EAST WEST UNIVERSITY

Project Report

Project Title : To-Do List
Course Title : Algorithms
Course Code : CSE246
Section : 02
Semester : Fall 2024

Submitted To,
Dr. Tania Sultana
Assistant Professor, Dept. of CSE

Submitted By,
Group : 07

Name	ID	Contribution
Tasmia Rahman	2022-3-60-164	33%
Md. Adnan	2022-3-60-179	33%
Mashiur Rahaman Mollah Niran	2022-3-60-183	33%

Date of Submission : 22th January, 2025

Table of Contents :

Introduction:

The To-Do List application is intended to help users manage their tasks more effectively. It lets users to add, update, delete, and display tasks, making it ideal for both personal and business use. The program focuses on simplicity and functionality, making it easy for users to organize and track their tasks. This project was created to meet the demand for efficient job management with low overhead.

Objective:

The primary goal is to enhance productivity by enabling users to manage their time effectively.The objective of this project is to develop a simple, command-line-based "To-Do List" application. The system allows users to:

1. Add tasks with unique identifiers, priorities, durations, and names.
2. Update task details as needed.
3. Delete tasks no longer required.
4. Display all current tasks in an organized manner.

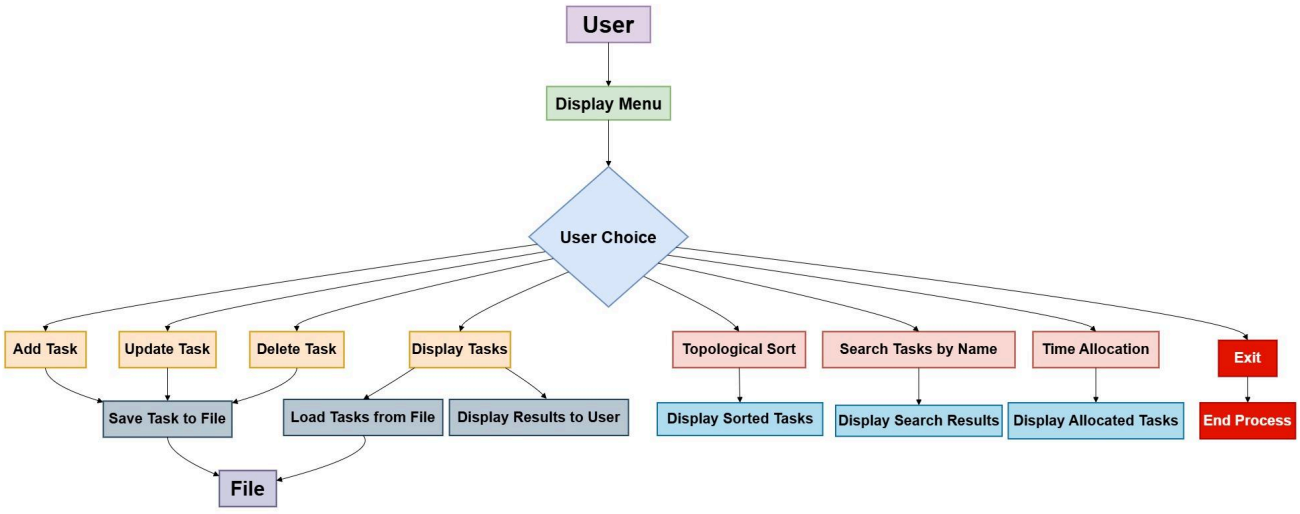
Technology Specification

- **Programming Language**: C++
- **Development Environment**: GCC Compiler, Linux/Windows terminal
- **Key Libraries**:
    - <iostream>: For input and output operations.
    - <queue>: For potential future extensions involving task prioritization.
    - <cstring>: For string manipulation.
    - <fstream>: For file handling.
    - <algorithm>: For manipulating collections of data.
- **File Management**: Tasks are stored in a text file (Tasks.txt) for persistence.
- **Key Algorithms**: Topological sorting, Greedy allocation, KMP algorithm for string matching.

- **Hardware Requirements**:
    - CPU: 1 GHz or faster processor.
    - Memory: 1 GB RAM.
    - Storage: Minimal storage for the executable and task data file.
- **Additional Considerations**:
    - Scalability: Supports up to 50 tasks.
    - Security: Data is stored in a plain-text file for simplicity, making it unsuitable for sensitive data.

Data Flow Diagram :

The flowchart shows the systemic process of the To-Do List step by step. It starts with the user interacting through a menu. Depending on the user's choice, the flow goes to different features like adding, updating, deleting, or displaying tasks. Other options include sorting tasks, searching tasks by name, and managing time, all shown side by side for simplicity. At the bottom, it shows how the system interacts with a file to save, load, or handle tasks. The process ends with options to exit and close the system.



Features List:

- **Add Task:** Allows the user to input and store task data.
- **Update Task:** Allows the modification of existing task properties.
- **Delete Task:** Deletes tasks from the system.
- **Display Tasks:** Shows all current tasks in a user-friendly style.
- **Sort Task:** Sort tasks based on priority and duration.
- **Time Allocation:** Allocate tasks based on available time using a greedy algorithm.
- **Find Task by Name:** Search for tasks by name using the KMP algorithm.
- **Save Task in File:** Persist tasks in a file for data recovery.

Implementation:

- **Adding Task :**
    - **Code:**

```cpp
// ADD TASK: Add a new task
void addTask(int id, const char* name, int priority, int duration)
{
    tasks[taskCount].id = id;
    strcpy(tasks[taskCount].name, name);
    tasks[taskCount].priority = priority;
    tasks[taskCount].duration = duration;
    tasks[taskCount].completed = false;

    taskCount++;
    saveTasksToFile();
    cout << "Task added successfully!\n";
}
```

```cpp
case 1:
{
    int id, priority, duration;
    char name[50];

    cout << "Enter Task ID: ";
    cin >> id;
    cin.ignore();
    cout << "Enter Task Name: ";
    cin.getline(name, 50);
    cout << "Enter Task Priority: ";
    cin >> priority;
    cout << "Enter Task Duration: ";
    cin >> duration;

    addTask(id, name, priority, duration);
    break;
}
```

    - **Result Console Screenshot:**

```
Enter Task ID: 1
Enter Task Name: Algorithm Project
Enter Task Priority: 10
Enter Task Duration: 5
Task added successfully!
```

- **Updating Task:**
  - **Code:**

```
// UPDATE: Modify an existing task
void updateTask(int id)
{
    int index = findTaskById(id);
    if (index == -1)
    {
        cout << "Task not found.\n";
        return;
    }

    cout << "Updating Task ID: " << id << "\n";
    cout << "Enter new Task Name (current: " << tasks[index].name << "): ";
    cin.ignore();
    cin.getline(tasks[index].name, 50);

    cout << "Enter new Priority (current: " << tasks[index].priority << "): ";
    cin >> tasks[index].priority;

    cout << "Enter new Duration (current: " << tasks[index].duration << "): ";
    cin >> tasks[index].duration;
    saveTasksToFile();
    cout << "Task updated successfully!\n";
}
```

```
case 2:
{
    int id;
    cout << "Enter Task ID to Update: ";
    cin >> id;
    updateTask(id);
    break;
}
```

  - **Result Console Screenshot:**

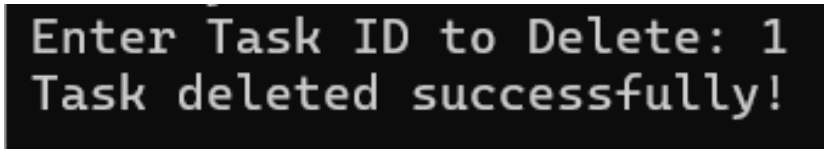
```
Enter Task ID to Update: 1
Updating Task ID: 1
Enter new Task Name (current: Algorithm Project): Algorithm Presentation
Enter new Priority (current: 10): 9
Enter new Duration (current: 5): 4
Task updated successfully!
```

- **Deleting Task :**
  - **Code :**

```
// DELETE: Remove a task
void deleteTask(int id)
{
    int index = findTaskById(id);
    if (index == -1)
    {
        cout << "Task not found.\n";
        return;
    }

    for (int i = index; i < taskCount - 1; i++)
    {
        tasks[i] = tasks[i + 1];
    }
    taskCount--;
    saveTasksToFile();
    cout << "Task deleted successfully!\n";
}
```

```
case 3:
{
    int id;
    cout << "Enter Task ID to Delete: ";
    cin >> id;
    deleteTask(id);
    break;
}
```

- **Result Console Screenshot:**

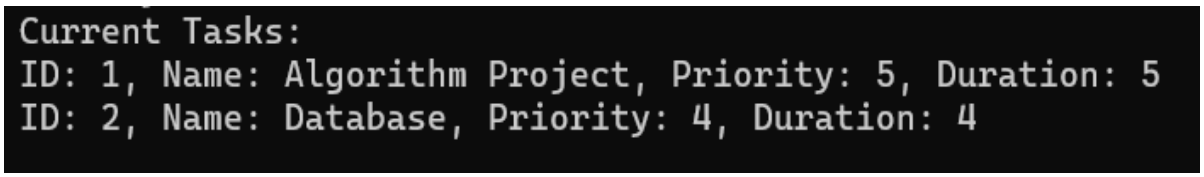


- **Displaying Task :**
  - **Code :**

```
// DISPLAY: Show all tasks
void displayTasks()
{
    if (taskCount == 0)
    {
        cout << "No tasks available.\n";
        return;
    }

    cout << "Current Tasks:\n";
    for (int i = 0; i < taskCount; i++)
    {
        cout << "ID: " << tasks[i].id << ", Name: " << tasks[i].name
             << ", Priority: " << tasks[i].priority
             << ", Duration: " << tasks[i].duration << "\n";
    }
}
```

```
case 4:
    displayTasks();
    break;
```

  - **Result Console Screenshot :**



- **Sorting Task :**
  - **Code :**

```
// TOPOLOGICAL SORT: Find task execution order
void topologicalSort()
{
    // Using simple priority queue for topological sorting based on priority and duration
    vector<Task> sortedTasks(tasks, tasks + taskCount);

    // Sort tasks by priority (descending order) and then by duration (ascending)
    sort(sortedTasks.begin(), sortedTasks.end(), [](Task &a, Task &b)
    {
        if (a.priority == b.priority)
            return a.duration < b.duration;
        return a.priority > b.priority;
    });

    cout << "Task Order (based on Priority and Duration):\n";
    for (Task &task : sortedTasks)
    {
        cout << task.name << " (ID: " << task.id << "), Priority: " << task.priority
             << ", Duration: " << task.duration << "\n";
    }
}
```

```
case 5:
    topologicalSort();
    break;
```

  - **Result Console Screenshot:**

```
Task Order (based on Priority and Duration):
Algorithm Project (ID: 1), Priority: 5, Duration: 5
Database (ID: 2), Priority: 4, Duration: 4
Algorithm Presentation (ID: 3), Priority: 3, Duration: 7
```

- **Searching Task :**
  - **Code :**

```
void searchTasksByName(const char* keyword)
{
    int keywordLength = strlen(keyword);
    int prefix[keywordLength];

    // Precompute prefix array
    computePrefixArray(keyword, prefix, keywordLength);

    cout << "Matching Tasks:\n";
    bool found = false;

    for (int i = 0; i < taskCount; i++)
    {
        const char* taskName = tasks[i].name;
        int taskNameLength = strlen(taskName);
        int j = 0; // Index for keyword

        for (int k = 0; k < taskNameLength; k++)
        {
            if (keyword[j] == taskName[k])
            {
                j++;
                if (j == keywordLength)
                {
                    // Match found
                    cout << "ID: " << tasks[i].id << ", Name: " << tasks[i].name
                         << ", Priority: " << tasks[i].priority
                         << ", Duration: " << tasks[i].duration << "\n";
                    found = true;
                    break;
                }
            }
            else
            {
                if (j != 0)
                {
                    j = prefix[j - 1];
                    k--; // Recheck the current character
                }
            }
        }
    }

    if (!found)
    {
        cout << "No matching tasks found.\n";
    }
}
```

```
case 6:
{
    char keyword[50];
    cout << "Enter keyword to search: ";
    cin.ignore();
    cin.getline(keyword, 50);
    searchTasksByName(keyword);
    break;
}
```

  - **Result Console Screenshot :**

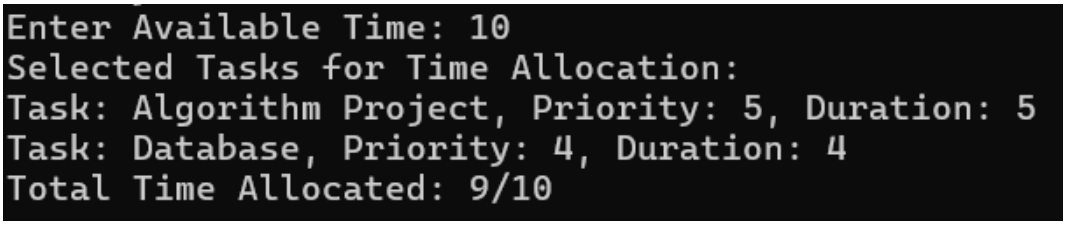
```
Enter keyword to search: Algorithm
Matching Tasks:
ID: 1, Name: Algorithm Project, Priority: 5, Duration: 5
ID: 3, Name: Algorithm Presentation, Priority: 3, Duration: 7
```

- **Time Allocating Task :**
  - **Code :**

```
// COIN CHANGE GREEDY: Allocate time to tasks based on priority
void timeAllocationGreedy(int totalTime)
{
    Task temp[MAX_TASKS];
    memcpy(temp, tasks, sizeof(tasks)); // Create a copy of tasks array for sorting

    // Sort tasks by priority (descending order)
    sort(temp, temp + taskCount, [](Task &a, Task &b)
    {
        return a.priority > b.priority;
    });

    int currentTime = 0; // Track total allocated time
    cout << "Selected Tasks for Time Allocation:\n";

    for (int i = 0; i < taskCount; i++)
    {
        if (currentTime + temp[i].duration <= totalTime)
        {
            currentTime += temp[i].duration;
            temp[i].completed = true;
            cout << "Task: " << temp[i].name << ", Priority: " << temp[i].priority << ", Duration: " << temp[i].duration << "\n";
        }
    }

    cout << "Total Time Allocated: " << currentTime << "/" << totalTime << "\n";
}
```

```
case 7:
{
    int totalTime;
    cout << "Enter Available Time: ";
    cin >> totalTime;
    timeAllocationGreedy(totalTime);
    break;
}
```

  - **Result Console Screenshot :**



- **Complexity:**

1. **Time Complexity**: Sorting tasks (O(n log n)), task iteration (O(n)).
2. **Space Complexity**: O(n) for task storage.

Total Complexity Analysis:

1. **Sorting**: Used in topological sort and greedy allocation (O(n log n)).
2. **Search (KMP**): Efficient pattern matching (O(n + m), where m is pattern length).
3. **File Operations**: Linear traversal during save/load (O(n)).

Results:

The application successfully fulfills the following functions:

1. Adding tasks with distinctive details.
2. Tasks are updated in response to user input.
3. Deleting tasks based on their ID.
4. Organizing and displaying all tasks.

Limitations:

1. The static array implementation limits task management to 50.
2. There is no advanced error handling for erroneous input or corrupted files.
3. Plain-text file storage lacks encryption, rendering it unsuitable for sensitive data.
4. The user interface is limited to console-based interaction.

Conclusion:

The To-Do List program offers a basic yet effective task management solution. While the existing implementation is working, future additions may include:

1. The **To-Do List Management System** efficiently manages tasks using algorithms like **KMP for searching**, **Greedy for time allocation** and **topological sort for sorting**.
2. It ensures reliable data persistence through **file handling**, making task organization seamless and effective.
3. This project highlights practical algorithm applications, enhancing productivity and time management.