UNIT –I : FUNDAMENTALS OF C-PROGRAMMING

Short Answer (2 mark) Questions

1. **Write difference between algorithm and flowchart.**

Ans :-

Difference between algorithm and flowchart:
An algorithm is a step-by-step procedure or formula for solving a problem, often expressed in plain English or a structured format that can be easily understood by humans and translated into programming code. It is more abstract and focuses on the logic of the solution.

A flowchart, on the other hand, is a visual representation of the algorithm. It uses different shapes and arrows to illustrate the sequence of steps and decision points in the algorithm. Flowcharts provide a graphical way to understand the logic and flow of the algorithm.

2. **Explain the importance of C language.**

Ans :-

Importance of C language:
C language is important for several reasons:

Efficiency: C is close to the hardware and allows for low-level manipulation, making it highly efficient.

Portability: Programs written in C can be easily ported to different platforms with minimal changes.

Widely Used: C is the foundation for many other programming languages and is used extensively in systems programming, embedded systems, and developing operating systems.

Flexibility: C provides low-level access to memory, which allows for better control over system resources.

Large Community: C has a large community of developers and a wealth of libraries and resources available.

3. **What is format specifier?**

Ans :-

Format specifier:
A format specifier is a special character used in printf() and scanf() functions in C and other programming languages to specify the type and format of data to be input or output.

Examples of format specifiers include %d for integers, %f for floating-point numbers, %c for characters, %s for strings, and %p for pointers.

4. **Define keyword, constant and variable.**

Ans :-

Keyword, constant, and variable:

- Keyword: Keywords are reserved words in a programming language that have predefined meanings and cannot be used for other purposes such as variable names or function names. Examples in C include int, float, if, while, etc.

- Constant: Constants are values that do not change during program execution. They are fixed values assigned to variables or used directly in expressions. Constants can be numeric constants, character constants, or string literals.

- Variable: Variables are named memory locations used to store data that may change during program execution. They have a data type and a unique identifier (name) within the scope of the program.

5. **Write a short note on type casting.**

Ans :-

Type casting:
Type casting is the process of converting a variable from one data type to another. It allows data of one type to be treated as another type for specific operations.

For example, converting an integer to a float or vice versa, or converting a character to an integer value.

Type casting can be implicit (automatically done by the compiler) or explicit (performed by the programmer using casting operators).

6. **Explain sizeof() with example?**

Ans :-

sizeof() is a built-in operator in C that returns the size of a variable or data type in bytes.

Example:

```
#include <stdio.h>
int main() {
   int num;
   printf("Size of int: %lu bytes\n", sizeof(num));
   return 0;
}
```

This program will output the size of the int data type in bytes.

7. **Why do we use header files?**

Ans :-

- Header files contain declarations of functions, constants, and data types used in a program.
- They allow for modular programming by separating interface (declaration) from implementation.
- Header files help in code organization, readability, and reusability.
- They provide a way to share code across multiple source files.
- Commonly used header files in C include <stdio.h>, <stdlib.h>, <math.h>, etc.

8. **Define relational operator?**

Ans :-

The relational operators in C are as follows:

Equal to (==): Checks if two operands are equal.
Not equal to (!=): Checks if two operands are not equal.
Greater than (>): Checks if the left operand is greater than the right operand.
Less than (<): Checks if the left operand is less than the right operand.
Greater than or equal to (>=): Checks if the left operand is greater than or equal to the right operand.
Less than or equal to (<=): Checks if the left operand is less than or equal to the right operand.

9. **What is the purpose of adding comments in a program?**

Ans :-

Purpose of adding comments in a program:
Comments are used to document code and improve its readability.

They provide explanations, notes, or reminders to programmers or other stakeholders about the purpose, logic, or functionality of the code.

Comments are ignored by the compiler and do not affect the execution of the program.

They make it easier to understand and maintain code, especially for complex or large programs.

10. **Differentiate between computer software and hardware?**

Ans :-

Difference between computer software and hardware:
Computer hardware refers to the physical components of a computer system, including the central processing unit (CPU), memory (RAM), storage devices (hard drive, SSD), input/output devices (keyboard, mouse, monitor), and peripheral devices (printer, scanner, etc.).

Computer software refers to the programs, applications, and data that instruct the hardware to perform specific tasks. Software includes system software (like operating systems) and application software (like word processors, web browsers, games, etc.).

In essence, hardware is the tangible, physical part of the computer system, while software is the intangible, logical part that controls and utilizes the hardware to perform tasks.

Essay Answer (10 mark) Questions

1. **Describe in detail about computer hardware and software.**

Ans :-

Computer hardware and software are two fundamental components of modern computing systems, each playing distinct but interconnected roles in the functioning of a computer.

Computer Hardware:
Computer hardware refers to the physical components that make up a computer system. These components can be touched, seen, and interacted with. Here are some key hardware components:

1. Central Processing Unit (CPU): Often referred to as the brain of the computer, the CPU performs most of the processing inside the computer. It executes instructions stored in the computer's memory.

2. Memory (RAM): Random Access Memory (RAM) is temporary storage that the computer uses to store data and instructions that are actively being used by the CPU. It is volatile, meaning it loses its contents when the power is turned off.

3. Storage Devices: Storage devices hold data permanently or semi-permanently. These can include Hard Disk Drives (HDDs), Solid State Drives (SSDs), and more.

4. Motherboard: The motherboard is the main circuit board of the computer. It houses the CPU, memory, and connectors for other peripherals.

5. Input Devices: Input devices allow users to input data and commands into the computer. Examples include keyboards, mice, touchscreens, and microphones.

6. Output Devices: Output devices display information processed by the computer. Common examples include monitors, printers, and speakers.

7. Peripheral Devices: These are additional devices that can be connected to the computer to extend its functionality, such as printers, scanners, webcams, etc.

Computer Software:
Computer software refers to the set of programs, instructions, and data that instruct the hardware how to perform specific tasks. It includes both system software and application software.

1. System Software: This software manages the hardware and provides a platform for running application software. Operating systems like Windows, macOS, Linux, and Unix are examples of system software.

2. Application Software: Application software is designed to perform specific tasks or functions for users. Examples include word processors, web browsers, games, spreadsheets, and photo editing software.

3. Utilities: Utilities are software programs that perform specific tasks related to managing computer resources or performing maintenance functions. Examples include antivirus software, disk cleanup tools, and backup software.

4. Device Drivers: These are software programs that allow the operating system to communicate with hardware devices such as printers, scanners, and graphics cards.

5. Programming Languages and Development Tools: These are tools used by developers to create software applications. Examples include compilers, interpreters, Integrated Development Environments (IDEs), and libraries.

Relationship between Hardware and Software:

Hardware and software work together to execute instructions, process data, and perform tasks. The software provides instructions to the hardware, which in turn executes those instructions and produces the desired output. Without software, hardware is essentially useless, and without hardware, software has no platform to run on. They are interdependent components of a computer system, each essential for its overall functionality.

2. **Write detailed notes on C data types.**

Ans :-

C data types are classifications used to specify the type of data that a variable can hold in a C program. Data types determine the size and type of values that can be stored in variables, as well as the operations that can be performed on those values. C provides a variety of data types to accommodate different types of data and optimize memory usage. Here are the main C data types:

Basic Data Types:

1.

int: The `int` data type is used to store integer values. It typically occupies 4 bytes of memory on most modern systems. The range of values that can be stored depends on the size of an integer on the specific platform, but it usually ranges from -2,147,483,648 to 2,147,483,647.

2.

char: The `char` data type is used to store a single character, such as 'a', 'b', 'c', etc. It typically occupies 1 byte of memory and can represent ASCII characters ranging from 0 to 255 or characters based on the system's character set.

3.

float: The `float` data type is used to store floating-point numbers, which represent real numbers with fractional parts. It typically occupies 4 bytes of memory and provides approximately 6 decimal digits of precision.

4.

double: The `double` data type is used to store double-precision floating-point numbers. It typically occupies 8 bytes of memory and provides approximately 15 decimal digits of precision.

Modifier Data Types:

1. short: The `short` modifier is used to reduce the range of values that an integer can hold. It typically occupies 2 bytes of memory.

2. long: The `long` modifier is used to extend the range of values that an integer can hold. It typically occupies 4 bytes of memory.

3. long long: The `long long` modifier is used to further extend the range of values that an integer can hold. It typically occupies 8 bytes of memory.

Other Data Types:

1. _Bool: The `_Bool` data type is used to store boolean values, which can be either `true` or `false`. It typically occupies 1 byte of memory.

2. void: The `void` data type is used to indicate that a function does not return any value or that a pointer does not point to any specific data type.

Derived Data Types:

1. Array: An array is a collection of elements of the same data type that are stored sequentially in memory.

2. Pointer: A pointer is a variable that stores the memory address of another variable. Pointers are used to indirectly access variables and dynamically allocate memory.

3. Structure: A structure is a composite data type that groups together variables of different data types under a single name.

4. Union: A union is a special data type that allows storing different data types in the same memory location. Only one member of a union can hold a value at any given time.

3. **Write an algorithm, flowchart and C program to find the sum of numbers from 1 to _n‘**

Ans :-

Below is the algorithm, flowchart, and C program to find the sum of numbers from 1 to _n‘

Algorithm:

1. Start

2. Initialize variables: `n`, `sum`, `i`.

3. Read the value of `n`.

4. Set `sum` to 0.

5. Use a loop to iterate from `i = 1` to `n`.

6. Inside the loop, add `i` to `sum`.

7. Increment `i`.

8. Repeat steps 6-7 until `i` reaches `n`.

9. Display the value of `sum`.

10. Stop.

Flowchart:

```
+---------------------+
|       Start         |
+---------------------+
|                     |
|    Read n           |
|   sum = 0           |
|      i = 1          |
|                     |
+----------+----------+
           |
           v
```

```
+----------+----------+
| i <= n ?            |
+----------+----------+
           |
           v
+----------+----------+
|  sum = sum + I   |
|       i = i + 1       |
+----------+----------+
           |
           v
+----------+----------+
|    End Loop         |
+----------+----------+
           |
           v
+----------+----------+
|  Display sum      |
+---------------------+
|          Stop         |
+---------------------+
```

C Program:

```c
#include <stdio.h>

int main() {

    int n, sum = 0, i;

    printf("Enter a positive integer n: ");

    scanf("%d", &n);

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

        sum += i;

    }

    printf("Sum of numbers from 1 to %d is: %d\n", n, sum);

    return 0;

}
```

This program first reads a positive integer \(n \) from the user, then it calculates the sum of numbers from 1 to \(n \) using a `for` loop, and finally, it displays the sum.

Q - 4. Discuss about the following operators in C language with example.

a. **Bitwise operators**
b. **Increment and decrement operators**
c. **Logical operators**

Ans :-

a. Bitwise Operators:

Bitwise operators in C language operate on the individual bits of integer operands. They are used to perform bit-level manipulation of data. Here are the bitwise operators in C:

1. Bitwise AND (`&`): Performs a bitwise AND operation between corresponding bits of two operands.

2. Bitwise OR (`|`): Performs a bitwise OR operation between corresponding bits of two operands.

3. Bitwise XOR (`^`): Performs a bitwise XOR (exclusive OR) operation between corresponding bits of two operands.

4. Bitwise NOT (`~`): Performs a bitwise NOT operation on each bit of the operand, effectively flipping all bits.

5. Left Shift (`<<`): Shifts the bits of the left operand to the left by a specified number of positions, filling the vacant bits with zeros.

6. Right Shift (`>>`): Shifts the bits of the left operand to the right by a specified number of positions, filling the vacant bits with zeros or the sign bit.

Example:

```
#include <stdio.h>

int main() {

   int a = 5;  // 0101 in binary

   int b = 3;  // 0011 in binary

   int result;
```

```
    // Bitwise AND
    result = a & b;  // 0101 & 0011 = 0001 (1 in decimal)
    printf("a & b = %d\n", result);

    // Bitwise OR
    result = a | b;  // 0101 | 0011 = 0111 (7 in decimal)
    printf("a | b = %d\n", result);

    // Bitwise XOR
    result = a ^ b;  // 0101 ^ 0011 = 0110 (6 in decimal)
    printf("a ^ b = %d\n", result);

    // Bitwise NOT
    result = ~a;  // ~0101 = 1010 (in decimal, it depends on the size of int)
    printf("~a = %d\n", result);

    // Left Shift
    result = a << 1;  // 0101 << 1 = 1010 (10 in decimal)
    printf("a << 1 = %d\n", result);

    // Right Shift
    result = a >> 1;  // 0101 >> 1 = 0010 (2 in decimal)
    printf("a >> 1 = %d\n", result);
    return 0;
}
```

b. Increment and Decrement Operators:

These operators are used to increase or decrease the value of a variable by 1.

1. Increment (`++`): Increases the value of a variable by 1.

2. Decrement (`--`): Decreases the value of a variable by 1.

Example:

```
#include <stdio.h>

int main() {

    int x = 5;

    printf("Initial value of x: %d\n", x);


    // Increment operator

    x++;

    printf("After incrementing, x: %d\n", x);


    // Decrement operator

    x--;

    printf("After decrementing, x: %d\n", x);


    return 0;

}
```

c. Logical Operators:

Logical operators in C language are used to perform logical operations on boolean values (0 or 1).

1. Logical AND (`&&`): Returns true if both operands are true.

2. Logical OR (`||`): Returns true if at least one of the operands is true.

3. Logical NOT (`!`): Returns true if the operand is false and vice versa.

```
#include <stdio.h>

int main() {

   int a = 5, b = 3;

   int result;


   // Logical AND

   result = (a > 4) && (b > 2);  // True && True = True

   printf("(a > 4) && (b > 2) = %d\n", result);


   // Logical OR

   result = (a > 4) || (b < 2);  // True || False = True

   printf("(a > 4) || (b < 2) = %d\n", result);


   // Logical NOT

   result = !(a == b);  // Not (False) = True

   printf("!(a == b) = %d\n", result);

   return 0;

}
```

Q - 5. Perform the following operations

a. 23>>3

b. 27<<2

c. 15&9

d. 15^9

e. 15 | 9

Ans :-

a. \(23 >> 3\):

\(23\) in binary is \(00010111\). Right shifting it by \(3\) positions gives \(00000010\), which is \(2\) in decimal.

b. \(27 << 2\):

\(27\) in binary is \(00011011\). Left shifting it by \(2\) positions gives \(01101100\), which is \(108\) in decimal.

c. \(15 & 9\):

\(15\) in binary is \(00001111\) and \(9\) in binary is \(00001001\). Performing bitwise AND operation between them results in \(00001001\), which is \(9\) in decimal.

d. \(15 ^ 9\):

\(15\) in binary is \(00001111\) and \(9\) in binary is \(00001001\). Performing bitwise XOR operation between them results in \(00000110\), which is \(6\) in decimal.

e. \(15 | 9\):

\(15\) in binary is \(00001111\) and \(9\) in binary is \(00001001\). Performing bitwise OR operation between them results in \(00001111\), which is \(15\) in decimal.

Code :-

```
#include <stdio.h>
int main() {
    int result;
    // a. 23 >> 3
    result = 23 >> 3;
    printf("23 >> 3 = %d\n", result);

    // b. 27 << 2
    result = 27 << 2;
    printf("27 << 2 = %d\n", result);

    // c. 15 & 9
    result = 15 & 9;
    printf("15 & 9 = %d\n", result);

    // d. 15 ^ 9
    result = 15 ^ 9;
    printf("15 ^ 9 = %d\n", result);

    // e. 15 | 9
    result = 15 | 9;
    printf("15 | 9 = %d\n", result);
    return 0;
}
```

Q - 6.

(a) Write the structure of C program and explain.
(b) Write a program to perform swapping of two numbers without using temporary variable.

Ans :-

a) Structure of a C Program:

A typical structure of a C program consists of several components:

1. Preprocessor Directives: These include directives such as `#include` to include header files, which are required for the program.

2. Global Declarations: These declarations include global variables and constants that are accessible throughout the program.

3. Function Prototypes: These declare the functions used in the program before they are called. It tells the compiler about the function name, return type, and parameters.

4. Main Function: The main function is the starting point of execution for every C program. It contains the statements that define the program's functionality.

Example Structure of a C Program:

```
// Preprocessor directive

#include <stdio.h>



// Function prototype

void swap(int *a, int *b);



// Main function
```

```
int main() {
    // Variable declarations
    int num1, num2;

    // Input
    printf("Enter value of num1: ");
    scanf("%d", &num1);
    printf("Enter value of num2: ");
    scanf("%d", &num2);

    // Function call to swap numbers
    swap(&num1, &num2);

    // Output
    printf("After swapping, num1 = %d, num2 = %d\n", num1, num2);

    return 0;
}

// Function definition for swapping numbers without using temporary variable
void swap(int *a, int *b) {
    *a = *a + *b;
    *b = *a - *b;
    *a = *a - *b;
}
```

Explanation:

- In the above program, we first include the necessary header file `<stdio.h>` for input and output operations.

- We declare a function prototype `void swap(int *a, int *b);` to inform the compiler about the function `swap()` before using it in the `main()` function.

- In the `main()` function, we declare two integer variables `num1` and `num2` to store the input numbers.

- We prompt the user to enter the values of `num1` and `num2` using `printf()` and `scanf()` functions.

- Then, we call the `swap()` function passing the addresses of `num1` and `num2` variables.

- Inside the `swap()` function, we perform swapping of numbers without using a temporary variable.

- Finally, we display the swapped values of `num1` and `num2` in the `main()` function using `printf()`.

b) Program to Perform Swapping of Two Numbers without Using Temporary Variable:

```
#include <stdio.h>

int main() {

    int num1, num2;



    // Input

    printf("Enter value of num1: ");

    scanf("%d", &num1);

    printf("Enter value of num2: ");

    scanf("%d", &num2);



    // Swapping without using a temporary variable

    num1 = num1 + num2;
    num2 = num1 - num2;

    num1 = num1 - num2;


    // Output

    printf("After swapping, num1 = %d, num2 = %d\n", num1, num2);


    return 0;

}
```

In this program, we directly perform arithmetic operations to swap the values of `num1` and `num2` without using a temporary variable. This approach utilizes the properties of arithmetic operations to perform the swapping operation.

Q - 7.

(a) Define algorithm. Write algorithm for finding factorial of a number.

(b) What is flowchart? Explain different symbols used for flowchart.

Ans :-

(a) Algorithm:

An algorithm is a step-by-step procedure or formula for solving a problem or accomplishing some end, especially by a computer. It's a precise set of instructions to perform a specific task.

Algorithm for Finding Factorial of a Number:

```
1. Start

2. Initialize variables: `number`, `factorial`, and `i`.

3. Read the value of `number` for which factorial is to be found.

4. Set `factorial` to 1.

5. Loop from `i = 1` to `number`.

   - Multiply `factorial` by `i`.

6. Repeat step 5 until `i` reaches `number`.
```

7. Display the value of `factorial`.

8. **End**

(b) Flowchart:

A flowchart is a graphical representation of an algorithm or process. It uses different symbols to represent different types of instructions or steps in a process.

Symbols used in a Flowchart:

1. Start/End Symbol: Represents the beginning or end of a process. Usually depicted as an oval shape with the word "Start" or "End" inside.

2. Process Symbol: Represents a task or operation to be performed. It can be any computational operation.

3. Input/Output Symbol: Represents input from the user or output to the user. It is usually depicted as a parallelogram shape.

4. Decision Symbol: Represents a decision point in the process where a decision is made based on a condition. It usually contains a question and has two or more arrows leaving it, each representing different possible outcomes based on the condition.

5. Connector Symbol: Used to connect different parts of the flowchart. It is represented by a small circle.

6. Flow Arrows: Arrows connecting different symbols indicating the flow or sequence of steps in the process.

Example Flowchart for Finding Factorial of a Number:

```
+-------------+       +-----------+
|   Start     |------->|  Input    |
+-------------+       +-----------+
                    |
                    v
+-------------+       +-----------+
|   Process   |------->|  Factorial|
+-------------+       +-----------+
                    |
                    v
+-------------+       +-----------+
|   Output    |<-------|    End    |
+-------------+       +-----------+
```

In this flowchart:

- Start: Beginning of the process.

- Input: User inputs the number.

- Process: Calculation of factorial.

- Output: Displaying the factorial.

- End: End of the process.

Each symbol represents a specific action or step in the process, and the flow arrows show the sequence of steps to be followed.

Q - 8.

(a) What is constant? Explain different constants in C.

(b) What is variable? Give the rules for variable declaration.

Ans :-

(a) Constants in C:

In C programming, a constant is a value that cannot be altered during program execution. Constants can be of various types, each with its own characteristics. Here are the different types of constants in C:

1. Integer Constants: Integer constants are whole numbers without any fractional or decimal parts. They can be decimal, octal, or hexadecimal.

   Example: `10`, `0x1A`, `075`

2. Floating-point Constants: Floating-point constants are numbers that have a fractional part or are represented in exponential form.

   Example: `3.14`, `2.5e-3`

3. Character Constants: Character constants represent individual characters enclosed within single quotes.

   Example: `'A'`, `'9'`, `'\n'`

4. String Constants: String constants are sequences of characters enclosed within double quotes.

   Example: `"Hello, World!"`, `"123"`

5. Enumeration Constants: Enumeration constants are symbolic names representing integer values. They are defined using the `enum` keyword.

Example:

```
enum Color   RED, GREEN, BLUE };
```

Here, `RED`, `GREEN`, and `BLUE` are enumeration constants with integer values `0`, `1`, and `2` respectively.

(b) Variables and Rules for Variable Declaration:

In C programming, a variable is a named storage location in memory where data can be stored and manipulated during program execution. Here are the rules for variable declaration in C:

1. Variable Names:

- Variable names must begin with a letter (either uppercase or lowercase) or an underscore `_`.

- After the initial character, variable names can include letters, digits, or underscores.

- Variable names are case-sensitive.

2. Keywords: Variable names cannot be the same as C keywords (reserved words).

3. Length of Variable Names: The length of variable names should be meaningful and easily understandable. However, C does not impose a strict limit on the length of variable names.

4. Type Declaration: Variables must be declared with a data type before they can be used.

For example:

```
int age;

float salary;

char grade;
```

5. Initialization: Variables can be initialized at the time of declaration, where an initial value is assigned to the variable.

```
int count = 0;

float pi = 3.14;

char letter = 'A';
```

6. Scope: Variables have a scope, which defines where they can be accessed within the program. Variables can be declared globally (outside all functions) or locally (inside a function).

7. Storage Class Specifiers: Variables can have different storage class specifiers like `auto`, `extern`, `static`, and `register`, which define their storage duration and scope.

8. Data Type: Variables must be declared with a valid data type that specifies the type of data the variable can hold.

Following these rules ensures that variables are properly declared and used in C programs, which helps maintain code clarity, readability, and correctness.

Q - 9.

(a) Write an algorithm and flowchart to generate Fibonacci series of numbers up to 'n'.

(b) Draw the flowchart to find the greatest of three numbers.

Ans :-

(a) Algorithm to Generate Fibonacci Series up to 'n':

1. Start

2. Initialize variables `a`, `b`, and `c` to 0 and `n` to the desired limit of the Fibonacci series.

3. Set `a` to 0 and `b` to 1.

4. Display `a` and `b` as the first two numbers of the series.

5. Repeat steps 6-8 until `c` is less than or equal to `n`.

6. Calculate the next Fibonacci number `c` by adding `a` and `b`.

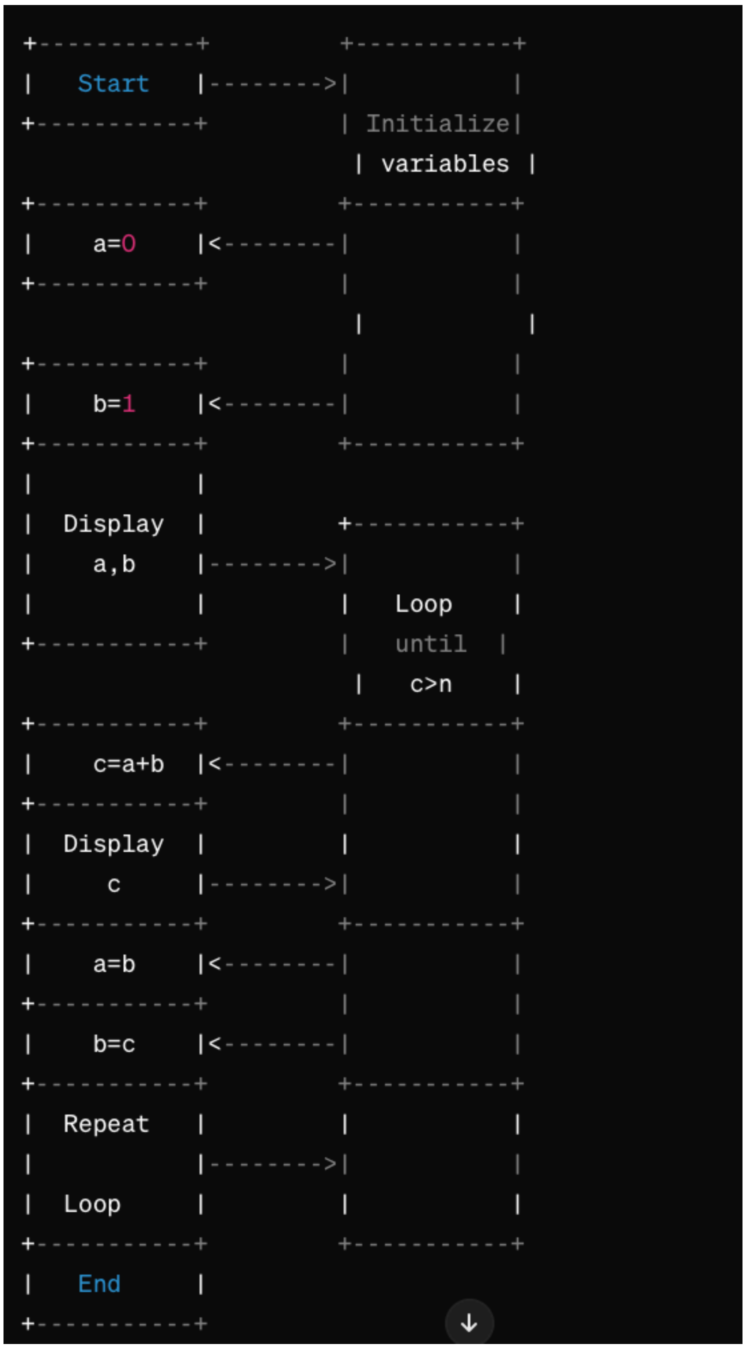
7. Display `c`.

8. Update `a` to the value of `b` and `b` to the value of `c`.

9. Repeat steps 5-8 until `c` exceeds `n`.

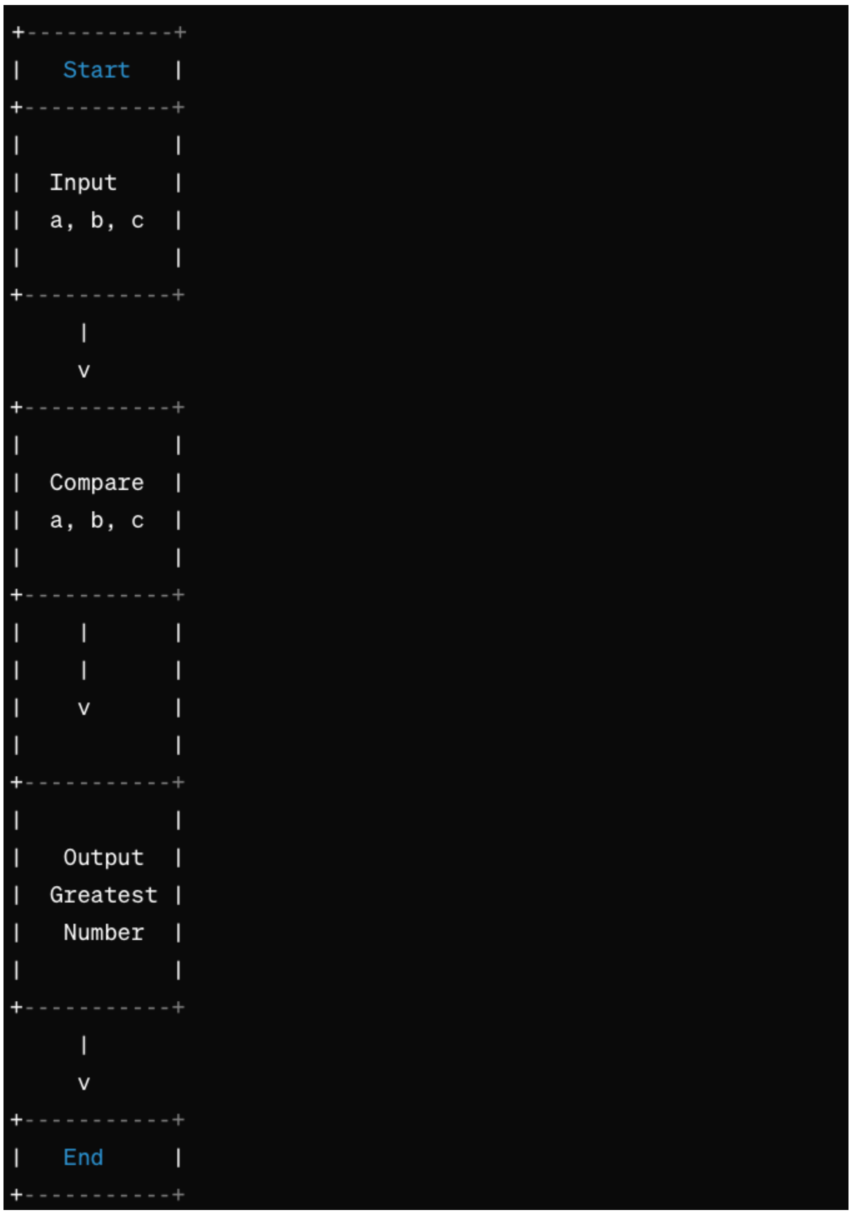
10. **End**

Flowchart to Generate Fibonacci Series up to 'n':



```
+-----------+         +-----------+
|   Start   |-------->|           |
+-----------+         | Initialize|
                       | variables |
+-----------+         +-----------+
|    a=0    |<--------|           |
+-----------+         |           |
                       |           |
+-----------+         |           |
|    b=1    |<--------|           |
+-----------+         +-----------+
|           |
|  Display  |         +-----------+
|    a,b    |-------->|           |
|           |         |   Loop    |
+-----------+         |   until   |
                       |    c>n    |
+-----------+         +-----------+
|   c=a+b   |<--------|           |
+-----------+         |           |
|  Display  |         |           |
|     c     |-------->|           |
+-----------+         +-----------+
|    a=b    |<--------|           |
+-----------+         |           |
|    b=c    |<--------|           |
+-----------+         +-----------+
|  Repeat   |         |           |
|           |-------->|           |
|  Loop     |         |           |
+-----------+         +-----------+
|   End     |
+-----------+
```

(b) Flowchart to Find the Greatest of Three Numbers:



```
+-----------+
|   Start   |
+-----------+
|           |
|  Input    |
|  a, b, c  |
|           |
+-----------+
     |
     v
+-----------+
|           |
|  Compare  |
|  a, b, c  |
|           |
+-----------+
|    |      |
|    |      |
|    v      |
|           |
+-----------+
|           |
|   Output  |
|  Greatest |
|   Number  |
|           |
+-----------+
     |
     v
+-----------+
|   End     |
+-----------+
```

In this flowchart:

- Start: Beginning of the process.

- Input: User inputs three numbers.

- Compare: Comparison is made between the three numbers to find the greatest.

- Output: Display the greatest number.

- End: End of the process.

Each symbol represents a specific action or step in the process, and the flow arrows show the sequence of steps to be followed.

Q - 10.

(a) Write an algorithm and flowchart to find whether the given number is prime or not.

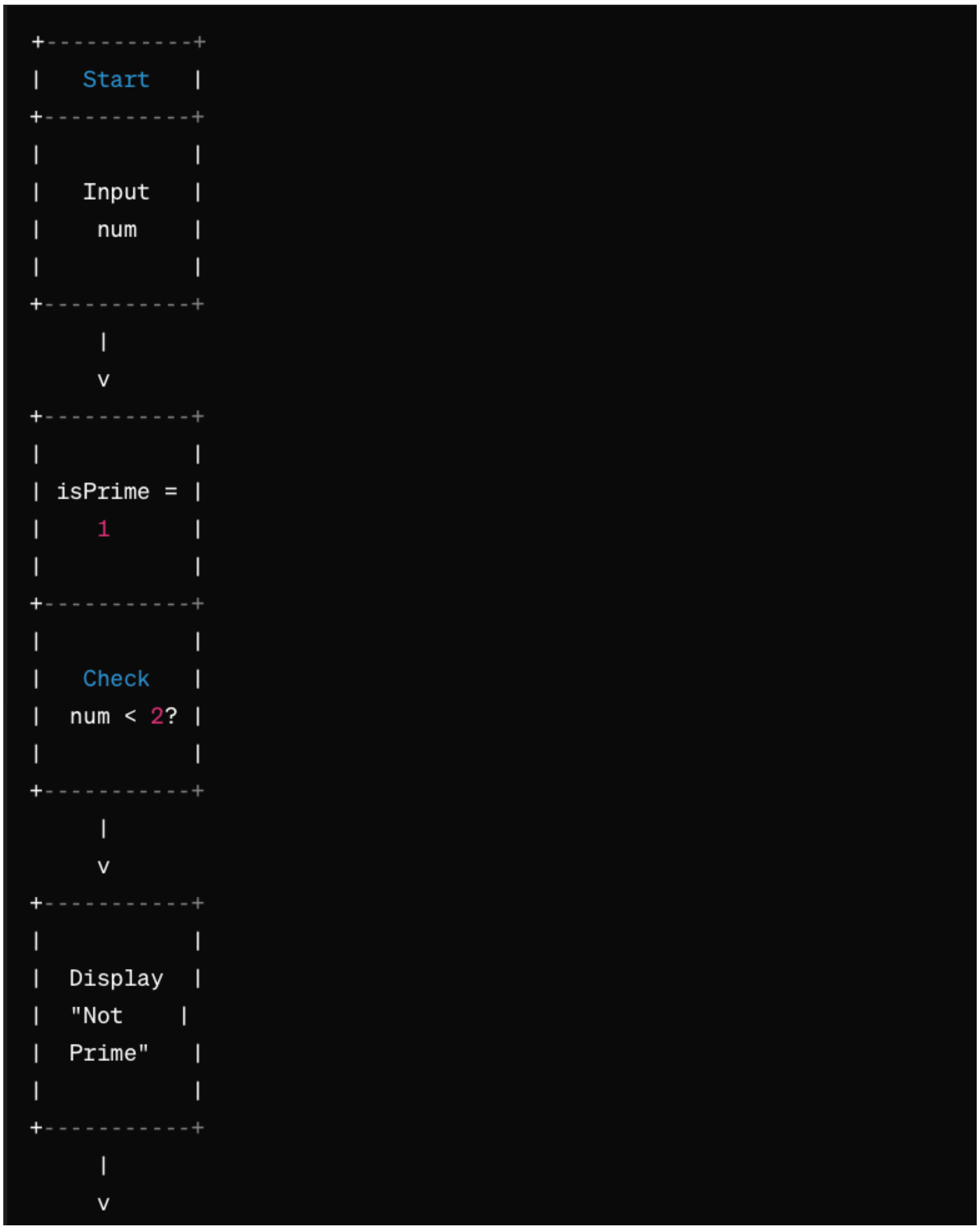
(b) Explain about type conversion in C.

Ans :-

(a) Algorithm to Determine Whether a Given Number is Prime or Not:

```
1. Start

2. Initialize variables `num`, `i`, and `isPrime`.

3. Read the value of `num`.

4. Set `isPrime` to 1 (assuming `num` is prime initially).

5. If `num` is less than 2, set `isPrime` to 0.

6. Loop from `i = 2` to `num/2`.

   - If `num` is divisible by `i`, set `isPrime` to 0 and break the loop.

7. If `isPrime` is 1, display `num` is prime, else display `num` is not prime.

8. End
```

(b) Flowchart to Determine Whether a Given Number is Prime or Not:



```
+-----------+
|   Start   |
+-----------+
|           |
|   Input   |
|    num    |
|           |
+-----------+
     |
     v
+-----------+
|           |
| isPrime = |
|    1      |
|           |
+-----------+
|           |
|   Check   |
|  num < 2? |
|           |
+-----------+
     |
     v
+-----------+
|           |
|  Display  |
|  "Not    |
|  Prime"   |
|           |
+-----------+
     |
     v
```

```
+-----------+
|           |
|   Loop    |
|  i = 2 to |
|  num/2    |
|           |
+-----------+
     |
     v
+-----------+
|           |
|   Check   |
|  num % i  |
|  == 0 ?   |
|           |
+-----------+
     |
     v
+-----------+
|           |
| isPrime = |
|     0     |
|   Break   |
|   Loop    |
|           |
+-----------+
     |
     v
+-----------+
|           |
|  Display  |
|  "Prime"  |
|           |
+-----------+
     |
     v
+-----------+
|    End    |
+-----------+
```

(b) Explanation of Type Conversion in C:

In C programming, type conversion refers to the process of converting a value from one data type to another. This can happen implicitly or explicitly.

1.Implicit Type Conversion: It occurs automatically by the compiler when an expression involving different data types is encountered. The conversion is performed in a way that preserves data and avoids loss of information.

Example:

```
int num1 = 10;
float num2 = num1;  // Implicit conversion from int to float
```

2. Explicit Type Conversion: Also known as type casting, it involves the programmer explicitly specifying the type to which the value is to be converted. It is done using casting operators like `(int)`, `(float)`, etc.

Example:

```
float num1 = 3.14;
int num2 = (int)num1;  // Explicit conversion from float to int
```

Type conversion is essential for performing operations involving different data types, ensuring compatibility between operands, and avoiding unintended errors in C programs. However, it's important to handle type conversions carefully to avoid loss of data or precision when converting between different data types.

UNIT –II : DECISION & LOOP CONTROL STATEMENTS

Short Answer (2 mark) Questions

1. **Classify the different types of decision making statements.**

Ans:-

Classify the different types of decision making statements.

- The different types of decision-making statements in C are:
- if statement: Executes a block of code if a specified condition is true.
- if-else statement: Executes one block of code if the condition is true and another block if the condition is false.
- nested if statement: An if statement inside another if statement.
- else-if ladder: Multiple if-else statements in sequence.
- switch statement: Evaluates an expression and executes code based on matching cases.

2. **How switch case works without break statement.**

Ans :-

How switch case works without break statement.

- If a `break` statement is not used within a `case`, the control will fall through to the next `case` statement and execute its code as well. This behavior is called fall-through.

3. Write the syntax for nested if and else-if ladder?

Ans:-

- Nested if:

```
if (condition1) {
   if (condition2) {
       // code to be executed if both condition1 and condition2 are true
   }
}
```

- else-if ladder:

```
if (condition1) {
   // code to be executed if condition1 is true
}
else if (condition2) {
   // code to be executed if condition2 is true and condition1 is false
}
else {
   // code to be executed if both condition1 and condition2 are false
}
```

4. **Write a program to check whether the person is eligible to vote.**

Ans:-

```
#include <stdio.h>

int main() {

    int age;



    printf("Enter your age: ");

    scanf("%d", &age);



    if (age >= 18) {

        printf("You are eligible to vote!\n");

    } else {

        printf("You are not eligible to vote yet.\n");

    }



    return 0;

}
```

5. **Write and explain syntax of ―for‖ loop.**

Ans:-

```
for (initialization; condition; update) {

    // code to be executed

}
```

- **initialization**: Initializes loop control variable(s).
- **condition**: Evaluates to true or false. If true, the loop continues; if false, the loop terminates.
- **update**: Updates the loop control variable(s) after each iteration.

6. **Distinguish between while and do-while statements.**

Ans:-

Distinguish between while and do-while statements.

- **while loop:** The condition is evaluated before the execution of the loop body. If the condition is false initially, the loop body may never execute.
- **do-while loop:** The condition is evaluated after the execution of the loop body. It guarantees that the loop body will execute at least once.

7. **Write a program to print the multiplication table from 1 to n?**

Ans:-

```
#include <stdio.h>

int main() {

   int n, i, j;



   printf("Enter the value of n: ");

   scanf("%d", &n);



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

      for (j = 1; j <= 10; j++) {

         printf("%d x %d = %d\n", i, j, i * j);

      }

      printf("\n");

   }



   return 0;

}
```

8. **Differentiate between break and continue.**

Ans:-

Differentiate between break and continue.

- **break**: Terminates the loop and transfers control to the statement immediately following the loop.
- **continue**: Skips the remaining code in the loop for the current iteration and proceeds to the next iteration.

9. **Define goto with an example.**

Ans:-

goto is a jump statement that transfers the control of the program to a specified label within the same function.

- Example:

```
#include <stdio.h>


int main() {

   int i = 0;


loop:

   printf("%d\n", i);

   i++;

   if (i < 5) {

      goto loop;

   }


   return 0;

}
```

10. Define exit and return statements.

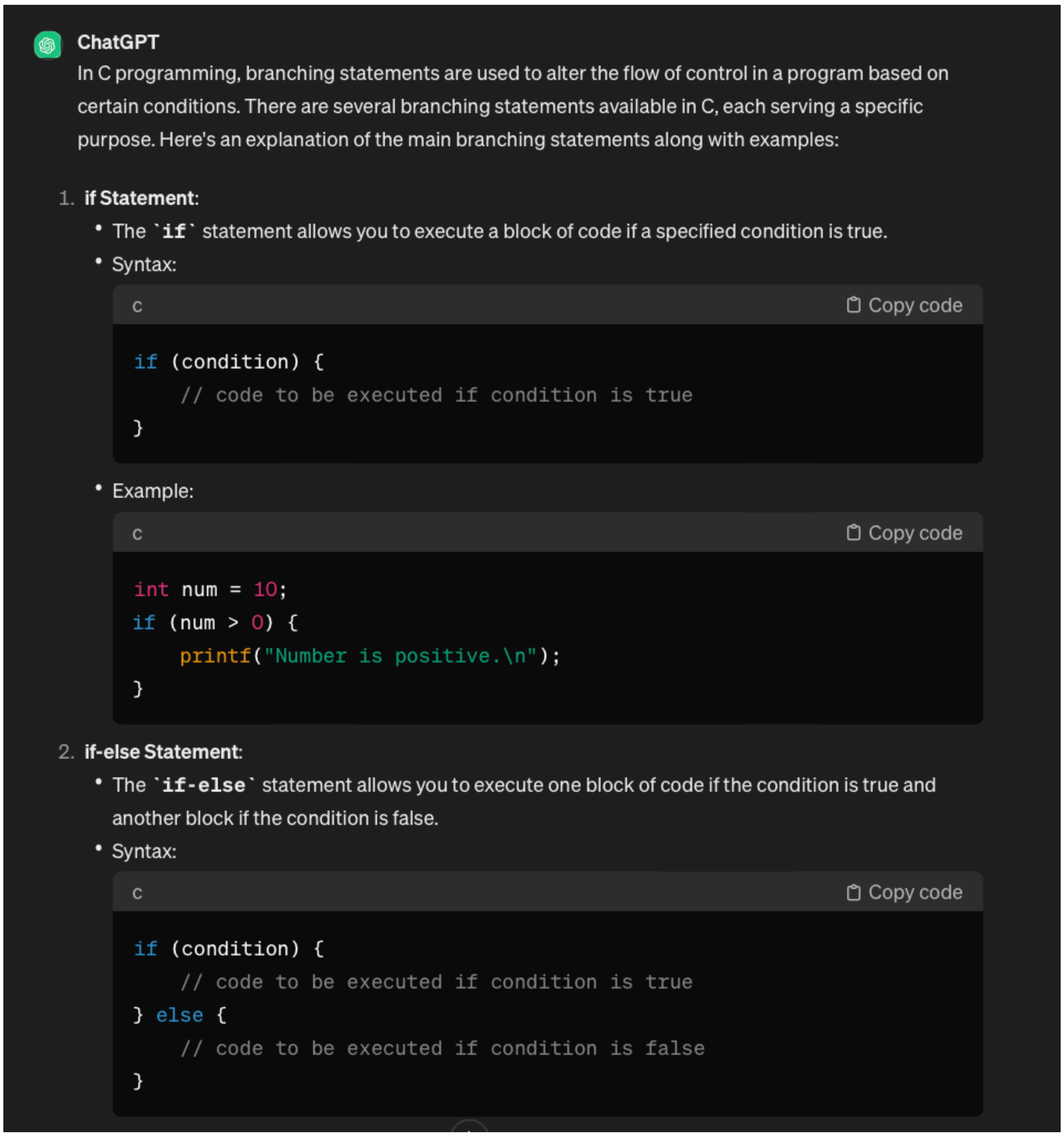
Ans:-

- **exit**: Terminates the entire program execution immediately.
- **return**: Exits the current function and returns a value to the calling function. It can also be used to terminate a loop or switch statement.

Essay Answer (10 mark) Questions

Q - 1. Explain various branching statements in C with examples.

Ans:-



ChatGPT

In C programming, branching statements are used to alter the flow of control in a program based on certain conditions. There are several branching statements available in C, each serving a specific purpose. Here's an explanation of the main branching statements along with examples:

1. **if Statement**:
   - The `if` statement allows you to execute a block of code if a specified condition is true.
   - Syntax:

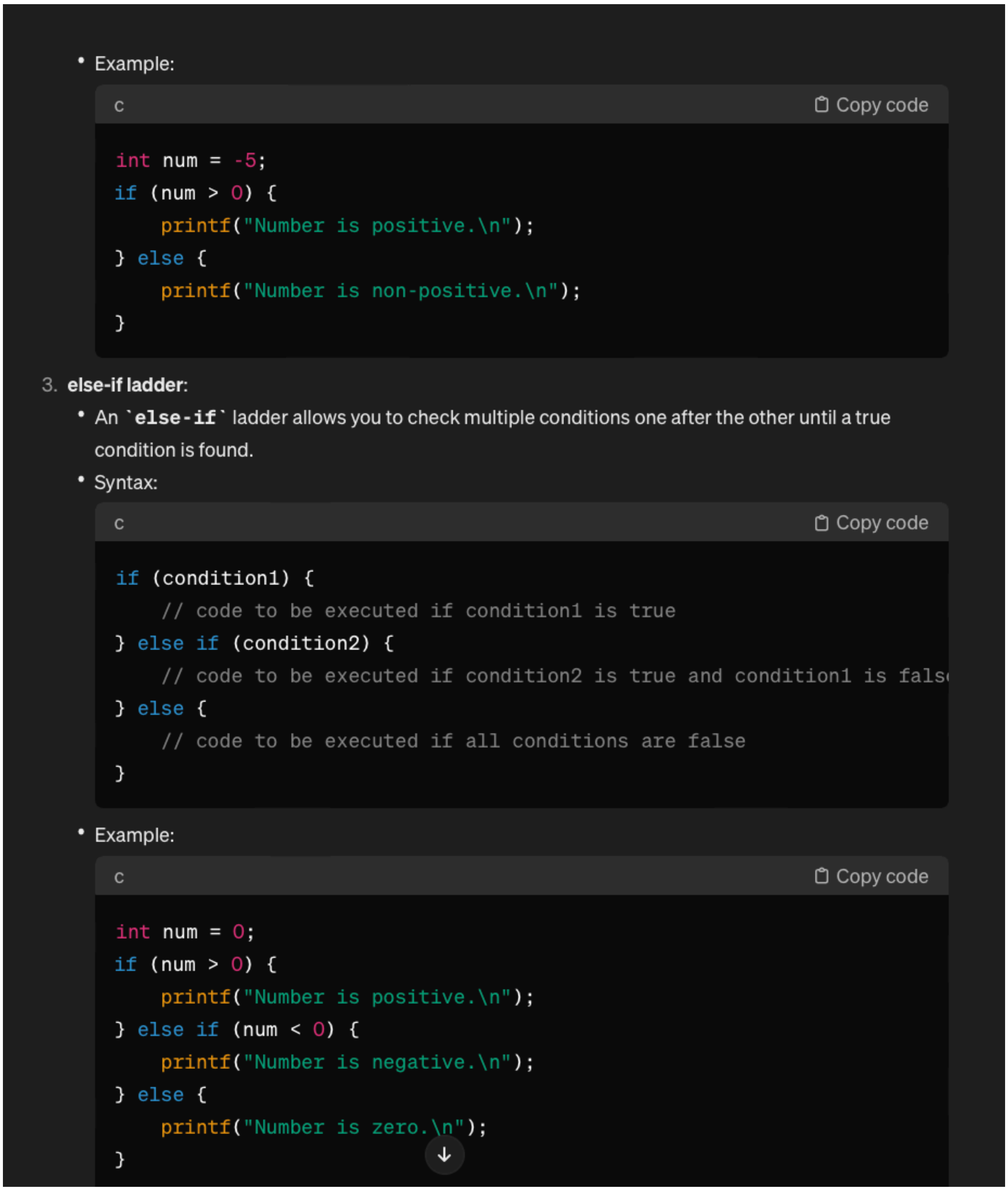
```c
if (condition) {
    // code to be executed if condition is true
}
```

   - Example:

```c
int num = 10;
if (num > 0) {
    printf("Number is positive.\n");
}
```

2. **if-else Statement**:
   - The `if-else` statement allows you to execute one block of code if the condition is true and another block if the condition is false.
   - Syntax:

```c
if (condition) {
    // code to be executed if condition is true
} else {
    // code to be executed if condition is false
}
```



- Example:

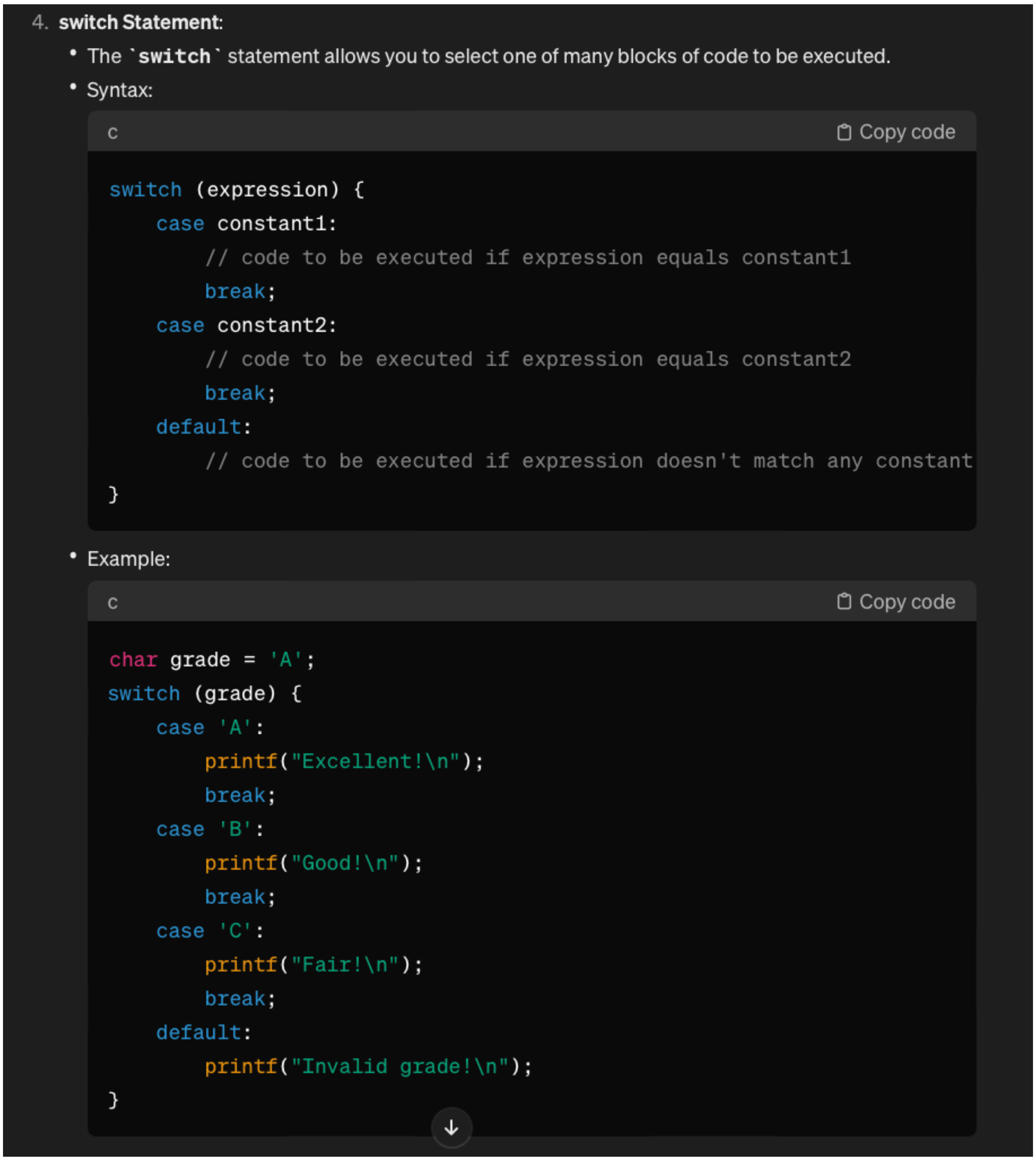
```c
int num = -5;
if (num > 0) {
    printf("Number is positive.\n");
} else {
    printf("Number is non-positive.\n");
}
```

3. **else-if ladder:**
   - An `else-if` ladder allows you to check multiple conditions one after the other until a true condition is found.
   - Syntax:

```c
if (condition1) {
    // code to be executed if condition1 is true
} else if (condition2) {
    // code to be executed if condition2 is true and condition1 is fals
} else {
    // code to be executed if all conditions are false
}
```

   - Example:

```c
int num = 0;
if (num > 0) {
    printf("Number is positive.\n");
} else if (num < 0) {
    printf("Number is negative.\n");
} else {
    printf("Number is zero.\n");
}
```



4. **switch Statement:**
   - The `switch` statement allows you to select one of many blocks of code to be executed.
   - Syntax:

```c
switch (expression) {
    case constant1:
        // code to be executed if expression equals constant1
        break;
    case constant2:
        // code to be executed if expression equals constant2
        break;
    default:
        // code to be executed if expression doesn't match any constant
}
```

   - Example:

```c
char grade = 'A';
switch (grade) {
    case 'A':
        printf("Excellent!\n");
        break;
    case 'B':
        printf("Good!\n");
        break;
    case 'C':
        printf("Fair!\n");
        break;
    default:
        printf("Invalid grade!\n");
}
```

These branching statements provide flexibility in controlling the flow of execution in a C program, allowing for the execution of different code paths based on varying conditions.

Q - 2.

(a) Write and explain about switch statement.
(b) Write a Program to perform arithmetic operations using switch.

Ans:-

(a) Switch Statement in C:

The `switch` statement in C is a control statement that allows a variable to be tested for equality against a list of values. It provides an efficient way to select among multiple alternative paths of execution based on the value of an expression.

Syntax of Switch Statement:

```
switch (expression) {

   case constant1:

      // code to be executed if expression equals constant1

      break;

   case constant2:

      // code to be executed if expression equals constant2

      break;

   ...

   default:

      // code to be executed if expression doesn't match any constant

}
```

- The `expression` is evaluated once and its value is compared with the values of the `case` constants.

- If a match is found, the corresponding block of code is executed.

- The `break` statement is used to terminate the `switch` statement and transfer control to the statement following the `switch` block.

- If no `case` matches the value of the `expression`, the `default` block of code (optional) is executed.

Example of Switch Statement:

```
#include <stdio.h>


int main() {
    int choice;
    printf("Enter your choice (1-3): ");
    scanf("%d", &choice);


    switch (choice) {
        case 1:
            printf("You chose option 1\n");
            break;
        case 2:
            printf("You chose option 2\n");
            break;
        case 3:
            printf("You chose option 3\n");
            break;
        default:
            printf("Invalid choice\n");
    }


    return 0;
}
```

(b) Program to Perform Arithmetic Operations Using Switch:

Here's a program that performs arithmetic operations (addition, subtraction, multiplication, and division) based on user input using the `switch` statement:

```
#include <stdio.h>

int main() {
  char operator;
  float num1, num2, result;

  printf("Enter operator (+, -, *, /): ");
  scanf("%c", &operator);

  printf("Enter two numbers: ");
  scanf("%f %f", &num1, &num2);

  switch (operator) {
    case '+':
      result = num1 + num2;
      printf("Sum: %.2f\n", result);
      break;
    case '-':
      result = num1 - num2;
      printf("Difference: %.2f\n", result);
      break;
    case '*':
      result = num1 * num2;

      printf("Product: %.2f\n", result);

      break;

    case '/':

      if (num2 != 0) {

          result = num1 / num2;

          printf("Quotient: %.2f\n", result);

      } else {

          printf("Error! Division by zero\n");

      }

      break;

    default:

      printf("Invalid operator\n");

  }


  return 0;

}
```

This program prompts the user to enter an arithmetic operator and two numbers. It then performs the specified operation using a `switch` statement and displays the result. If the user tries to divide by zero, it outputs an error message.

Q - 3. List and explain loop control (or) iteration statements in C.

Ans:-

Loop control statements, also known as iteration statements, in C programming allow you to execute a block of code repeatedly as long as a specified condition is true. These statements provide powerful mechanisms for performing repetitive tasks efficiently. Here are the loop control statements in C:

1.for Loop:

- The `for` loop is used to execute a block of code a specified number of times.

- **Syntax:**

```
for (initialization; condition; update) {

   // code to be executed

}
```

- `initialization`: Initializes loop control variable(s).
- `condition`: Evaluates to true or false. If true, the loop continues; if false, the loop terminates.
- `update`: Updates the loop control variable(s) after each iteration.

- Example:

```
for (int i = 0; i < 5; i++) {

   printf("%d ", i);

}
```

2.while Loop:

- The `while` loop is used to execute a block of code as long as a specified condition is true.

- **Syntax:**

```
 while (condition) {
   // code to be executed
}
```

- `condition`: Evaluates to true or false. If true, the loop continues; if false, the loop terminates.

- **Example:**

```
int count = 0;
while (count < 5) {
   printf("%d ", count);
   count++;
}
```

3. do-while Loop:

- The `do-while` loop is similar to the `while` loop, but it executes the block of code at least once, even if the condition is false.

- **Syntax:**

```
do {
   // code to be executed
} while (condition);
```

- `condition`: Evaluates to true or false. If true, the loop continues; if false, the loop terminates.

- **Example:**

```
 int num = 5;
do {
   printf("%d ", num);
   num--;
} while (num > 0);
```

Loop Control Statements:

1. break Statement:

- The `break` statement is used to terminate the loop immediately and transfer control to the statement following the loop.

- It can be used with `for`, `while`, and `do-while` loops.

- **Example:**

```
for (int i = 0; i < 10; i++) {
   if (i == 5) {
      break;
   }
   printf("%d ", i);
}
```

2. continue Statement:

- The `continue` statement is used to skip the remaining code in the current iteration of the loop and proceed to the next iteration.

- It can be used with `for`, `while`, and `do-while` loops.

- Example:

```
for (int i = 0; i < 10; i++) {

   if (i % 2 == 0) {

       continue;

   }

   printf("%d ", i);

}
```

Loop control statements provide flexibility and control over the execution of loops in C programs, allowing developers to write efficient and concise code for repetitive tasks.

Q - 4.

(a) Write and explain syntax of –for‖ loop.
(b) Write a program to generate prime numbers between 1 and ‗n'.

Ans:-

(a) Syntax of "for" Loop:

The `for` loop in C is a control flow statement that allows you to execute a block of code repeatedly for a fixed number of times. It consists of three parts: initialization, condition, and update.

Syntax:

```
for (initialization; condition; update) {

   // code to be executed

}
```

Initialization: It is an expression where you initialize loop control variable(s). It is executed only once at the beginning of the loop.

Condition: It is a boolean expression that is evaluated before each iteration of the loop. If the condition evaluates to true, the loop continues; otherwise, the loop terminates.

Update: It is an expression that updates loop control variable(s) after each iteration of the loop.

Example:

```
#include <stdio.h>



int main() {

   // Example of a for loop to print numbers from 1 to 5

   for (int i = 1; i <= 5; i++) {

      printf("%d ", i);

   }

   return 0;

}
```

In this example:

- Initialization: `int i = 1;` initializes the loop control variable `i` to 1.

- Condition: `i <= 5;` checks if `i` is less than or equal to 5.

- Update: `i++` increments `i` by 1 after each iteration.

(b) Program to Generate Prime Numbers between 1 and 'n':

Here's a program to generate prime numbers between 1 and 'n':

```
#include <stdio.h>

#include <stdbool.h>

int main() {

    int n;

    printf("Enter the value of n: ");

    scanf("%d", &n);

    printf("Prime numbers between 1 and %d are: ", n);

    for (int num = 2; num <= n; num++) {

        bool isPrime = true;

        for (int i = 2; i <= num / 2; i++) {

            if (num % i == 0) {

                isPrime = false;

                break;

            }

        }

        if (isPrime) {

            printf("%d ", num);

        }

    }

return 0;

}
```

This program takes an input 'n' from the user and prints all prime numbers between 1 and 'n'. It uses a nested `for` loop to check if each number is prime or not. The outer loop iterates from 2 to 'n', and the inner loop checks if the current number is divisible by any number from 2 to its half. If it is divisible, the number is not prime; otherwise, it is prime.

Q - 5.

(a) Write a program to check whether the given number is palindrome or not.

(b) Write a program to check whether the given number is ―Even‖ or ―Odd‖ using GOTO statement.

Ans :-

(a) Program to Check Whether a Given Number is Palindrome or Not:

```
#include <stdio.h>


int main() {
    int num, reversedNum = 0, originalNum, remainder;


    printf("Enter an integer: ");
    scanf("%d", &num);


    originalNum = num;


    // Reversing the number
    while (num != 0) {
        remainder = num % 10;
        reversedNum = reversedNum * 10 + remainder;
        num /= 10;
    }


    // Checking if the original number is equal to its reverse
    if (originalNum == reversedNum) {
        printf("%d is a palindrome.\n", originalNum);
    } else {
        printf("%d is not a palindrome.\n", originalNum);
    }

    return 0;
}
```

This program takes an integer input from the user and checks if it is a palindrome or not. It reverses the number and compares it with the original number to determine if they are the same.

(b) Program to Check Whether the Given Number is "Even" or "Odd" Using GOTO Statement:

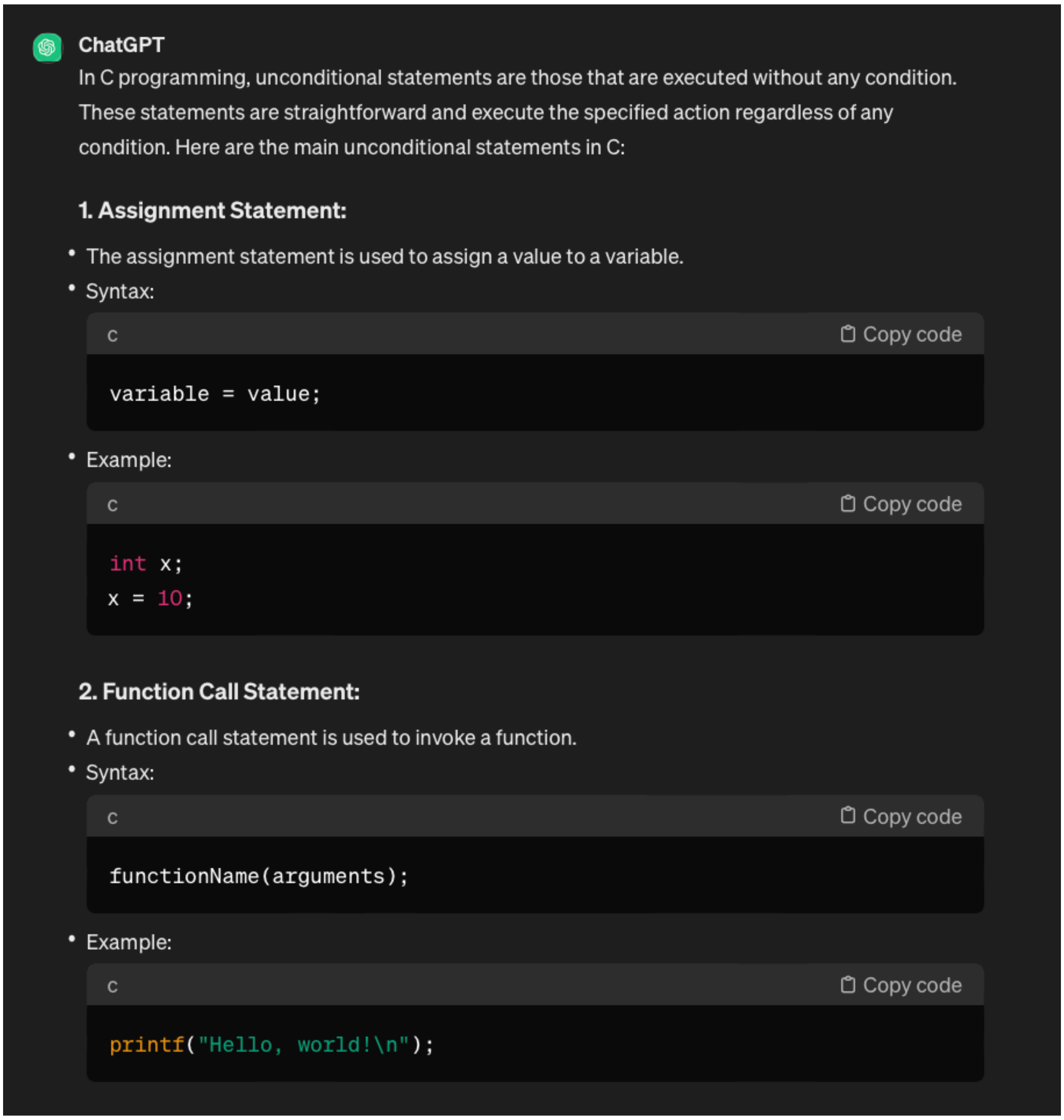
```
#include <stdio.h>

int main() {
    int num;

input:
    printf("Enter an integer: ");
    scanf("%d", &num);

    if (num % 2 == 0) {
        printf("%d is Even.\n", num);
    } else {
        printf("%d is Odd.\n", num);
    }
    goto input; // Jump to input label to repeat the process



    return 0;

}
```

This program continuously prompts the user to enter an integer. It then checks if the entered number is even or odd using the modulo operator. After printing the result, it uses the `goto` statement to jump back to the `input` label, allowing the user to input another number. This process continues indefinitely until the program is terminated manually.

Note: The use of `goto` statements is generally discouraged because it can make the code difficult to understand and maintain. It is better to use loops for repetitive tasks.

Q - 6.List and explain unconditional statements in C with examples.

Ans:-



ChatGPT

In C programming, unconditional statements are those that are executed without any condition. These statements are straightforward and execute the specified action regardless of any condition. Here are the main unconditional statements in C:

1. Assignment Statement:

- The assignment statement is used to assign a value to a variable.
- Syntax:

```c
variable = value;
```

- Example:

```c
int x;
x = 10;
```

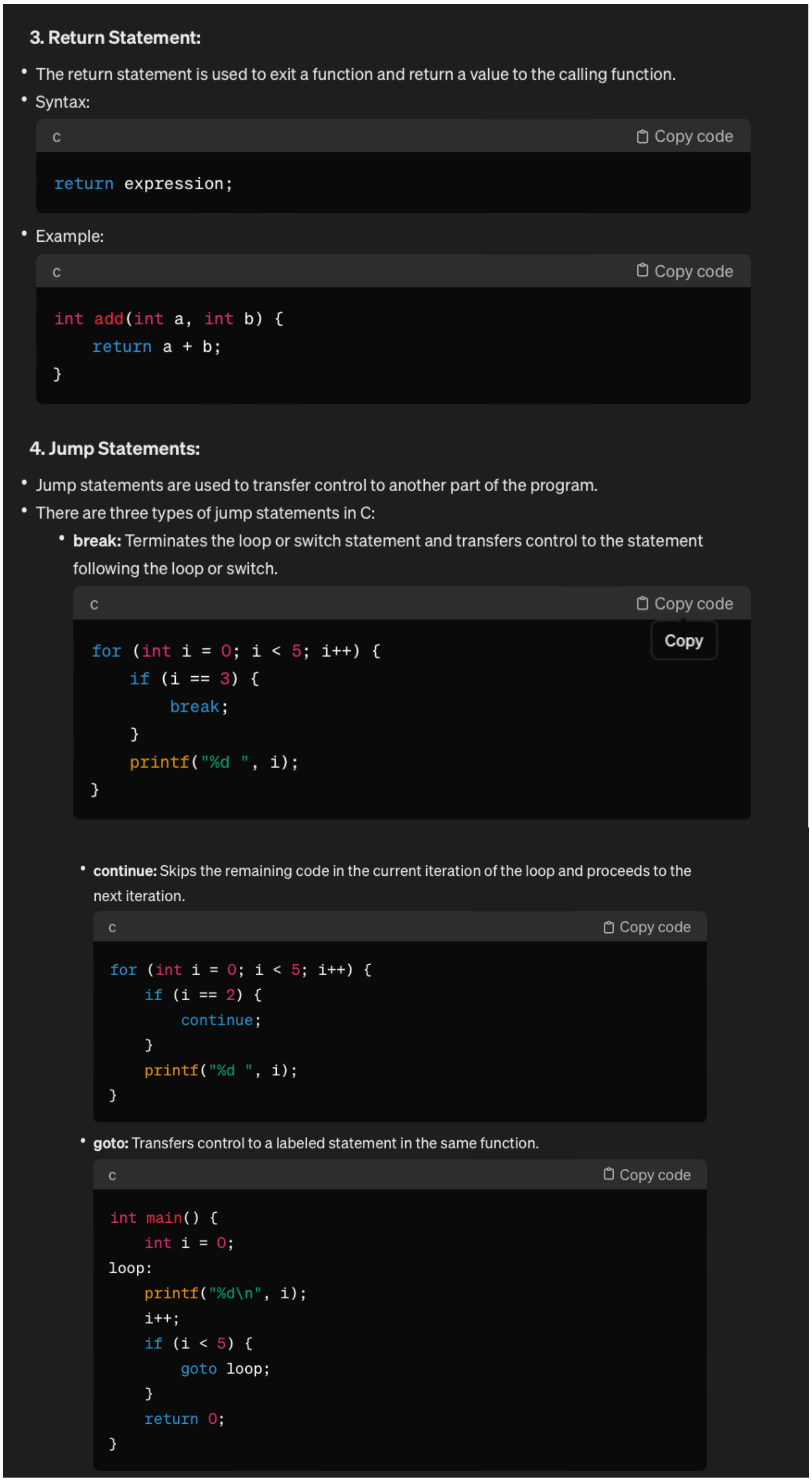
2. Function Call Statement:

- A function call statement is used to invoke a function.
- Syntax:

```c
functionName(arguments);
```

- Example:

```c
printf("Hello, world!\n");
```



3. Return Statement:

- The return statement is used to exit a function and return a value to the calling function.
- Syntax:

```c
return expression;
```

- Example:

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

4. Jump Statements:

- Jump statements are used to transfer control to another part of the program.
- There are three types of jump statements in C:
  - **break:** Terminates the loop or switch statement and transfers control to the statement following the loop or switch.

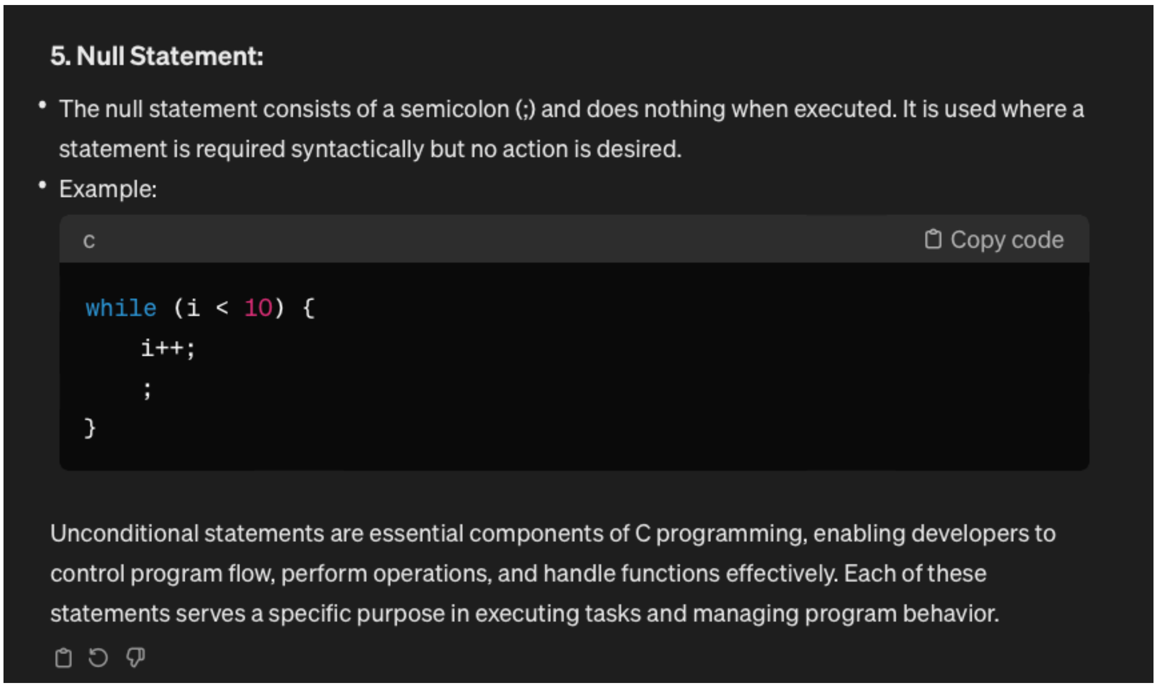
```c
for (int i = 0; i < 5; i++) {
    if (i == 3) {
        break;
    }
    printf("%d ", i);
}
```

  - **continue:** Skips the remaining code in the current iteration of the loop and proceeds to the next iteration.

```c
for (int i = 0; i < 5; i++) {
    if (i == 2) {
        continue;
    }
    printf("%d ", i);
}
```

  - **goto:** Transfers control to a labeled statement in the same function.

```c
int main() {
    int i = 0;
loop:
    printf("%d\n", i);
    i++;
    if (i < 5) {
        goto loop;
    }
    return 0;
}
```


5. Null Statement:

- The null statement consists of a semicolon (;) and does nothing when executed. It is used where a statement is required syntactically but no action is desired.
- Example:

```c
while (i < 10) {
    i++;
    ;
}
```

Unconditional statements are essential components of C programming, enabling developers to control program flow, perform operations, and handle functions effectively. Each of these statements serves a specific purpose in executing tasks and managing program behavior.

Q - 7.

(a) Write a program to find sum of the individual digits of a given number.

(b) Write a program to find the sum of even and odd numbers from 1 to n.

Ans :-

(a) Program to Find Sum of the Individual Digits of a Given Number:

```c
#include <stdio.h>


int main() {
    int num, sum = 0, digit;


    printf("Enter a number: ");
    scanf("%d", &num);
    while (num != 0) {
        digit = num % 10;   // Extract the last digit
        sum += digit;       // Add the digit to the sum
        num /= 10;          // Remove the last digit
    }

    printf("Sum of individual digits: %d\n", sum);

    return 0;
}
```

Explanation:

- We take an integer input from the user (`num`).
- Inside the while loop, we extract the last digit of `num` using the modulus operator `%`.
- We add the extracted digit to the `sum` variable.
- Then, we update the value of `num` by removing its last digit using integer division `/`.
- The loop continues until `num` becomes 0.

(b) Program to Find the Sum of Even and Odd Numbers from 1 to n:

```
#include <stdio.h>

int main() {
    int n, sumEven = 0, sumOdd = 0;

    printf("Enter a positive integer: ");
    scanf("%d", &n);

    for (int i = 1; i <= n; i++) {
        if (i % 2 == 0) {
            sumEven += i;  // Add even numbers to sumEven
        } else {
            sumOdd += i;   // Add odd numbers to sumOdd
        }
    }

    printf("Sum of even numbers from 1 to %d: %d\n", n, sumEven);
    printf("Sum of odd numbers from 1 to %d: %d\n", n, sumOdd);

    return 0;
}
```

Explanation:

- We take an integer input from the user (`n`).

- We use a for loop to iterate from 1 to `n`.

- Inside the loop, we check if the current number `i` is even or odd using the modulus operator `%`.

- If `i` is even, we add it to the `sumEven` variable; otherwise, we add it to the `sumOdd` variable.

- After the loop finishes, we print the sums of even and odd numbers.

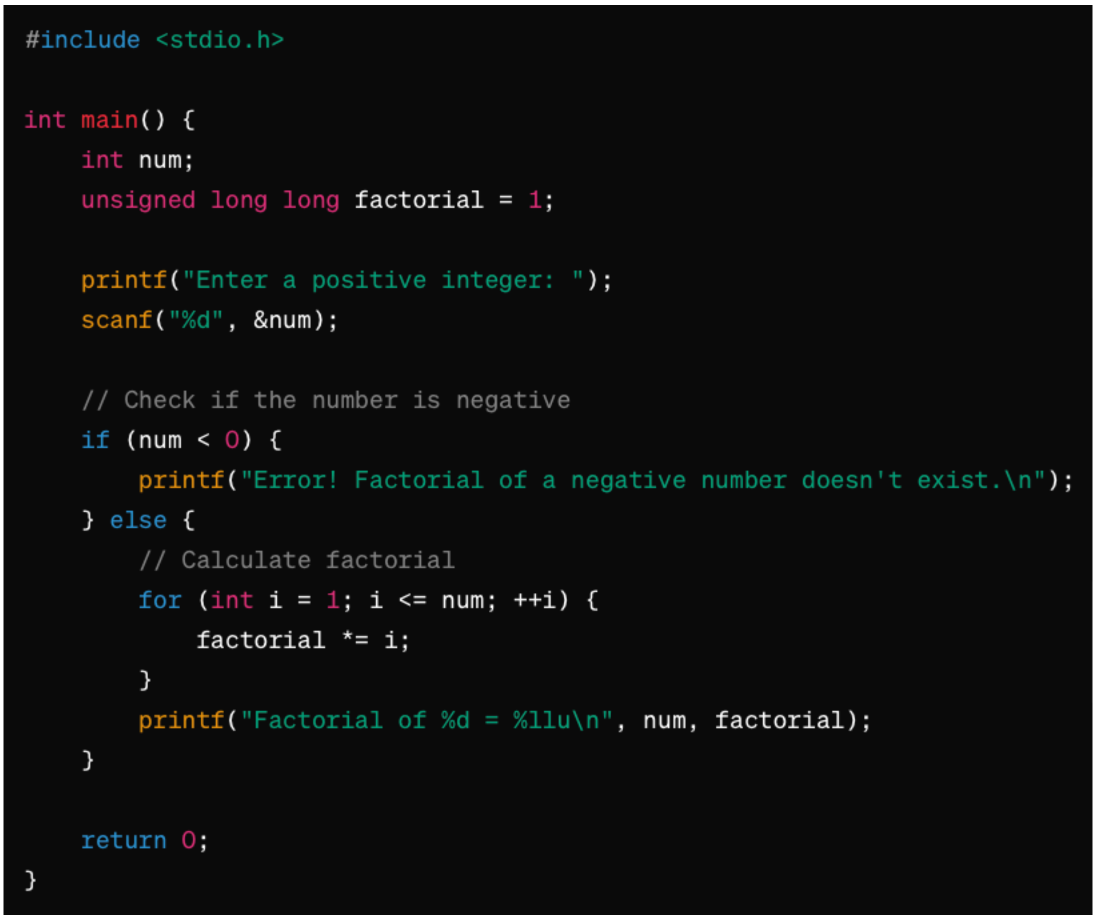
Q - 8.

(a) Write a program to find the factorial of a given number.

(b) Write a program to generate ‗n‘ Fibonacci numbers.

Ans :-

(a) Program to Find the Factorial of a Given Number:



```c
#include <stdio.h>

int main() {
    int num;
    unsigned long long factorial = 1;

    printf("Enter a positive integer: ");
    scanf("%d", &num);

    // Check if the number is negative
    if (num < 0) {
        printf("Error! Factorial of a negative number doesn't exist.\n");
    } else {
        // Calculate factorial
        for (int i = 1; i <= num; ++i) {
            factorial *= i;
        }
        printf("Factorial of %d = %llu\n", num, factorial);
    }

    return 0;
}
```

Explanation:

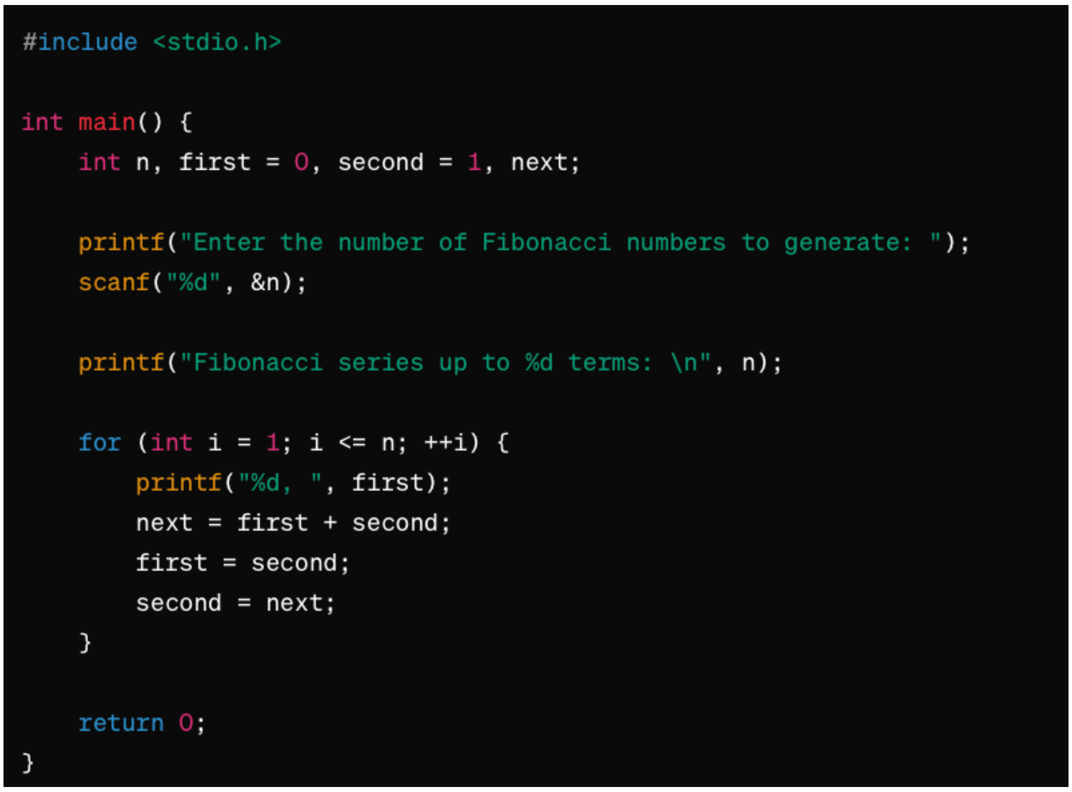
- We take an integer input from the user (`num`).

- We initialize `factorial` to 1 because the factorial of 0 and 1 is 1.

- We check if the number is negative. If it is, we print an error message.

- Otherwise, we use a for loop to calculate the factorial of the number.

(b) Program to Generate 'n' Fibonacci Numbers:



```c
#include <stdio.h>

int main() {
    int n, first = 0, second = 1, next;

    printf("Enter the number of Fibonacci numbers to generate: ");
    scanf("%d", &n);

    printf("Fibonacci series up to %d terms: \n", n);

    for (int i = 1; i <= n; ++i) {
        printf("%d, ", first);
        next = first + second;
        first = second;
        second = next;
    }

    return 0;
}
```

Explanation:

- We take an integer input from the user (`n`) to determine the number of Fibonacci numbers to generate.

- We initialize `first` and `second` as the first two Fibonacci numbers.

- We use a for loop to generate Fibonacci numbers up to the specified `n`.

- In each iteration, we print the value of `first`, calculate the next Fibonacci number (`next`), and update `first` and `second` accordingly.

Q - 9.

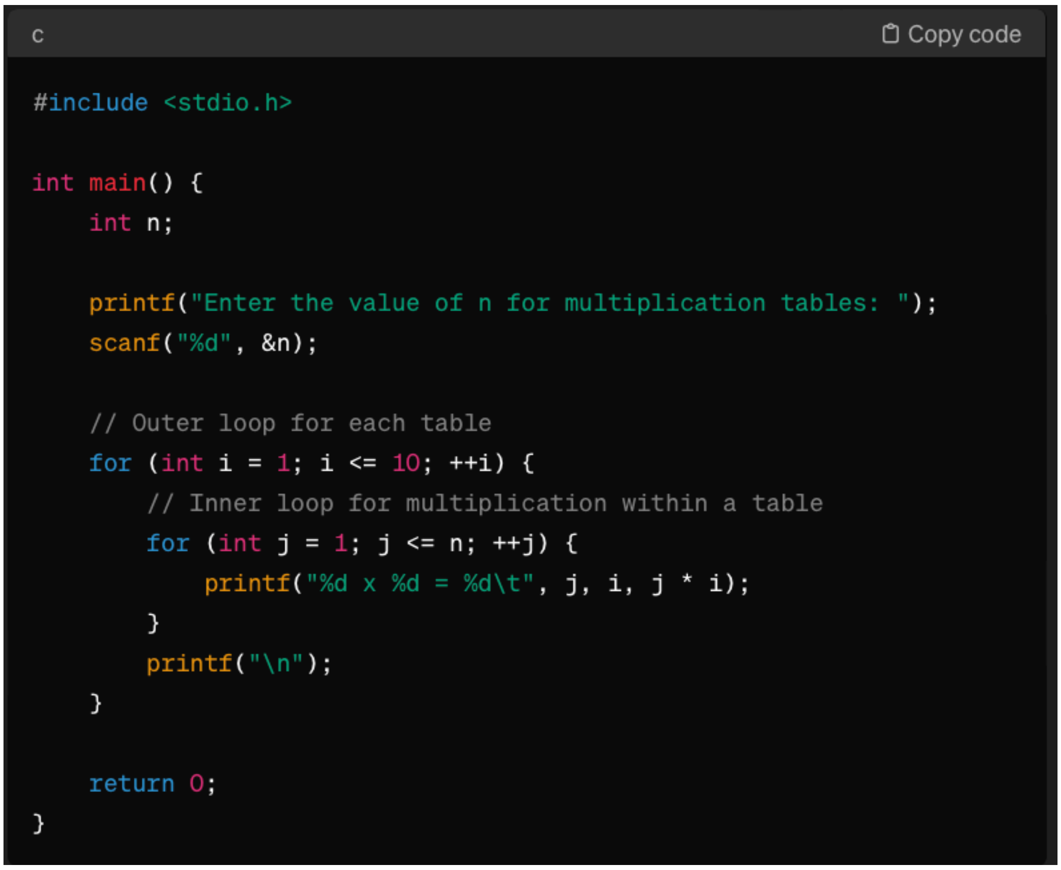
(a) What is a nested loop? Write a program to display multiplications tables from 1 to n.

(b) Write a program to display the following pattern.(ALL CLASS & LAB)

Ans :-

(a) Nested Loop and Program to Display Multiplication Tables from 1 to n:

A nested loop is a loop inside another loop. In C programming, you can have loops inside loops. The inner loop will be executed fully when the outer loop is executed once. This is useful for situations where you need to perform repetitive tasks within another repetitive task.



```c
#include <stdio.h>

int main() {
    int n;

    printf("Enter the value of n for multiplication tables: ");
    scanf("%d", &n);

    // Outer loop for each table
    for (int i = 1; i <= 10; ++i) {
        // Inner loop for multiplication within a table
        for (int j = 1; j <= n; ++j) {
            printf("%d x %d = %d\t", j, i, j * i);
        }
        printf("\n");
    }

    return 0;
}
```

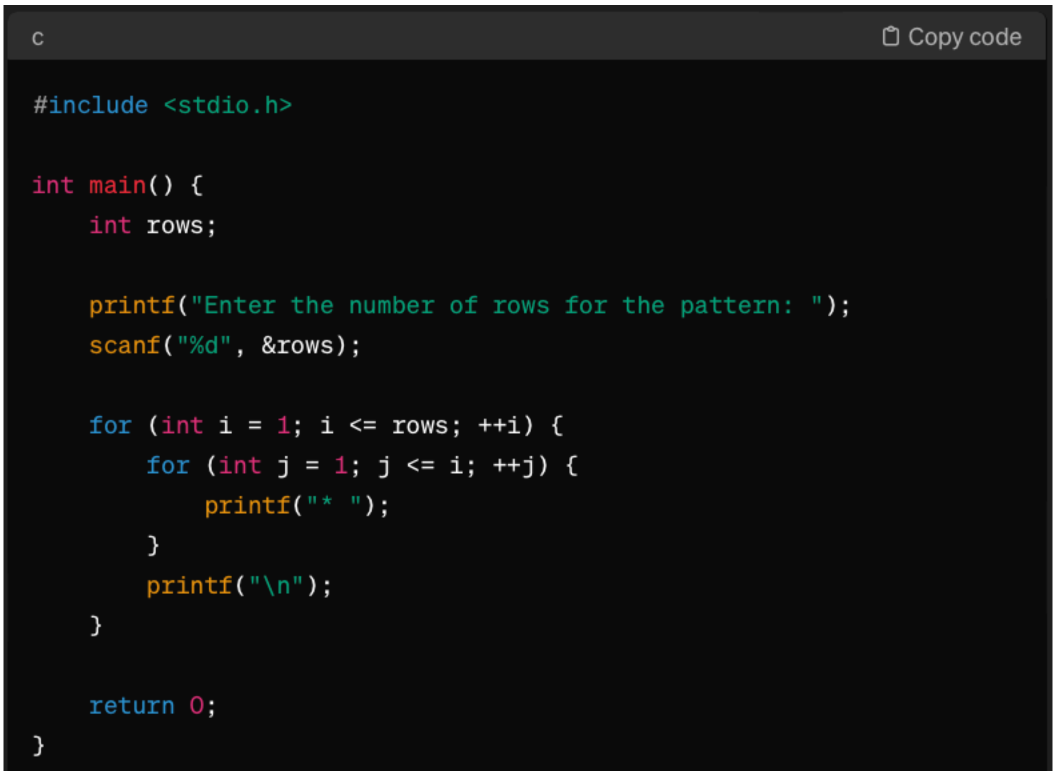
Explanation:

- The outer loop (`i`) runs from 1 to 10, representing each multiplication table.

- The inner loop (`j`) runs from 1 to `n`, displaying the multiplication for each number in the table.

- The result is formatted in a tabular form.

(b) Program to Display the Following Pattern:


c Copy code

```c
#include <stdio.h>

int main() {
    int rows;

    printf("Enter the number of rows for the pattern: ");
    scanf("%d", &rows);

    for (int i = 1; i <= rows; ++i) {
        for (int j = 1; j <= i; ++j) {
            printf("* ");
        }
        printf("\n");
    }

    return 0;
}
```

Explanation:

- The outer loop (`i`) represents the number of rows in the pattern.

- The inner loop (`j`) represents the number of asterisks (*) to be printed in each row.

- As the outer loop progresses, the number of asterisks in each row increases, creating a pattern.

Q - 10.

(a) Explain else-if ladder with the help of flowchart and program.

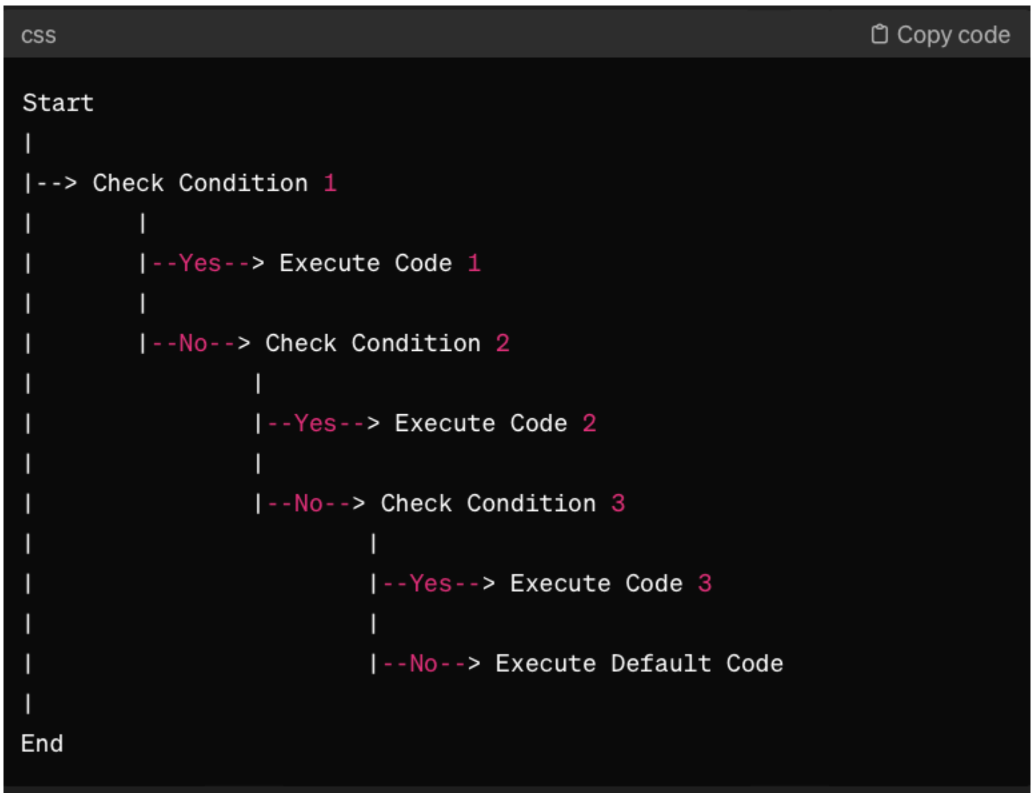
(b) How does nested if-else works explain with an example?

Ans :-

(a) Explanation of else-if Ladder with Flowchart and Program:

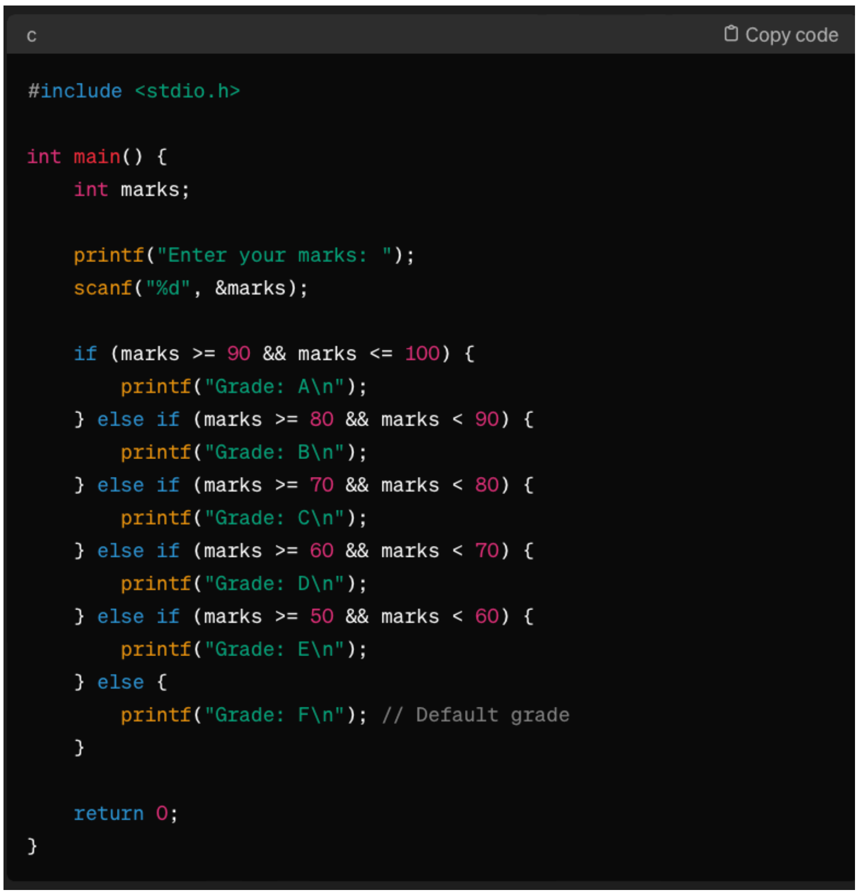
The else-if ladder is a series of if-else statements where each condition is checked only if the previous condition(s) in the ladder is false. It provides a structured way to handle multiple conditions in a program.

Flowchart for else-if Ladder:



```
Start
|
|--> Check Condition 1
|        |
|        |--Yes--> Execute Code 1
|        |
|        |--No--> Check Condition 2
|                  |
|                  |--Yes--> Execute Code 2
|                  |
|                  |--No--> Check Condition 3
|                           |
|                           |--Yes--> Execute Code 3
|                           |
|                           |--No--> Execute Default Code
|
End
```

Program Example of else-if Ladder:



```c
#include <stdio.h>

int main() {
    int marks;

    printf("Enter your marks: ");
    scanf("%d", &marks);

    if (marks >= 90 && marks <= 100) {
        printf("Grade: A\n");
    } else if (marks >= 80 && marks < 90) {
        printf("Grade: B\n");
    } else if (marks >= 70 && marks < 80) {
        printf("Grade: C\n");
    } else if (marks >= 60 && marks < 70) {
        printf("Grade: D\n");
    } else if (marks >= 50 && marks < 60) {
        printf("Grade: E\n");
    } else {
        printf("Grade: F\n"); // Default grade
    }

    return 0;
}
```

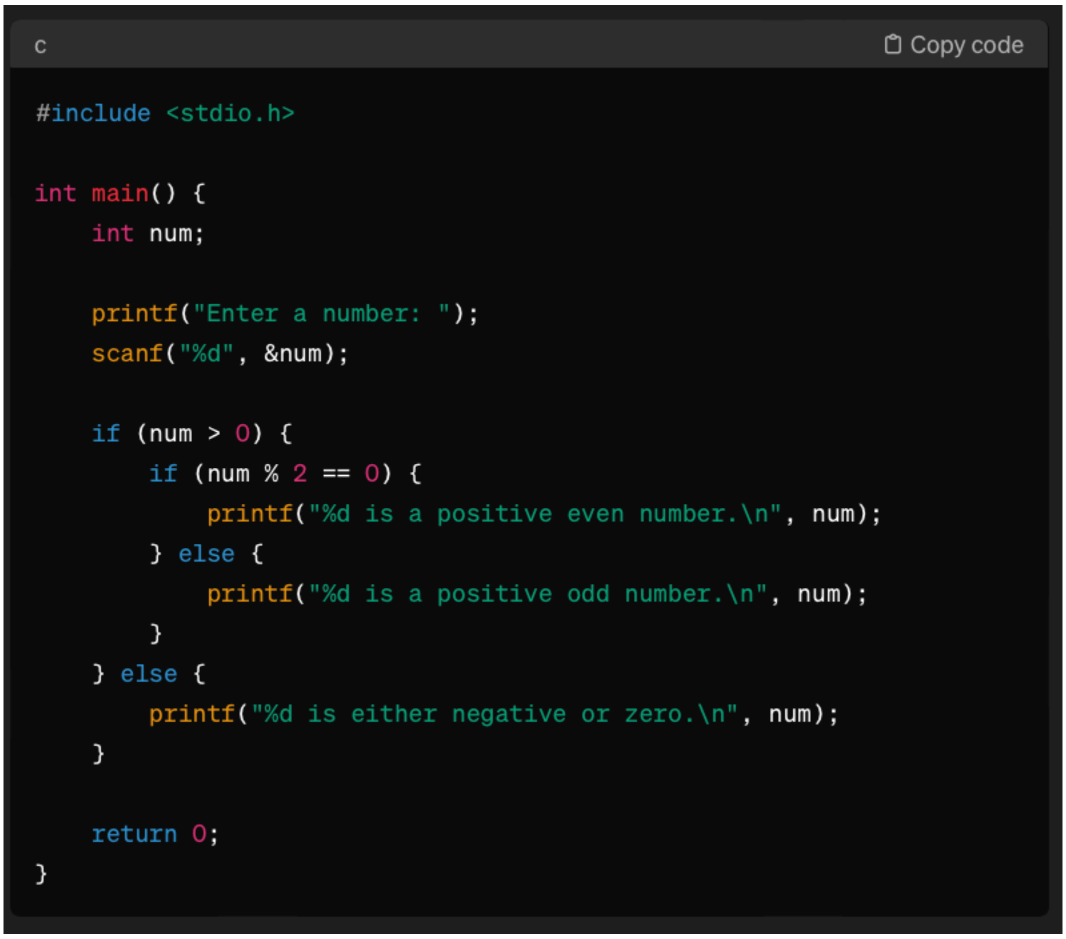
Explanation:

- The program takes input as marks and determines the grade based on the range of marks using the else-if ladder.

- It checks each condition one by one until a true condition is found or the default condition is executed if none of the conditions match.

(b) Explanation of Nested if-else with Example:

Nested if-else statements are if-else statements inside another if or else block. This allows for more complex decision-making within a program.



```c
#include <stdio.h>

int main() {
    int num;

    printf("Enter a number: ");
    scanf("%d", &num);

    if (num > 0) {
        if (num % 2 == 0) {
            printf("%d is a positive even number.\n", num);
        } else {
            printf("%d is a positive odd number.\n", num);
        }
    } else {
        printf("%d is either negative or zero.\n", num);
    }

    return 0;
}
```

Explanation:

- In this example, we have a nested if-else statement.

- The outer if-else statement checks if the number is positive or negative.

- If the number is positive, the inner if-else statement checks if it's even or odd.

- Based on the conditions, appropriate messages are printed.

UNIT –III : Functions Short Answer (2 mark) Questions

1. **What is a function? Write the types of functions.**

Ans:-

- A function is a self-contained block of code that performs a specific task. It is designed to carry out a particular operation, and it can be called from other parts of the program. Functions enhance code modularity, reusability, and maintainability.

- Types of functions:

1. Built-in functions: These are predefined functions provided by the programming language or standard libraries, such as printf() and scanf() in C.

2. User-defined functions: These are functions created by the programmer to perform specific tasks within the program.

2. **What is meant by call-by value and call-by reference?**

Ans :-

- Call-by-value: In call-by-value, the actual value of the argument is passed to the function parameter. Any changes made to the parameter inside the function do not affect the original argument outside the function.

- Call-by-reference: In call-by-reference, the memory address (reference) of the argument is passed to the function parameter. Any changes made to the parameter inside the function directly affect the original argument outside the function.

3. **What is recursion?**

Ans :-

- Recursion is the process in which a function calls itself directly or indirectly. It is a programming technique used to solve problems by breaking them down into smaller instances of the same problem. Recursion involves two main components: a base case that defines when the recursion should stop, and a recursive case that breaks down the problem into smaller subproblems.

4. **Write and explain the syntax of function?**

Ans :-

- Syntax of a function:

```
return_type function_name(parameters) {
   // Function body
   // Statements
   return value; // Optional return statement
}
```

- return_type: Specifies the data type of the value returned by the function
- function_name: Name of the function, which is used to call the function.
- parameters: List of variables passed to the function (optional).
- Function body: Contains statements that define what the function does.
- return statement: Optional statement used to return a value from the function to the calling code.

5. **What is #include, #define directives.**

Ans :-

- #include directive: It is a preprocessor directive used to include the contents of another file into the current file. It allows you to use functions, macros, and other declarations defined in the included file.

- #define directive: It is a preprocessor directive used to define constants or macros. It associates an identifier with a value or a piece of code, which can be used throughout the program to represent that value or code.

Essay Answer (10 mark) Questions

.

1. **(a) What are the advantages of functions?**

   (b) Write a C program using function to exchange two numbers using pointers.

Ans :-

(a) Advantages of Functions:

1. Modularity: Functions allow breaking down a program into smaller, manageable, and modular pieces. Each function performs a specific task, making the code easier to understand, debug, and maintain.

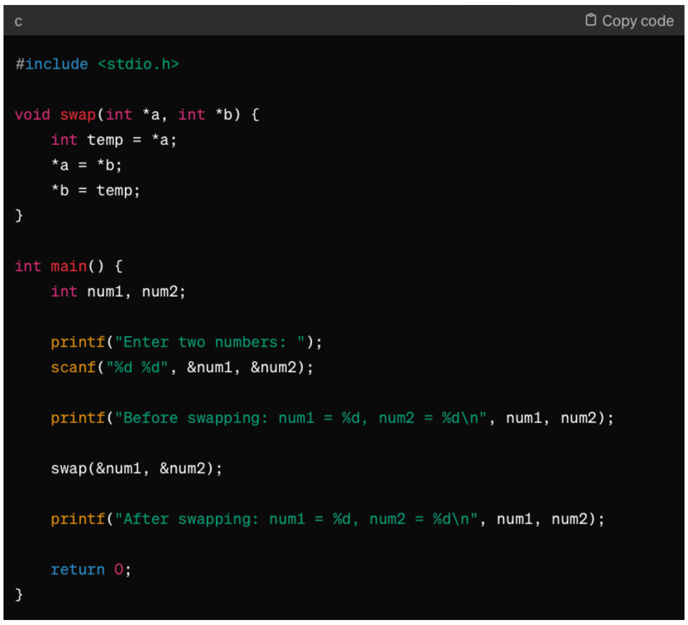
2. Code Reusability: Functions can be reused in multiple parts of a program or even in different programs altogether. This saves time and effort by eliminating the need to rewrite the same code.

3. Abstraction: Functions provide an abstraction layer by hiding the implementation details from the calling code. Users only need to know what a function does and how to use it, without needing to understand its internal workings.

4. Ease of Debugging: Since functions are modular and focused, it becomes easier to isolate and debug errors or issues within a specific function without affecting the rest of the program.

5. Teamwork: Functions facilitate teamwork by allowing different developers to work on different functions simultaneously. It promotes code collaboration and improves productivity.

(b) C Program to Exchange Two Numbers Using Pointers:



```c
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int num1, num2;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);

    swap(&num1, &num2);

    printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);

    return 0;
}
```

2. (a) Discuss about the different categories of functions.

(b) Write a C program to illustrate call-by-value parameter passing technique.

Ans :-

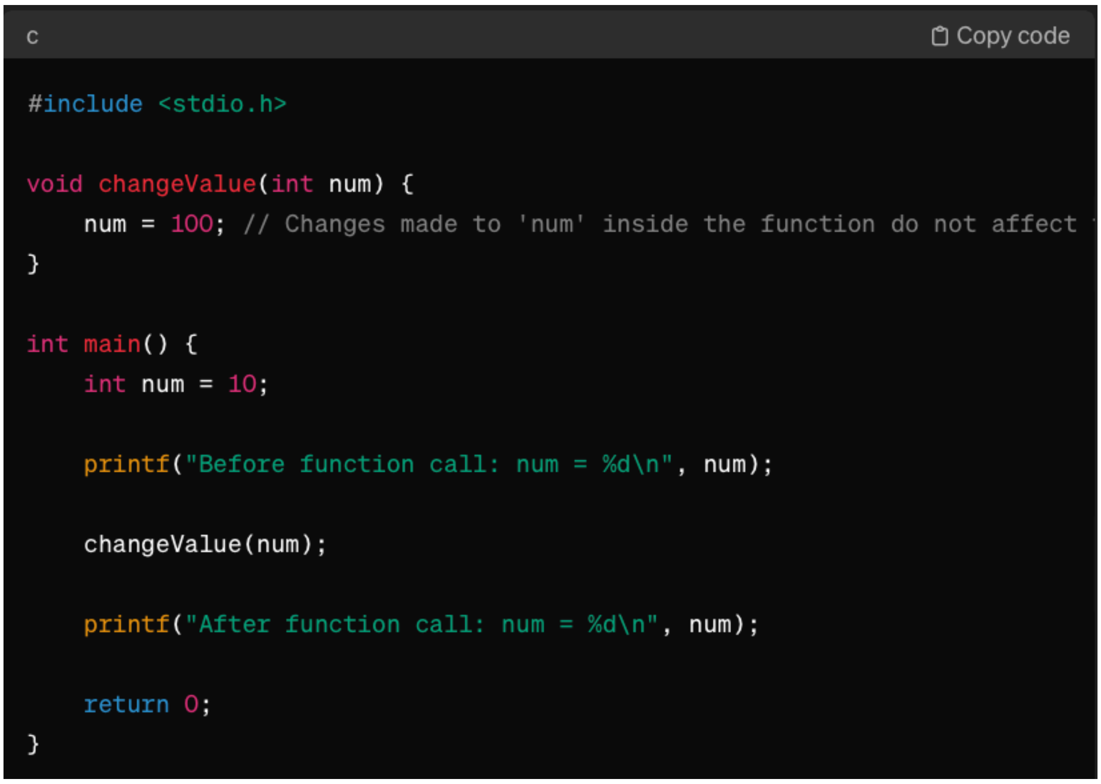
(a) Different Categories of Functions:

1. Built-in Functions: These are predefined functions provided by the programming language or standard libraries, such as printf() and scanf() in C.

2. User-defined Functions: These are functions created by the programmer to perform specific tasks within the program. They enhance code modularity and reusability.

3. Library Functions: These are functions provided by external libraries or header files. They offer additional functionalities beyond the standard library functions and can be included in the program using the #include directive.

(b) C Program to Illustrate Call-by-Value Parameter Passing:



```c
#include <stdio.h>

void changeValue(int num) {
    num = 100; // Changes made to 'num' inside the function do not affect
}

int main() {
    int num = 10;

    printf("Before function call: num = %d\n", num);

    changeValue(num);

    printf("After function call: num = %d\n", num);

    return 0;
}
```

3. **(a) Write short notes on nested functions.**

(b) Write a C program to explain call-by-reference parameter passing technique.

Ans :-

(a) Nested Functions:

- Nested functions are functions defined within another function.

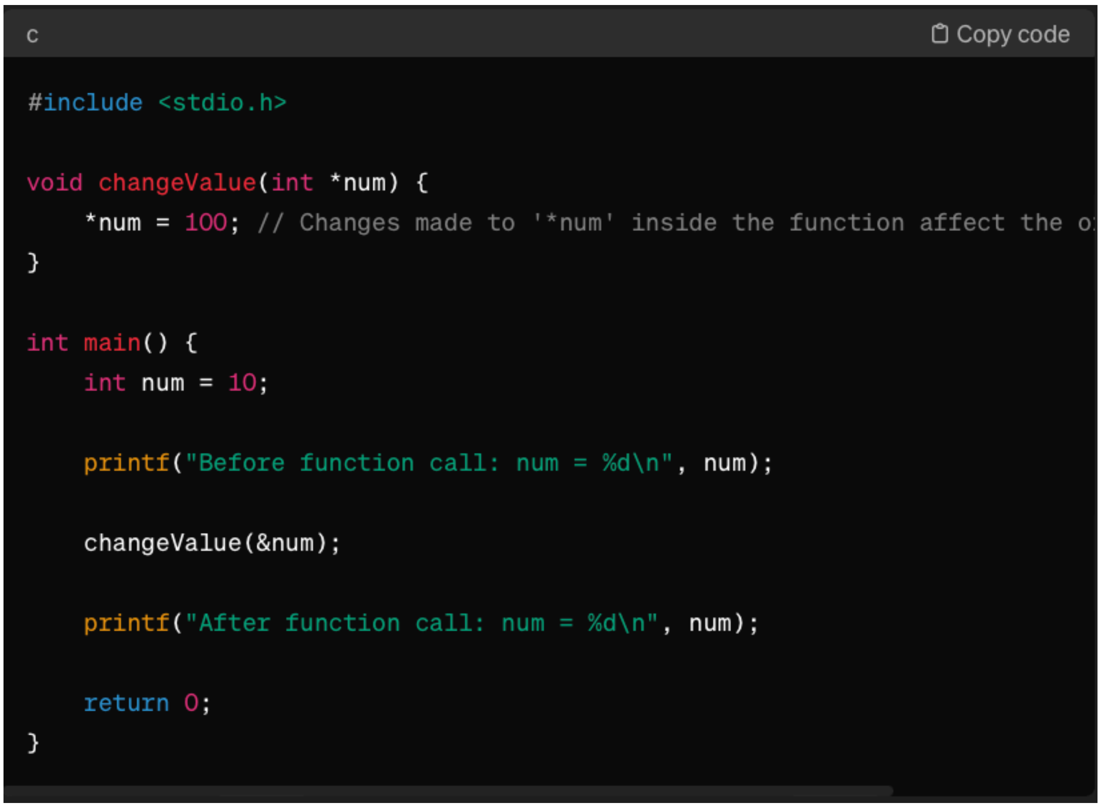
- They are scoped to the enclosing function and cannot be accessed or called from outside the enclosing function.

- Nested functions have access to variables in the enclosing function's scope, including local variables and parameters.

- They promote encapsulation and help in organizing code by grouping related functionalities together.

- Nested functions are not supported in standard C, but they are available in some other programming languages like C++ and Python.

(b) C Program to Explain Call-by-Reference Parameter Passing:



```c
#include <stdio.h>

void changeValue(int *num) {
    *num = 100; // Changes made to '*num' inside the function affect the o
}

int main() {
    int num = 10;

    printf("Before function call: num = %d\n", num);

    changeValue(&num);

    printf("After function call: num = %d\n", num);

    return 0;
}
```

4. (a) What is recursion? What are the advantages and Disadvantages of recursion?

(b) Write a C program to find the factorial of a given number using recursion.

Ans :-

(a) Recursion:

- Recursion is a programming technique where a function calls itself directly or indirectly to solve a problem.

- Advantages of Recursion:

1. Simplifies code and problem-solving by breaking complex problems into smaller, more manageable subproblems.

2. Offers an elegant and concise solution for problems that exhibit repetitive or self-similar structures.

3. Facilitates the implementation of algorithms like tree traversal, sorting, and searching.

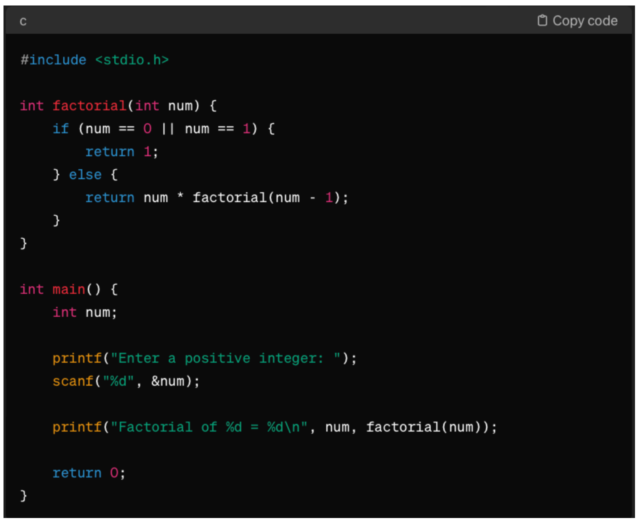
- Disadvantages of Recursion:

1. Consumes more memory and stack space compared to iterative solutions, especially for deep recursive calls.

2. May result in slower execution speed due to the overhead of function calls and stack management.

3. Recursion can be difficult to understand and debug, leading to potential logical errors and infinite loops.

(b) C Program to Find the Factorial of a Given Number Using Recursion:



```c
#include <stdio.h>

int factorial(int num) {
    if (num == 0 || num == 1) {
        return 1;
    } else {
        return num * factorial(num - 1);
    }
}

int main() {
    int num;

    printf("Enter a positive integer: ");
    scanf("%d", &num);

    printf("Factorial of %d = %d\n", num, factorial(num));

    return 0;
}
```

5. **Distinguish between the following:**

    a. **Actual and formal arguments**
    b. **Global and local variables**
    c. **Automatic and static variables**

Ans :-

a. Actual and Formal Arguments:

- Actual arguments are the values passed to a function when it is called. They represent the data that the function will operate on.

- Formal arguments are the parameters declared in the function definition to receive the values of the actual arguments. They act as placeholders for the data provided during the function call.

b. Global and Local Variables:

- Global variables are declared outside of any function and can be accessed and modified by any function within the program. They have a global scope and lifetime, meaning they exist throughout the execution of the program.

- Local variables are declared within a function and can only be accessed and modified within that function's scope. They have a local scope and lifetime, meaning they are created when the function is called and destroyed when the function exits.

c. Automatic and Static Variables:

- Automatic variables are declared within a function without the static keyword. They are created when the function is called and destroyed when the function exits. They have a local scope and are reinitialized every time the function is called.

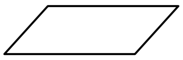
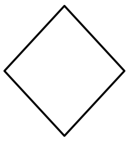
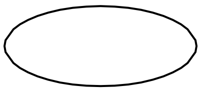
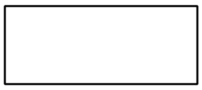
- Static variables are declared with the static keyword within a function. They retain their value between function calls and have a local scope limited to the function in which they are declared. Static variables are initialized only once and persist throughout the program's execution.

UNIT-1 : FUNDAMENTALS OF **C-PROGRAMMING(MCQ)**

1. Which of the following is used to perform computations on the entered data?
   (A) Memory **(B) Processor** (C) Input device (D) Output device
2. Which of the following is not an input device?
   (A) Plotter (B) Scanner (C) Keyboard (D) Mouse
3. Which of the following is not an output device?
   (a) Plotter **(b) Scanner** (c) Printer (d) Speaker
4. Which of the following is used as a primary memory of the computer?
   (a) Magnetic storage device **(b) RAM**
   (c) Optical storage device (d) Magneto-optical storage device
5. Which one of the following is a volatile memory?
   (a) **RAM** (b) Auxiliary memory (c) ROM (d) Secondary memory

2. Software is defined as
   (a) Set of memory cells **(b) Set of Programs** (c) Set of hardware (d) None

7. Which statement is a valid?
   (a) **1KB=1024 bytes** (b) 1 MB=2018 bytes
   (c) 1 MB=10000 kilobytes (d) i KB=100 bytes
8. symbol is used for input/output in flowchart

(a)  (b)  (c)  (d) 

9. Which of the following is a pictorial representation of an algorithm?
   (a) Program **(b) Flowchart** (c) Algorithm (d) Pseudo code
10. Among the following, which converts assembly language into machine language (a) Interpreter (b) Compiler **(c) Assembler** (d) Algorithm
11. Which one of the following is known as the ―language of computer‖?
    (a) Programming language (b) High-level language
    (c) Machine language (d) Assembly language
12. translates high level language into machine language
    (a) Compiler (b) Translator (c) Processor (d) Loader
13. Which of the following is not a valid variable declaration
    (a) int 2class; (b) int class2; (c) int class_2; (d) int ELSE;

14. The range of ―unsigned int‖ data type is___
    (a) -32768 to 32767 **(b) 0 to 65535** (c) -65536 to 65535 (d) -128 to 127
15. The size of ―long double‖ data type in 16-bit machine is
    (a) 8 bytes **(b) 10 bytes** (c) 2 bytes (d) 4bytes
16. The range of ―char‖ data type is ______
    (a) **-128 to 127** (b) 0 to 255 (c) -32768 to 32767 (d) None

17. The size of ―char‖ data type is _
(a) 1 byte (b) 2 bytes (c) 4 bytes (d) 10 bytes
18. The format specifier that is used to read or write a character is _______
(a) %f (b) %d **(c) %c** (d) %s
19. Which one of the following is a string constant
(a) ‗3' **(b) "hello"** (c) 30 (d) None
20. If no precision is specified for floating point number then printf() prints ______ decimal positions.
(a) Two (b)Four **(c) Six** (d) Zero
21. What is the result of 8 | 4 ?
(a) 0 (b) 1 (c) 4 **(d) 12**
22. Which of the following operator is used to combine two or more relational expressions
(a) ^ (b) ~ (c) & **(d) &&**
23. ~(100111) gives ______
(a) 010010 **(b) 011000** (c) 010100 (d) 111001
24. 10<<3 gives _____
(a) 40 (b) 1 **(c) 80** (d) 30
25. Shifting a number ‗n' by ‗s' bits to left is equivalent to which of the following?
(a) $2^s/n$ (b) $n/2^s$ (c) s^2/n **(d) $n*2^s$**
26. Shifting a number ‗n' by ‗s' bits to right is equivalent to which of the following?
(a) $2^s/n$ **(b) $n/2^s$** (c) s^2/n (d) $n*2^s$
27. Based on the precedence levels and associativity the 8+4*5+6/2 expression yields

(a) 43 (b) 34 **(c) 31** (d) 41
28. ______operators are used for shifting bits to right and left
(a) **>> and <<** (b) > and < (c) ?and : (d) None
29. The expression a++ is referred as
(a) Pre increment **(b) Post increment** (c) Before increment (d) After increment 30. The expression ++a referred as
(a) **Pre increment** (b) Post increment (c) Before increment (d) After increment
31. If a=3, b=5 the value of the expression ++a+b++ is_____
(a) 10 **(b) 9** (c) 8 (d) None of the above
32. ___defines the order of evaluation when operators have the same precedence
(a) Priority (b) Precedence **(c) Associativity** (d) None of the above
33. Which one of the following is having highest precedence
(a) ++ (b) && **(c) ( )** (d) ,
34. Which one of the following is having least precedence
(a) ++ (b) && (c) ( ) **(d) ,**
35. String constants are enclosed in

(a) ‗ ‗ **(b) " "** (c) ( ) (d) [ ]
36. Character constants are enclosed in

(a) „ „ (b) — — (c) ( ) (d) [ ]

37. The escape sequence character __causes the cursor to move to the next line on the screen
    (a) \t **(b) \n** (c) \r (d) \v
38. The assignment statement ―sum=sum+i;‖ is equivalent to
    (A) sum=+i; **(B) sum+=i;** (C) sum= =sum+i; (D) None
39. sizeof() operator returns the size of an operand in ______
    (A) Bits (B) Nibble **(C) Bytes** (D) None
40. Which of the following is the correct way of using type casting **(A) c=(int)a/b;** (B) c=a(int)/b; (C) c=int a/b; (D) None

<u>UNIT-2 : DECISION & LOOP CONTROL STATEMENTS</u>

1. Which of the following is not a loop structure?
   (a) for (b) do-while **(c) repeat-until** (d) while
2. If statement is a ――――――statement
   (a) One-way decision (b) Multi-way decision **(c) Two way decision** (d) Loop construct
3. ‗break' statement in a loop is used for
   (a) **Terminating the loop** (b) De-allocating memory
   (c) Terminating the program (d) Terminating the function
4. The keyword ―else‖ can be used with
   (a) for statement (b) do.. while ( ) statement **(c) if statement** (d) switch ( ) statement
5. The two different ways to implement a multiway selection in C are
   (a) Simple if and if-else (b) if-else and nested if-else
   (c) else-if ladder and switch (d) None
6. The minimum number of time that a do-while loop executes
   (a) 0 **(b) 1** (c) infinitely (d) variable

3. The while loop is terminated when the conditional expression returns

(a) 1 (b) 2 (c) 3 **(d) Zero**

8. C provides __________as a convenient alternative to the traditional if-else for two way selection.

(a) **Conditional operator** (b) Short hand assignment (c) Increment (d) None

9. The statement used to send back any value to the calling function is

(a) break (b) continue (c) exit **(d) return**

10. The______statement is used to skip the remaining part of the statements in a loop and continue with next iteration.

(a) break (b) goto **(c) continue** (d) exit

11. should be avoided as part of structured programming approach

(a) break **(b) goto** (c) continue (d) exit

12. The minimum number of times ―for‖ loop executes

(a) 2 (b) can‗t be predicted **(c) 0** (d) 1

13. What will be output when you will execute following c code?

```
void main()
{  int fruit=1; switch(fruit+2)
    { default:printf("apple"); case 4:
            printf(" banana");
         case 5: printf(" orange");
         case 8: printf(" grape");
    }
}
```

(a) **applebanana orange grape** (b) grape (c) orange (d) banana orange grape

14. Which for loop has range of similar indexes of 'i' used in for (i = 0;i< n; i++)?

(a) for (i = n; i>0; i–) (b) for (i = n; i >= 0; i–)

(c) for (i = n-1; i>0; i–) **(d) for (i = n-1; i>-1; i–)**

15. What will be output when you will execute following C code?

```
void main()
{ int check=2;
        switch(check)
        { case 2: printf("1");
           break; case 3:
           printf(" 2"); break;
        }
}
```

(a) 12 (b) 2 **(c) 1** (d) Compilation error

16. Which one among the following is the correct syntax of for loop?

(a) for(i=0;i<n;i++); (b) for(i<n;i=0;i++);

(c) for(i=0;i<n:i++)**;** (d) None

17. ‗for‗ loop in C program , if the condition is missing

(a) assumed to be present and taken to be false

(b) **assumed to be present and taken to be true** (c) syntax error

(d) execution will be terminated abruptly

18. if c is initialized to 1, how many times following loop is executed

```
While((c>0)&&(c<60))
{          c++; }
```

(a) 60 **(b) 59** (c) 61 (d)1

19. The library function exit () causes an exit from

(a) loop (b) block (c)function **(d) None**

20. break statement can use with

i) loop ii)switch iii) block

(a) onlyi,ii (b) only ii,iii (c) only i, iii (d) All

21. What is the output of this C code?

```
int main()
   { while () printf("In
while loop ");
printf("After loop\n");
   }
```

(a) In while loop after loop (b) After loop (**c) Compile time error**(d) Infinite loop

22. Which among the following is not checked in switch case

(a) character (b) integer **(c) float** (d) None

23. What is the output of the following program main()

```
        {
                int i;
                for(i=1;i<5;i++)
                {        if(i==3)
                                break;
                        Printf(—%d‖,i);
                }
        }
```

(a) 12345 (b)124 (c)1245 **(d)12**

24. What is the output of the following program main()

```
        {        int i;
                for(i=1;i<5;i++)
                { if(i==3) continue;
                        Printf(—%d‖,i);
                }
        }
```

(a) 12345 **(b)124** (c)1245 (d)12

25. What are the entry controlled loops among the following

i. while ii. Do-while iii. For

(a) only i (b) only ii,iii (c) only iii **(d) only i, iii**

26. What is the output of the following program? main()

```
{ int i=1; while(i<=5)
        printf(―%d‖,i);


}
```

(a) 12345 (b)1234 (c) 2345 **(d) Leads to infinite loop**

27. for(;;) can be terminated by

(a) break (b) exit(0) (c) return **(d) All the above**

28. What is the output of the following program main()

```
{ for(i=1;i<=5;i++);
        printf(―%d‖,i);
}
```

(a) 12345 (b)1234 **(c) 6** (d) leads to infinite loop

29. What is the correct syntax of for loop

(a) for(i=0;i<n;i++){ } (b) for(i<n;i=0;i++){ }

(c) for(i=0;i<n:i++){} (d) for(i=0:i<n:i++){ }

30. Array is an example of which of the following?

(a) **Derived types** (b) Fundamental types (c) User-defined types (d) None

31. Which of the following is used to display a string on the screen?

(a) %s (b) %c (c) %d (d) %f

32. What is the final value of x when the code int x; for(x=0; x<10; x++) {} is run?

(a) 10 (b) 9 (c) 0 (d) 1

33. Which of the following is exit controlled loop

(a) for (b) while **(c) do-while** (d) None

34. The default statement is executed when

(a) **All the case statements are false** (b) One of the case is true

(c) One of the case is false (d) None

35. How many times the following C code prints ―Hello‖ int main()

```
   {   while   (1)
printf("Hello
");
   }
```

(a) One (b) zero **(c) Infinite** (d) Produce error

36. How many times the following C code prints ―Hello‖ int main()

```
   { do
      { printf("Hello
");
}while(0);
}
```

(a) **One** (b) zero (c) Infinite (d) Produce error

37. How many bytes the array **price** occupies. float price[10];

(a) 10 bytes (b) 4 bytes **(c) 40 bytes** (d) 20 bytes

38. Which of the following is syntactically correct?

(a) for(); (b) for(;); (c) for(,); **(d) for(;;);**

39. What is the output of the following code main()

```
{ int a= 0,b = 20; char x
        =1,y =10;
        if(a,b,x,y)
                printf("hello");
}
```

(a) Syntax error **(b) hello** (c) 10 (d) None

40. is used to terminate from the entire program (a) return (b) break **(c) exit** (d) goto

UNIT-3 : Arrays and Functions

1. Array is an example of which of the following?

(a) Derived types (b) fundamental types (c) user-defined types (d) None

2. Array elements are stored in

(a) Scattered memory locations (b) **Sequential memory locations**

(c) Direct memory locations (d) None

3. int a[10] will reserve how many locations in the memory?

(a) 10 (b) 9 (c) 11 (d) None of the above

4. Which one of the following is the correct syntax for initialization of one-dimensional arrays?

(a) int num[3]={0 0 0}; **(b) int num[3]={0,0,0};**

(c) int num[3]={0;0;0}; (d) int num[3]=0;

5. Under which of the following conditions, the size of the array need not be specified? (a) When the compiler is smart **(b) When initialization is a part of definition** (c) Both (d) None

6. Which of following is correct array declaration

A) int num(25); B) int array num[25]; **C) int num[25];** D) num[25];

7. Array subscripts in ‗C‘ starts from

A) 0 B) compiler dependent C) 1 D) -1

8. Array elements are stored in

A) Column major order B) in diagonal order

C) Row major order D) either in row major or column major order

9. Which of the following statements is used to read a string of characters into the array **words**?

A) scanf(―%d‖, words); B) scanf(‖% \n‖, words);

C) scanf("%s", words); D) scanf(― %c‖, words);

10. A string constant is one dimensional array of characters terminated by a

A) Comma B) Full stop C) Semicolon **D) Null character („\0‟)**

11. Which of the following multi-dimensional array declaration is correct for realizing a 2 X 3 matrix

(a) **int m[2][3];** (b) int m[3][2]; (c) int m[3],m[2]; (d) None

12. Which of the following is the correct syntax for initialization of two-dimensional arrays?

(a) **table[2][3]={0,0,0,1,1,1}** (b) table[2][3]={{0,0,0,}{1,1,1}}

(c) table[2][3]={0,1},{0,1}{0,1} (d) None

13. What will be assigned for marks[3] and marks[4] in the following initialization int marks[5]={30,45,80};

(a) 80 and garbage (b) garbage and garbage **(c) 0 and 0** (d) None

14. Which of the following is correct initialization of string TITAN

(a) char name[ ]=‖TITAN\0‖ (b) char name[10]=‖TITAN\0‖

(c) char name[ ]="TITAN" (d) char name[10]={―TITAN‖}

15. Which of the following initialization is wrong

(a) x[5]=15 **(b) x[10.3]=30** (c) x[0]=20 (d) None

16. char ch[ ]={‗a',‗b',‗c',‗\0'}; int sum=ch[1]+ch[2];

What is the value of sum?

(a) 195 **(b) 197** (c) ab (d) error

17. What happens if we initialize an array as int group[20]={0};

(a) Produce an error (b) Only 0th element is initialized with zero

(c) Every element is initialized with zero (d) None

18. To store a table of values which of the following is used

(a) One dimensional array **(b) Two dimensional array**

(c) Three dimensional array (d) None

19. int rank[3]={3,2,4,1,5};

(a) **Compile time error** (b) Initializes only 3 elements with first 3 values

(c) Initializes only 3 elements with last 3 values (d) Initialize all elements with zeros

20. How to refer an element in ith row jth column of a two dimensional array

(a) x[i,j] **(b) x[i][j]** (c) x[ij] (d) x[i]x[j]

21. A function can be called in a program

A. Only two times B. Only once **C. Any number of times** D. Only three times

22. When you pass an array as an argument to a function, what actually gets passed

A. **Address of the array** B. Values of the elements of the array

C. Number of elements of the array D. None

23. The statement used to send back any value to the calling function is

A. break B. continue C. exit **D. return**

24. The function sqrt( ) is part of header file.

A. conio.h B. stdio.h **C. math.h** D. iostream.h

25. A function can return only____value

A. Zero **B. One** C. two D. three

26. Actual and formal parameters must agree in

A. Data types **B. Number of arguments and Data types**

C. Names and Data type D. None

27. Any function can be called from any other function. This statement is A. **True sometimes** B. Neither true nor false C. False D. True

28. The header file that must be included at the beginning of a C program to use a library function cos() is

A. stdlib.h B. conio.h C. dos.h **D. math.h**

29. function is said to be function calling itself.

A. Call by reference B. Call by value **C. Recursive** D. All above

30. void funct (void);

The above function declaration indicates A. it returns

(a) value and had arguments B. it returns nothing and had arguments

C. it returns a value and no arguments **D. it returns nothing and no arguments**

31. The parameters of the called function(function definition) are called A. Casual parameters **B. formal parameters** C. usual parameters D. actual parameter

32. Recursion means **A. Function calling same function** B. Function calling a function

C. Both D. None

33. A function is one that returns no value has___return type **A. Void** B. Integer C. Float D. Recursive

34. The parameters in a function call are

A. Real parameters B. Formal parameters **C. Actual parameters** D. Dummy parameters

35. Based on arguments and return types, functions are classified into

A. 1 type B. 2 types C. 3 types **D. 4 types**

36. Maximum number of arguments can be passed to a function are

A. 2 B. 3 C. 4 **D. Any**

37. The default parameter passing mechanism is

(a) **Call by value** (b) Call by reference (c) Call by name (d) None

38. Any C program _____

(a) **Must contain at least one function** (b) need not contain any function

(c) Needs input data (d) None

39. Call by reference is also known as

(a) **Call by address or Call by location** (b) Call by address or Call by value

(c) Call by value or Call by name (d) None

40. Determine output:

```
main()
{ int i=abc(10);
        printf(—%d‖,--
        i);
}
int abc(int i)
{       return(i++);    }
```

(a) 10 **(b) 9** (c) 11 (d) None