Advance C Programming

Module 1 -Assignment

Topic 1: Structures

Question:
Write a C program that represents a calendar for a week. Each day has:

- **dayName (e.g., "Monday")**
- **tasks (array of strings with maximum 3 tasks per day)**

Note:

1. **Define appropriate structures.**
2. **Allow the user to input tasks for any day.**
3. **Display all tasks grouped by the day.**

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

// Define a structure for one day
struct Day {
    char name[10];            // Day name (like "Monday")
    char tasks[3][100];       // Up to 3 tasks, each up to 100 characters
    int taskCount;            // Number of tasks for the day
};

int main() {
    // Array of 7 days with predefined names
    struct Day week[7] = {
        {"Monday", {}, 0},
        {"Tuesday", {}, 0},
        {"Wednesday", {}, 0},
```

```
    {"Thursday", {}, 0},
    {"Friday", {}, 0},
    {"Saturday", {}, 0},
    {"Sunday", {}, 0}
};

int choice;
char dayName[10];
int i;

while (1) {
    printf("\n1. Add Task\n2. View Tasks\n0. Exit\nEnter choice: ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter day name (e.g., Monday): ");
        scanf("%s", dayName);

        // Find the matching day
        for (i = 0; i < 7; i++) {
            if (strcmp(week[i].name, dayName) == 0) {
                if (week[i].taskCount < 3) {
                    printf("Enter task: ");
                    scanf(" %[^\n]", week[i].tasks[week[i].taskCount]); // read full line
                    week[i].taskCount++;
                    printf("Task added.\n");
                } else {
                    printf("Maximum 3 tasks allowed per day.\n");
```

```
                }
                break;
            }
        }
    } else if (choice == 2) {
        printf("\n--- Weekly Tasks ---\n");
        for (i = 0; i < 7; i++) {
            printf("%s:\n", week[i].name);
            if (week[i].taskCount == 0) {
                printf("  No tasks\n");
            } else {
                for (int j = 0; j < week[i].taskCount; j++) {
                    printf("  - %s\n", week[i].tasks[j]);
                }
            }
        }
    } else if (choice == 0) {
        break;
    } else {
        printf("Invalid choice.\n");
    }
}

return 0;
}
```

```
1. Add Task
2. View Tasks
0. Exit
Enter choice: 1
Enter day name (e.g., Monday): Tuesday
Enter task: cleaning
Task added.

1. Add Task
2. View Tasks
0. Exit
Enter choice: 2

--- Weekly Tasks ---
Monday:
  No tasks
Tuesday:
  - cleaning
Wednesday:
  No tasks
Thursday:
  No tasks
Friday:
  No tasks
Saturday:
  No tasks
Sunday:
  No tasks

1. Add Task
2. View Tasks
0. Exit
Enter choice:
=== Session Ended. Please Run the code again ===
```

Topic 2: Pointers

Question:

Write a function in C that takes a pointer to an integer array and its size, and then rearranges the array in-place such that all even numbers appear before odd numbers, preserving the original relative order using only pointer arithmetic (no indexing with []).

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

void rearrangeEvenOdd(int *arr, int size) {
    int *temp = (int *)malloc(size * sizeof(int));
    if (!temp) return;

    int *p = arr;
    int *t = temp;

    // First copy even numbers in order
    for (int i = 0; i < size; i++) {
        if (*(p + i) % 2 == 0) {
            *t = *(p + i);
            t++;
        }
    }
    // Then copy odd numbers in order
    for (int i = 0; i < size; i++) {
        if (*(p + i) % 2 != 0) {
            *t = *(p + i);
            t++;
        }
```

```
    }
    // Copy back to original array
    for (int i = 0; i < size; i++) {
        *(p + i) = *(temp + i);
    }
    free(temp);
}
// Test
int main() {
    int arr[] = {3, 8, 5, 12, 10, 7, 6};
    int size = sizeof(arr) / sizeof(arr[0]);
    rearrangeEvenOdd(arr, size);
    for (int *p = arr; p < arr + size; p++) {
        printf("%d ", *p);
    }
    printf("\n");
    return 0;
}
```

```
8 12 10 6 3 5 7


=== Code Execution Successful ===
```

Topic 3: Arrays

Question:

You are given a 2D matrix of size n x n where each row and each column is sorted in increasing order. Write a C function to determine whether a given key exists in the matrix using the most efficient approach.

```
#include <stdio.h>


// Function to search for a key in a sorted 2D matrix
int searchMatrix(int matrix[][100], int n, int key) {
    int row = 0;
    int col = n - 1;
    while (row < n && col >= 0) {
        if (matrix[row][col] == key) {
            return 1; // Key found
        } else if (matrix[row][col] > key) {
            col--; // Move left
        } else {
            row++; // Move down
        }
    }
    return 0; // Key not found
}
int main() {
    int matrix[4][100] = {
        {10, 20, 30, 40},
        {15, 25, 35, 45},
        {27, 29, 37, 48},
        {32, 33, 39, 50}
    };
```

```
    int n = 4;
    int key;
    printf("Enter the key to search: ");
    scanf("%d", &key);
    if (searchMatrix(matrix, n, key)) {
        printf("Key %d found in the matrix.\n", key);
    } else {
        printf("Key %d not found in the matrix.\n", key);
    }
    return 0;
}
```

```
Enter the key to search: 50
Key 50 found in the matrix.


=== Code Execution Successful ===
```