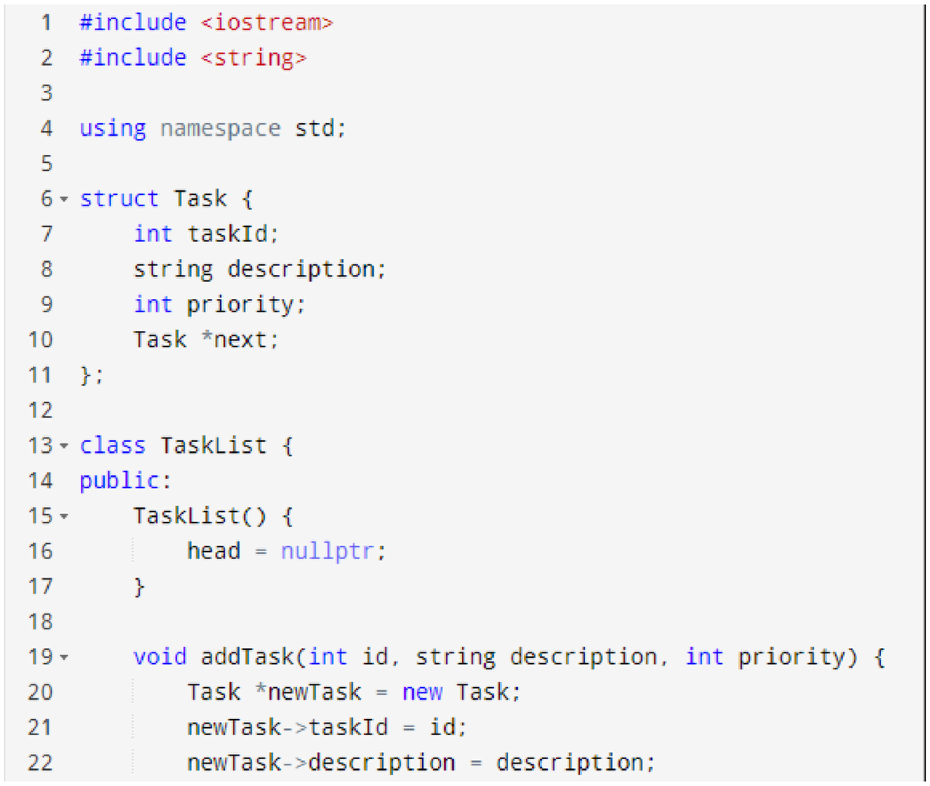
COMSATS UNIVERSITY ISLAMABAD

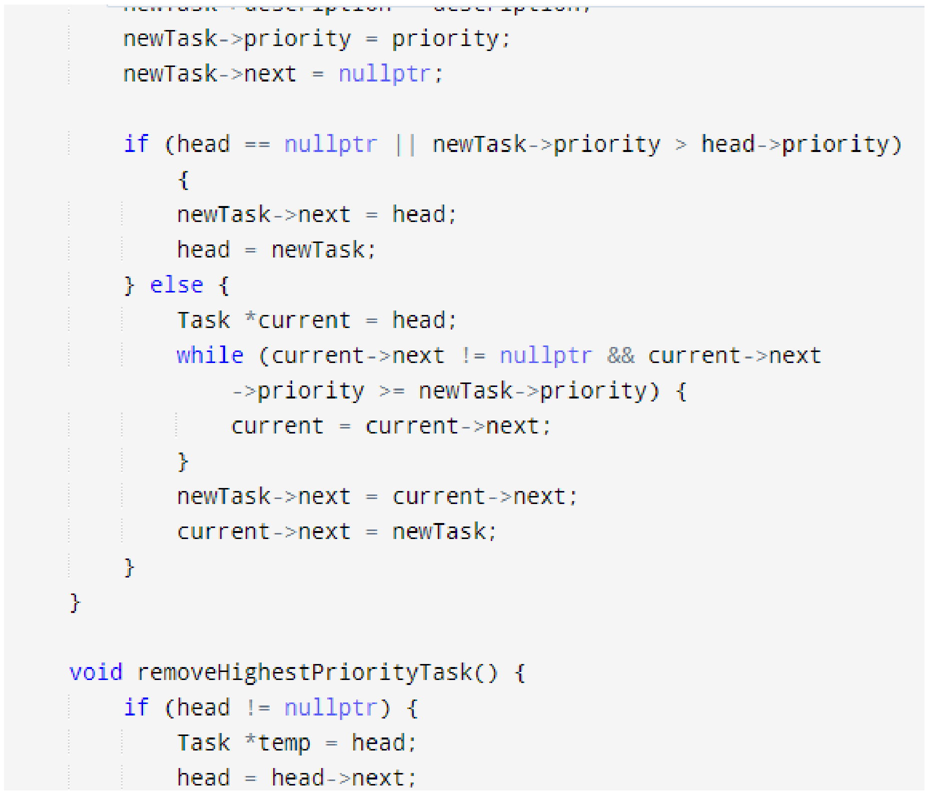
ATTOCK CAMPUS

DEPARTMENT OF COMPUTER SCIENCE

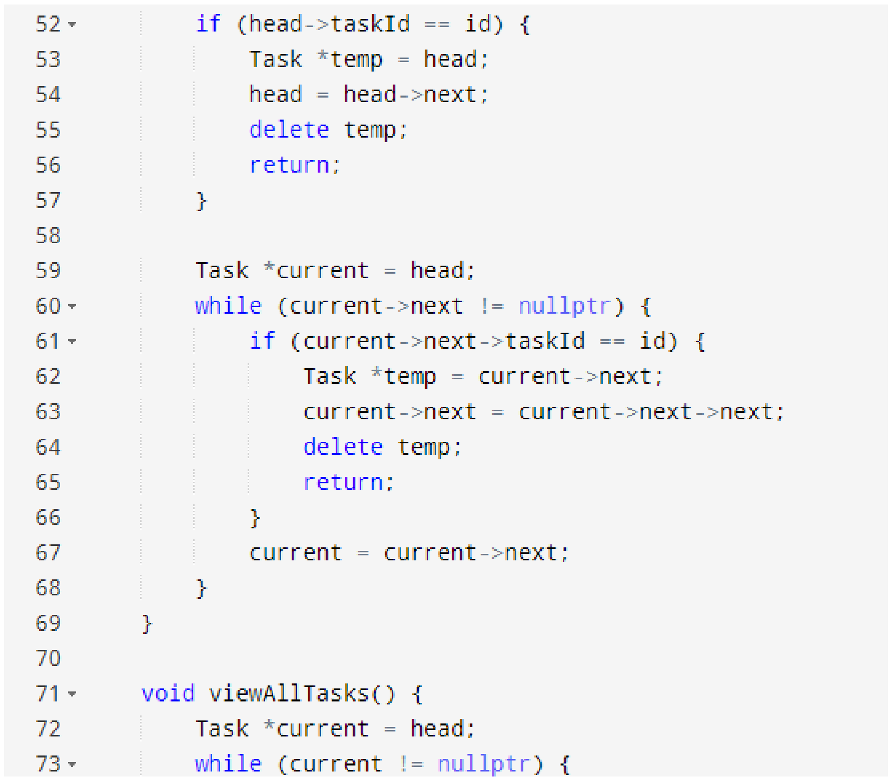
NAME :	**AKASHA AKMAL**
REG. NO:	**FA22-BSE-038**
SUBJECT:	**Data Structures**
ASSIGNMENT:	**Assignment #1**
Date:	**24rd September, 2024**
SUBMITTED TO:	**sir Kamran**



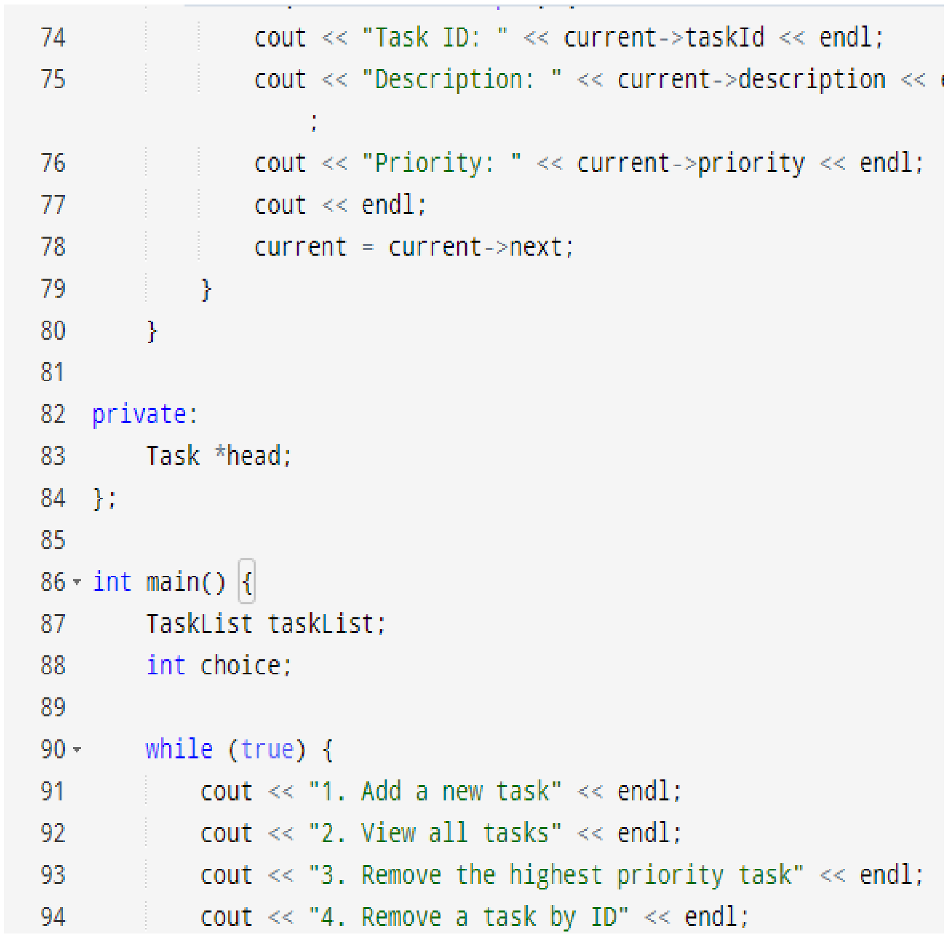
```cpp
#include <iostream>
#include <string>

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;
```



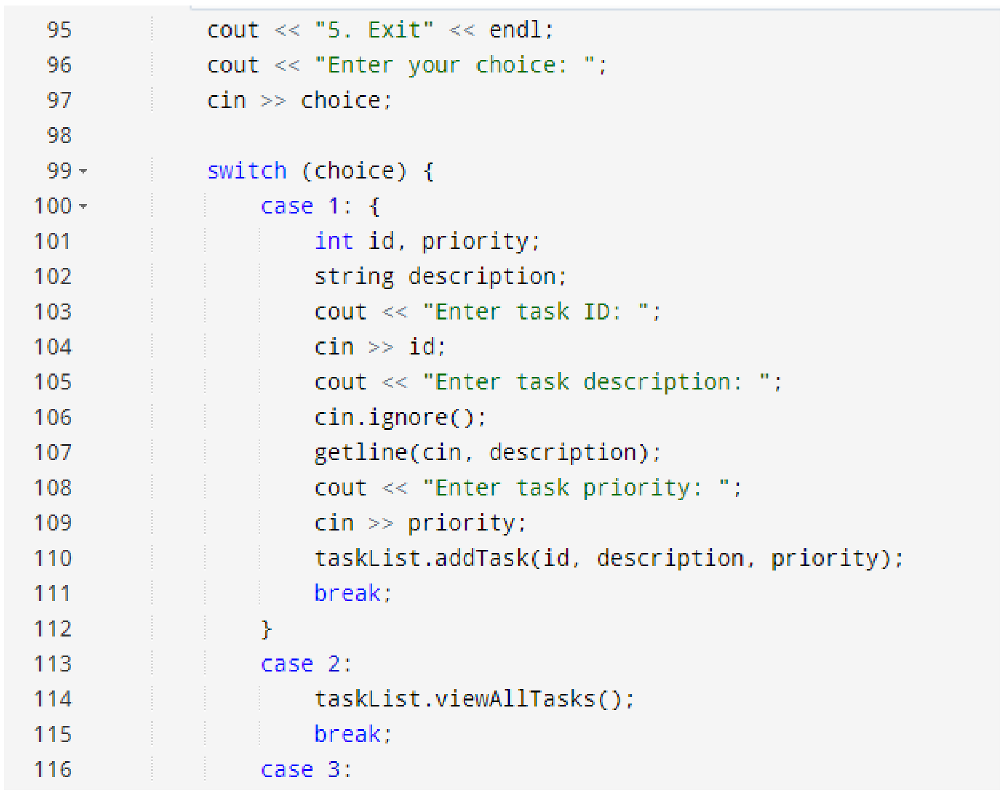
```
    newTask->description = description;
    newTask->priority = priority;
    newTask->next = nullptr;

    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;
```



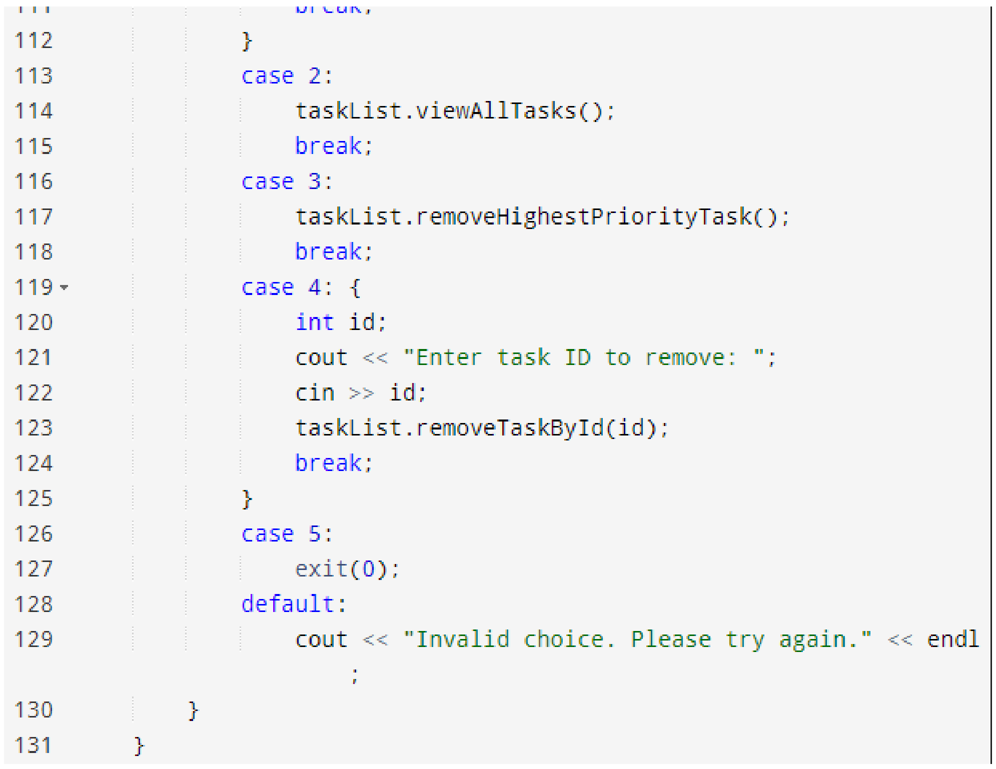
```cpp
        if (head->taskId == id) {
            Task *temp = head;
            head = head->next;
            delete temp;
            return;
        }

        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) {
```



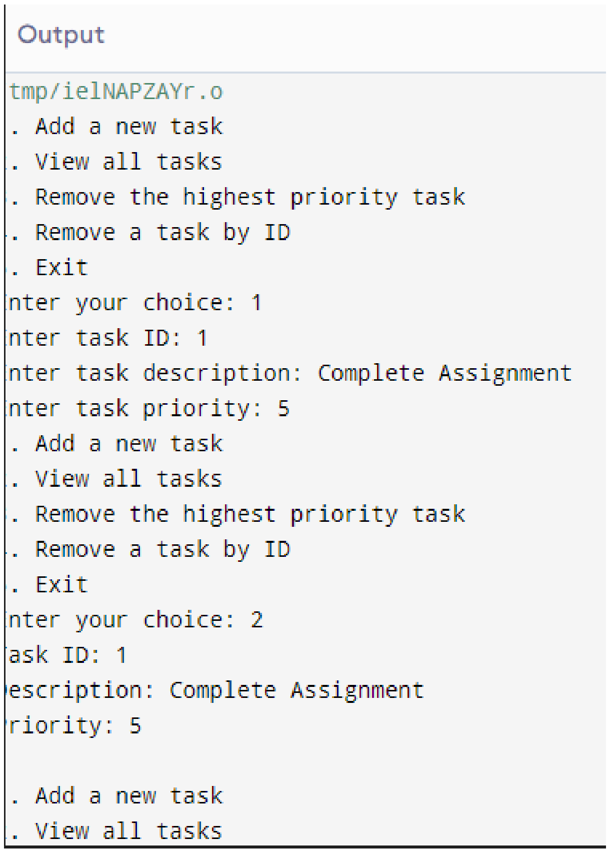
```cpp
            cout << "Task ID: " << current->taskId << endl;
            cout << "Description: " << current->description << 
                ;
            cout << "Priority: " << current->priority << endl;
            cout << endl;
            current = current->next;
        }
    }

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;
        }
    }
```


Output

```
tmp/ielNAPZAYr.o
. Add a new task
. View all tasks
. Remove the highest priority task
. Remove a task by ID
. Exit
nter your choice: 1
nter task ID: 1
nter task description: Complete Assignment
nter task priority: 5
. Add a new task
. View all tasks
. Remove the highest priority task
. Remove a task by ID
. Exit
nter your choice: 2
ask ID: 1
escription: Complete Assignment
riority: 5

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

Report:

Intro:

The objective of this assignment is to create a task management system using Object-Oriented Programming (OOP) in C++. The system is implemented using a linked list structure to manage tasks, where each task has an ID, a description, and a priority. The program allows the user to add new tasks, remove tasks based on priority or ID, and view all the tasks in order of their priority. The key objective is to learn how to manipulate dynamic memory, create linked list data structures, and practice handling basic task management functions efficiently.

Code Explanation:

1. Struct Task:

- This defines the structure for each task. Each task contains:
    - Task Id: An integer to uniquely identify each task.
    - description: A string to store a brief explanation of the task.
    - priority: An integer representing the task's priority level (higher priority tasks will come first in the list).
    - next: A pointer that links one task to the next one, forming a linked list.

2. Class Task List:

- The Task List class manages all the tasks and provides operations such as adding, removing, and viewing tasks.
- **Constructor:**
    - Initializes the task list by setting the head pointer to nullptr. The head pointer is used to keep track of the start of the linked list.
- **Destructor:**
    - The destructor ensures that all dynamically allocated memory is released when the object is destroyed, preventing memory leaks. It traverses the list and deletes each node.
- **add Task(int id, string description, int priority):**
    - This function adds a new task to the list in the correct position based on its priority. If the list is empty or the new task has a higher priority than the head task, it becomes the

new head. Otherwise, the function traverses the list to find the correct position for the new task.

    - Logic ensures that tasks with higher priority appear earlier in the list. Lower priority tasks are appended to the end or in between based on their priority.

- **removeHighestPriorityTask ():**
    - This function removes the task with 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 no tasks are present, it displays a message stating that there are no tasks to remove.
- **removeTaskById (int id):**
    - This function removes a task based on its ID. If the task with the given ID is the head, it deletes the head. Otherwise, it traverses the list to find the task with the matching ID and removes it.
    - If no task with the specified ID is found, the function displays an appropriate message.
- **viewAllTasks ():**
    - This function prints all the tasks in the list, starting from the head, displaying the task ID, description, and priority of each task.
    - If the list is empty, it informs the user that there are no tasks to display.

3. Main Function:

- **Menu Loop:**
    - The main () function contains an infinite loop that displays a menu to the user, allowing them to choose one of the operations: adding a new task, viewing all tasks, removing the highest-priority task, removing a task by ID, or exiting the program.
    - Each option invokes the corresponding method from the TaskList class to perform the desired operation.
    - The loop continues until the user selects the "Exit" option.
    - The program uses input validation to ensure that the user provides valid inputs

Conclusion:

the programs also helped me practice error handling and input validation, which are critical for making programs robust. Lastly, I learned how important it is to clean up dynamically allocated memory properly, as failing to do so can lead to memory leaks and bugs that may be hard to trace in more complex programs.