Sudarshan Maghade

1). Embedded C

Q1.1) Write an Embedded C program to multiply any number by 9 in the fastest manner.

Answer -

```
#include <stdio.h>


// Function to multiply a number by 9 using bitwise operations

int multiply(int num)
 {
    return (num << 3) + num;  // Equivalent to (num * 8) + num
}

int main()
 {
    int number;

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

      int result = multiply(number);

      printf("%d * 9 = %d\n", number, result);

    return 0;
}
```

Q1.2) Write a program to print numbers from 1 to 1000 without using conditional operators.

```
#include <stdio.h>


void printNumbers(int n) {
```

```
    printf("%d\n", n);
    if (n < 1000) {     1000
        printNumbers(n + 1);
    }
}

int main() {
    printNumbers(1);
    return 0;
}
```

Q1.3) Write a MIN macro program that takes two arguments and returns the smallest of both arguments.

```
#include <stdio.h>

// Macro to find the minimum of two numbers
#define MIN(a, b) ((a) < (b) ? (a) : (b))

int main()
{
    int x = 10, y = 20;

        printf("Minimum of %d and %d is %d\n", x, y, MIN(x, y));

    return 0;
}
```

Q2.1) Write Program to Implement a Stack The program creates a stack and allows users to perform push, pop, and display operations on it.

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100 // Maximum size of the stack
```

```
// Stack structure
typedef struct
{
    int items[MAX_SIZE];
    int top;
} Stack;

// Function to initialize the stack
void initialize(Stack *s)
 {
    s->top = -1; // Stack is empty initially
}

// Function to check if the stack is full
int isFull(Stack *s)
{
    return s->top == MAX_SIZE - 1;
}

// Function to check if the stack is empty
int isEmpty(Stack *s)
{
    return s->top == -1;
}

// Function to push an element onto the stack
void push(Stack *s, int value)
{
    if (isFull(s))
 {
        printf("Stack Overflow! Cannot push %d.\n", value);
    }
else
{
```

```
        s->top++;
        s->items[s->top] = value;
        printf("%d pushed to the stack.\n", value);
    }
}

// Function to pop an element from the stack
int pop(Stack *s)
  {
    if (isEmpty(s))
  {
        printf("Stack Underflow! Cannot pop from an empty stack.\n");
        return -1; // Return -1 to indicate underflow
    }
 else
 {
        int value = s->items[s->top];
        s->top--;
        printf("%d popped from the stack.\n", value);
        return value;
    }
}

// Function to display the stack
void display(Stack *s)
{
    if (isEmpty(s))
 {
        printf("Stack is empty.\n");
    } else
 {
        printf("Stack elements:\n");
        for (int i = s->top; i >= 0; i--)
 {
            printf("%d\n", s->items[i]);
```

```
        }
    }
}

int main()
 {
    Stack s;
    initialize(&s); // Initialize the stack

    int choice, value;

    while (1)
{
        printf("\nStack Operations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
 {
            case 1:
                printf("Enter the value to push: ");
                scanf("%d", &value);
                push(&s, value);
                break;
            case 2:
                pop(&s);
                break;
            case 3:
                display(&s);
                break;
            case 4:
```

```
            printf("Exiting the program.\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
  }

  return 0;
}
```

Q2.2) Write a program to C program to convert 545 numbers from Decimal to Binary.

```
#include <stdio.h>

// Function to convert decimal to binary
void decimalToBinary(int n)
{
  int binary[32]; // Array to store binary digits
  int i = 0;

  // Convert decimal to binary
  while (n > 0)
 {
    binary[i] = n % 2; // Store the remainder
    n = n / 2;        // Divide the number by 2
    i++;
  }

  // Print binary digits in reverse order
  printf("Binary of 545 is: ");

  for (int j = i - 1; j >= 0; j--) {
    printf("%d", binary[j]);
  }
  printf("\n");
```

```
}

int main()
 {
    int decimalNumber = 545; // Decimal number to convert
    decimalToBinary(decimalNumber); // Convert to binary
    return 0;
}
```

Q2.3) Given two numbers a and b, the task is to find the GCD of the two numbers?
Input: a = 20, b = 28
Output: 4
Explanation:?

```
#include <stdio.h>

// Function to find GCD using Euclidean Algorithm
int gcd(int a, int b)
{
    while (b != 0)
 {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int main()
{
    int a = 20, b = 28;

    int result = gcd(a, b);

    printf("GCD of %d and %d is: %d\n", a, b, result);
```

```
    return 0;
}
```

Q3.1) Python | Play a video using OpenCV.(external camera)

```
import cv2

# Try opening external camera (change index if necessary)
cap = cv2.VideoCapture(1)  # 0 for default camera, 1 for external


if not cap.isOpened():

    print("Error: Could not open external camera. Try changing index (0, 1,
2...).")
    exit()

while True:
    ret, frame = cap.read()  # Read frame from camera

    if not ret:
        print("Failed to capture frame.")
        break

    cv2.imshow("External Camera", frame)  # Display the frame

    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()  # Release the camera
cv2.destroyAllWindows()  # Close all OpenCV windows
```

Q3.2) Python | Find Circles, and Ellipses in an Image using OpenCV

```
import cv2
```

```
# Open the external camera (usually camera index 0)
cap = cv2.VideoCapture(0)

# Check if the camera is opened successfully
if not cap.isOpened():
    print("Error: Could not open camera.")
    exit()

# Loop to read and display video frames
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Check if the frame was captured successfully
    if not ret:
        print("Error: Could not read frame.")
        break

    # Display the frame in a window
    cv2.imshow("Camera Feed", frame)

    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the camera and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

#4). Python:

Q4.1) Factorial Program in Python.

```
def factorial(n):

    # Base case: factorial of 0 or 1 is 1
    if n == 0 or n == 1:
```

```
        return 1
    # Recursive case: n! = n * (n-1)!
    else:
        return n * factorial(n - 1)


num = int(input("Enter a non-negative integer: "))

# Check if the input is valid

if num < 0:

    print("Error: Factorial is not defined for negative numbers.")
else:

    result = factorial(num)

    print(f"The factorial of {num} is: {result}")
```

Q4.2) Automorphic number or not. For example, 5 is an automorphic number, 5*5 =25. The last digit is 5 which same as the given number. It has only a positive single-digit number. If the number is not valid, it should display "Invalid input".

```
def is_automorphic(num):

    if num < 0 or num > 9:

        print("Invalid input")

        return

    square = num * num

    last_digit = square % 10  # Extract last digit of square
```

```
    if last_digit == num:
        print(f"{num} is an Automorphic number.")
    else:
        print(f"{num} is NOT an Automorphic number.")

# Example usage
num = int(input("Enter a single-digit positive number: "))
is_automorphic(num)
```

Q4.3) Print the following pattern:

0
0 1
0 1 2
0 1 2 3
0 1 2 3 4

```
# Loop to print the pattern
for i in range(5):  # Number of rows

    for j in range(i + 1):  # Print numbers from 0 to i

        print(j, end=" ")

    print()  # Move to next line
```

Q4.4) Write a Program to Implement a Stack The program creates a stack and allows users to perform push and pop operations on it. (implement using class)

```
class Stack:
    def __init__(self):

        self.stack = []  # Initialize an empty list to store stack elements

    def push(self, value):
```

```
    """Push an element onto the stack."""
    self.stack.append(value)
    print(f"{value} pushed onto stack.")
def pop(self):
    """Pop an element from the stack."""
    if not self.is_empty():
        popped_value = self.stack.pop()
        print(f"{popped_value} popped from stack.")
        return popped_value
    else:
        print("Stack Underflow! Cannot pop.")
def display(self):
    """Display the current stack elements."""
    if not self.is_empty():
        print("Stack elements:", *self.stack)
    else:
        print("Stack is empty!")
def is_empty(self):
```

```
        """Check if the stack is empty."""
        return len(self.stack) == 0

# Example usage
stack = Stack()
while True:
    print("\nStack Operations:")
    print("1. Push\n2. Pop\n3. Display\n4. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        value = int(input("Enter value to push: "))
        stack.push(value)
    elif choice == 2:
        stack.pop()
    elif choice == 3:
        stack.display()
    elif choice == 4:
        print("Exiting...")
```

```
        break
    else:
        print("Invalid choice! Please try again.")
```

 # 5). Linux:

Q5.1) create a new folder “test_new” in the home directory.

```
mkdir ~/test_new

ls ~/
```

Q5.2) remove all permissions of the “test_new” directory.

```
chmod 000 ~/test_new

ls -ld ~/test_new
```

Q5.3) create a “new.txt” file in the “test_new” directory.

```
touch ~/test_new/new.txt
```

To check the file is run or not

```
ls -l ~/test_new
```

Q5.4) delete the “test_new” directory.

```
rm -r ~/test_new
```

Alternative force delete

```
sudo rm -rf ~/test_new
```

 # 6) Practical Arduino:

Q6.1) write a code to operate the relay with serial terminal input.
input: **1**

Output**: Relay on**
input: **1**
Output**: Relay off**
input: **1**
Output**: Relay on**

```
#include <xc.h>   // Include for PIC microcontrollers (use appropriate header for your MCU)
#include <stdio.h>

#define RELAY_PIN LATBbits.LATB0  // Define relay control pin (for PIC, modify as needed)

void UART_Init(void);
char UART_Read(void);
void UART_Write(char data);
void UART_Write_String(const char *str);

int main()
{
    UART_Init();  // Initialize UART
    TRISBbits.TRISB0 = 0;  // Set relay pin as output
    RELAY_PIN = 0;  // Initially, relay is OFF

    UART_Write_String("Enter '1' to toggle relay ON/OFF\n");

    while (1)
    {
        char input = UART_Read();  // Read input from serial terminal

        if (input == '1')
        {
            RELAY_PIN = !RELAY_PIN;  // Toggle relay state

            if (RELAY_PIN)
```

```
            {
                UART_Write_String("Relay ON\n");
            }
 else
{
                UART_Write_String("Relay OFF\n");
            }
        }
    }

    return 0;
}

// Function to initialize UART
void UART_Init(void) {
    // For PIC18F4580 (Modify settings based on your MCU)
    TXSTAbits.BRGH = 1; // High Baud Rate
    SPBRG = 25;  // Baud Rate 9600 for 4MHz Clock
    TXSTAbits.SYNC = 0;  // Asynchronous mode
    RCSTAbits.SPEN = 1;  // Enable serial port
    TXSTAbits.TXEN = 1;  // Enable transmitter
    RCSTAbits.CREN = 1;  // Enable receiver
}

// Function to read a character from UART
char UART_Read(void)
 {
    while (!PIR1bits.RCIF);
    return RCREG;
}

void UART_Write(char data)
 {
    while (!TXSTAbits.TRMT);  // Wait until TX buffer is empty
    TXREG = data;  // Transmit data
```

```
  }

// Function to send a string via UART
void UART_Write_String(const char *str)
{
    while (*str)
     {
        UART_Write(*str++);
     }
}
```

Q6.2) Write an Arduino code for ESP32 that updates the main code in the controller

```
#include <WiFi.h>
#include <ArduinoOTA.h>

const char* ssid = "Your_SSID";  // Replace with your Wi-Fi SSID
const char* password = "Your_PASSWORD";  // Replace with your Wi-Fi
password

void setup() {
    Serial.begin(115200);

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi...");
    }
    Serial.println("Connected to WiFi!");

    // Start OTA service
    ArduinoOTA.setHostname("ESP32-OTA");
```

```
    // Actions during OTA update
    ArduinoOTA.onStart([]() {
        Serial.println("Start updating...");
    });

    ArduinoOTA.onEnd([]() {
        Serial.println("\nUpdate Complete");
    });

    ArduinoOTA.onProgress([](unsigned int progress, unsigned int total)
{
        Serial.printf("Progress: %u%%\r", (progress * 100) / total);
    });

    ArduinoOTA.onError([](ota_error_t error) {
        Serial.printf("Error[%u]: ", error);
        if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");

        else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");

        else if (error == OTA_CONNECT_ERROR) Serial.println("Connect
Failed");
        else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive
Failed");
        else if (error == OTA_END_ERROR) Serial.println("End Failed");
    });

    ArduinoOTA.begin();
    Serial.println("OTA Ready");
}

void loop()
 {
    ArduinoOTA.handle();
    // Your main code here
```

```
    Serial.println("ESP32 Running...");
    delay(2000);
}
```

Q6.3) Creating a Time-Sensitive Button Debounce Mechanism

Task:
Implement a button debounce mechanism on the ESP32 that ensures a button press is only recognized if the button is held down for exactly 500 milliseconds, no more, no less. If the button is held down for less or more than 500 milliseconds, the press should be ignored.

Requirements:
1. Use the GPIO pin 0 for the button input.
2. Use an LED connected to GPIO pin 2 to indicate a successful button press.
3. The solution should involve FreeRTOS tasks to manage the timing requirements effectively.

```
#include <Arduino.h>

// Pin definitions
#define BUTTON_PIN 0
#define LED_PIN 2

// Task handles
TaskHandle_t ButtonTaskHandle = NULL;

TaskHandle_t LEDTaskHandle = NULL;

// Function prototypes
void ButtonTask(void *pvParameters);

void LEDTask(void *pvParameters);
```

```
void setup()
{
   Serial.begin(115200);

      pinMode(BUTTON_PIN, INPUT_PULLUP);

   pinMode(LED_PIN, OUTPUT);

   digitalWrite(LED_PIN, LOW);

   // Create FreeRTOS tasks
   xTaskCreatePinnedToCore(ButtonTask, "ButtonTask", 2048, NULL, 1,
&ButtonTaskHandle, 0);

   xTaskCreatePinnedToCore(LEDTask, "LEDTask", 1024, NULL, 1,
&LEDTaskHandle, 1);
}

// Button Monitoring Task
void ButtonTask(void *pvParameters)
{
   while (1)
{
       // Wait for button press
       if (digitalRead(BUTTON_PIN) == LOW) {
           uint32_t pressStart = millis();  // Record press time

           // Wait until button is released
           while (digitalRead(BUTTON_PIN) == LOW) {
               vTaskDelay(10 / portTICK_PERIOD_MS);  // Avoid CPU overload
           }

           uint32_t pressDuration = millis() - pressStart;  // Calculate press
duration
```

```
            // Check if the press duration is exactly 500ms
            if (pressDuration >= 495 && pressDuration <= 505) {
                Serial.println("Valid Button Press (500ms)");
                xTaskNotifyGive(LEDTaskHandle);  // Notify LED Task
            }
        else
          {
                Serial.println("Invalid Press Duration");
            }
        }

        vTaskDelay(50 / portTICK_PERIOD_MS);
    }
}

// LED Control Task
void LEDTask(void *pvParameters)
{
    while (1)
{
        ulTaskNotifyTake(pdTRUE, portMAX_DELAY);  // Wait for button event
        digitalWrite(LED_PIN, HIGH);
        vTaskDelay(100 / portTICK_PERIOD_MS);  // Blink LED for 100ms
        digitalWrite(LED_PIN, LOW);
    }
}

void loop()
{
    vTaskDelay(portMAX_DELAY);
}
```