ADVANCE C PROGRAMMING TRAINING MODULE 1 ASSIGNMENT SOLUTION

-BY SAKTHI KUMAR S

Topic 1: Structures

1. 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.

SOLUTION:

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

#define MAX_TASKS 3
#define MAX_DAYS 7
#define MAX_TASK_LEN 100

typedef struct {
    char name[10];
    char tasks[MAX_TASKS][MAX_TASK_LEN];
    int taskCount;
} Day;

// Initializing days

void initializeWeek(Day week[]) {
    const char *names[MAX_DAYS] = {
        "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"
    };
    for (int i = 0; i < MAX_DAYS; i++) {
        strcpy(week[i].name, names[i]);
        week[i].taskCount = 0;
    }
}

// Finding day index by name

int findDayIndex(Day week[], const char *dayName) {
    for (int i = 0; i < MAX_DAYS; i++) {
        if (strcasecmp(week[i].name, dayName) == 0)
            return i;
    }
    return -1;
}
```

// To add Tasks

```
void addTask(Day week[]) {
   char input[20];
   printf("Enter day name: ");
   scanf("%s", input);

   int index = findDayIndex(week, input);
   if (index == -1) {
      printf("Invalid day name. Try again.\n");
      return;
   }

   if (week[index].taskCount >= MAX_TASKS) {
      printf("Task limit reached for %s (max %d tasks).\n", week[index].name,
MAX_TASKS);
      return;
   }

   getchar();
   printf("Enter task: ");
   fgets(week[index].tasks[week[index].taskCount], MAX_TASK_LEN, stdin);

week[index].tasks[week[index].taskCount][strcspn(week[index].tasks[week[index].taskCo
unt], "\n")] = '\0';
   week[index].taskCount++;

   printf("Task added to %s.\n", week[index].name);
}
```

// Displaying Tasks

```
void displayTasks(const Day week[]) {
   printf("\n--- Weekly Tasks ---\n");
   for (int i = 0; i < MAX_DAYS; 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]);
         }
      }
   }
   printf("--------------------\n");
}
```

```
int main() {
    Day week[MAX_DAYS];
    int choice;

    initializeWeek(week);

    do {
        printf("\n--- Weekly To-Do List Menu ---\n");
        printf("1. Add Task\n");
        printf("2. Display Tasks\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addTask(week);
                break;
            case 2:
                displayTasks(week);
                break;
            case 3:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please enter 1, 2, or 3.\n");
        }
    } while (choice != 3);

    return 0;
}
```

OUTPUT:

```
--- Weekly To-Do List Menu ---
1. Add Task
2. Display Tasks
3. Exit
Enter your choice: 1
Enter day name: monday
Enter task: go to temple
Task added to Monday.

--- Weekly To-Do List Menu ---
1. Add Task
2. Display Tasks
3. Exit
Enter your choice: 1
Enter day name: tuesday
Enter task: assignment
Task added to Tuesday.

--- Weekly To-Do List Menu ---
1. Add Task
2. Display Tasks
3. Exit
Enter your choice: 1
Enter day name: monday
Enter task: review
Task added to Monday.
```

```
--- Weekly To-Do List Menu ---
1. Add Task
2. Display Tasks
3. Exit
Enter your choice: 2

--- Weekly Tasks ---
Monday:
  - go to temple
  - review
Tuesday:
  - assignment
Wednesday:
  No tasks
Thursday:
  No tasks
Friday:
  No tasks
Saturday:
  No tasks
Sunday:
  No tasks
--------------------

--- Weekly To-Do List Menu ---
1. Add Task
2. Display Tasks
3. Exit
Enter your choice: 3
Exiting program.
```

```
--- Weekly To-Do List Menu ---
1. Add Task
2. Display Tasks
3. Exit
Enter your choice: 1
Enter day name: wednesday
Enter task: task1
Task added to Wednesday.

--- Weekly To-Do List Menu ---
1. Add Task
2. Display Tasks
3. Exit
Enter your choice: 1
Enter day name: wednesday
Enter task: task2
Task added to Wednesday.

--- Weekly To-Do List Menu ---
1. Add Task
2. Display Tasks
3. Exit
Enter your choice: 1
Enter day name: wednesday
Enter task: task3
Task added to Wednesday.

--- Weekly To-Do List Menu ---
1. Add Task
2. Display Tasks
3. Exit
Enter your choice: 1
Enter day name: wednesday
Task limit reached for Wednesday (max 3 tasks).
```

Topic 2: Pointers

2. 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 []).

SOLUTION:

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

// Rearranging array so that even numbers come first, preserving relative order

void rearrange(int *arr, int size) {
    int *temp = (int *)malloc(size * sizeof(int));
    if (temp == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

    int *writePtr = temp;

    // Copying even numbers first
    for (int *ptr = arr; ptr < arr + size; ptr++) {
        if (*ptr % 2 == 0) {
            *writePtr = *ptr;
            writePtr++;
        }
    }

    // Then Copying odd numbers
    for (int *ptr = arr; ptr < arr + size; ptr++) {
        if (*ptr % 2 != 0) {
            *writePtr = *ptr;
            writePtr++;
        }
    }

    // Copying back to the original array
    for (int i = 0; i < size; i++) {
        *(arr + i) = *(temp + i);
    }

    free(temp);
}
```

```
// Printing array values using pointer arithmetic

void printArray(const int *arr, int size) {
    for (const int *ptr = arr; ptr < arr + size; ptr++) {
        printf("%d ", *ptr);
    }
    printf("\n");
}

int main() {
    int size;

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

    if (size <= 0) {
        printf("Invalid array size.\n");
        return 1;
    }

    int *arr = (int *)malloc(size * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    printf("Enter %d integers:\n", size);
    for (int *ptr = arr; ptr < arr + size; ptr++) {
        scanf("%d", ptr);
    }

    printf("\nOriginal:   ");
    printArray(arr, size);

    rearrange(arr, size);

    printf("Rearranged: ");
    printArray(arr, size);

    free(arr);
    return 0;
}
```

OUTPUT:

```
Enter the number of elements: 6
Enter 6 integers:
1 2 3 4 5 6

Original:   1 2 3 4 5 6
Rearranged: 2 4 6 1 3 5
```

```
Enter the number of elements: 6
Enter 6 integers:
3 4 6 7 8 9

Original:   3 4 6 7 8 9
Rearranged: 4 6 8 3 7 9
```

```
Enter the number of elements: 6
Enter 6 integers:
5 2 9 4 1 6

Original:   5 2 9 4 1 6
Rearranged: 2 4 6 5 9 1
```

Topic 3: Arrays

3. 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.

SOLUTION:

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

bool searchMatrix(int matrix[][100], int n, int key) {
    int row = 0, col = n - 1;  // Start from top-right

    while (row < n && col >= 0) {
        if (matrix[row][col] == key)
            return true;
        else if (matrix[row][col] > key)
            col--;  // Move left
        else
            row++;  // Move down
    }
    return false;
}

void printMatrix(int matrix[][100], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%4d", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int n, key;
    int matrix[100][100];

    printf("Enter the size of the matrix (n x n): ");
    scanf("%d", &n);

    printf("Enter the elements: \n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &matrix[i][j]);

    printf("\nMatrix:\n");
    printMatrix(matrix, n);

    printf("\nEnter the key to search: ");
```

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

    if (searchMatrix(matrix, n, key))
        printf("Key %d found.\n", key);
    else
        printf("Key %d not found.\n", key);

    return 0;
}
```

OUTPUT:

```
Enter the size of the matrix (n x n): 3
Enter the elements:
1 2 3
4 5 6
7 8 9

Matrix:
   1   2   3
   4   5   6
   7   8   9

Enter the key to search: 9
Key 9 found.
```

```
Enter the size of the matrix (n x n): 2
Enter the elements:
1 2
3 4

Matrix:
   1   2
   3   4

Enter the key to search: 6
Key 6 not found.
```