2 Marks

1. Explain the structure of a C program and describe the role of each section (e.g., header files, main function, etc.) in program execution.

Structure of a C Program

- **Header Files:**
  - Included at the beginning of the program using #include directive.
  - Provide access to predefined functions and data types (e.g., stdio.h for standard input/output, math.h for mathematical functions).
- **Main Function:**
  - The entry point of the program.
  - Execution always begins within the main() function.
  - May return an integer value (typically 0 for successful execution).
- **Variables:**
  - Used to store data (e.g., integers, characters, floating-point numbers).
  - Declared within the program using appropriate data types.
- **Statements:**
  - Instructions that perform actions (e.g., assignment, arithmetic operations, control flow statements).
- **Functions:**
  - Reusable blocks of code that perform specific tasks.
  - Can be defined within the program or declared in header files.

2. Demonstrate timeout mechanism crucial in embedded systems? How do loop timeouts aid in managing the flow of an embedded program and ensuring its operational efficiency?

Timeout Mechanism in Embedded Systems

- **Crucial for:**
    - Preventing indefinite waiting for events that may not occur.
    - Ensuring timely responses and preventing system hangs.
    - Managing resource allocation efficiently.
- **Loop Timeouts:**
    - Implement a counter within a loop.
    - Increment the counter on each iteration.
    - If the counter exceeds a predefined threshold, exit the loop.
    - Example: Waiting for a sensor reading while monitoring a timeout counter.

3.Construct a C program to check if a given number is prime or not using control structures.

```
#include <stdio.h>
#include<conio.h>
int main() {
   int num, i, isPrime = 1;
    printf("Enter an integer: ");
   scanf("%d", &num);
        if (num <= 1) {
                isPrime = 0;
         } else {
                for (i = 2; i <= num / 2; ++i) {
                      if (num % i == 0) {
                              isPrime = 0;
                             break;
                       }
                }
```

```
        }
        if (isPrime)
                printf("%d is a prime number.\n", num);
        }else{
           printf("%d is not a prime number.\n", num);
        }
         getch();
}
```

4. Write a C program to demonstrate the use of nested loops by printing a pyramid pattern of stars.

```
#include <stdio.h>
#include<conio.h>
int main() {
    int rows, i, j;

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

    for (i = 1; i <= rows; ++i) {
        for (j = 1; j <= rows - i; ++j) {
            printf(" ");
        }
        for (j = 1; j <= 2 * i - 1; ++j) {
            printf("*");
        }
        printf("\n");
    }
```

```
    return 0;
}
```

5. Build a C program that demonstrates the use of structures to organize information about a student, including their name, roll number, and marks.

```
#include <stdio.h>
#include <conio.h>
struct Student {
    char name[50];
    int rollNo;
    float marks;
};

int main() {
    struct Student student;

    printf("Enter student name: ");
    scanf("%s", student.name);

    printf("Enter roll number: ");
    scanf("%d", &student.rollNo);

    printf("Enter marks: ");
    scanf("%f", &student.marks);

    printf("\nStudent Information:\n");
    printf("Name: %s\n", student.name);
    printf("Roll No: %d\n", student.rollNo);
```

```
    printf("Marks: %.2f\n", student.marks);


    return 0;
}
```

6. Extend the importance of header files in structuring embedded C code and how they simplify large-scale programming projects.

Importance of Header Files in Embedded C

- **Modularity:** Break down code into smaller, reusable units.
- **Code Reusability:** Define functions and data structures once, use them throughout the project.
- **Improved Readability:** Enhance code organization and maintainability.
- **Reduced Errors:** Centralized declarations minimize inconsistencies.

7. Write a program that demonstrates how to implement a time delay using 8051 timers for 500 milliseconds.

8051 Timer for 500ms Delay (Conceptual)

- **Timer Configuration:**
    - Select appropriate timer mode (e.g., Timer 0, Mode 1).
    - Calculate the required timer count for 500ms delay based on system clock frequency.
- **Timer Interrupt:**
    - Generate an interrupt after the desired delay.
    - In the interrupt service routine, perform the required actions.

8. Construct a program to configure Port 0 of the 8051 as an output port and blink an LED with a 1-second delay.

8051 Blink LED (Conceptual)

- **Port Configuration:**

    - Configure Port 0 as output using appropriate special function registers (SFRs).
- **Delay Loop:**
    - Implement a delay loop to create the 1-second interval.
- **Toggle LED:**
    - Invert the output on Port 0 to toggle the LED state.

9. Build an Embedded C program to perform the logical NOT operation on an 8-bit input data and display the result using the 8051's I/O ports.

8051 Logical NOT Operation (Conceptual)

- **Read Input:**
    - Read the 8-bit input data from an input port.
- **Perform NOT Operation:**
    - Invert each bit of the input data using bitwise NOT operator (~).
- **Write Output:**
    - Write the inverted data to an output port.

10. Explain how logic operations (AND, OR, XOR) are used in 8051 and write a program to toggle a bit in Port 1 using an AND operation.

🎬 **Logic Operations:**

- AND, OR, XOR are used for bit-level manipulation.
- Example: To set a specific bit, perform a bitwise OR with a mask containing 1 at the desired bit position.

🎬 **Toggle Bit Using AND:**

- Create a mask with 0 at the bit position to be toggled and 1s elsewhere.
- Perform bitwise AND with the input data to clear the desired bit.
- Perform bitwise XOR with the mask to invert the cleared bit.