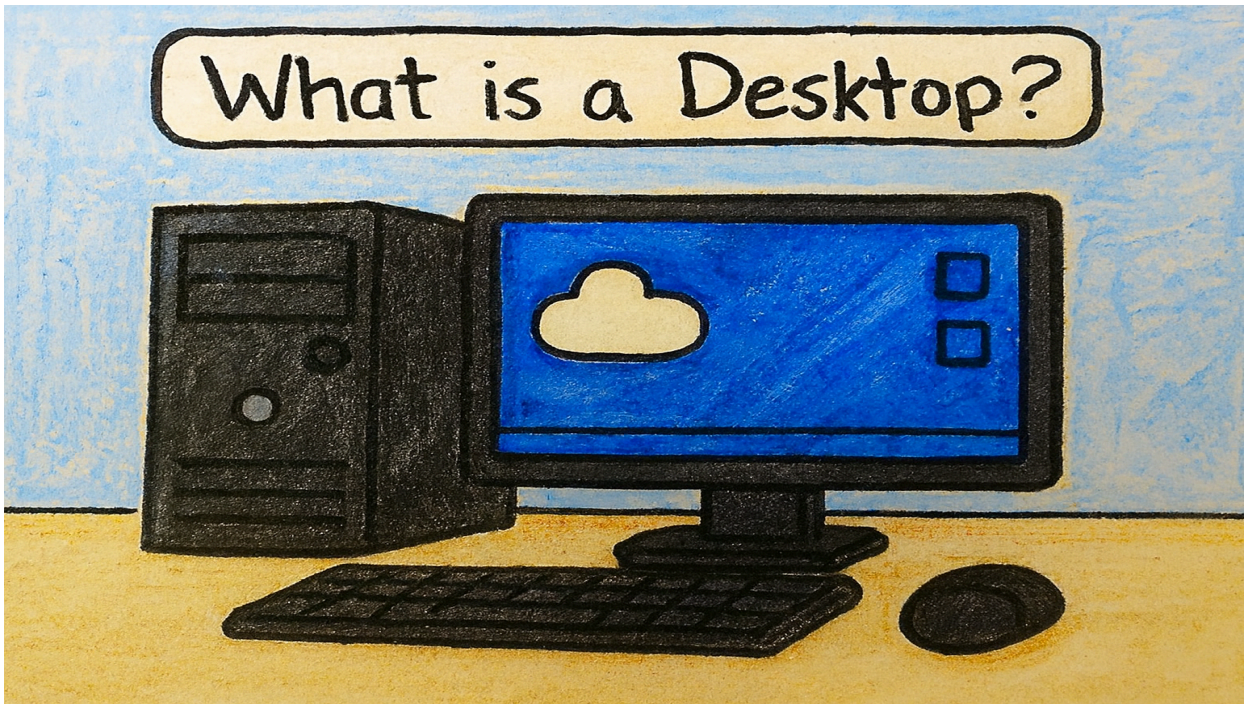

Class: 12th

Subject: Computer

Chapter 13: FUNCTIONS IN C

📍 Important MCQs:

1. Modular programming is inspired by:

(a) Software testing

(b) Computer networking

(c) Hardware manufacturing ✅

(d) Data processing

2. Functions in C are known as:

(a) Program errors

(b) Building blocks of programs ✅

(c) Variables

(d) Operators

3. A function is best described as:

(a) A loop with conditions

(b) A self-contained block of code with a specific purpose ✅

(c) A collection of variables

(d) An input device

4. Writing the entire program logic inside the main function is called:

(a) Structured programming

(b) Modular programming

(c) Object-oriented programming

(d) Unstructured programming ✅

5. Structured programming divides the program into:

(a) Files

(b) Loops

(c) Functions or modules ✅

(d) Data types

6. In structured programming, the main function is used to:

(a) Write the whole code

(b) Store data

(c) Call other functions when needed ✅

(d) Debug errors

7. Which of the following is NOT an advantage of functions?

(a) Code reusability

(b) Easy maintenance

(c) Parallel development

(d) Increase code repetition ✅

8. The C standard library is an example of:

(a) Unstructured programming

(b) Debugging

(c) Code reusability ✅

(d) Hardware components

9. When an error occurs in a program using functions, debugging becomes easier because:

(a) The whole program is deleted

(b) Only the affected function is checked ✅

(c) Errors are ignored

(d) Program stops permanently

10. Functions are especially useful in large programs because:

(a) Only one programmer can work

(b) Programs become longer

(c) Different programmers can write different functions simultaneously ✅

(d) Code cannot be reused

11. How many types of functions are there in C?

(a) One

(b) Two ✅

(c) Three

(d) Four

12. Which of the following is a type of function in C?

(a) System functions

(b) Library functions

(c) Built-in and user-defined functions ✅

(d) Main and loop functions

13. Built-in functions are:

(a) Written by the programmer

(b) Predefined functions provided by C libraries ✅

(c) Used only in main()

(d) Optional functions

14. Built-in functions are stored in:

(a) Variables

(b) Loops

(c) Libraries ✅

(d) Arrays

15. Which of the following is an example of a built-in function?

(a) myFunction()

(b) calculate()

(c) printf() ✅

(d) userInput()

16. To use printf() and scanf(), which header file must be included?

(a) conio.h

(b) ctype.h

(c) stdlib.h

(d) stdio.h ✅

17. getch() and getche() functions belong to which library?

(a) stdio.h

(b) ctype.h

(c) conio.h ✅

(d) math.h

18. Why are built-in functions used in C?

(a) To make programs longer

(b) To avoid writing complex code again and again ✅

(c) To increase errors

(d) To replace main()

19. Functions written by programmers according to problem requirements are called:

(a) Built-in functions

(b) Library functions

(c) User-defined functions ✅

(d) Special functions

20. Which function is mandatory in every C program?

(a) printf()

(b) scanf()

(c) main() ✅

(d) getch()

21. A C function mainly consists of:

(a) Only a function body

(b) Only a return statement

(c) Function header and function body ✅

(d) Only parameters

22. The first line of a function definition is called:

(a) Function body

(b) Return statement

(c) Function header ✅

(d) Parameter list

23. Which of the following is NOT a part of a function header?

(a) Return type

(b) Function name

(c) Parameters

(d) Executable statements ✅

24. If a function does not return any value, its return type should be:

(a) int

(b) float

(c) char

(d) void ✅

25. What happens when a return statement is executed in a function?

(a) The function continues execution

(b) The program stops completely

(c) The function stops execution and returns a value ✅

(d) The loop restarts

26. Why does the compiler need a function prototype?

(a) To execute the function faster

(b) To allocate memory permanently

(c) To check correct use of a function ✅

(d) To remove syntax errors

27. A function prototype provides information about:

(a) Only the function name

(b) Only the return type

(c) Function name, parameters, and return type ✅

(d) Function body

28. The general form of a function prototype ends with:

(a) Curly braces

(b) A colon

(c) A semicolon ✅

(d) No symbol

29. A function prototype is similar to a function header but:

(a) Has a body

(b) Has no return type

(c) Ends with a semicolon ✅

(d) Has executable statements

30. Where are function prototypes usually placed in a C program?

(a) At the end of the program

(b) Inside main()

(c) At the beginning, before main() function ✅

(d) Inside loops

31. What is a function call?

(a) Defining a function

(b) Declaring a variable

(c) Invoking a function to perform a task ✅

(d) Ending a program

32. A function call statement includes:

(a) Only function name

(b) Function name and arguments

(c) Function name, arguments, and semicolon ✅

(d) Return type

33. What happens when a function call is executed?

(a) Program stops

(b) Control transfers to the called function ✅

(c) Compiler rechecks the program

(d) Variables are destroyed

34. After a function finishes execution, control:

(a) Terminates the program

(b) Goes to the operating system

(c) Returns to the calling function ✅

(d) Goes to main() only

35. The time period during which a variable exists in memory is called:

(a) Scope

(b) Size

(c) Lifetime ✅

(d) Storage class

36. The scope of a variable refers to:

(a) Its data type

(b) Its memory size

(c) The region where it can be accessed ✅

(d) Its lifetime

37. Variables declared inside a block are called:

(a) Global variables

(b) Static variables

(c) Local variables ✅

(d) External variables

38. The scope of a local variable is:

(a) The entire program

(b) Only main() function

(c) From declaration to the end of its block ✅

(d) Till program termination

39. Accessing a variable outside its scope causes:

(a) Logical error

(b) Runtime error

(c) Compiler error ✅

(d) No error

40. Local variables are destroyed when:

(a) Program ends

(b) Control moves out of their block ✅

(c) Function is called

(d) Compiler starts execution

41. Variables declared outside all functions and blocks are called:

(a) Local variables

(b) Static variables

(c) Global variables ✅

(d) Automatic variables

42. The scope of a global variable is:

(a) Only inside main()

(b) Only inside one function

(c) Throughout the file after its declaration ✅

(d) Only inside loops

43. The lifetime of a global variable is:

(a) Till the end of a block

(b) Till the function ends

(c) From program start to program termination ✅

(d) Only when it is used

44. Global variables are created in memory:

(a) When a function is called

(b) After execution of main()

(c) Before the execution of main() starts ✅

(d) Inside loops

45. A global variable can be accessed by:

(a) Only one function

(b) Only main()

(c) All functions in the same file after declaration ✅

(d) Only loops

46. A function that returns no value and accepts no arguments has return type:

(a) int

(b) float

(c) char

(d) void ✅

47. In a function without arguments, the parameter list may contain:

(a) int

(b) float

(c) void or be empty ✅

(d) char

48. After a function without arguments completes its task, control:

(a) Stops the program

(b) Returns to the calling function ✅

(c) Goes to the operating system

(d) Repeats the function

49. Functions that return a value must specify the data type of:

(a) Parameters only

(b) Function name

(c) Returned value in the return type ✅

(d) Local variables

50. Which of the following is an example of a built-in function that returns a value and accepts arguments?

(a) printf()

(b) getch()

(c) sqrt() ✅

(d) voidFunction()

📍 Important Short Questions:

1. What is modular programming?

Answer:

👉 Modular programming is a programming approach in which a program is divided into smaller, independent modules or functions, each designed to perform a specific task.

Example: A washing machine is made of separate components like motor, drum, and timer, each performing a specific function, similar to modules in a program.

2. Why are functions called the building blocks of C programs?

Answer:

👉 Functions are called the building blocks of C programs because they encapsulate code for a specific operation, allowing it to be reused multiple times in the program.

Example: Using a printf() function multiple times to display messages in different parts of a program.

3. What is unstructured programming?

Answer:

👉 Unstructured programming is a style where the entire program logic is written inside a single main() function without breaking it into smaller functions.

Example: A program that calculates the sum, average, and maximum of numbers entirely inside main().

4. How is structured programming different from unstructured programming?

Answer:

👉 In structured programming, the program is divided into multiple functions or modules, whereas in unstructured programming, all logic is written in one main() function.

Example: In a structured program, one function calculates sum, another calculates average, and the main function calls them as needed.

5. State two advantages of using functions in C.

Answer:

👉 1. Functions make programs easier to understand and maintain.

👉 2. Functions increase code reusability, allowing the same code to be used in multiple places or programs.

Example: The sqrt() function from C library can be used in any program to calculate square roots without rewriting its logic.

6. What are the two types of functions in C?

Answer:

👉 There are two types of functions in C:

1. Built-in functions
2. User-defined functions

Example: printf() is a built-in function, while a function AddNumbers() written by the programmer is user-defined.

7. What are built-in functions?

Answer:

👉 Built-in functions are predefined functions provided by C libraries to perform common tasks without writing the code from scratch.

Example: scanf(), getch(), toupper().

8. Why do we use built-in functions in C?

Answer:

👉 Built-in functions save time and effort by providing ready-to-use code for complex tasks, so programmers don't have to reinvent the wheel.

Example: Using sqrt() to find square root instead of writing the algorithm manually.

9. What are user-defined functions?

Answer:

👉 User-defined functions are functions written by the programmer to perform specific tasks that are not covered by built-in functions.

Example: A function CalculateArea() to find the area of a triangle.

10. What is the general structure of a function in C?

Answer:

👉 **A function in C consists of:**

1. Function header (return type, function name, parameters)
2. Function body (curly braces containing executable statements)
3. Return statement (optional if return type is void)

Example:

```
int Add(int a, int b) {
  return a + b;
}
```

11. What is a function header?

Answer:

👉 The function header is the first line of a function that specifies its return type, name, and parameters.

Example:

int Multiply(int x, int y)

12. What are the three parts of a function header?

Answer:

👉 A function header consists of:

1. Return type
2. Function name
3. Parameter list (enclosed in parentheses)

Example: float Area(int base, int height)

13. What is the role of the function body?

Answer:

👉 The function body contains variable declarations and the program logic enclosed in curly braces. It performs the task assigned to the function.

Example:

```
{
  int sum;
```

```
  sum = a + b;
  return sum;
}
```

14. What is the purpose of the return statement in a function?

Answer:

👉 The return statement sends a value back to the calling function and stops execution of the function.

Example:

```
return a + b;
```

15. When is a return statement not required in a function?

Answer:

👉 If the function's return type is void (i.e., it does not return a value), then a return statement is optional.

Example:

```
void PrintMessage() {
  printf("Hello!");
```

}

16. What is a function prototype in C?

Answer:

👉 A function prototype is a statement that provides the compiler with information about a function, including its name, return type, and parameters, so that the function can be used correctly before its definition.

Example:

int Add(int a, int b);

17. Where should a function prototype be placed in a C program?

Answer:

👉 A function prototype should be placed at the beginning of the source file, usually just before the main() function or before the function call.

Example:

#include <stdio.h>

int Add(int a, int b); // prototype

void main() { ... }

18. What is a function call in C?

Answer:

👉 A function call is a statement used to invoke a function to perform its specific task. Control is transferred to the called function, and after execution, control returns to the calling function.

Example:

sum = Add(5, 10); // calling Add function

19. What are local variables in C?

Answer:

👉 Local variables are variables declared inside a block (like a function or loop). They are accessible only within that block and are destroyed when the block ends.

Example:

void main() {

  int nCount = 0; // local variable

```
}
```

20. What is the scope and lifetime of a local variable?

Answer:

👉 **Scope**: The region in the program where the variable is accessible (from its declaration to the end of the block).

👉 **Lifetime**: The duration in memory from when the block is entered until it is exited.

Example:

```
if (nCount == 0) {
   int chk = 10; // scope limited to this block
}
// chk cannot be used here, it is destroyed
```

21. What are global variables in C?

Answer:

👉 Global variables are variables declared outside all functions and blocks. They can be accessed by any function in the file after their declaration.

Example:

```
int nCount = 0; // global variable

void Counter() { nCount++; }

void main() { Counter(); printf("%d", nCount); }
```

22. What is the scope and lifetime of a global variable?

Answer:

👉 **Scope**: Global variables are visible throughout the file in all functions after their declaration.

👉 **Lifetime**: They exist in memory from the start of program execution to its termination.

Example: The variable nCount can be used in both main() and Counter() functions.

23. What is a function without arguments?

Answer:

👉 A function without arguments is a function that does not take any input parameters. Its return type is usually void.

Example:

```
void PrintAsterisks() { printf("***"); }

void main() { PrintAsterisks(); }
```

24. How does a function without arguments work in a program?

Answer:

👉 When called, control transfers to the function, memory is allocated for its variables, statements are executed, and control returns to the calling function. Memory is then deallocated.

Example: The function PrintAsterisks() prints asterisks and then returns control to main().

25. What is a function that returns a value and accepts arguments?

Answer:

👉 It is a function that takes input parameters, performs calculations, and returns a value to the calling function.

Example:

```
int Add(int a, int b) { return a + b; }
```

void main() { int sum = Add(5, 10); }

📍 Exercise 13c

1. Fill in the blanks:

1. A _____ is a self-contained piece of code.

Answer: Function

Explain:

👉 A function is a block of code designed to perform a specific task. It allows code to be reused and organized into manageable parts.

Example:

void PrintMessage() { printf("Hello!"); }

2. Pre-defined functions are packaged in _____.

Answer: Libraries

Explain:

👉 Libraries are collections of pre-written functions provided by C to perform common tasks. Using them avoids rewriting code.

Example: stdio.h contains printf() and scanf() functions.

3. A _____ provides basic information about the function to the compiler.

Answer: Function prototype

Explain:

👉 A function prototype tells the compiler the function's name, return type, and parameters before it is called, allowing error checking.

Example:

int Add(int a, int b); // prototype

4. The duration for which a variable exists in memory is called its _____.

Answer: Lifetime

Explain:

👉 Lifetime is the period during which a variable occupies memory. For local variables, it ends when the function/block finishes.

Example: Local variable in a function is created on entry and destroyed on exit.

5. _____ of a variable refers to the region of the program where it can be referenced.

Answer: Scope

Explain:

👉 Scope defines where a variable can be accessed in the program. Local variables have block scope, and global variables have file scope.

Example:

```
int n = 5; // global scope

void main() { int x = 10; } // x has local scope
```

6. _____ variables are declared outside all blocks.

Answer: Global

Explain:

👉 Global variables are accessible in all functions after their declaration and exist throughout program execution.

Example:

int count = 0; // global variable

7. A function cannot return more than _____ value(s) through return statement.

Answer: One

Explain:

👉 A C function can only return a single value through the return statement. Multiple values must be returned using pointers or structures.

Example:

int Add(int a, int b) { return a + b; }

8. The parameters specified in the function header are called _____ parameters.

Answer: Formal

Explain:

👉 Formal parameters are variables defined in the function header that accept values passed during a function call.

Example:

int Add(int a, int b) { return a + b; } // a and b are formal parameters

9. The parameters passed to a function in the function call are called _____ parameters.

Answer: Actual

Explain:

👉 Actual parameters are the real values provided to a function when it is called.

Example:

int sum = Add(5, 10); // 5 and 10 are actual parameters

10. Functions help to achieve _____ programming.

Answer: Modular / Structured

Explain:

👉 Functions break a program into small, manageable, reusable modules, making it easier to understand, maintain, and debug.

Example:

A program to calculate area and perimeter can have two separate functions: Area() and Perimeter().

2. Choose the correct option:

1. Function prototypes for built-in functions are specified in:

a) source files

b) header files ✅

c) object files

d) image files

2. Global variables are created in:

a) RAM ✅

b) ROM

c) hard disk

d) cache

3. Which of the following is true about a function call?

a) Stops the execution of the program

b) Transfers control to the called function ✅

c) Transfers control to the main function

d) Resumes the execution of the program

4. Which of the following looks for the prototypes of functions used in a program?

a) linker

b) loader

c) compiler ✅

d) parser

5. Memory is allocated to a local variable at the time of its:

a) declaration

c) definition

b) destruction

d) first reference ✅

6. The name of actual and formal parameters:

a) may or may not be same ✅

b) must be same

c) must be different

d) must be in lowercase

7. Formal arguments are also called:

a) actual arguments

b) dummy arguments ✅

c) original arguments

d) referenced arguments

8. printf() is a:

a) built-in function ✅

b) user-defined function

c) local function

d) keyword

9. A built-in function:

a) can not be redefined ✅

b) can be redefined

c) can not return a value

d) should be redefined

10. In a C program, two functions can have:

a) same name

b) same parameters

c) same name and same parameters

d) same name but different parameters ✅

3. Write T for true and F for false statement.

(i) In C, arguments can be passed to a function only by value.

Answer: F ❌ (Correct: Arguments can be passed by value or by reference using pointers)

(ii) C was derived from earlier programming language named B.

Answer: T ✅

(iii) The B was developed by Ken Thompson in 1980.

Answer: F ❌ (Correct: B was developed by Ken Thompson in 1969–70)

(iv) A function can be called anywhere in the program.

Answer: T ✅

(v) In C, every function must return a value.

Answer: F ❌ (Correct: Functions with return type void do not return any value)

(vi) A user-defined function can not be called in another user-defined function.

Answer: F ❌ (Correct: A user-defined function can be called inside another user-defined function)

(vii) A function can be called only once in a program.

Answer: F ❌ (Correct: A function can be called multiple times from any part of the program)

(viii) Scope of a local variable is the block in which it is defined.

Answer: T ✅

(ix) Global variables exist in memory till the execution of the program.

Answer: T ✅

(x) An unstructured program is more difficult to debug than a structured program.

Answer: T ✅

(xi) Function body is an optional part of the function.

Answer: F ❌ (Correct: A function body is mandatory to perform operations)

🌟 Q.4: What is a function? How many types of functions are used in C? Discuss the difference between them.

❖ **Answer:**

Definition of a Function:

- A function is a self-contained block of code that performs a specific task.
- It allows a programmer to write code once and reuse it wherever needed.
- Functions make programs modular, reduce repetition, and make them easier to debug and maintain.

Types of Functions in C:

1. Built-in Functions (Predefined Functions):

- Provided by the C language in libraries or header files.
- Perform common tasks so that programmers don't need to write code from scratch.
- Require inclusion of corresponding header files (e.g., #include <stdio.h> for printf() and scanf()).
- **Examples**: printf(), scanf(), sqrt(), toupper(), getch().

2. User-defined Functions:

- Written by the programmer to perform program-specific tasks.
- Used when built-in functions are not sufficient.
- Divide the program into smaller, manageable modules.
- **Example:**

```
int Add(int a, int b) {
    return a + b;
}
```

Differences Between Built-in and User-defined Functions:

- **Definition:** Built-in functions are predefined; user-defined functions are written by the programmer.
- **Usage:** Built-in functions can be used in any program; user-defined functions are specific to a program.
- **Header Files:** Built-in functions require header files; user-defined functions do not.
- **Purpose:** Built-in functions perform common tasks; user-defined functions provide flexibility for custom tasks.
- **Examples:** Built-in: printf(), scanf(); User-defined: Add(), Print_Asterisks().

✦ **Summary:**

- Functions are the building blocks of a C program.
- Built-in functions save time and effort.
- User-defined functions allow customization and modularity.
- Both together make programming structured, readable, and easy to maintain.

🌟 **Q.5: Differentiate the following:**

(i) Function Definition and Function Declaration

Function Definition:

- It contains the actual body of the function with executable statements.
- It tells the compiler what the function does.

Example:

```
int Add(int a, int b) {
    return a + b;
}
```

Function Declaration (Prototype):

- It provides the compiler with the function's name, return type, and parameters without the body.
- It tells the compiler that the function will be defined later.

Example:

```
int Add(int a, int b);
```

(ii) Global and Local Variables

Global Variables:

- Declared outside all functions.
- Accessible throughout the file from their point of declaration.

- **Lifetime:** Exists until the program terminates.

Local Variables:

- Declared inside a block or function.
- Accessible only within the block where they are declared.
- **Lifetime:** Exists only while the block executes.

(iii) Scope and Lifetime of a Variable

Scope:

- Refers to the region of the program where a variable can be accessed.
- **Example:** A local variable's scope is limited to its block.

Lifetime:

- Refers to the duration for which a variable exists in memory.
- **Example:** A global variable exists until the program ends; a local variable exists only during block execution.

(iv) Function Prototype and Function Header

Function Prototype:

- Declaration of a function providing its name, return type, and parameters, ending with a semicolon.

- Placed before the main function or function calls.
- **Example:** int Add(int a, int b);

Function Header:

- The first line of the function definition includes return type, name, and parameters.
- Followed by the function body in curly braces.
- **Example**: int Add(int a, int b) { ... }

(v) Formal Parameters and Actual Parameters of a Function

Formal Parameters:

- Variables declared in the function header to receive values.
- **Example:** In int Add(int a, int b), a and b are formal parameters.

Actual Parameters:

- Values or variables passed to the function when it is called.
- **Example:** In Add(10, 20), 10 and 20 are actual parameters.

🌟 Q.6: How is a function call made in a C program? Discuss briefly.

❖ Answer:

Definition:

- A function call is a statement used to invoke a function to perform a specific task in a C program.
- When a function is called, the control of the program is transferred to that function.

How a Function Call Works:

1. **The function name, required arguments (if any), and a semicolon (;) are used to call a function.**
    - **Example:** Add(10, 20);
2. **When the function call is executed:**
    - The program control moves to the called function.
    - Memory is allocated to variables declared inside the function (local variables).
    - Statements inside the function body are executed.
3. **After the function finishes execution:**
    - Control returns to the statement immediately following the function call in the calling function.

- Memory allocated to local variables is released back to the system.

Key Points:

- Functions can be called from main() or even from other user-defined functions.
- A function can be called multiple times from different parts of the program.
- Arguments passed during the function call are called actual parameters.
- If the function returns a value, it can be stored or used in expressions.

Example:

```
#include <stdio.h>

int Add(int a, int b);  // Function prototype

void main() {

   int sum;

   sum = Add(10, 20);  // Function call

   printf("Sum = %d", sum);

}
```

```
int Add(int a, int b) {  // Function definition
    return a + b;
}
```

✦ **Summary:**

- Function call transfers control to the called function.
- **It allows modular,** reusable, and structured programming.
- Memory management is automatic for local variables in the called function.

🌟 **Q.7: Answer the following:**

(i) What is the purpose of a function argument?

- Function arguments are used to pass data from the calling function to the called function.
- They allow the function to perform operations on different values without changing its code.
- **Example:** In Add(10, 20), 10 and 20 are arguments that the function Add uses to calculate the sum.

(ii) How many (maximum) values can a function return using return statement?

- A function can return only one value using the return statement.
- If multiple values are needed, arrays, structures, or pointers can be used.

(iii) When is a function executed, and where should a function prototype and function definition appear in a source program?

- **Execution:** A function is executed when it is called in the program.
- **Function prototype:** Usually placed at the beginning of the source file, before the main() function or any function calls.
- **Function definition:** Can appear after the main() function or any other calling function, as long as the prototype is declared before the call.

(iv) Write three advantages of functions:

- Functions make programs easier to understand and maintain by dividing the code into smaller modules.

- Functions increase code reusability, as the same function can be used multiple times in the same program or in different programs.
- Functions allow parallel development, as multiple programmers can work on different functions simultaneously.

🌟 **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 <stdio.h>

// Function prototypes

void Draw_Horizontal(void);

void Draw_Vertical(void);

void main() {

   // Draw top horizontal line

   Draw_Horizontal();
```

```
    // Draw vertical lines
    Draw_Vertical();
    // Draw bottom horizontal line
    Draw_Horizontal();
}
// Function to draw horizontal lines
void Draw_Horizontal(void) {
    for(int i = 1; i <= 10; i++) {
        printf("*");
    }
    printf("\n");
}
// Function to draw vertical lines
void Draw_Vertical(void) {
    for(int i = 1; i <= 4; i++) {
        printf("*");
```

```
        for(int j = 1; j <= 8; j++) {
            printf(" "); // Space between vertical lines
        }
        printf("*\n");
    }
}
```

✦ Explanation:

1. Draw_Horizontal function:

- Prints a row of * symbols to form the top and bottom of the rectangle.
- Uses a for loop to print 10 * symbols in a row.

2. Draw_Vertical function:

- Prints vertical sides of the rectangle.
- Each line starts and ends with *, with spaces in between to form the width.
- **Uses nested for loops:** the outer loop controls the number of lines, the inner loop controls spaces.

3. main() function:

- Calls Draw_Horizontal to print the top line.
- Calls Draw_Vertical to print the sides.
- Calls Draw_Horizontal again to print the bottom line.

Output:

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

Key Points for Exam:

- Functions reduce repetitive code.
- Draw_Horizontal and Draw_Vertical are user-defined functions without arguments and without return values.
- The program demonstrates modular programming and function call mechanism in C.

🌟 Q.9: Write a program that prompts the user for the Cartesian coordinates of two points (x1, y1) and (x2, y2) and displays the distance between them. Write a function

Distance() with four input parameters that computes and returns the distance.

❖ **Answer:**

```
#include <stdio.h>
#include <math.h>  // Required for sqrt() and pow() functions
// Function prototype
double Distance(int x1, int y1, int x2, int y2);
void main() {
    int x1, y1, x2, y2;
    double dist;
    // Input coordinates from user
    printf("Enter coordinates of first point (x1 y1): ");
    scanf("%d %d", &x1, &y1);
    printf("Enter coordinates of second point (x2 y2): ");
    scanf("%d %d", &x2, &y2);
    // Function call to compute distance
    dist = Distance(x1, y1, x2, y2);
```

```
    // Display the distance
    printf("Distance between the points is: %.2lf\n", dist);
}
// Function definition to calculate distance
double Distance(int x1, int y1, int x2, int y2) {
    double result;
    result = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
    return result;
}
```

✦ Explanation:

1. Distance function:

- Takes four input parameters: x1, y1, x2, y2.
- Computes distance using the formula:

$$\text{distance} = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

2. main() function:

- Prompts user to input coordinates of two points.
- Calls Distance() function to calculate the distance.
- Stores the returned value in dist and displays it using printf().

Key points for exam:

- Demonstrates user-defined function with arguments and return value.
- Uses math library functions: sqrt() for square root and pow() for power.
- Enhances modularity, making the program easy to read and maintain.

Sample Output:

Enter coordinates of first point (x1 y1): 3 4

Enter coordinates of second point (x2 y2): 7 8

Distance between the points is: 5.66

🌟 Q.10: Write a program that reverses a number using a function Reverse.

❖ **Answer:**

#include <stdio.h>

```
// Function prototype
int Reverse(int num);
void main() {
    int num, rev;
    // Input number from user
    printf("Enter a number: ");
    scanf("%d", &num);
    // Function call to reverse the number
    rev = Reverse(num);
    // Display reversed number
    printf("Reversed number is: %d\n", rev);
}
// Function definition to reverse a number
int Reverse(int num) {
    int rev = 0;
    while(num != 0) {
```

```
        rev = rev * 10 + (num % 10); // extract last digit and add to
rev
        num = num / 10;              // remove last digit from num
    }
    return rev; // return reversed number
}
```

✦ Explanation:

1. Reverse function:

- Accepts a number as input.
- Initializes rev to 0.
- Extracts the last digit using num % 10 and adds it to rev.
- Removes the last digit from num using num / 10.
- Repeats until num becomes 0.
- Returns the reversed number.

2. main() function:

- Takes input from the user.
- Calls the Reverse function.
- Prints the reversed number.

Sample Output:

Enter a number: 2765

Reversed number is: 5672

🌟 **Q.11: Write a program to print the asterisks pattern using a function Draw_Asterisks.**

❖ **Answer:**

```
#include <stdio.h>

// Function prototype

void Draw_Asterisks(void);

void main() {

    // Function call to print asterisks

    Draw_Asterisks();

}

// Function definition to print the pattern

void Draw_Asterisks(void) {

    int outer, inner;
```

```
    // Loop for each row
    for(outer = 9; outer >= 1; outer -= 2) {
        // Loop to print asterisks in a row
        for(inner = 1; inner <= outer; inner++) {
            printf("*");
        }
        printf("\n"); // Move to next line after each row
    }
}
```

✦ **Explanation:**

1. Function Draw_Asterisks():

- No parameters and no return value (void).
- **Uses two loops:**

➢ **Outer loop:** controls the number of rows and decreases by 2 each time (9, 7, 5, 3, 1).

➢ **Inner loop:** prints the asterisks in each row.

- Prints a newline after each row.

2. main() function:

- Calls the Draw_Asterisks function to print the pattern.

Output:

****

****

*****

***

*

🌟 Q.12: Write a function Is_Prime to check whether a number is prime.

❖ **Answer**

```
#include <stdio.h>

// Function prototype

int Is_Prime(int num);

void main() {

   int n, result;
```

```
    // Input number from user
    printf("Enter a number: ");
    scanf("%d", &n);
    // Function call to check prime
    result = Is_Prime(n);
    // Display result
    if(result == 1)
        printf("%d is a prime number.\n", n);
    else
        printf("%d is not a prime number.\n", n);
}
// Function definition to check prime
int Is_Prime(int num) {
    int i;
    if(num <= 1)
        return 0; // numbers <= 1 are not prime
```

```
    for(i = 2; i <= num / 2; i++) { // check divisibility up to num/2
        if(num % i == 0)
            return 0; // divisible, not prime
    }
    return 1; // prime
}
```

✦ Explanation:

1. Function Is_Prime(int num):

- Takes an integer num as input.
- Returns 1 if num is prime, 0 otherwise.
- Checks divisibility from 2 to num/2.

2. main() function:

- Reads a number from the user.
- Calls Is_Prime to check if it is prime.
- Prints whether the number is prime or not.

Sample Output:

Enter a number: 17

17 is a prime number.

Enter a number: 20

20 is not a prime number.

🌟 Q.13: using functions Add(), Subtract(), Multiply(), and Divide(). The program handles the user's choice and performs the operations.

❖ **Answer:**

C Program for a simple calculator using functions

```
#include <stdio.h>

#include <stdlib.h> // for exit()

// Function prototypes

int Add(int a, int b);

int Subtract(int a, int b);

int Multiply(int a, int b);

float Divide(int a, int b);

void main() {

   int num1, num2, choice;
```

```
float result;
// Input two numbers
printf("Enter two integers: ");
scanf("%d %d", &num1, &num2);
while(1) { // Infinite loop until user chooses to exit
    // Display menu
    printf("\nChoose an operation:\n");
    printf("1. Add\n");
    printf("2. Subtract\n");
    printf("3. Multiply\n");
    printf("4. Divide\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch(choice) {
        case 1:
```

```
        result = Add(num1, num2);
        printf("Sum = %d\n", (int)result);
        break;
    case 2:
        result = Subtract(num1, num2);
        printf("Difference = %d\n", (int)result);
        break;
    case 3:
        result = Multiply(num1, num2);
        printf("Product = %d\n", (int)result);
        break;
    case 4:
        result = Divide(num1, num2);
        if(result != -1)
            printf("Division = %.2f\n", result);
        break;
```

```
            case 5:
                printf("Exiting program.\n");
                exit(0);
            default:
                printf("Invalid choice! Try again.\n");
        }
    }
}
// Function definitions
int Add(int a, int b) {
    return a + b;
}
int Subtract(int a, int b) {
    return a - b;
}
int Multiply(int a, int b) {
```

```
    return a * b;
}
float Divide(int a, int b) {
    if(a == 0 && b == 0) {
        printf("Cannot divide zero by zero.\n");
        return -1;
    }
    if(b == 0 || a == 0) {
        if(a == 0 && b != 0)
            return 0; // 0 divided by any number = 0
        else {
            printf("Division by zero error!\n");
            return -1;
        }
    }
    if(a > b)
```

```
        return (float)a / b;
    else
        return (float)b / a;
}
```

✦ Explanation:

1. Function Prototypes:

- Add(), Subtract(), Multiply(), Divide() declared at the top.

2. main() function:

- Inputs two integers from the user.
- Displays a menu repeatedly using while(1) loop.
- Switch statement executes the function based on user choice.
- Choice 5 exits the program using exit(0).

3. Functions:

- **Add**: Returns sum of two numbers.
- **Subtract**: Returns difference of two numbers.
- **Multiply**: Returns product of two numbers.

- **Divide**: Divides the larger number by the smaller one. Handles division by zero.

4. Safety:

- Checks for zero in division.
- Ensures the program does not crash if the denominator is zero.

Sample Output:

Enter two integers: 12 4

Choose an operation:

1. Add

2. Subtract

3. Multiply

4. Divide

5. Exit

Enter your choice: 1

Sum = 16

Enter your choice: 4

Division = 3.00

Enter your choice: 5

Exiting program.

🌟 Q.14: using a function Factorial() that returns the factorial of a number.

C Program to compute factorial using a function

```
#include <stdio.h>

// Function prototype

long Factorial(int n);

void main() {

    int num;

    long result;

    // Input a number

    printf("Enter a number: ");

    scanf("%d", &num);
```

```
    // Call the function to compute factorial
    result = Factorial(num);
    // Display the result
    printf("Factorial of %d is %ld\n", num, result);
}
// Function to calculate factorial
long Factorial(int n) {
    long fact = 1;
    int i;
    for(i = 1; i <= n; i++) {
        fact = fact * i;
    }
    return fact;
}
```

✦ Explanation:

1. Function Prototype:

- long Factorial(int n); informs the compiler about the function.

2. main() Function:

- Inputs a number from the user.
- Calls Factorial() and stores the returned value in result.
- Prints the factorial using printf().

3. Factorial() Function:

- Accepts one integer parameter n.
- Uses a for loop to multiply all numbers from 1 to n.
- Returns the computed factorial as long to handle larger numbers.

Sample Output:

Enter a number: 5

Factorial of 5 is 120

🌟 Q.15: using a function GCD() to compute the greatest common divisor:

C Program to find GCD using a function

#include <stdio.h>

```
// Function prototype
int GCD(int a, int b);
void main() {
    int num1, num2, result;
    // Input two numbers
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);
    // Call the function to compute GCD
    result = GCD(num1, num2);
    // Display the result
    printf("GCD of %d and %d is %d\n", num1, num2, result);
}
// Function to calculate GCD
int GCD(int a, int b) {
    int i, gcd = 1;
    // Find the minimum of a and b
```

```
    int min = (a < b) ? a : b;

    // Check for the greatest common divisor

    for(i = 1; i <= min; i++) {

        if(a % i == 0 && b % i == 0) {

            gcd = i;

        }

    }

    return gcd;

}
```

✦ Explanation:

1. Function Prototype:

- int GCD(int a, int b); informs the compiler about the function.

2. main() Function:

- Inputs two numbers from the user.
- Calls GCD() and stores the returned value in result.
- Prints the GCD using printf().

3. GCD() Function:

- Accepts two integer parameters a and b.
- Finds the smaller of the two numbers.
- Check all numbers from 1 to the smaller number to find the greatest number that divides both.
- Returns the greatest common divisor.

Sample Output:

Enter two numbers: 24 36

GCD of 24 and 36 is 12

Note:

This chapter is designed to provide a solid foundation of knowledge, with the goal of deepening understanding and encouraging further exploration of the subject. The content has been carefully selected to support effective learning and inspire students to engage with the topic more deeply.

Author: Muhammad Asghar

Purpose: To contribute to education by offering insightful, valuable content that enhances learning and understanding.