Comsats University Islamabad, Attock Campus

Department of Computer Science

Task Management System using Singly Linked List

Name:	Muhammad Fahad
Enrollment Number:	SP23-BSE-010
Date:	24/09/2024
Assignment No:	01
Submitted To:	Sir M. Kamran

Introduction

In this assignment, we're building a simple **Task Management System** using a **Singly Linked List**. Each task is represented by a node in the list, containing a **unique task ID**, **description**, and a **priority level**. The main goal of this system is to manage tasks based on their priority—higher priority tasks are placed first.

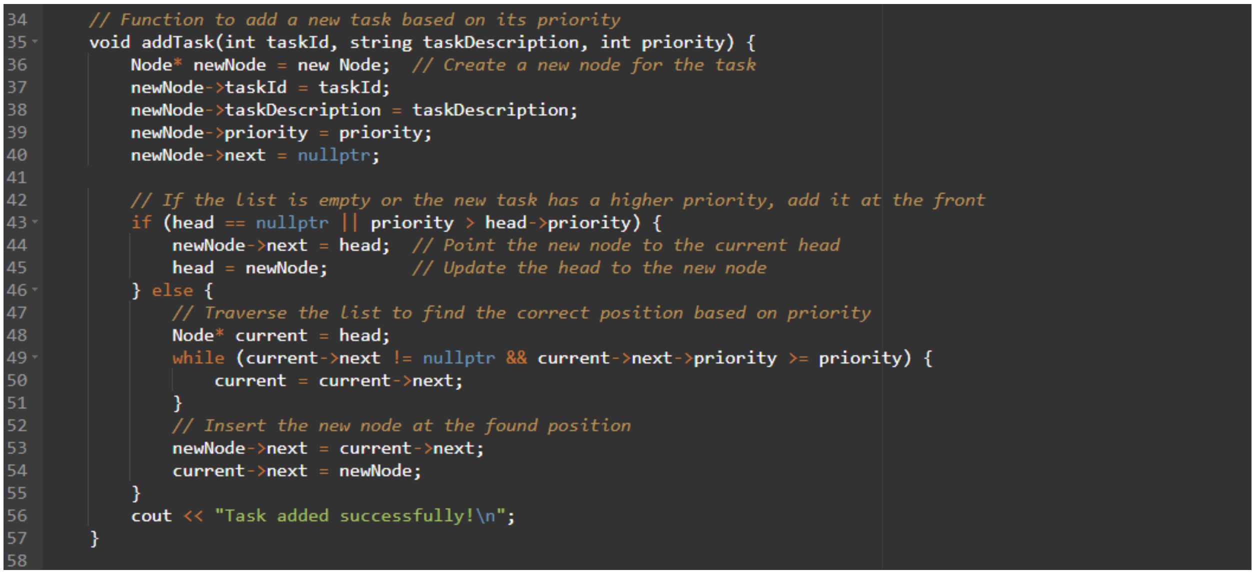
Operations Implemented

1. **Add Task**: Add a new task at the correct position in the list based on its priority.
2. **Remove Highest Priority Task**: Removes the task with the highest priority (the first node).
3. **Remove Task by ID**: Removes a specific task from the list by its ID.
4. **View All Tasks**: Displays all tasks in the list.

Code Explanation

1. Adding a Task

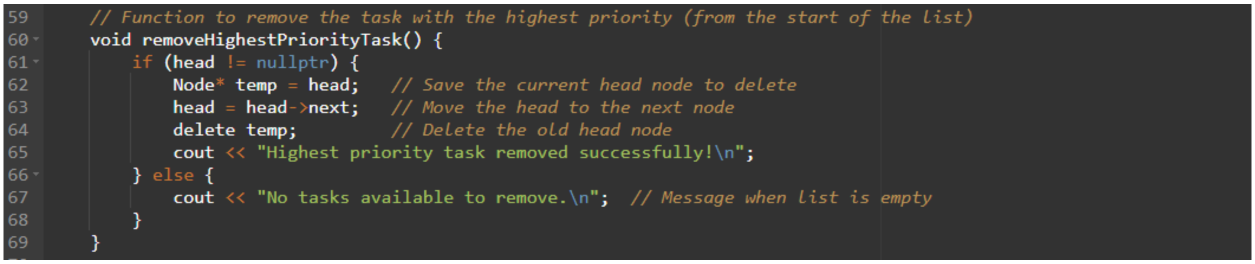
This function allows the user to add a task by providing an ID, description, and priority. The task is inserted in the list at a position that maintains the order of priority. If the new task has the highest priority, it's added to the front of the list.



```
    // Function to add a new task based on its priority
    void addTask(int taskId, string taskDescription, int priority) {
        Node* newNode = new Node;  // Create a new node for the task
        newNode->taskId = taskId;
        newNode->taskDescription = taskDescription;
        newNode->priority = priority;
        newNode->next = nullptr;

        // If the list is empty or the new task has a higher priority, add it at the front
        if (head == nullptr || priority > head->priority) {
            newNode->next = head;  // Point the new node to the current head
            head = newNode;        // Update the head to the new node
        } else {
            // Traverse the list to find the correct position based on priority
            Node* current = head;
            while (current->next != nullptr && current->next->priority >= priority) {
                current = current->next;
            }
            // Insert the new node at the found position
            newNode->next = current->next;
            current->next = newNode;
        }
        cout << "Task added successfully!\n";
    }
```

2. Removing the Highest Priority Task

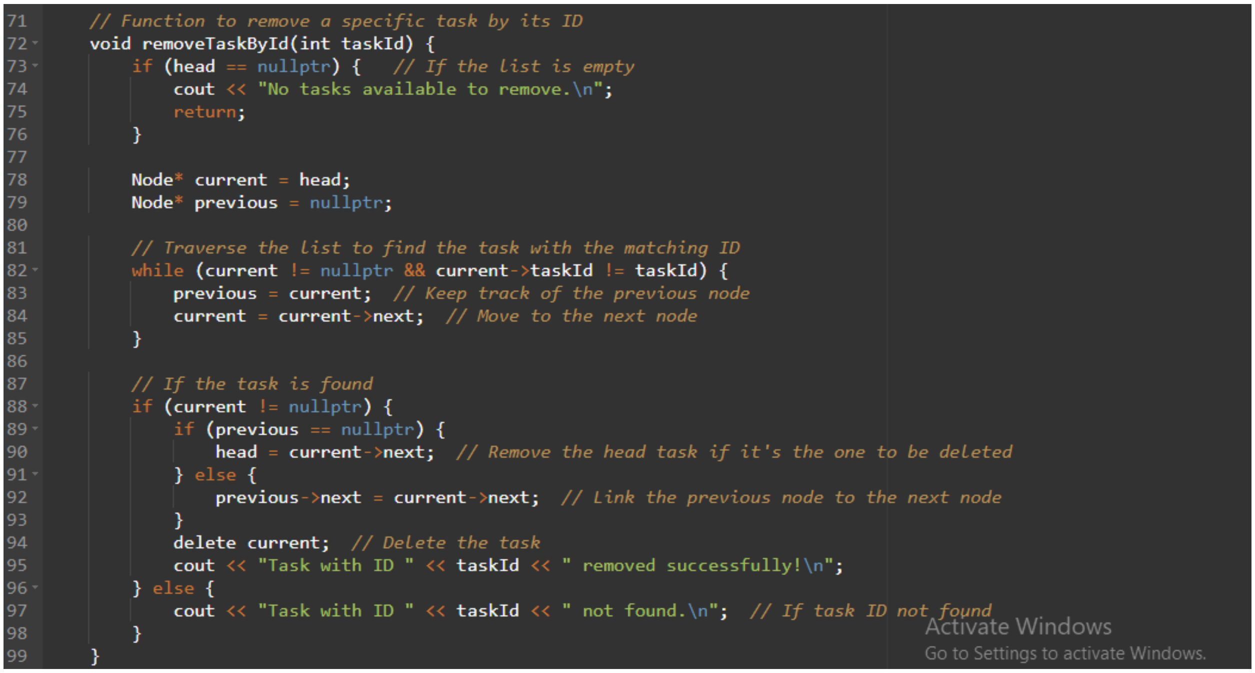
This function removes the first task in the list, which is always the one with the highest priority. If the list is empty, it prints a message.



```cpp
// Function to remove the task with the highest priority (from the start of the list)
void removeHighestPriorityTask() {
    if (head != nullptr) {
        Node* temp = head;   // Save the current head node to delete
        head = head->next;   // Move the head to the next node
        delete temp;         // Delete the old head node
        cout << "Highest priority task removed successfully!\n";
    } else {
        cout << "No tasks available to remove.\n";  // Message when list is empty
    }
}
```

3. Removing a Task by ID

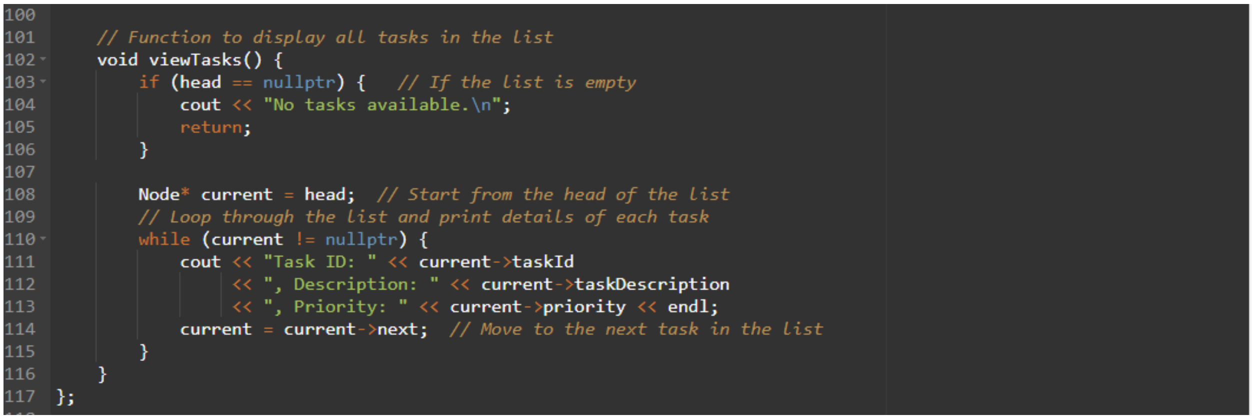
This function allows the user to remove a task by specifying its ID. It traverses the list to find the task, and if found, removes it. Otherwise, it notifies the user that the task ID wasn’t found.



```cpp
// Function to remove a specific task by its ID
void removeTaskById(int taskId) {
    if (head == nullptr) {   // If the list is empty
        cout << "No tasks available to remove.\n";
        return;
    }

    Node* current = head;
    Node* previous = nullptr;

    // Traverse the list to find the task with the matching ID
    while (current != nullptr && current->taskId != taskId) {
        previous = current;  // Keep track of the previous node
        current = current->next;  // Move to the next node
    }

    // If the task is found
    if (current != nullptr) {
        if (previous == nullptr) {
            head = current->next;  // Remove the head task if it's the one to be deleted
        } else {
            previous->next = current->next;  // Link the previous node to the next node
        }
        delete current;  // Delete the task
        cout << "Task with ID " << taskId << " removed successfully!\n";
    } else {
        cout << "Task with ID " << taskId << " not found.\n";  // If task ID not found
    }
}
```

4. Viewing All Tasks

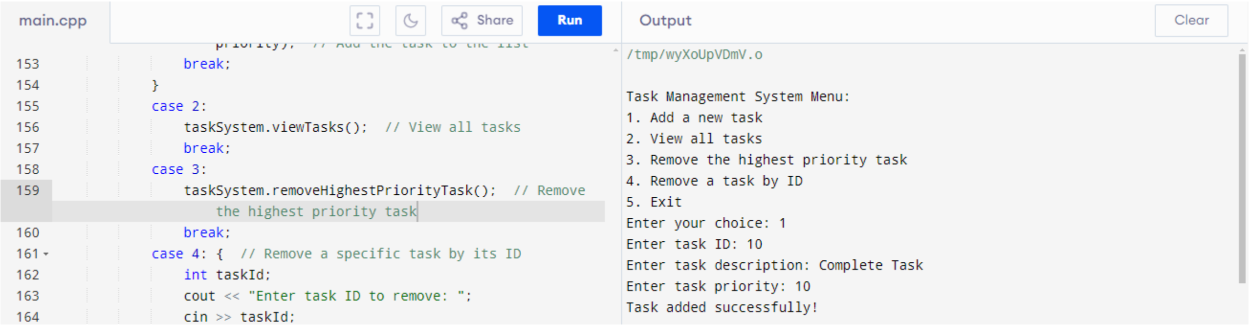
This function prints all tasks in the list, showing their ID, description, and priority. If the list is empty, it shows a message saying there are no tasks to display.



```cpp

    // Function to display all tasks in the list
    void viewTasks() {
        if (head == nullptr) {   // If the list is empty
            cout << "No tasks available.\n";
            return;
        }

        Node* current = head;  // Start from the head of the list
        // Loop through the list and print details of each task
        while (current != nullptr) {
            cout << "Task ID: " << current->taskId
                 << ", Description: " << current->taskDescription
                 << ", Priority: " << current->priority << endl;
            current = current->next;  // Move to the next task in the list
        }
    }
};
```

Program Output (Screenshots)

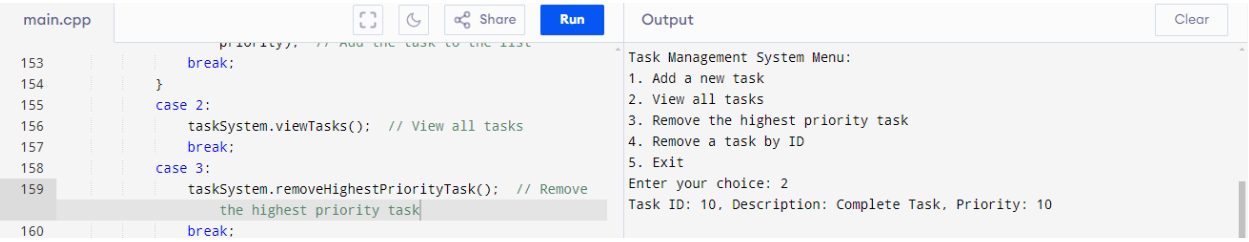
- **Adding a Task**



```cpp
                priority);  // Add the task to the list
            break;
        }
        case 2:
            taskSystem.viewTasks();  // View all tasks
            break;
        case 3:
            taskSystem.removeHighestPriorityTask();  // Remove
                the highest priority task
            break;
        case 4: {  // Remove a specific task by its ID
            int taskId;
            cout << "Enter task ID to remove: ";
            cin >> taskId;
```

```
/tmp/wyXoUpVDmV.o

Task Management System Menu:
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: 10
Enter task description: Complete Task
Enter task priority: 10
Task added successfully!
```

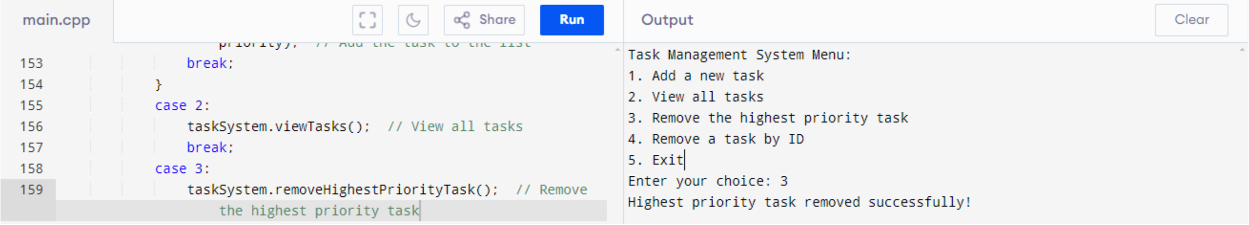
- **Viewing Tasks**



```cpp
                priority);  // Add the task to the list
            break;
        }
        case 2:
            taskSystem.viewTasks();  // View all tasks
            break;
        case 3:
            taskSystem.removeHighestPriorityTask();  // Remove
                the highest priority task
            break;
```

```
Task Management System Menu:
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
Task ID: 10, Description: Complete Task, Priority: 10
```

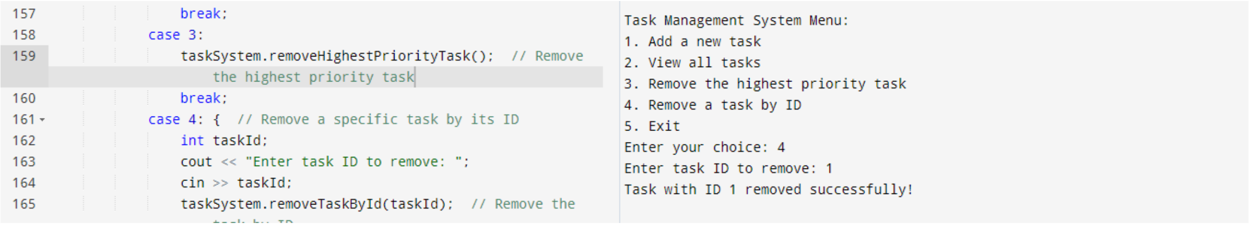
- **Removing Highest Priority Task**



```
                priority);  // Add the task to the list
            break;
        }
        case 2:
            taskSystem.viewTasks();  // View all tasks
            break;
        case 3:
            taskSystem.removeHighestPriorityTask();  // Remove
                the highest priority task
```

```
Task Management System Menu:
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: 3
Highest priority task removed successfully!
```

- **Removing Task by ID**



```
            break;
        case 3:
            taskSystem.removeHighestPriorityTask();  // Remove
                the highest priority task
            break;
        case 4: {  // Remove a specific task by its ID
            int taskId;
            cout << "Enter task ID to remove: ";
            cin >> taskId;
            taskSystem.removeTaskById(taskId);  // Remove the
                task by ID
```

```
Task Management System Menu:
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: 4
Enter task ID to remove: 1
Task with ID 1 removed successfully!
```

Conclusion

This task management system was a great way to apply the concept of **singly linked lists**. By implementing task insertion based on priority, we learned how to traverse and manipulate linked lists efficiently. One of the challenges was ensuring that tasks were inserted in the correct order, but after some careful list traversal, it worked out perfectly! 🎉

THE END