COMSATS UNIVERSITY ISLAMABAD ATTOCK CAMPUS
DEPARTMENT OF COMPUTER SCIENCE

NAME KUBRA

REGISTRATION NUMBER SP23-BSE-008

ASSIGNMENT NO 01

COURSE DATA STRUCTURE

SUBMITTED TO MUHAMMAD KAMRAN

DATE 25 SEPTEMBER 2024

Report on Task Management System Using Singly Linked List

Introduction

The objective of this assignment is to develop a task management system using a singly linked list. This program allows users to manage tasks efficiently by enabling operations such as adding tasks with varying priority levels, viewing all tasks, removing the highest priority task, and removing a task by its ID. The linked list structure ensures that tasks are stored in order of priority, allowing for efficient retrieval and deletion.

Code Explanation

1. Structure Definitions

Task Structure Represents an individual task.

TaskID: Unique identifier for each task.

description : A brief description of the task.

Priority: The priority level of the task (higher values indicate higher priority).

next: Pointer to the next task in the list.

TaskList Structure: Represents the entire list of tasks.

head: Pointer to the first task in the list.

Constructor

cpp

```
TaskList() {
    head = NULL;  // Initialize head to NULL
}
```

- Initializes the task list with no tasks by setting the `head` pointer to `NULL`.

Functions:

1. addTask(int id, string description, int priority)

   - Adds a new task to the list in descending order of priority.

2.`removeHighestPriorityTask()

- Removes the task with the highest priority (the head of the list).

3.`removeTaskById(int id)

- Removes a specific task from the list based on its task ID.

4. viewAllTasks() const`

- Displays all tasks in the list with their details.

Let me know if you need anything else!

3. Adding a Task

```cpp
void addTask(int id, string description, int priority) {
    Task newTask = new Task;
    newTask->taskId = id;
    newTask->description = description;
    newTask->priority = priority;
    newTask->next = NULL;

    if (head == NULL || newTask->priority > head->priority) {
        newTask->next = head;
        head = newTask;
    } else {
        Task* current = head;
        while (current->next != NULL && current->next->priority > newTask->priority) {
            current = current->next;
        }
        newTask->next = current->next;
```

```
        current->next = newTask;
    }
}
```

- This function creates a new task and inserts it into the linked list in descending order of priority.
- If the list is empty or the new task has a higher priority than the current head, it becomes the new head.
- Otherwise, it traverses the list to find the appropriate position for the new task.

4. Removing the Highest Priority Task

```cpp
void removeHighestPriorityTask() {
    if (head != NULL) {
        Task* temp = head;
        head = head->next;
        delete temp;
        cout << "Highest priority task removed." << endl;
    } else {
        cout << "No tasks to remove." << endl;
    }
}
```

- Removes the task at the head of the list (highest priority).
- If the list is empty, it informs the user.

5. Removing a Task by ID

```cpp
void removeTaskById(int id) {
    if (head == NULL) {
        cout << "Task list is empty." << endl;
        return;
```

```
    }
    if (head->taskId == id) {
        Task* temp = head;
        head = head->next;
        delete temp;
        cout << "Task with ID" << id << " removed." << endl;
        return;
    }

    Task* current = head;
    while (current->next != NULL) {
        if (current->next->taskId == id) {
            Task* temp = current->next;
            current->next = temp->next;
            delete temp;
            cout << "Task with ID" << id << " removed." << endl;
            return;
        }
        current = current->next;
    }
    cout << "Task with ID" << id << " not found." << endl;
}
```

- This function removes a specific task by its ID.
- If the list is empty, it notifies the user. If the task is found, it is removed; otherwise, an error message is displayed.

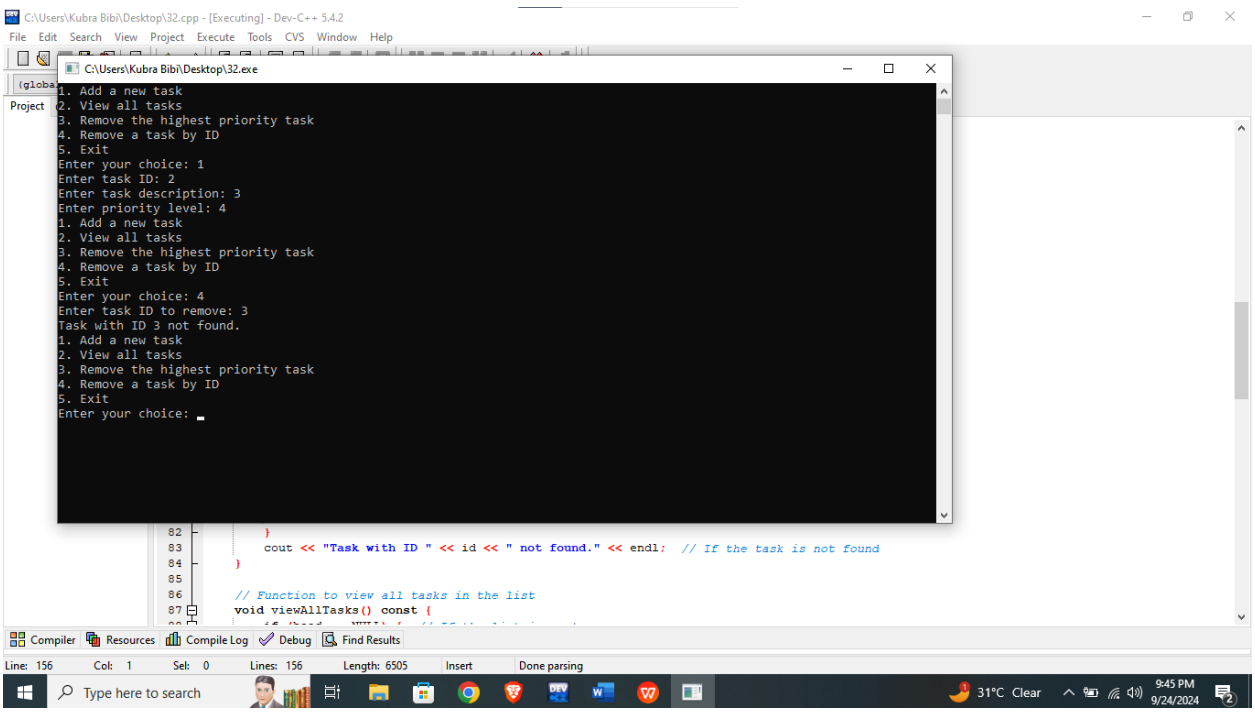
6. Viewing All Tasks

```cpp
void viewAllTasks() const {
```

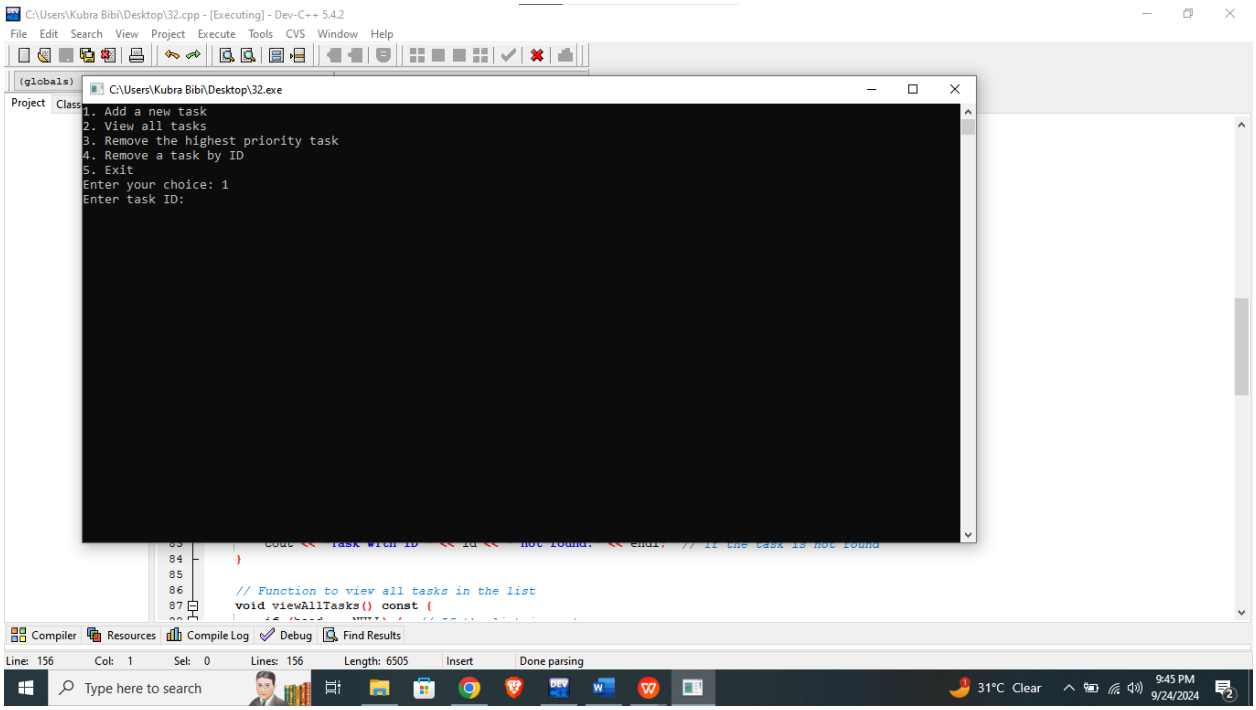
```
    if (head == NULL) {
        cout << "No tasks in the list." << endl;
        return;
    }

    Task* current = head;
    while (current != NULL) {
        cout << "Task ID: " << current->taskId << endl;
        cout << "Description: " << current->description << endl;
        cout << "Priority: " << current->priority << endl;
        cout << endl;
        current = current->next;
    }
}
```

- This function displays all tasks in the list, showing their ID, description, and priority.

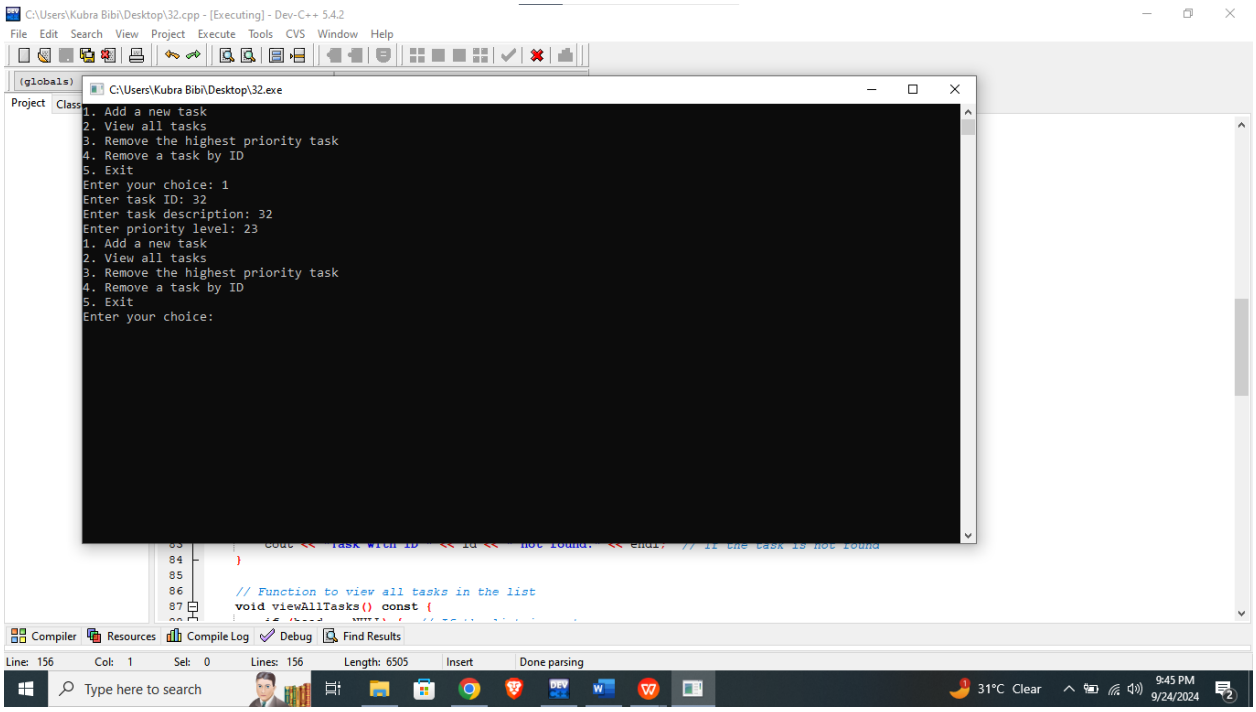
Screenshots



VEIWING TASKS



REMOVING



1. **Adding a Task**: Capture the output after adding a new task.

   ![Add Task](add_task_screenshot.png)

2. **Viewing All Tasks**: Capture the output after viewing all tasks.

   ![View All Tasks](view_all_tasks_screenshot.png)

3. **Removing the Hghest Priority Task**: Capture the output after removing the highest priority task.

   ![Remove Hghest Priority Task](remove_highest_priority_task_screenshot.png)

4. **Removing a Task by ID**: Capture the output after removing a specific task by its ID.

   ![Remove Task by ID](remove_task_by_id_screenshot.png)

5. Attempting to View or Remove from an Empty List**: Capture the output when trying to view or remove from an empty list.

   ![Empty List](empty_list_screenshot.png)

Conclusion

This task management system effectively demonstrates the use of a singly linked list for dynamic task management. The operations implemented allow for efficient addition, viewing, and removal of tasks based on priority, showcasing the advantages of using linked lists over arrays for such applications.Operations are used in the link list are following adding task,viewing ,removing and further more