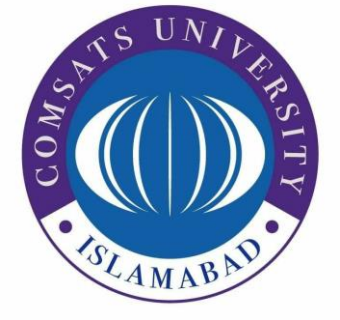
COMSATS University Islamabad

Attock Campus



Department Of Computer Sciences

NAME:	**Amber Saba**
REG.NO:	**Sp23-bcs-062**
SUBMITTED TO:	**Sir Kamran**
COURSE TITLE:	**Data Structures**
DATE:	**23/09/2024**
ASSIGNMENT :	**1**

REPORT

Introduction:

The objective of this assignment is to create a task management system using a **singly linked list** in C++. Each task is represented as a node in the linked list, and the system is designed to manage tasks based on their **priority**. The higher the priority, the earlier the task appears in the list. The key operations implemented in this system include:

1. Adding a new task in the appropriate position based on priority.
2. Removing the task with the highest priority.
3. Removing a specific task by its ID.
4. Viewing all tasks in the system.

These operations provide efficient management of tasks in a dynamic environment where tasks can be added or removed based on priority and task ID.

Code Explanation:

1. **`addTask(int id, string description, int priority)`:**
   - This function is responsible for adding a new task to the task list. A new task is represented as a node with a unique ID, description, and priority.
   - The logic ensures that the new task is inserted at the correct position in the linked list based on its priority. Tasks with higher priority are placed before tasks with lower priority.
   - If the list is empty or the new task has a higher priority than the current head, the new task becomes the head of the list.
   - Otherwise, the function traverses the list and inserts the task at the appropriate position.
2. **`removeHighestPriorityTask()`:**
   - This function removes the task with the highest priority, which is always located at the head of the list.
   - If the list is empty, it notifies the user that there are no tasks to remove.
   - Otherwise, the head node is deleted, and the head pointer is updated to the next task in the list.
3. **`removeTaskById(int id)`:**
   - This function removes a specific task by its unique ID.
   - It first checks if the head contains the task with the matching ID. If so, the head is removed.
   - If the task is not at the head, the function traverses the list to locate the task with the matching ID and removes it by adjusting the pointers of the previous node.
   - If no task with the given ID is found, it notifies the user that the task does not exist.
4. **`viewTasks()`:**
   - This function displays all the tasks in the list, starting from the head.
   - If the list is empty, it informs the user that there are no tasks to display.
   - It iterates through the list and prints the ID, description, and priority of each task.
   -

5. **`TaskManager()`:**
    - The destructor is responsible for freeing up memory by deleting all nodes in the linked list when the program ends.
    - It ensures that all dynamically allocated memory for tasks is properly released to prevent memory leaks.

Code Output:

```
Task Management System
1. Add a new task
2. View all tasks
3. Remove the highest priority task
4. Remove a task by ID
5. Exit
Enter your choice: 1
Enter task ID: 12
Enter task description: adding accounts
Enter task priority: 1
Task added successfully!
```

```
Task Management System
1. Add a new task
2. View all tasks
3. Remove the highest priority task
4. Remove a task by ID
5. Exit
Enter your choice: 2
ID: 12, Description: adding accounts , Priority: 1
```

Summary:

This task management system uses a singly linked list to manage tasks dynamically, allowing efficient task insertion based on priority, task removal by priority or ID, and viewing the current task list. The operations are straightforward, ensuring ease of use and maintaining tasks in an organized way.

==========