Final Project

The Task Manager - A Todo List (Linked List)	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** December 2, 2024
Section: CPE21S4	**Date Submitted:** December 2, 2024
Name: - ESTEBAN, Prince Wally G. - FERNANDEZ, Don Eleazar T. - SANCHEZ, Christan Ray R. - TANDAYU, Leoj Jeam B. - VALLESER, Kenn L.	**Instructor:** Engr. Maria Rizette Sayo

1. Objective(s)

The project aims to implement a linked list-based structure for task storage, utilizing a doubly linked list to manage tasks, which allows users to add and process them in any order. It focuses on designing and implementing efficient functions for adding, removing, and prioritizing tasks. Additionally, the project includes developing a user-friendly console interface to enable users to interact with tasks and monitor their status in real-time. Finally, it will involve testing and evaluating the task manager's performance, emphasizing the efficiency of task operations, specifically insertion, deletion, and retrieval using linear data structures.

The specific objective of this program is to enhance the functionality of the task manager by improving key operations such as insertion, deletion, retrieval, and saving of tasks. These operations are essential for the smooth management of tasks within the system, and by optimizing their implementation, the overall performance and efficiency of the program can be significantly improved. Specifically, these functions will allow for the addition of new tasks, the effective removal of completed or outdated tasks, the quick retrieval of information, and the reliable saving of tasks, all of which contribute to a more streamlined and efficient user experience.

2. Intended Learning Outcomes (ILOs)

After this activity, the student should be able to:

- To implement linked list-based structure for storing tasks.
- To apply a DOUBLY LINKED LISTS data structure for task management, allowing users to add tasks and process them in ANY ORDER.
- To design and implement functions for adding, removing, and prioritizing tasks efficiently.
- To develop a user-friendly interface (CONSOLE) that allows users to interact with tasks and view their status in real-time.
- To test and evaluate the performance of the task manager, focusing on the efficiency of task

operations (insertion, deletion, retrieval) using linear data structures.

3. Discussion

Most people are often busy with their assigned tasks. Many of them have not just one, but multiple tasks to complete for the day. And so, they sometimes feel lost or not sure about what to do next. Task management software is a tool that helps teams organize and prioritize tasks, view deadlines and check off tasks. Many task managers have functionalities that link tasks based on dependencies, send automated reminders for upcoming due dates, set up recurring tasks and build workflows for complex projects. An effective task manager tool helps teams streamline work to improve efficiency, productivity and, ultimately, profitability (Williams, 2024).

Benjamin Franklin, a contemporary of the Scottish economist with plenty of productivity theories of his own, put forth what might be considered the first "to-do" list in 1791. The productivity measure in Franklin's list of tasks (wash, work, read, work, put things in their places) was less likely to be measured in hard numbers. Franklin's assessment was simple: Start the day asking what good shall be done, and at the end of the day evaluate based on what was accomplished (Dishman, 2018).

To support busy professionals in managing and even performing their official tasks, the aim is to provide an analysis of daily efficiency and time utilization in completing day to day tasks and checking the daily progress. It also helps in maintaining a daily streak of successful tasks completion in order to achieve a long term goal (Kejriwal et al., 2020, p. 969).

How do we start the project so first we will begin by analyzing the functional requirements of the task manager, specifically focusing on how tasks will be managed using linked lists. This involves defining the core features of the console interface, which will provide the following options for user interaction in C++:

- Add Task
- Display Tasks
- Complete Task
- Cancel Task
- Delete Task
- Save to Memory File
- Exit

These options will form the basis of the task manager's interaction flow, allowing users to efficiently manage tasks.

In System Design, we will design the system architecture, emphasizing the use of a doubly linked list to provide flexible task management. To guide implementation, we will create flowcharts illustrating how tasks are added, removed, and retrieved. Additionally, we will design the core functions needed for task operations, such as insertion, deletion, retrieval, and saving task data to a local file.

The Implementation in this phase, the task manager will be implemented using a doubly linked list to store tasks. Key functionalities will include:

- Adding tasks to the list
- Removing and marking tasks (as complete or pending)
- Managing the list through the console interface described earlier

The user interface will be console-based, providing a simple and intuitive way for users to interact with the task manager.

Doubly Linked List - According to P. Basemera, A doubly linked list is a linear data structure that consists of a series of nodes, each containing data and two pointers: one pointing to the previous node and another to the next node. This unique characteristic allows for bidirectional traversal of the list, making it a versatile data structure (Basemera, 2023).

Linked List - According to Ravikiran AS, A linked list is the most sought-after data structure when it comes to handling dynamic data elements. A linked list consists of a data element known as a node. And each node consists of two fields: one field has data, and in the second field, the node has an address that keeps a reference to the next node. (Rakiviran, 2024)

Operations of the Task Manager	
Insertion	The insertion adds a new element to the linked list (*Linked List Operations: Traverse, Insert and Delete*, n.d.).
Deletion	The deletion removes the existing elements (*Linked List Operations: Traverse, Insert and Delete*, n.d.).
Retrieval	The retrieve is the process of obtaining information or data from a storage location (*What Is Retrieve? How Does Data Retrieval Work in Databases*, n.d.).
Saving	The save is the process to copy the document, record or image being worked on to a storage medium (*Save*, n.d.).

The Testing and Debugging in the project will start once the system is implemented, we will perform unit testing on individual task operations (e.g., adding, deleting, completing tasks) to ensure correctness. Integration testing will follow to ensure that the doubly linked list works seamlessly with the task manager's overall functionality. Performance testing will be conducted to evaluate how well the system handles large sets of tasks.

Finally, in the evaluation phase, we will evaluate the efficiency of key task operations, such as insertion, deletion, and retrieval, by measuring the time complexity of each. User feedback will be gathered to assess the usability and effectiveness of the console interface, ensuring that the task manager meets both functional and user experience requirements.

4. Materials and Equipments

- A Desktop Computer with Dev C++
- A Laptop with Dev C++
- A Windows Operating System

5. Procedure

Flowchart

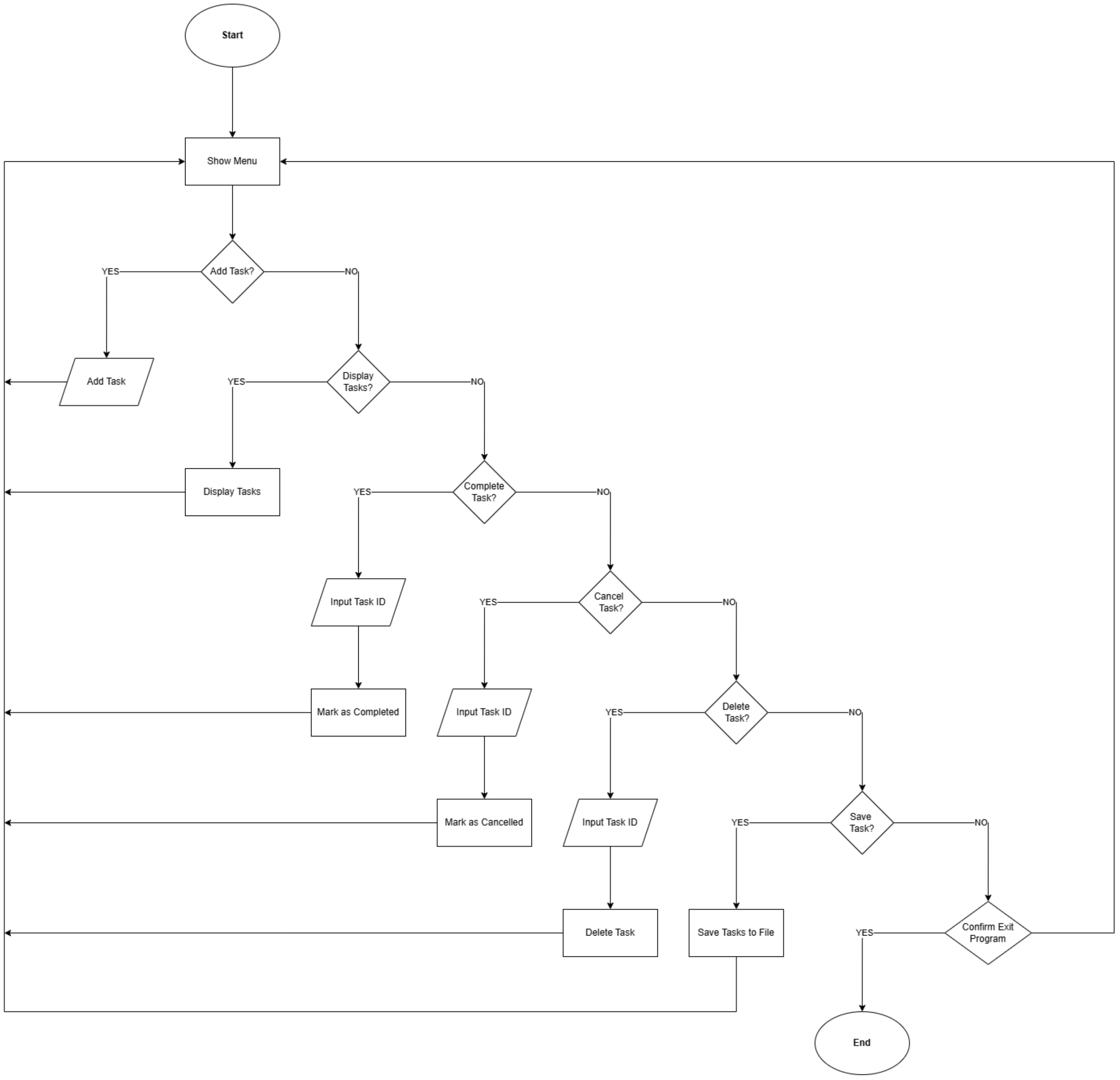


Figure 1: The flowchart of the Task Manager.

Algorithm

1. Start
2. Menu
3. Choose Action (Add task, Display task, Complete Task, Cancel Task, Delete Task, Save Tasks to File)
4. If the user wants to continue, Loop.
5. Else Exit Program
6. End

Pseudocode

```
Define a structure "Task" with attributes:
    - id (int)
    - name (string)
    - isCompleted (boolean)
    - isCancelled (boolean)
Define a class "TaskManager" with:
    - Private attributes:
        - tasks (list of Task): stores the tasks
        - taskMap (unordered_map of int to iterator): maps task ID to task iterator in the list
        - nextId (int): counter for assigning unique task IDs
    - Public methods:
        - Constructor: initializes nextId to 1
        - addTask(taskName): adds a new task to tasks list and maps it in taskMap
        - displayTasks(): displays all tasks with their status
        - completeTask(taskId): marks task as completed by ID
        - cancelTask(taskId): marks task as cancelled by ID
        - deleteTask(taskId): deletes a task by ID from both tasks list and taskMap
        - saveToFile(filename): saves all tasks and their status to a specified file
Define main function:
    - Initialize TaskManager instance (taskManager)
    - Loop until the user exits:
        - Display menu options:
            1. Add Task
            2. Display Tasks
            3. Complete Task
            4. Cancel Task
```

```
          5. Delete Task
          6. Save to Memory File
          7. Exit
        - Get user's choice and perform the corresponding action:
          - If choice is 1:
            - Get task name input from the user
            - Call taskManager.addTask() with task name
          - If choice is 2:
            - Call taskManager.displayTasks()
          - If choice is 3:
            - Get task ID input from the user
            - Call taskManager.completeTask() with task ID
          - If choice is 4:
            - Get task ID input from the user
            - Call taskManager.cancelTask() with task ID
          - If choice is 5:
            - Get task ID input from the user
            - Call taskManager.deleteTask() with task ID
          - If choice is 6:
            - Call taskManager.saveToFile() with filename "memory.txt"
          - If choice is 7:
            - Print exit message and break loop
          - Otherwise:
            - Print "Invalid choice" message
    - End of main function
```

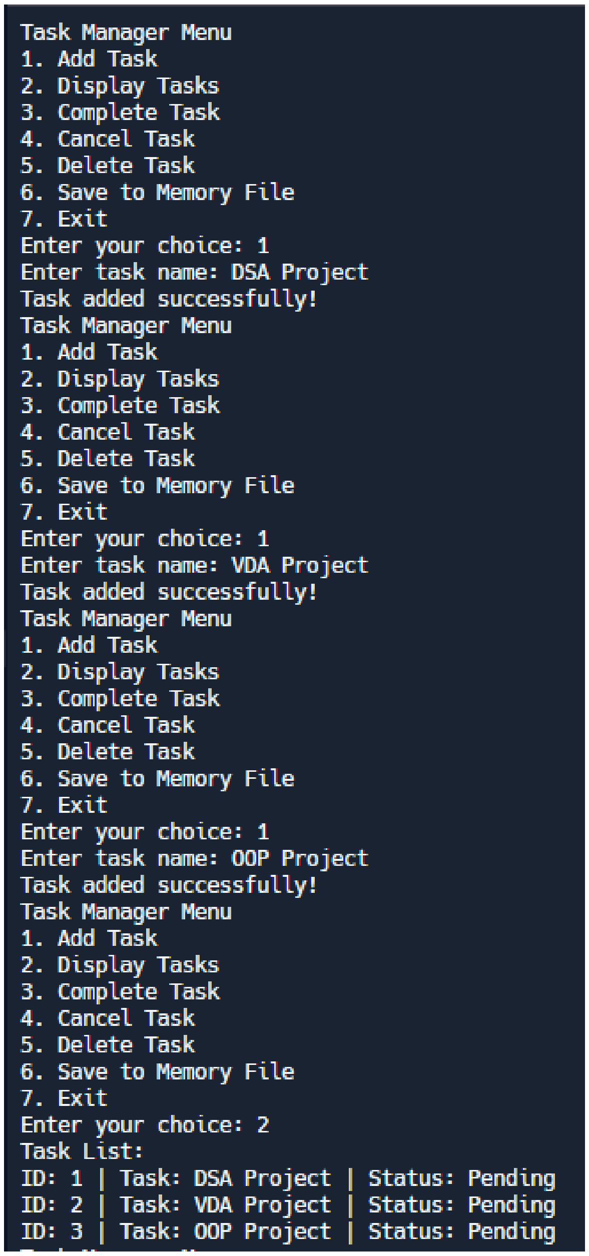
Actual Code

```
#include <iostream>
#include <fstream>
#include <list>
#include <unordered_map>
#include <string>

using namespace std;

struct Task {
    int id;
    string name;
    bool isCompleted, isCancelled;
};
```

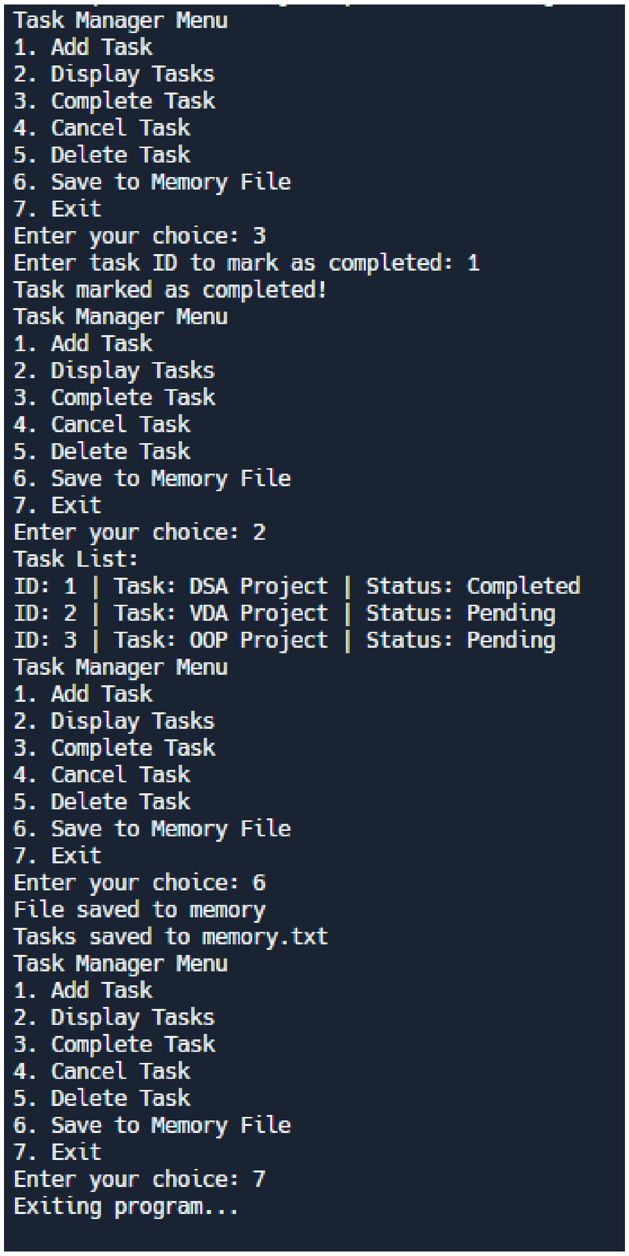
```
class TaskManager {
private:
    list<Task> tasks;                          // Stores the list of tasks
    unordered_map<int, list<Task>::iterator> taskMap;  // Maps task ID to list iterator
    int nextId;                                // ID counter for tasks

public:
    TaskManager() : nextId(1) {}

    void addTask(const string& taskName) {
        tasks.push_back({nextId, taskName, false});
        taskMap[nextId] = --tasks.end();  // Map the task ID to the newly added task
        cout << "Task added successfully!" << endl;
        ++nextId;
    }

    void displayTasks() const {
        if (tasks.empty()) {
            cout << "No tasks available." << endl;
        } else {
            cout << "Task List:" << endl;
            for (const auto& task : tasks) {
                cout << "ID: " << task.id << " | Task: " << task.name
                                 << " | Status: " << (task.isCompleted ? "Completed" : (task.isCancelled ?
"Cancelled" : "Pending")) << endl;
            }
        }
    }

    void completeTask(int taskId) {
        auto it = taskMap.find(taskId);
        if (it != taskMap.end()) {
            it->second->isCompleted = true;
            cout << "Task marked as completed!" << endl;
        } else {
            cout << "Task not found!" << endl;
        }
    }

    void cancelTask(int taskId) {
        auto it = taskMap.find(taskId);
        if (it != taskMap.end()) {
            it->second->isCancelled = true;
            cout << "Task marked as cancelled!" << endl;
        } else {
            cout << "Task not found!" << endl;
        }
    }

    void deleteTask(int taskId) {
        auto it = taskMap.find(taskId);
        if (it != taskMap.end()) {
            tasks.erase(it->second);  // Erase the task from the list
            taskMap.erase(it);        // Remove task from the map
            cout << "Task deleted successfully!" << endl;
        } else {
            cout << "Task not found!" << endl;
        }
    }

    void saveToFile(const std::string &filename) {
        ofstream outFile(filename);
        if (outFile.is_open()) {
            for (const auto& task : tasks) {
                        outFile << "ID: " << task.id << " | Task: " << task.name << " | Status: " <<
(task.isCompleted ? "Completed" : (task.isCancelled ? "Cancelled" : "Pending")) << endl;
            }
            outFile.close();
            cout << "Tasks saved to " << filename << std::endl;
        } else {
        cout << "Unable to open file " << filename << std::endl;
    }
}
};

int main() {
    TaskManager taskManager;
    int choice, id;
    string taskName;

    while (true) {
        cout << "-----------------------"<<endl;
        cout << "Task Manager Menu" << endl;
        cout << "1. Add Task" << endl;
        cout << "2. Display Tasks" << endl;
        cout << "3. Complete Task" << endl;
        cout << "4. Cancel Task" << endl;
        cout << "5. Delete Task" << endl;
        cout << "6. Save to Memory File" << endl;
        cout << "7. Exit" << endl;
        cout << "-----------------------"<<endl;
        cout << "Enter your choice: ";
        cin >> choice;

if (choice == 1) {
            cout << "Enter task name: ";
            cin.ignore();
            getline(cin, taskName);
            taskManager.addTask(taskName);
        } else if (choice == 2) {
            taskManager.displayTasks();
        } else if (choice == 3) {
            cout << "Enter task ID to mark as completed: ";
            cin >> id;
            taskManager.completeTask(id);
        } else if (choice == 4) {
            cout << "Enter task ID to Cancel: ";
            cin >> id;
            taskManager.cancelTask(id);
        } else if (choice == 5) {
            cout << "Enter task ID to delete: ";
            cin >> id;
            taskManager.deleteTask(id);
        } else if (choice == 6) {
            cout << "File saved to memory" << endl;
            taskManager.saveToFile("memory.txt");
        } else if (choice == 7) {
            cout << "Exiting program..." << endl;
            break;
        } else {
            cout << "Invalid choice!" << endl;
            break;
        }
```

```
    }
    return 0;
}
```

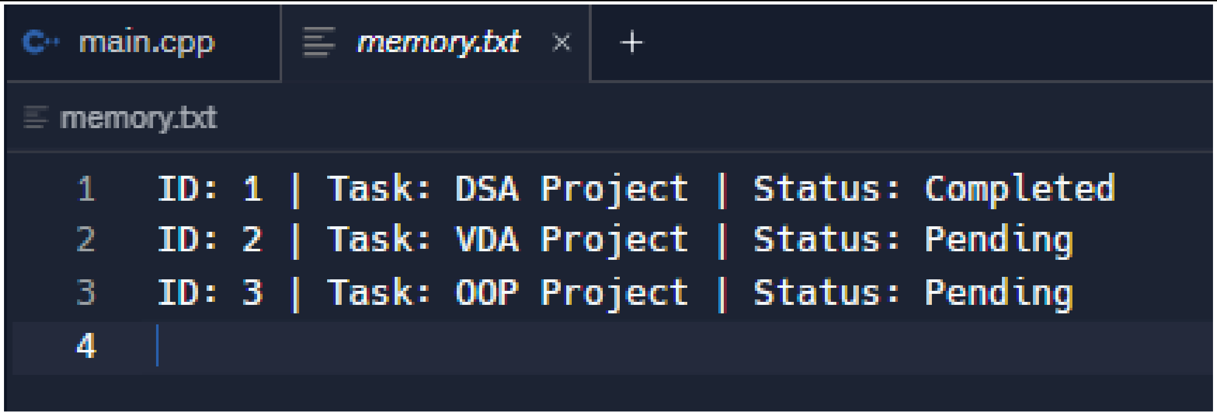
6. Output



```
Task Manager Menu
1. Add Task
2. Display Tasks
3. Complete Task
4. Cancel Task
5. Delete Task
6. Save to Memory File
7. Exit
Enter your choice: 1
Enter task name: DSA Project
Task added successfully!
Task Manager Menu
1. Add Task
2. Display Tasks
3. Complete Task
4. Cancel Task
5. Delete Task
6. Save to Memory File
7. Exit
Enter your choice: 1
Enter task name: VDA Project
Task added successfully!
Task Manager Menu
1. Add Task
2. Display Tasks
3. Complete Task
4. Cancel Task
5. Delete Task
6. Save to Memory File
7. Exit
Enter your choice: 1
Enter task name: OOP Project
Task added successfully!
Task Manager Menu
1. Add Task
2. Display Tasks
3. Complete Task
4. Cancel Task
5. Delete Task
6. Save to Memory File
7. Exit
Enter your choice: 2
Task List:
ID: 1 | Task: DSA Project | Status: Pending
ID: 2 | Task: VDA Project | Status: Pending
ID: 3 | Task: OOP Project | Status: Pending
```



```
Task Manager Menu
1. Add Task
2. Display Tasks
3. Complete Task
4. Cancel Task
5. Delete Task
6. Save to Memory File
7. Exit
Enter your choice: 3
Enter task ID to mark as completed: 1
Task marked as completed!
Task Manager Menu
1. Add Task
2. Display Tasks
3. Complete Task
4. Cancel Task
5. Delete Task
6. Save to Memory File
7. Exit
Enter your choice: 2
Task List:
ID: 1 | Task: DSA Project | Status: Completed
ID: 2 | Task: VDA Project | Status: Pending
ID: 3 | Task: OOP Project | Status: Pending
Task Manager Menu
1. Add Task
2. Display Tasks
3. Complete Task
4. Cancel Task
5. Delete Task
6. Save to Memory File
7. Exit
Enter your choice: 6
File saved to memory
Tasks saved to memory.txt
Task Manager Menu
1. Add Task
2. Display Tasks
3. Complete Task
4. Cancel Task
5. Delete Task
6. Save to Memory File
7. Exit
Enter your choice: 7
Exiting program...
```



```
ID: 1 | Task: DSA Project | Status: Completed
ID: 2 | Task: VDA Project | Status: Pending
ID: 3 | Task: OOP Project | Status: Pending
```

7. Conclusion

In conclusion, the **Task Manager** project effectively demonstrates the practical application of **doubly linked lists** for managing dynamic and mutable data, specifically in the context of a to-do list application. By utilizing a doubly linked list, the system offers efficient task management operations such as **insertion**, **deletion**, **retrieval**, and **saving** in an intuitive, real-time user interface. This approach allows for both **forward** and **backward traversal** of the task list, enhancing the flexibility of the task manager.

The project achieved its objectives by incorporating key functionalities, including task prioritization, marking tasks as complete or cancelled, and saving the task list to a memory file, while ensuring efficient insertion, deletion, and retrieval of tasks. These functionalities were designed to be both **user-friendly** and **responsive**, enabling users to manage their daily tasks effectively.

In summary, the Task Manager demonstrates the power of **linked lists** in building **real-world applications**, specifically in the domain of **task management**. The project not only reinforces the theoretical concepts of data structures but also provides a practical solution for daily productivity management, offering users an efficient and reliable tool for task organization and monitoring.

8. References

Basemera, P. (2023, October 2). *Introduction to Doubly Linked Lists*. Medium. Retrieved November 13, 2024, from https://medium.com/@pbasemera/understanding-doubly-linked-lists-a-practical-implementation-with-browser-history-management-7cef5cb01d53

Dishman, L. (2018, September 12). *The dark history of our obsession with productivity*. Fast Company. Retrieved October 20, 2024, from https://www.fastcompany.com/90230330/how-our-obsession-with-productivity-evolved

Kejriwal, S., Vishal, V., Gulati, A., & Gambhir, G. (2020, December). A Review Of Daily Productivity Growth Using Todo Manager. *International Research Journal of Modernization in Engineering Technology and Science*, *2*(12), pp. 969–974. https://www.irjmets.com/uploadedfiles/paper/volume2/issue_12._december_2020/5480/1628083219.pdf

Linked List Operations: Traverse, Insert and Delete. (n.d.). Programiz. Retrieved November 13, 2024, from https://www.programiz.com/dsa/linked-list-operations

Rakiviran, A. S. (2024). Software Development. In *Data Structure Tutorial for Beginners* (3 of 64). Simpl;learn. Linked List in Data Structure

Save. (n.d.). PCMag. Retrieved November 13, 2024, from https://www.pcmag.com/encyclopedia/term/save

Williams, C. (2024, January 25). *Why You Need Task Manager Software — Not Just a To-Do List*. Accelo. Retrieved October 20, 2024, from https://www.accelo.com/post/why-task-manager-software

What is Retrieve? How Does Data Retrieval Work in Databases. (n.d.). Lenovo. Retrieved November 13, 2024, from https://www.lenovo.com/ph/en/glossary/what-is-retrieve/?orgRef=https%253A%252F%252Fwww.google.com%252F&srsltid=AfmBOoqFdqEqIRtKeAOJP1a5nQwTvbqcDZ5aowZj5HGOJNEVNy5aev00