b) arguments
   (c) data type of return value
   (d) all of these
3. Which return type cannot return any value to the caller?
   (a) int (b) float
   (c) void (d) double
4. Memory is allocated for a function when the function is
   (a) declared (b) defined
   (c) called (d) returned
5. Which keyword allows a variable to have file scope?
   (a) auto (b) static
   (c) register (d) extern
6. The default storage class of global variables is
   (a) auto (b) static
   (c) register (d) extern
7. Which variable retains its value in-between function calls?
   (a) auto (b) static
   (c) register (d) extern
8. The default storage class of a local variable is
   (a) auto (b) static
   (c) register (d) extern

State True or False

1. The calling function must pass parameters to the called function.
2. Function header is used to identify the function.
3. The name of a function is global
4. No function can be declared within the body of another function.
5. The function call statement invokes the function.
6. Auto variables are stored inside CPU registers.
7. Extern variables are initialized by default.
8. The default storage class of local variables is `extern`.
9. Recursion follows a divide-and-conquer technique to solve problems.
10. Local variables overwrite the value of global variables.
11. A C function can return only one value.
12. A function must have at least one argument.
13. A function can be declared and defined before the `main()`.
14. A function can be defined in the `main()`.
15. Variable names in the function definition must match with those specified in the function declaration.
16. Specifying variable names in the function declaration is optional.
17. The `main()` is a user-defined function.

Review Questions

1. Define a function. Why are they needed?
2. Explain the concept of making function calls.
3. Differentiate between function declaration and function definition.
4. Differentiate between formal parameters and actual parameters.
5. How many types of storage classes C language supports? Why do we need different types of such classes?
6. Give the features of each storage class.
7. Explain the concept of recursive functions with example.
8. Differentiate between an iterative function and a recursive function. Which one will you prefer to use and in what circumstances?
9. What will happen when the actual parameters are less than formal parameters in a function?
10. What will happen when data type of a variable in the function declaration does not match with the corresponding variable in the function header?
11. What will happen when a function returns a value that does not match with the return type of the function?
12. Write a program to calculate factorial of a number using recursion. Also write a non recursive procedure to do the same job.
13. Explain the tower of Hanoi problem.
14. Write a program using function that calculates the hypotenuse of a right-angled triangle.
15. Write a function that accepts a number `n` as input and returns the average of numbers from 1 to `n`.
16. Differentiate between call by value and call by reference using suitable examples.
17. What do you understand by scope of a variable? Explain in detail with suitable examples.
18. Why function declaration statement is placed prior to function definition?
19. Write a function to reverse a string using recursion.
20. Write a function is_prime that returns a 1 if the argument passed to it is a prime a number and a 0 otherwise.
21. Write a function that accepts an integer between 1-12 to represent the month number and displays the corresponding month of the year (For example if month = 1, then display JANUARY)
22. Write a function is_leap_year which takes the year as its argument and checks whether the year is a leap year or not and then displays an appropriate message on the screen.
23. Write a program to concatenate two strings using recursion.
24. Write a program to read an integer number. Print the reverse of this number using recursion
25. Write a program to swap two variables that are defined as global variables.
26. Write a program to compute `F(x, y)` where

    ```
    F(x, y) = F(x-y, y) + 1 if y≤x
    And F(x, y) = 0 if x<y
    ```

27. Write a program to compute `F(n, r)` where `F(n, r)` can be recursively defined as

    ```
    F(n, r) = F(n-1, r) + F(n-1, r-1)
    ```

28. Write a program to compute Lambda(n) for all positive values of n where Lambda(n) can be recursively defined as

    ```
    Lambda(n) = Lambda(n/2) + 1 if n>1
    AND Lambda(n) = 0 if n =1
    ```

29. Write a program to compute `F(M, N)` where `F(M, N)` can be recursively defined as

    ```
    F(M,N) = 1 if M=0 or M≥N≥1
    AND F(M,N) = F(M-1,N) + F(M-1, N-1)
      otherwise
    ```

30. Write a menu driven program to add, subtract, multiply, and divide two integers using functions.
31. Write a program to find the smallest of three integers using functions.
32. Write a program to calculate area of a triangle using function.
33. Write a program to find whether a number is divisible by two or not using functions.
34. Write a program to illustrate call by value technique of passing arguments to a function.
35. Write a program to illustrate call by reference technique of passing arguments to a function.
36. Write a program to swap two integers using call by value method of passing arguments to a function.
37. Write a program using function to calculate x to the power of y, where y can be either negative or positive.

38. Write a program using function to calculate compound interest given the principal, rate of interest and number of years.
39. Write a program to swap two integers using call by reference method of passing arguments to a function.
40. Write a program to calculate factorial of a number (a) using recursion (b) without using recursion.
41. Write a program to convert the given string "hello world" to "dlrow olleh" without using recursion.
42. Write a program to find HCF of two numbers (a) using recursion (b) without using recursion.
43. Write a program to calculate x^y (a) using recursion (b) without using recursion.
44. Write a program to print the Fibonacci series (a) using recursion (b) without using recursion.
45. Write a program using functions to perform calculator operations.
46. Write a function that converts temperature given in Celsius into Fahrenheit.
47. Write a function that prints the conversion table of Degrees Fahrenheit into Degrees Celsius ranging from 32-212 degrees Celsius.
48. Write a function to draw the following pattern on the screen

```
********
!      !
!      !
!      !
********
```

49. Write a function to print a table of binomial coefficients which is given by the formula-

```
B(m, x) = m!/ (x! (m-x)!) where m > x
```
Hint: `B(m,0) = 1, B(0,0) = 1 and`
```
B(m,x) = B(m, x-1) * [(m - x + 1)/x]
```

50. Write a program to evaluate

$f(x) = x - x^3/3! + x^5/5! - x^7/7! + \ldots$

Program Output

Find out the output of the following codes.

1.
```
#include <stdio.h>
int func();
main()
{
   printf("%d", func());
   printf("%d", func());
   printf("%d", func());
   printf("%d", func());
   return 0;
}
int func()
{
   int counter=0;
   return counter;
}
```

2.
```
#include <stdio.h>
int func();
main()
{
   printf("%d", func());
   printf("%d", func());
   printf("%d", func());
   printf("%d", func());
   return 0;
}
int func()
{
   static int counter=0;
   return counter;
}
```

3.
```
#include <stdio.h>
int func();
int counter = 5;
main()
{
   printf("%d", func());
   printf("%d", func());
   printf("%d", func());
   printf("%d", func());
   return 0;
}
int func()
{
   return counter++;
}
```

4.
```
#include <stdio.h>
int add(int, int);
main()
```

```
{
   int a=2, b=3;
   printf("%d %d %d", a, b, add(a, b));
   return 0;
}
int add(int a, int b)
{
   int c;
   c = a+b
   return;
}
```

5.
```
#include <stdio.h>
int add(int, int);
main()
{
   int a=2, b=3;
   printf("%d %d %d", a, b, add(a, b));
   return 0;
}
int add(int a, int b)
{
   int c;
   c = a+b
}
```

6.
```
#include <stdio.h>
int func(int);
main()
{
   int a=2;
   printf("%d", func(a));
   return 0;
}
int func(int a)
{
   if(a>1)

   return func(--a) + 10;
   else

   return 0;
}
```

7.
```
#include <stdio.h>
void func(int);
main()
{
   int a=127;
         printf("%d", a);
         func(a);
      return 0;
}
void func(int a)
{
   a++;
         printf("%d", a);
}
```

8.
```
#include <stdio.h>
void func(int);
auto int a;
main()
{
   int a=10;
         printf("%d", a);
         func(a);
   return 0;
}
void func(int a)
{
   a++;
         printf("%d", a);
}
```

9.
```
#include <stdio.h>
static int add(int val)
(
         static int sum;
         sum += val;
         return sum;
}
main()
{
   int i, n=10;
         for(i=0;i<10;i++)
                  add(i);
         printf("\n SUM = %d", func(0));
   return 0;
}
```

10.
```
#include <stdio.h>
int func(int);
main()
{
   int a=2;
   printf("%d", func(a));
   return 0;
}
int func(int a)
{
    {
              ;
          }
              {
              { ;
              }
              return a;
          }
  }
```

11.
```
#include <stdio.h>
void func(int);
int a=10;
main()
{
   int a=2;
        printf("%d", a);
        func(a);
   printf("%d", a);
   return 0;
}
void func(int a)
{
   a=20;
}
```

12.
```
#include <stdio.h>
void func(char);
main()
{
   char ch=256;
        func(ch);
   return 0;
}
void func(char a)
{
        printf("%d", a);
}
```

13.
```
#include <stdio.h>
int a;
static int func()
(
       return a++;
}
main()
{
   a=10;
        printf("%d", func());
        a *= 10;
        printf("%d", func());
   return 0;
}
```

14.
```
#include <stdio.h>
int prod(int x, int y)
{
   return (x*y);
}
main()
{
   int x=2, y=3, z;
   z = prod(x,prod(x,y));
   printf("%d", z);
   return 0;
}
```