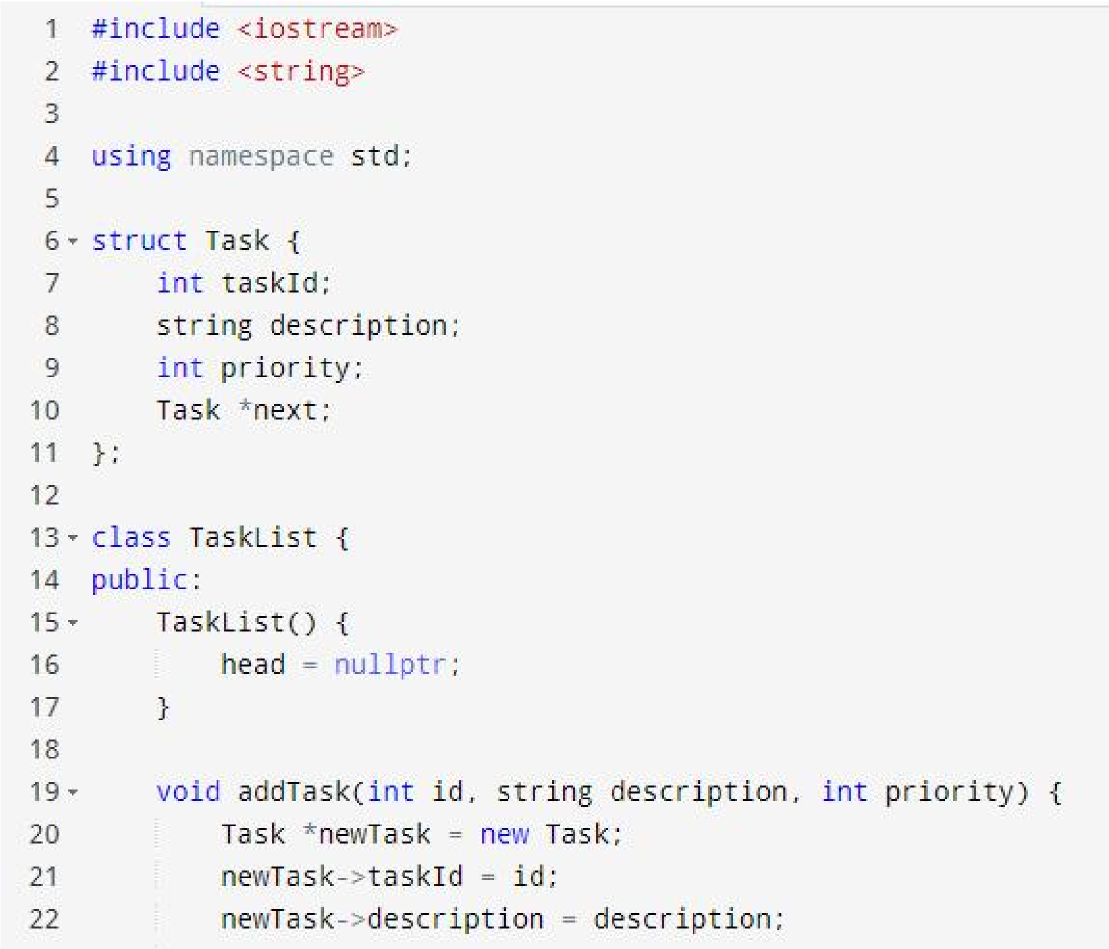
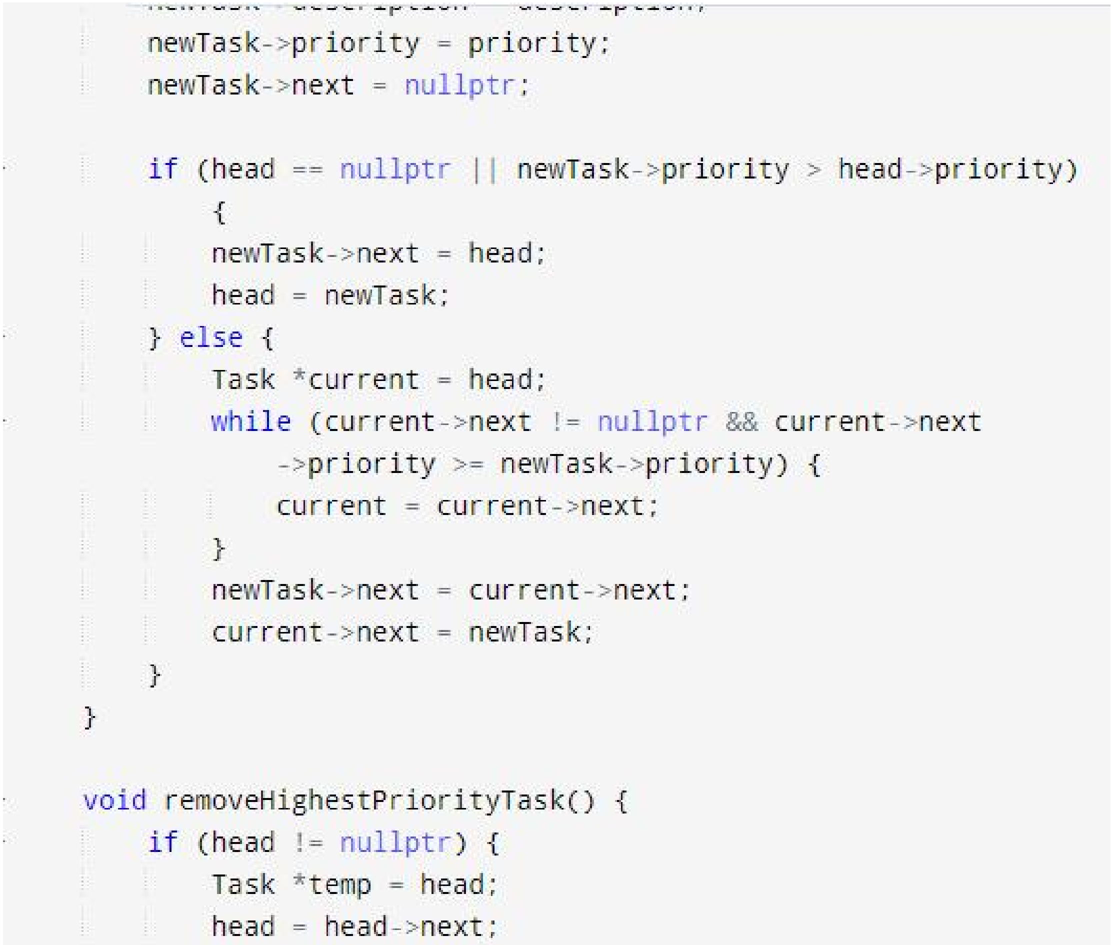
COMSATS UNIVERSITY ISLAMABAD ATTOCK CAMPUS

DEPARTMENT OF COMPUTER SCIENCE

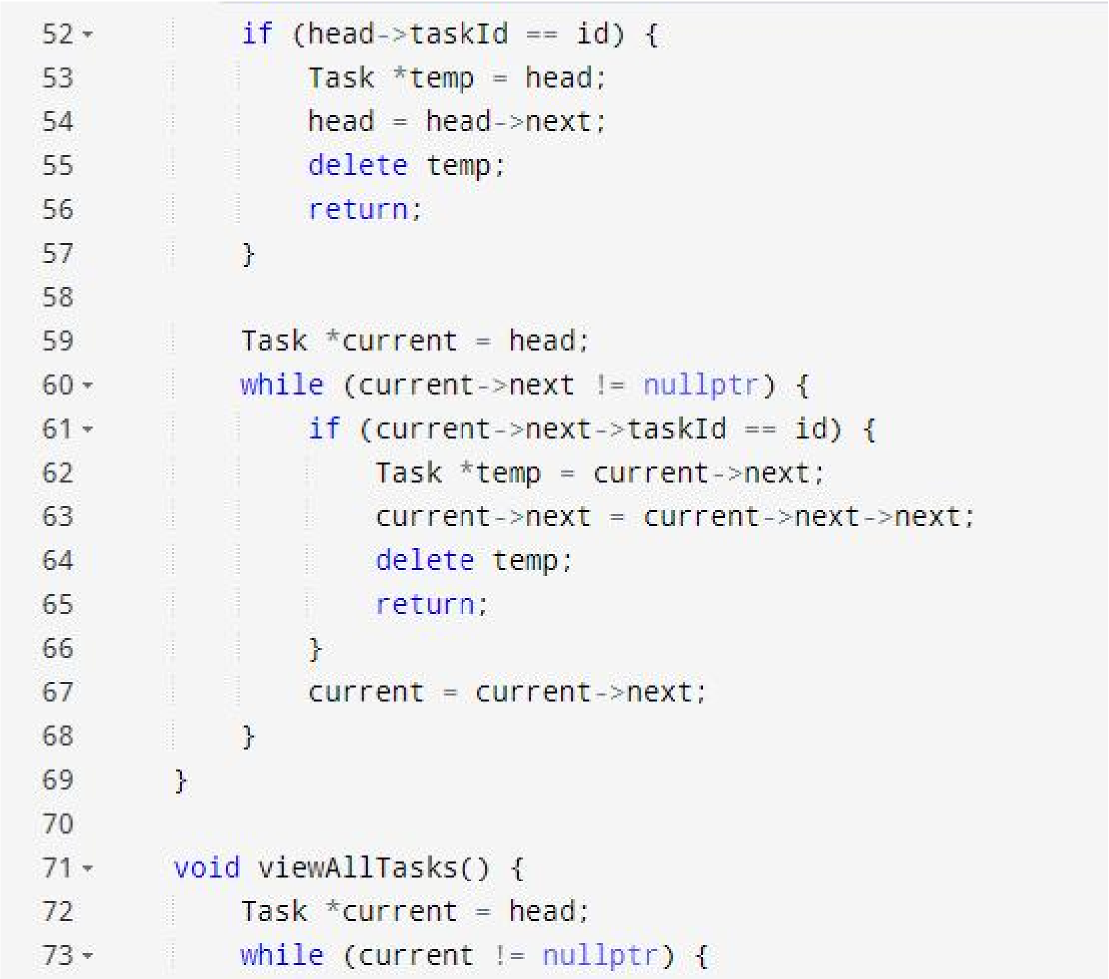
NAME	***:***	***Hassan Fayyaz***
REG. NO	***:***	***FA22-BSE-040***
SUBJECT	***:***	***Data Structures***
ASSIGNMENT	***:***	***01***
Date	***:***	***24 September, 2024***
SUBMITTED TO	***:***	***Mr Kamran***



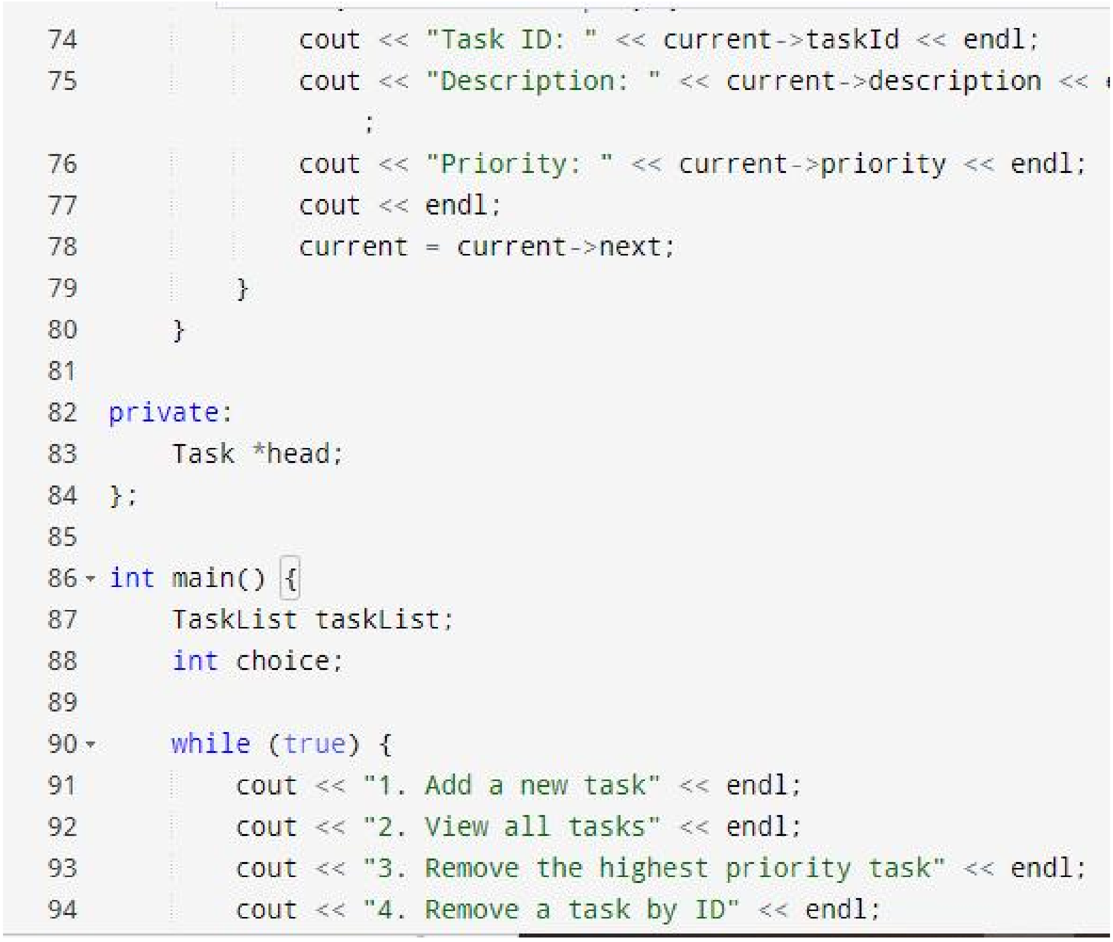
```
#include <iostream>
#include <string>

using namespace std;

struct Task {
    int taskId;
    string description;
    int priority;
    Task *next;
};

class TaskList {
public:
    TaskList() {
        head = nullptr;
    }

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



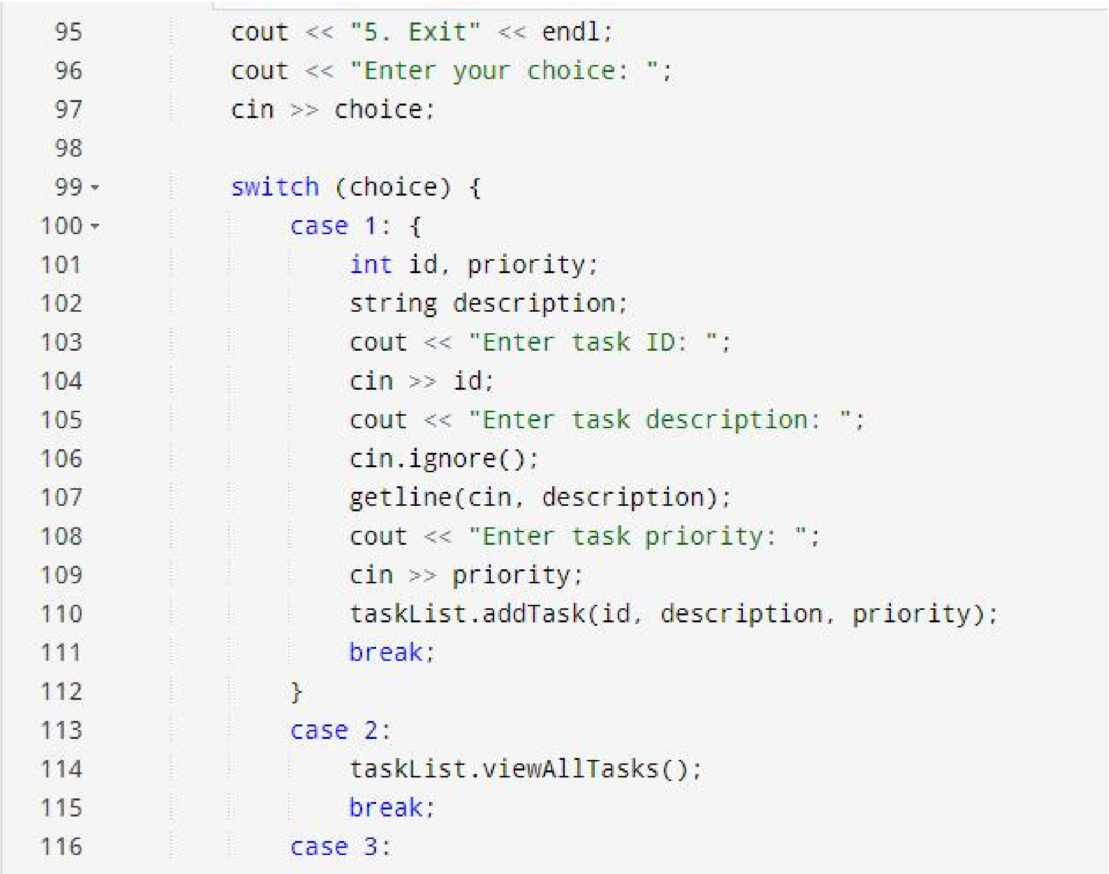
```
    newTask->description = description;
    newTask->priority = priority;
    newTask->next = nullptr;

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

void removeHighestPriorityTask() {
    if (head != nullptr) {
        Task *temp = head;
        head = head->next;
```



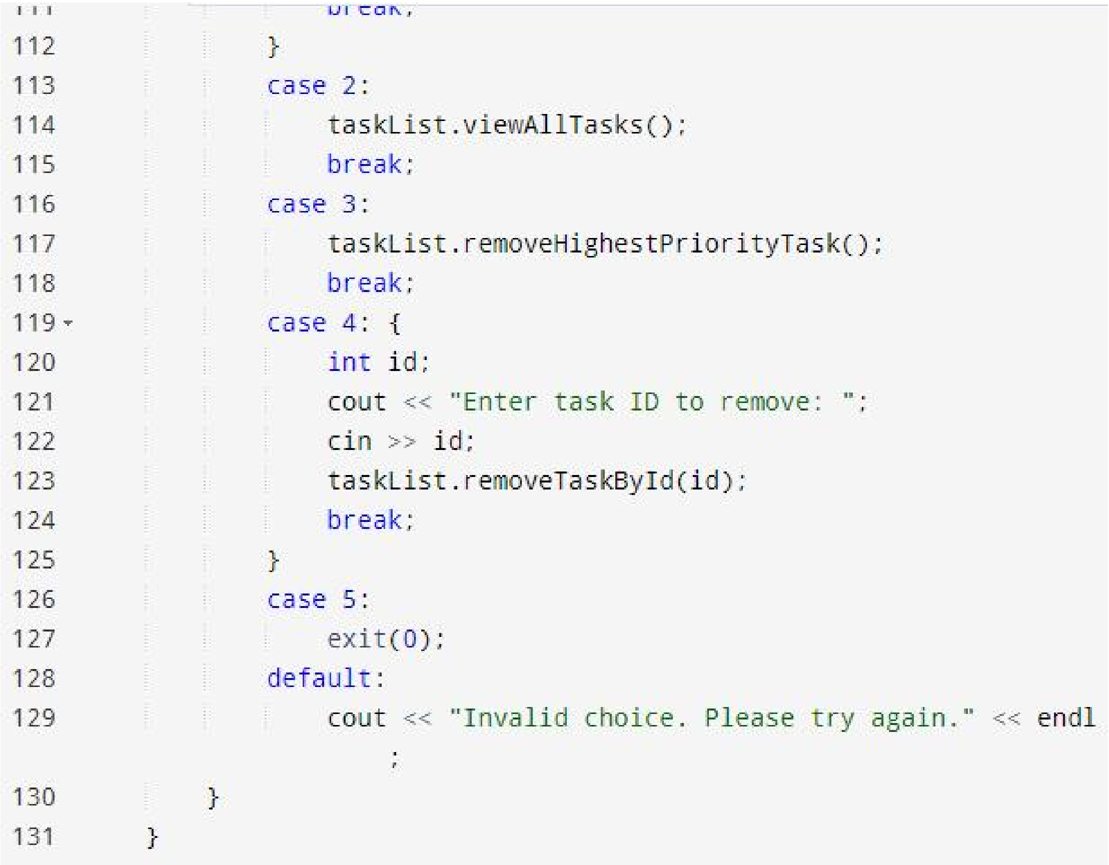
```
        if (head->taskId == id) {
            Task *temp = head;
            head = head->next;
            delete temp;
            return;
        }

        Task *current = head;
        while (current->next != nullptr) {
            if (current->next->taskId == id) {
                Task *temp = current->next;
                current->next = current->next->next;
                delete temp;
                return;
            }
            current = current->next;
        }
    }

    void viewAllTasks() {
        Task *current = head;
        while (current != nullptr) {
```



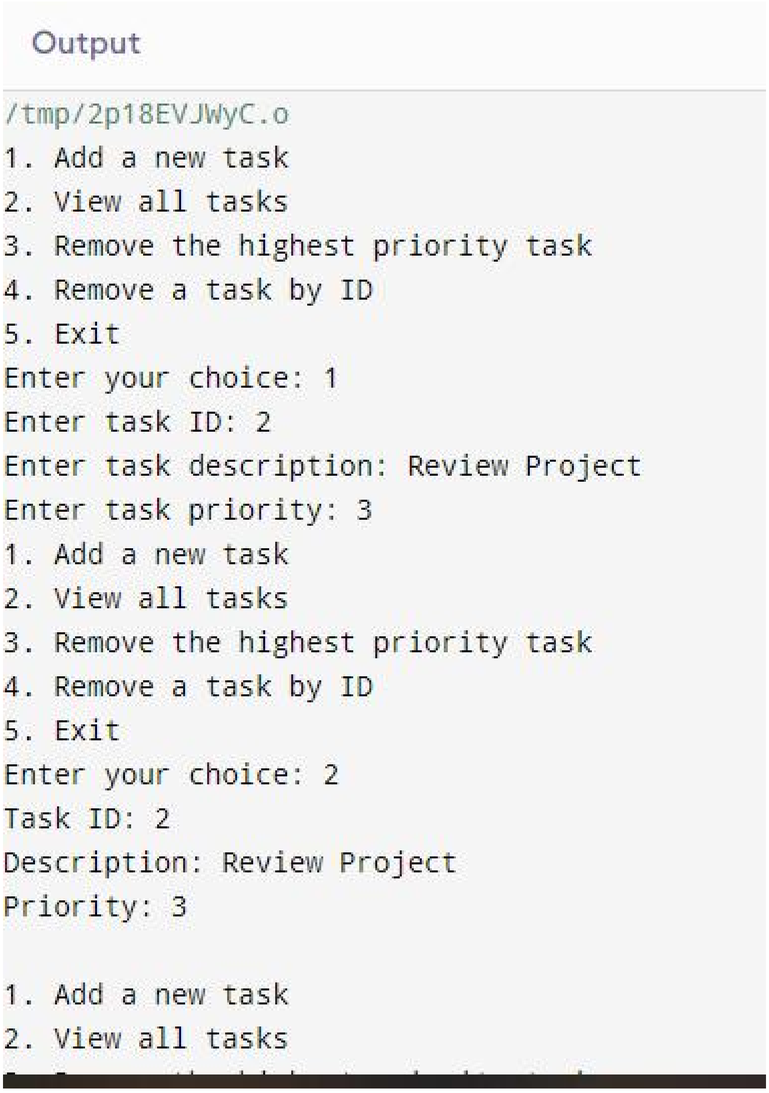
```
            cout << "Task ID: " << current->taskId << endl;
            cout << "Description: " << current->description << 
                ;
            cout << "Priority: " << current->priority << endl;
            cout << endl;
            current = current->next;
        }
    }

private:
    Task *head;
};

int main() {
    TaskList taskList;
    int choice;

    while (true) {
        cout << "1. Add a new task" << endl;
        cout << "2. View all tasks" << endl;
        cout << "3. Remove the highest priority task" << endl;
        cout << "4. Remove a task by ID" << endl;
```



```cpp
    cout << "5. Exit" << endl;
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
        case 1: {
            int id, priority;
            string description;
            cout << "Enter task ID: ";
            cin >> id;
            cout << "Enter task description: ";
            cin.ignore();
            getline(cin, description);
            cout << "Enter task priority: ";
            cin >> priority;
            taskList.addTask(id, description, priority);
            break;
        }
        case 2:
            taskList.viewAllTasks();
            break;
        case 3:
```



```
                break;
            }
            case 2:
                taskList.viewAllTasks();
                break;
            case 3:
                taskList.removeHighestPriorityTask();
                break;
            case 4: {
                int id;
                cout << "Enter task ID to remove: ";
                cin >> id;
                taskList.removeTaskById(id);
                break;
            }
            case 5:
                exit(0);
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    }
```



```
/tmp/2p18EVJWyC.o
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: 2
Enter task description: Review Project
Enter task priority: 3
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: 2
Description: Review Project
Priority: 3

1. Add a new task
2. View all tasks
```

Report

Introduction:

The purpose of this assignment is to create a task management system using object-oriented programming (OOP) in C++. The system uses a linked list structure to manage tasks, giving each task an ID, a description, and a priority. Using the program, the user can add new tasks, remove tasks based on ID or priority, and view all of the tasks in order of priority. The main objectives are to work with dynamic memory, construct linked list data structures, and gain proficiency with basic task management operations.

Interpretation of Code

Work Organization:

This establishes the framework for every task. Each assignment consists of:
taskId: A unique task identifier represented by an integer.
description: A string containing an assignment synopsis.
priority: An integer that indicates the task's level of importance; tasks with higher priorities will be listed first.
next: A pointer that joins one task to the next to form a linked list.

Class Task List:

The TaskList class, which also provides functionality for adding, removing, and viewing tasks, is responsible for managing all of the tasks.

Builder:

Initializes the task list by setting the head pointer to nullptr. The head pointer keeps track of the beginning of the linked list.

Destroyer:

Memory leaks are avoided by the destructor, which makes sure that all dynamically allocated memory is released when the object is destroyed. As the list is scanned, every node is removed.

AddTask(int priority, int id, strl description):

This function adds a new task to the list at the appropriate position based on its priority. If the list is empty or the new task is more important than the previous task, it becomes the new head. If not, the function moves the new task through the list iteratively to decide where it belongs. Logic dictates the order in which higher priority tasks are listed. Lower priority tasks are arranged at the end or in between depending on their significance.

Delete Task With Maximum Priority:

This function removes the task that has the highest priority, which is always the first task in the list (the head). The function deletes the head and updates the head pointer to the next task.
If there are no tasks to remove, a message stating as much is displayed.

TaskById extraction (int id):

This function removes a task based on its ID. In the event that the task with the given ID is the head, the head is deleted. If not, it removes the task by looking through the list for the one that matches the ID.
If there are no tasks with the specified ID that the function can find, it displays an appropriate message.

ViewAllTasks:

This function, starting at the top of the list, prints the task ID, description, and priority for each task in the list.

If there are no tasks to display, the user is informed that the list is empty.

Principal Role:

Menu Loop: An infinite loop in the main() function shows the user a menu with options to add a new task, view all tasks, remove the task with the highest priority, remove a task by ID, or quit the application.

For each option, the corresponding method from the TaskList class is called in order to accomplish the desired action.

Until the user chooses the "Exit" option, the loop never ends.

Input validation is used by the program to make sure the user enters valid data.

Conclusion:

I learned how to comprehend and use dynamic data structures like linked lists in C++ thanks to this assignment. Important concepts of Object-Oriented Programming (OOP) like data abstraction, constructors, destructors, and pointers were reinforced. The largest difficulty I encountered was making sure that every operation—particularly those involving dynamic memory allocation and task addition or removal—was completed in a timely manner.

I also gained experience with error handling and input validation, two things that are essential to the robustness of programs, thanks to the program.Finally, I discovered that it's critical to correctly clean up dynamically allocated memory because doing otherwise can result in memory leaks and bugs that might be challenging to find in more intricate programs.