1. Problem Statement

In daily life, managing tasks and to-do lists on paper or by memory can often lead to forgotten work, disorganization, or missed deadlines. Whether it's academic responsibilities, work-related assignments, or personal goals, keeping track of tasks manually becomes inefficient and unreliable over time.

To address this issue, we have developed a To-Do List Management System using **C++**. This system allows users to digitally create, manage, update, and track their tasks in a structured and organized manner. It eliminates the chances of forgetting important activities and helps users stay productive.

The program uses a doubly linked list structure to store tasks, allowing efficient addition, deletion, and navigation through tasks in both directions. Each task is assigned a unique ID, a title, and a description, along with a status indicating whether it is completed or pending.

The system supports essential features such as adding a new task, editing existing tasks, marking tasks as completed, deleting tasks, filtering by status, searching by ID, and viewing all tasks. The interactive console-based menu makes it user-friendly and straightforward for all types of users.

This project not only helps in managing day-to-day tasks effectively but also demonstrates the practical application of data structures and object-oriented programming in solving real-world problems.

This system helps the user to:

- **Add New Tasks:**
  Add a task by entering its ID, title, and description. This helps keep track of all pending or completed tasks.
- **Edit Existing Tasks:**
  Users can update the title or description of a task anytime.
- **Mark Tasks as Completed:**
  When a task is finished, the user can mark it as completed. This helps in distinguishing between pending and done tasks.
- **Delete Tasks:**
  Tasks that are no longer needed can be deleted to keep the list clean and relevant.
- **Filter Tasks by Status:**
  Users can view only completed or only pending tasks based on their requirement.
- **Search Tasks by ID:**
  A task can be searched using its unique ID to quickly locate its details.
- **View All Tasks:**
  Displays a full list of all tasks with their ID, title, description, and status.

2. Tools and Technologies

- **Tool Used:**
  Visual Studio Code was used to write and test the C++ code.
- **Technologies Used:**
  - **C++ Programming Language:** The core language used to build the system.
  - **Doubly Linked List Data Structure:** To store tasks dynamically and support two-way traversal.
  - **Object-Oriented Programming (OOP):** Tasks and operations are organized into classes and objects.
  - **Console-based Application:** Runs in the terminal/command prompt for simple interaction.

3. Data Structure Used

The project uses a **doubly linked list** to store tasks. Each node represents a task with the following details:

- **ID**: Unique identifier for the task.
- **Title**: Short name of the task.
- **Description**: Details about the task.
- **isComplete**: A boolean to indicate if the task is done.
- **Previous & Next Pointers**: For bidirectional linking between tasks.

Justification

- **Efficient Insertion and Deletion:** Tasks can be added or removed without shifting other data (unlike arrays).
- **No Fixed Size:** The list grows dynamically as tasks are added.
- **Two-way Navigation:** Using a doubly linked list helps navigate forward or backward if needed.
- **Memory Efficient:** Only memory for actual tasks is used.

4. Code Flow Explanation

Overview

This project is a **console-based To-Do List Management System in C++**, designed using a **doubly linked list**. It supports common task management features including addition, editing, deletion, and filtering.

Structure: Node

Each task is represented as a node with:

- `Id`: Unique task identifier
- `title`: Name of the task
- `description`: Details of the task
- `iscomplete`: Completion status
- `next` and `previous`: Pointers to connect tasks in both directions

Class: ToDoList

Manages all operations including:

1. addTask() – Add a New Task

Adds a new task to the end of the doubly linked list. It creates a new task node using user-provided details such as ID, title, and description. If the list is empty, the new task becomes the head node. Otherwise, it is inserted at the end by adjusting the previous and next pointers accordingly. The new task is initially marked as not completed.

2. editTask() – Edit a Task

Updates an existing task by searching for its ID. Once found, the function takes new input for the task's title and description and updates the values. If the task ID does not exist, an appropriate message is shown.

3. markdone() – Mark Task as Completed

Changes the completion status of a task to "completed" using its ID. It searches for the specified task in the list and sets its iscomplete flag to true to indicate that the task has been finished.

4. deleteTask() – Delete a Task

Removes a task from the list by its ID. The function searches for the task and, once located, updates the links of the neighboring nodes to bypass the target node. The node is then deleted from memory to free up space.

5. filterTasks() – Filter by Completion Status

Displays only the tasks that match a certain completion status. The user is prompted to enter 1 for completed or 0 for pending tasks. The function then traverses the list and displays tasks that match the selected status.

6. searchTask() – Search for a Task by ID

Searches for a specific task using its ID and displays its details. If the task is found, its title and description are shown. If not, a message is displayed indicating that no task with the entered ID exists.

7. viewTasks() – View All Tasks

Displays all the tasks in the list along with their details. Each task's ID, title, description, and status (completed or pending) is shown in order, starting from the head of the list.

Main Function – Menu System

The main function presents a menu with the following options:

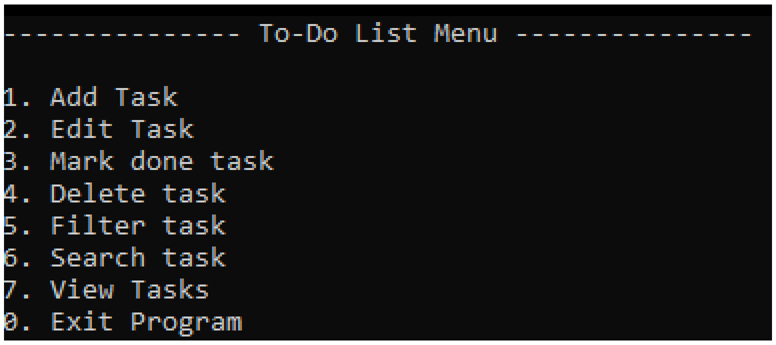
1. Add Task
2. Edit Task
3. Mark Task as Completed
4. Delete Task
5. Filter Tasks
6. Search Task
7. View All Tasks
8. Exit

Each option corresponds to a member function in the `ToDoList` class. The program continues running until the user selects the exit option.

5.Output Screen Shots

- **Main Interface(Program Start)**

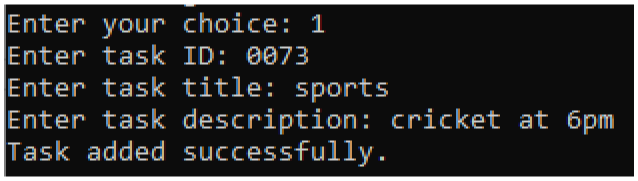
This is the first screen that appears when the program starts.
It shows all options available in the system.



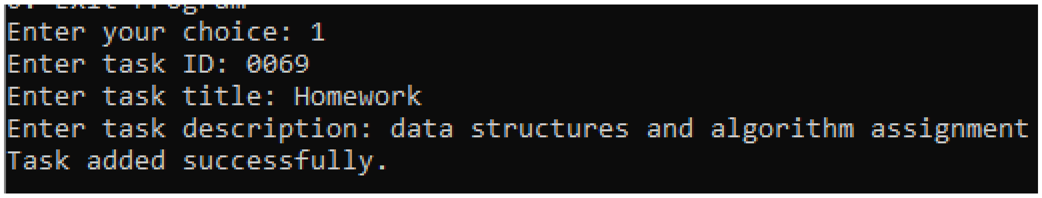
```
--------------- To-Do List Menu ---------------

1. Add Task
2. Edit Task
3. Mark done task
4. Delete task
5. Filter task
6. Search task
7. View Tasks
0. Exit Program
```

1. Add Task Function
This screen allows the user to add a new task.
You enter the task ID, title, and description to create a new task.

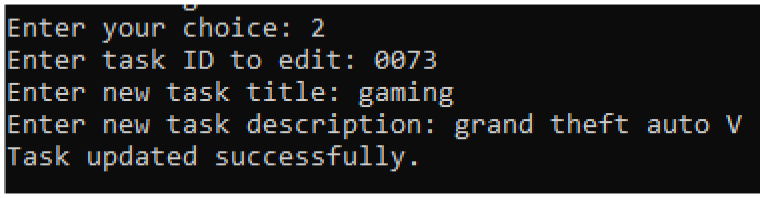


```
Enter your choice: 1
Enter task ID: 0073
Enter task title: sports
Enter task description: cricket at 6pm
Task added successfully.
```



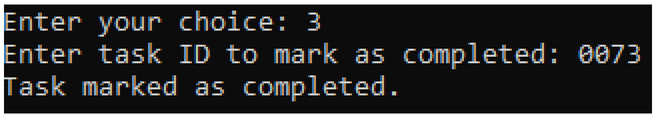
```
Enter your choice: 1
Enter task ID: 0069
Enter task title: Homework
Enter task description: data structures and algorithm assignment
Task added successfully.
```

2. Edit Task Function
This shows how an existing task's title and description can be updated using its ID.



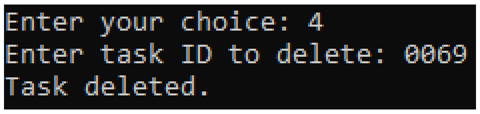
```
Enter your choice: 2
Enter task ID to edit: 0073
Enter new task title: gaming
Enter new task description: grand theft auto V
Task updated successfully.
```

3. Mark Task as Completed Function
This shows marking a specific task as completed using its ID.



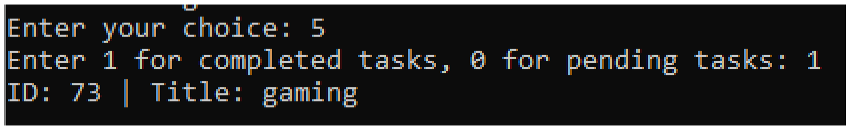
```
Enter your choice: 3
Enter task ID to mark as completed: 0073
Task marked as completed.
```

4. Delete Task Function
This shows deleting a task from the list by entering its ID.



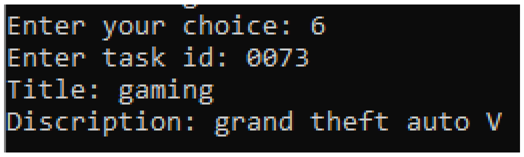
```
Enter your choice: 4
Enter task ID to delete: 0069
Task deleted.
```

5. Filter Tasks Function
This shows how to filter and display tasks based on their status ,either completed or pending.



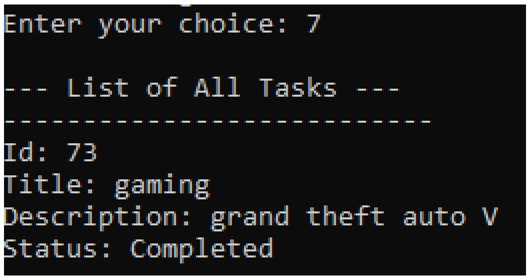
```
Enter your choice: 5
Enter 1 for completed tasks, 0 for pending tasks: 1
ID: 73 | Title: gaming
```

6. Search Task Function
This shows searching for a task by its ID and displaying its details including title and description.



```
Enter your choice: 6
Enter task id: 0073
Title: gaming
Discription: grand theft auto V
```

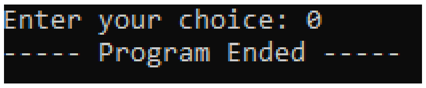
7. View All Tasks Function
This shows the full list of tasks currently in the to-do list, including their status.



```
Enter your choice: 7

--- List of All Tasks ---
---------------------------
Id: 73
Title: gaming
Description: grand theft auto V
Status: Completed
```

8. Exit
Entering 0 exits the program safely.



```
Enter your choice: 0
----- Program Ended -----
```

6. Summary

- This is a **To-Do List Manager built in C++** using **doubly linked lists**.
- The system supports adding, editing, deleting, searching, and filtering tasks.
- Doubly linked list enables dynamic memory usage and flexible insertion/deletion.
- The application runs in a console and is suitable for students or individuals who want to manage tasks programmatically.
- The code follows object-oriented design and is simple to present and understand.

7. Challenges and Conclusion

Challenges
• Managing Doubly Linked List Pointers: Handling dynamic memory with both `next` and `previous` pointers was tricky, especially during deletion and insertion.
• Task ID Management: Ensuring that each task has a unique ID and verifying it while performing operations like edit, delete, and search required consistent checking.
• Input Handling: Switching between integer inputs (like ID) and string inputs (like title and description) required careful handling of `cin` and `getline()` to avoid input being skipped or mixed.

• Completion Status Tracking: Implementing a clean way to mark tasks as completed and show that status in the view function took extra logic.
• Filtering Tasks: Writing a function to filter only completed or only pending tasks without skipping any node required good use of conditions and flags.
• No Sorting Implemented: Tasks are displayed in the order added. Features like sorting by ID, title, or status could enhance usability but are not implemented.
• Edge Cases: Handling edge cases such as empty list conditions, invalid task IDs, and editing non-existent tasks added complexity to the code.
• Console UI Formatting: Keeping the output neat and readable in a console-only environment without using extra libraries required extra effort.

Conclusion

• This project successfully implements a basic To-Do List Management System using a doubly linked list in C++.
• The program allows users to add tasks, edit them, delete them, search by ID, mark them as completed, view all tasks, and filter based on their completion status.
• It provides a hands-on understanding of object-oriented programming, especially the use of classes, constructors, and member functions.
• Helped reinforce the concepts of dynamic memory allocation and linked list traversal.
• Also gave practical experience with input/output handling and condition-based task filtering.
• The program can be further enhanced by adding features like data persistence using file handling, task priority levels, due dates, and automatic task sorting.
• Overall, the project was a useful exercise in building a functional application using core C++ concepts without relying on external libraries.