BANGABANDHU SHEIKH MUJIBUR RAHMAN DIGITAL UNIVERSITY, BANGLADESH

Project Report

Course Title : Data Structure and Algorithms Sessional

Course Code : CSE 114

SUBMITTED BY

Pritom Kumar Bhowmik (2101005)
Redwan Bin Jalal (2101006)
Department of IRE
Session: 2021-2022
Bangabandhu Sheikh Mujibur Rahman Digital University, Bangladesh

SUBMITTED TO

Suman Saha
Lecturer
Department of IRE
Bangabandhu Sheikh Mujibur Rahman Digital University, Bangladesh

Date of Submission: 07 November, 2023

DSA Lab

Project on:- To-Do List Application

Table of Contents

1. Introduction:

A to-do list project implemented with Data Structures and Algorithms (DSA) is a practical application that leverages DSA concepts to efficiently manage tasks and prioritize work. This project allows users to add, remove, complete, search for, and organize tasks in a systematic and organized manner. By using DSA principles, such as data structures and algorithms, you can create a more efficient and scalable solution for task management.

2. Project Objectives:

The objectives of a To-Do List Project in Data Structures and Algorithms (DSA) are as follows:

1. Task Management: Create a system for efficient and organized task management, allowing users to add, view, update, and remove tasks.
2. Efficient Searching: Implement search algorithms to quickly locate tasks by name, date, or other criteria. Ensure that searching is efficient even for a large number of tasks.
3. Task Prioritization: Allow users to prioritize tasks based on deadlines, importance, or categories. Implement sorting algorithms to maintain an organized task list.
4. Task Completion Tracking: Enable users to mark tasks as complete or incomplete, providing a sense of accomplishment and progress tracking.
5. Memory Management: Properly manage memory allocation and deallocation, preventing memory leaks when adding or removing tasks.
6. User Interface: Develop a user-friendly interface for easy interaction with the to-do list, including input, display, and task manipulation.
7. Scalability: Choose appropriate data structures and algorithms that can scale to handle a growing number of tasks without significant performance degradation.
8. Learning and Practice: Provide a practical application for learning and practicing DSA concepts, such as arrays, linked lists, searching, sorting, and dynamic memory management.

9. Customization and Organization: Allow users to customize the to-do list, such as categorizing tasks, setting priorities, and organizing tasks based on due dates.
10. Error Handling: Implement error handling mechanisms to address potential issues, such as input validation and handling data structure-related errors.
11. User Feedback: Incorporate user feedback and improvements based on user experience to make the to-do list more user-friendly and effective.
12. Project Documentation: Create clear and comprehensive documentation to explain how the project works, including data structures, algorithms, and how to use the application.
13. Testing and Debugging: Thoroughly test the application to identify and fix any bugs or issues. This includes testing various scenarios and edge cases.
14. Performance Optimization: Optimize the performance of the application to ensure that it responds quickly, even with a large number of tasks.
15. User Education: Provide information and guidance within the application to educate users on how to make the most of the to-do list features and functionalities.
16. Security and Privacy: Ensure that user data is handled securely and that the application respects user privacy.
17. Project Completion: Successfully implement all planned features, ensuring that the to-do list application fulfills its intended purpose effectively.
18. Project Maintenance: Plan for ongoing maintenance, updates, and improvements to keep the application relevant and bug-free.

3. Implementation Details of this code:

1. Data Structures:

- **Task Structure:** Define a structure to represent a task, including fields like name, description, date, and completion status.

```
struct Task {
char name[50];
char description[100];
char date[20];
int completed;
};
```

- **Task List:** Create a data structure to store and manage tasks, which can be implemented using an array or a linked list. Choose the appropriate data structure based on the project's requirements.

```
struct Task tasks[MAX_TASKS];
```

- **Dynamic Memory Allocation (Optional):** If using a linked list, dynamically allocate memory for tasks as they are added to the list.

```
struct Node {
    struct Task task;
    struct Node* next;
};
```

2. Task Management:

- **Add Task:** Implement a function to add a new task to the task list. This function should include memory allocation (if using a linked list) and updating the task count.

```
void addTask(struct Task tasks[], int *taskCount, char name[], char description[], char
date[]);
```

- **Remove Task:** Create a function to remove a task from the list, updating the task count and memory deallocation (if using a linked list).

```
void removeTask(struct Task tasks[], int *taskCount, char name[]);
```

- **Mark Task as Complete:** Implement a function to mark a task as complete or incomplete.

```
void markTaskAsComplete(struct Task tasks[], int taskCount, char name[]);
```

3. **Searching and Sorting:**
   - **Search Task by Name:** Develop a function to search for a task by name and return its index in the list.

```
int searchTaskByName(struct Task tasks[], int taskCount, char name[]);
```

   - **Search Task by Date:** Create a function to search for a task by date and return its index in the list.

```
int searchTaskByDate(struct Task tasks[], int taskCount, char date[]);
```

   - **Sorting Tasks:** Implement sorting algorithms (e.g., merge sort, quicksort) to arrange tasks based on due date, priority, or completion status.

4. User Interface:

- **Command-Line Interface (CLI):** Design a user-friendly command-line interface for users to interact with the to-do list. Provide options for adding, removing, marking tasks, and displaying the task list.

5. Memory Management:

- Ensure proper memory management, including memory allocation and deallocation, to prevent memory leaks when adding or removing tasks (applies to linked list implementations).

6. Error Handling:

- Implement error handling mechanisms for scenarios like invalid input, memory allocation failures, and data structure-related errors.

7. User Education:

- Provide in-app guidance or help documentation to educate users on how to use the application effectively.

8. Testing and Debugging:

- Thoroughly test the application to identify and fix any bugs or issues. Test various scenarios and edge cases to ensure robust functionality.

9. Performance Optimization:

- Optimize the application's performance to ensure quick and responsive task management, even with a large number of tasks.

10. Security and Privacy:

- Ensure that user data is handled securely and that the application respects user privacy.

11. Documentation:

- Create comprehensive documentation that explains how the project works, including data structures, algorithms, and how to use the application.

12. User Feedback:

- Incorporate user feedback and improvements based on user experience to make the to-do list more user-friendly and effective.

13. Project Completion:

- Successfully implement all planned features, ensuring that the to-do list application fulfills its intended purpose effectively.

14. Project Maintenance:

- Plan for ongoing maintenance, updates, and improvements to keep the application relevant and bug-free.

4. Uses Of DSA:

Data Structures:

1. Array: The primary data structure used in this code is an array of struct Task elements to store and manage the to-do list items. The array is fixed in size with a maximum of 100 tasks, as specified by MAX_TASKS.
2. Struct: The struct Task structure is used to represent a task, with fields for name, description, date, and completion status.

Algorithms and Concepts:

1. Searching: The code uses linear search to find tasks by name or date. The searchTaskByName and searchTaskByDate functions iterate through the task list to locate the desired task or date.
2. Sorting (Optional): While sorting is not explicitly implemented in this code, it's common in to-do list applications to sort tasks based on different criteria such as due date, priority, or completion status. Sorting algorithms such as merge sort or quicksort can be added if desired.
3. Input Handling: The code uses input handling to receive user input for adding, removing, marking tasks, and searching for tasks by name or date.
4. Dynamic Memory Allocation (Optional): Although dynamic memory allocation is not used in this code, it can be implemented if a more flexible, dynamic-sized data structure like a linked list is desired for managing tasks.
5. Error Handling: While not explicitly shown in the provided code, error handling concepts can be added to handle cases such as invalid input or memory allocation failures.

Overall, the code provides a basic implementation of a to-do list with rudimentary data structures and algorithms for task management, searching, and a user interface. Depending on the specific requirements and desired features, more advanced data structures and algorithms can be incorporated to enhance the functionality and efficiency of the application.

❖ Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#define MAX_TASKS 100

struct Task {
    char name[50];
    char description[100];
    char date[20];
    int completed;
};


void addTask(struct Task tasks[], int *taskCount, char name[], char description[], char date[]) {
    if (*taskCount < MAX_TASKS) {
        struct Task newTask;
        strcpy(newTask.name, name);
        strcpy(newTask.description, description);
        strcpy(newTask.date, date);
        newTask.completed = 0;
        tasks[*taskCount] = newTask;
        (*taskCount)++;
        printf("Task '%s' with name '%s' added to the to-do list with the date '%s'.\n", description, name, date);
    } else {
        printf("The to-do list is full. Cannot add more tasks.\n");
    }
}


void markTaskAsComplete(struct Task tasks[], int taskCount, char name[]) {
    int taskIndex = searchTaskByName(tasks, taskCount, name);
    if (taskIndex != -1) {
        tasks[taskIndex].completed = 1;
        printf("Task '%s' has been marked as complete.\n", name);
    } else {
        printf("Task '%s' not found in the to-do list.\n", name);
    }
}


int searchTaskByName(struct Task tasks[], int taskCount, char name[]) {
    for (int i = 0; i < taskCount; i++) {
        if (strcmp(tasks[i].name, name) == 0) {
            return i;
        }
    }
    return -1;
}
```

```c
int searchTaskByDate(struct Task tasks[], int taskCount, char date[]) {
    int left = 0;
    int right = taskCount - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        int cmp = strcmp(tasks[mid].date, date);

        if (cmp == 0) {
            return mid;
        } else if (cmp < 0) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return -1;
}


void displayTasks(struct Task tasks[], int taskCount) {
    if (taskCount == 0) {
        printf("Your to-do list is empty.\n");
    } else {
        printf("Your to-do list:\n");
```

```c
    }
    return -1;
}


void displayTasks(struct Task tasks[], int taskCount) {
    if (taskCount == 0) {
        printf("Your to-do list is empty.\n");
    } else {
        printf("Your to-do list:\n");
        for (int i = 0; i < taskCount; i++) {
            printf("%d. Task Name: %s, Description: %s (Date: %s, %s)\n", i + 1, tasks[i].name, tasks[i].description, tasks[i].date, tasks[i].completed ? "Complete" : "Incomplete
        }
    }
}



int main() {
    struct Task tasks[MAX_TASKS];
    int taskCount = 0;
    char name[50];
    char description[100];
    char date[20];

    while (1) {
        int choice;
        printf("\nWelcome To This Application\n\nTo-Do List Menu:\n");
```

How manu like for this...

```c
        int choice;
        printf("\nWelcome To This Application\n\nTo-Do List Menu:\n");
        printf("1. Add Task\n");
        printf("2. Remove Task\n");
        printf("3. Display Tasks\n");
        printf("4. Mark Task as Complete\n");
        printf("5. Search for Task by Task Name\n");
        printf("6. Search for Task by Date\n");
        printf("7. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter task name: ");
                scanf(" %[^\n]", name);
                printf("Enter task description: ");
                scanf(" %[^\n]", description);
                printf("Enter task date (YYYY-MM-DD): ");
                scanf(" %[^\n]", date);
                addTask(tasks, &taskCount, name, description, date);
                break;
            case 2:
                printf("Enter task description to remove: ");
                scanf(" %[^\n]", description);
                int taskIndex = searchTaskByName(tasks, taskCount, description);
                if (taskIndex != -1) {
                    for (int j = taskIndex; j < taskCount - 1; j++) {
```

```c
                    for (int j = taskIndex; j < taskCount - 1; j++) {
                        tasks[j] = tasks[j + 1];
                    }
                    taskCount--;
                    printf("Task '%s' removed from the to-do list.\n", description);
                } else {
                    printf("Task '%s' not found in the to-do list.\n", description);
                }
                break;
            case 3:
                displayTasks(tasks, taskCount);
                break;
            case 4:
                printf("Enter task name to mark as complete: ");
                scanf(" %[^\n]", name);
                markTaskAsComplete(tasks, taskCount, name);
                break;
            case 5:
                printf("Enter task name to search for: ");
                scanf(" %[^\n]", name);
                int searchResult = searchTaskByName(tasks, taskCount, name);
                if (searchResult != -1) {
                    printf("Task with name '%s' found at position %d. (%s)\n", name, searchResult + 1, tasks[searchResult].completed ? "Complete" : "Incomplete");
                } else {
                    printf("Task with name '%s' not found in the to-do list.\n", name);
                }
                break;
```

```c
                break;
            case 6:
                printf("Enter task date to search for (YYYY-MM-DD): ");
                scanf(" %[^\n]", date);
                int searchResultByDate = searchTaskByDate(tasks, taskCount, date);
                if (searchResultByDate != -1) {
                    printf("Task with date '%s' found at position %d. (%s)\n", date, searchResultByDate + 1, tasks[searchResultByDate].completed ? "Complete" : "Incomplete");
                } else {
                    printf("No task with date '%s' found in the to-do list.\n", date);
                }
                break;
            case 7:
                printf("Exiting the to-do list application.\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

❖ Output:

```
Welcome To This Application

To-Do List Menu:
1. Add Task
2. Remove Task
3. Display Tasks
4. Mark Task as Complete
5. Search for Task by Task Name
6. Search for Task by Date
7. Quit
Enter your choice: 1
Enter task name: pritom
Enter task description: Redwan
Enter task date (YYYY-MM-DD): 2023-2-4
Task 'Redwan' with name 'pritom' added to the to-do list with the date '2023-2-4'.
```

```
Welcome To This Application

To-Do List Menu:
1. Add Task
2. Remove Task
3. Display Tasks
4. Mark Task as Complete
5. Search for Task by Task Name
6. Search for Task by Date
7. Quit
Enter your choice: 1
Enter task name: Dsa Project
Enter task description: To do list
Enter task date (YYYY-MM-DD): 2023-11-07
Task 'To do list' with name 'Dsa Project' added to the to-do list with the date '2023-11-07'.

Welcome To This Application

To-Do List Menu:
1. Add Task
2. Remove Task
3. Display Tasks
4. Mark Task as Complete
5. Search for Task by Task Name
6. Search for Task by Date
7. Quit
Enter your choice:
```

❖ Conclusion:

The To-Do List project is a simple console-based application that allows users to manage their tasks efficiently. It demonstrates the use of data structures and algorithms to create a functional task management system. In conclusion, here are the key points and takeaways from this project:

1. **Functionality**: The project provides a range of essential features, including adding tasks, marking tasks as complete, searching for tasks by name or date, and displaying the current to-do list. Users can perform common to-do list operations with ease.
2. **Data Structure**: The code utilizes a structure (**struct Task**) to represent tasks. An array of these structures is used to store and manage tasks, demonstrating a basic application of data structures in organizing and manipulating data.
3. **Searching and Retrieving**: The project implements functions for searching tasks by both name and date. These search operations showcase basic algorithms for retrieving information from a dataset.
4. **User Interface**: The text-based user interface is simple and user-friendly. It allows users to interact with the application easily through a menu system.
5. **Error Handling**: The project includes error handling, such as notifying the user when attempting to add a task when the to-do list is full or when trying to mark a non-existent task as complete.

In summary, this project demonstrates the application of data structures and algorithms in building a practical task management system. It can serve as a foundation for more complex and feature-rich to-do list applications and be extended to include additional functionalities like task prioritization, due dates, and user accounts.