

**COMSATS University Islamabad, Attock Campus**

|  |  |
| --- | --- |
| Name | Muhammad Awais |
| Assignment | **01** |
| Subject | DS(Theory) |
| Registration No | Sp22-bse-020 |
| Submitted To | Mr. Muhammad Kamran |

Department of Computer Science

**FALL 2024 Assignment**

**Course: -DS**

**Program: BSE-II**

**Introduction:**

The objective of this assignment is to create a simple task management system using a singly linked list in C++. The system allows users to manage tasks, each represented as a node in the list. Each task contains a unique task ID, a description, and a priority level. The tasks are organized based on priority, with higher-priority tasks appearing first in the list. The implemented system provides the following operations:

1. **Add a new task** based on its priority.

2. **Remove the highest priority task** (i.e., the task at the head of the list).

3. **Remove a task by its unique ID.**

4. **View all tasks in the system.**

This task management system is built using a singly linked list, which allows efficient insertion and removal of tasks while maintaining task order based on priority.

**Code Explanation:**

**1. Task Structure:**

* The `Task` structure represents each task. It contains:
* `id`: A unique integer ID for each task.
* `description`: A string describing the task.
* `priority`: An integer representing the task's priority (higher numbers indicate higher priority).
* `next`: A pointer to the next task in the list.

**2. TaskList Class:**

- This class manages the singly linked list and provides methods to add, remove, and view tasks.

**3. `addTask(int id, const string& description, int priority)`:**

- **Logic:** This function inserts a new task into the list based on its priority. If the list is empty or if the task has the highest priority, it is added to the front of the list. Otherwise, the function traverses the list to find the correct position where the new task should be inserted, ensuring that tasks are always sorted by priority.

**4. `removeHighestPriorityTask()`:**

- **Logic:** This function removes the task with the highest priority, which is always the task at the head of the list. If the list is empty, the function informs the user that there are no tasks to remove. Otherwise, it removes the head node and shifts the head pointer to the next task.

**5. `removeTaskById(int id)`:**

**- Logic:** This function removes a specific task by its unique ID. It traverses the list to find the task with the matching ID. If the task is found at the head of the list, the head is updated. If the task is found further down the list, the `next` pointers are adjusted to skip over the node to be deleted.

**6. `viewTasks()`:**

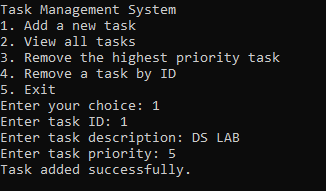
**- Logic:** This function displays all tasks in the list. It traverses the list from the head to the end, printing each task's ID, description, and priority. If the list is empty, it informs the user that no tasks are present.

**7. Destructor (`~TaskList()`):**

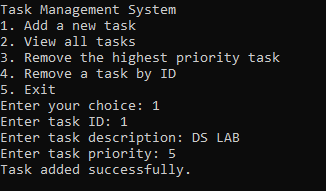
**- Logic:** This destructor ensures that all dynamically allocated memory for the tasks is freed when the `TaskList` object is destroyed. It does this by traversing the list and deleting each node.

**Screenshots:**

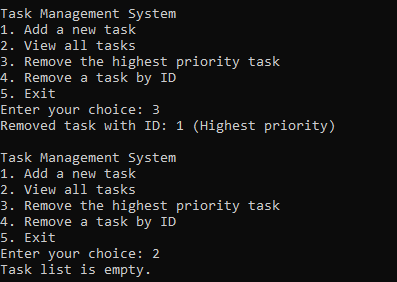
**Adding a Task:**



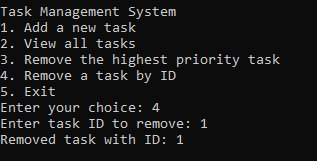
**Viewing tasks:**



**Removing the highest priority task.**



**Removing a task by ID.**



**Conclusion:**

Through this assignment, I gained a deeper understanding of how linear data structures, specifically singly linked lists, can be used to efficiently manage dynamic data such as tasks in a task management system. Implementing task prioritization using a linked list enhanced my knowledge of list traversal and insertion at the correct position based on priority. I also learned the importance of memory management in C++, as the creation and deletion of nodes required careful handling to avoid memory leaks.

One of the main challenges I faced was ensuring that the tasks were always inserted in the correct order while maintaining an efficient insertion time. Additionally, handling edge cases, such as removing tasks from an empty list or searching for a task by an invalid ID, required thoughtful checks and error handling. Overall, this project strengthened my problem-solving skills and provided valuable insights into data structure implementation in C++.