COMSATS UNIVERSITY ISLAMABAD

ATTOCK CAMPUS

DEPARTMENT OF COMPUTER SCIENCE

NAME:	**Samar Safdar Khan**
REG. NO:	**SP23-BSE-043**
SUBJECT:	**Data Structures**
ASSIG:	**1**
Date:	**24th September, 2024**
SUBMITTED TO:	**Mr. Muhammad Kamran**

1 Introduction:

This task management system helps users organize and prioritize their tasks using a singly linked list. It allows for easy addition of tasks based on priority, removal of the highest priority task, and deletion of specific tasks by ID, all through a simple console interface.

2 Explanation:

2.1 TaskNode Struct:

The TaskNode struct is the basic building block of the linked list. It represents each individual task and contains essential information, including a unique task ID to identify the task, a description that explains what the task entails, a priority level indicating the urgency or importance of the task, and a pointer to the next task in the list. This structure allows for dynamic memory allocation and linked organization of tasks.

2.2 TaskManager Class:

The TaskManager class serves as the controller for managing the linked list of tasks. It maintains a pointer to the head of the list, which starts as NULL, indicating that there are no tasks initially. This class is responsible for all operations related to task management.

2.2.1 Add Task Function

This function is responsible for adding a new task to the list. It evaluates the priority of the new task against the current tasks. If the list is empty or the new task has a higher priority than the existing head task, it becomes the new head of the list. If not, the function traverses the list to find the appropriate position to insert the new task, ensuring that tasks remain ordered by priority. This allows the system to quickly access the highest priority task at the beginning of the list.

2.2.2 Remove Highest Priority Task

This function removes the task with the highest priority, which is located at the head of the list. Before removal, it checks if the list is empty and informs the user if there are no tasks to remove. If a task is present, it updates the head pointer to point to the next task and deallocates the memory used by the removed task, preventing memory leaks.

2.2.3 Remove Task by ID

This function allows users to remove a specific task by its unique ID. It starts by checking if the list is empty and notifies the user if there are no tasks available. If the task to be removed is the head task, it updates the head pointer accordingly. If the task is located further down the list, the function searches for it and removes it while maintaining the integrity of the linked list structure.

2.2.4 Display Tasks

The display tasks function provides a way for users to see all current tasks in the system. It checks if the list is empty and informs the user if no tasks are available. If there are tasks present, it iterates through the list and prints the details of each task, including the ID, description, and priority. This function is essential for users to track their tasks effectively.

2.2.5 Destructor

The destructor is a special function that ensures memory cleanup when the TaskManager object is destroyed. It systematically removes all tasks from the list, freeing up the allocated memory to prevent leaks. This is crucial for managing resources efficiently and ensuring that the program runs smoothly without consuming unnecessary memory.

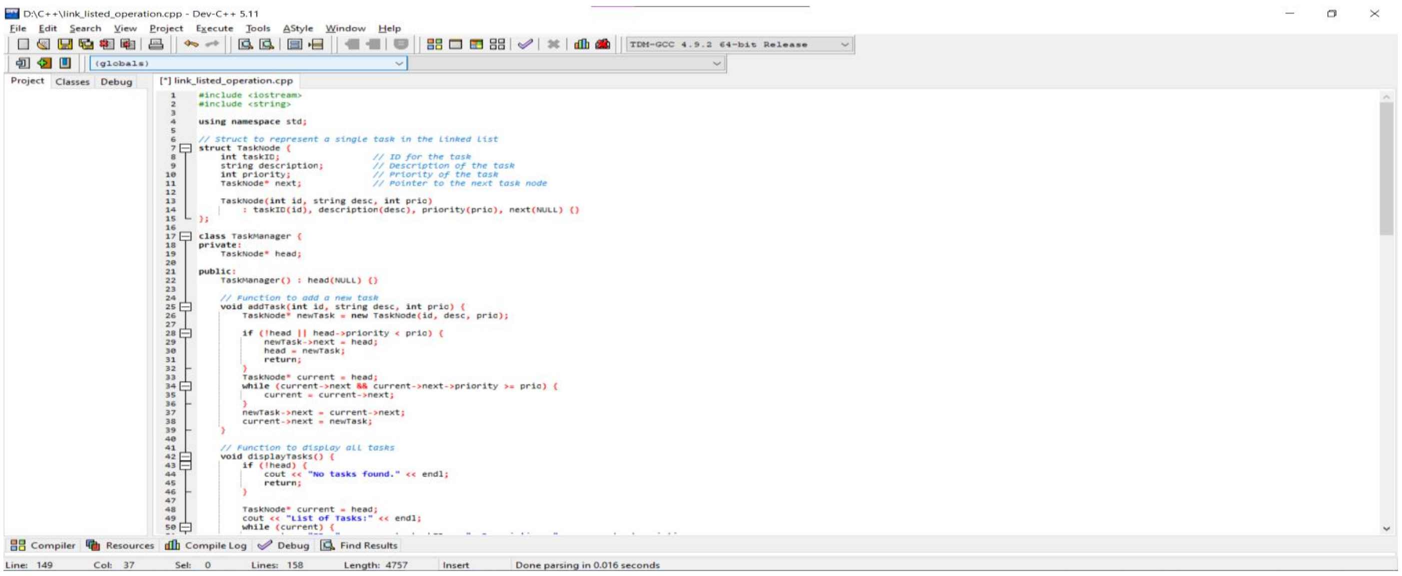
2.2.6 Display Menu

This function provides a user-friendly console interface that allows users to interact with the task management system. It presents a set of options for adding tasks, viewing the list of tasks, removing tasks, and exiting the program. Based on the user's selection, it invokes the corresponding methods in the TaskManager class, facilitating a smooth user experience.

2.3 Main Function

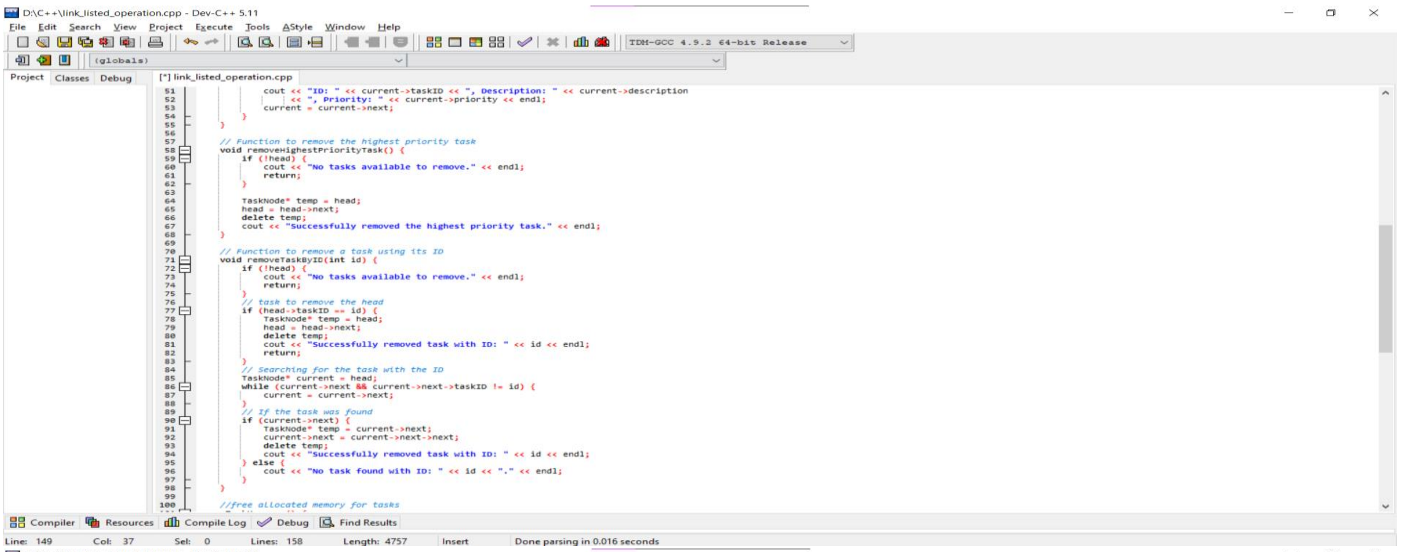
The main function serves as the entry point of the program. It initiates the task management system by calling the display menu function, allowing users to begin interacting with the task management features. This function essentially sets the program in motion, leading to user engagement with the task management system.

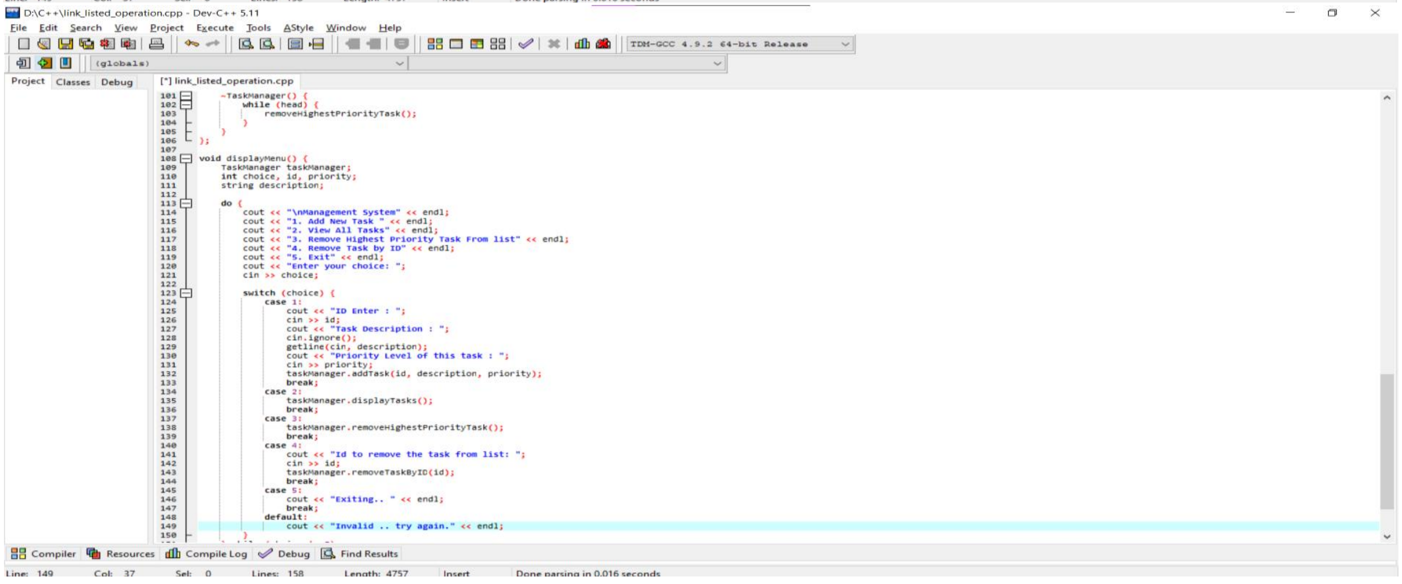
3 CODE:



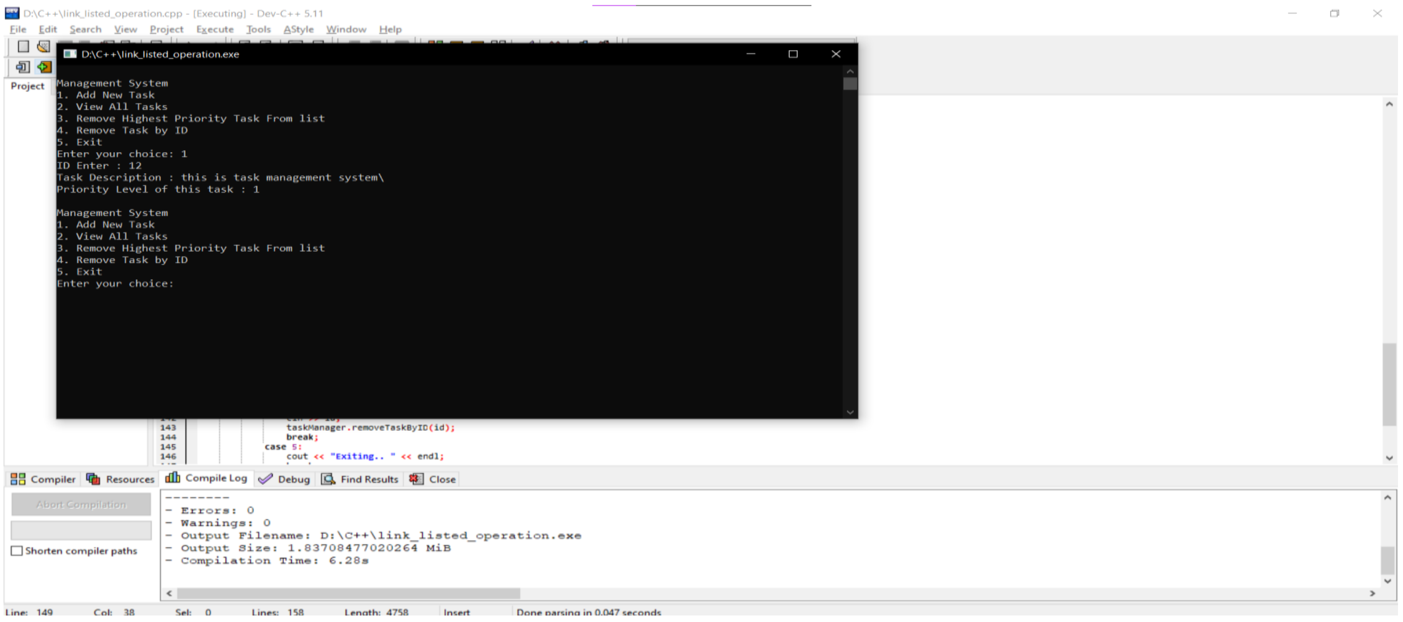
```cpp
#include <iostream>
#include <string>

using namespace std;

// Struct to represent a single task in the linked list
struct TaskNode {
    int taskID;                 // ID for the task
    string description;         // Description of the task
    int priority;               // Priority of the task
    TaskNode* next;             // Pointer to the next task node

    TaskNode(int id, string desc, int prio)
        : taskID(id), description(desc), priority(prio), next(NULL) {}
};

class TaskManager {
private:
    TaskNode* head;

public:
    TaskManager() : head(NULL) {}

    // Function to add a new task
    void addTask(int id, string desc, int prio) {
        TaskNode* newTask = new TaskNode(id, desc, prio);

        if (!head || head->priority < prio) {
            newTask->next = head;
            head = newTask;
            return;
        }
        TaskNode* current = head;
        while (current->next && current->next->priority >= prio) {
            current = current->next;
        }
        newTask->next = current->next;
        current->next = newTask;
    }

    // Function to display all tasks
    void displayTasks() {
        if (!head) {
            cout << "No tasks found." << endl;
            return;
        }

        TaskNode* current = head;
        cout << "List of Tasks:" << endl;
        while (current) {
```

Line: 149   Col: 37   Sel: 0   Lines: 158   Length: 4757   Insert   Done parsing in 0.016 seconds


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

    // Function to remove the highest priority task
    void removeHighestPriorityTask() {
        if (!head) {
            cout << "No tasks available to remove." << endl;
            return;
        }

        TaskNode* temp = head;
        head = head->next;
        delete temp;
        cout << "Successfully removed the highest priority task." << endl;
    }

    // Function to remove a task using its ID
    void removeTaskByID(int id) {
        if (!head) {
            cout << "No tasks available to remove." << endl;
            return;
        }
        // task to remove the head
        if (head->taskID == id) {
            TaskNode* temp = head;
            head = head->next;
            delete temp;
            cout << "Successfully removed task with ID: " << id << endl;
            return;
        }
        // Searching for the task with the ID
        TaskNode* current = head;
        while (current->next && current->next->taskID != id) {
            current = current->next;
        }
        // If the task was found
        if (current->next) {
            TaskNode* temp = current->next;
            current->next = current->next->next;
            delete temp;
            cout << "Successfully removed task with ID: " << id << endl;
        } else {
            cout << "No task found with ID: " << id << "." << endl;
        }
    }

    //free allocated memory for tasks
```


```cpp
    ~TaskManager() {
        while (head) {
            removeHighestPriorityTask();
        }
    }
};

void displayMenu() {
    TaskManager taskManager;
    int choice, id, priority;
    string description;

    do {
        cout << "\nManagement System" << endl;
        cout << "1. Add New Task " << endl;
        cout << "2. View All Tasks" << endl;
        cout << "3. Remove Highest Priority Task From list" << endl;
        cout << "4. Remove Task by ID" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "ID Enter : ";
                cin >> id;
                cout << "Task Description : ";
                cin.ignore();
                getline(cin, description);
                cout << "Priority Level of this task : ";
                cin >> priority;
                taskManager.addTask(id, description, priority);
                break;
            case 2:
                taskManager.displayTasks();
                break;
            case 3:
                taskManager.removeHighestPriorityTask();
                break;
            case 4:
                cout << "Id to remove the task from list: ";
                cin >> id;
                taskManager.removeTaskByID(id);
                break;
            case 5:
                cout << "Exiting.. " << endl;
                break;
            default:
                cout << "Invalid .. try again." << endl;
        }
```

OUTPUT:


```
Management System
1. Add New Task
2. View All Tasks
3. Remove Highest Priority Task From list
4. Remove Task by ID
5. Exit
Enter your choice: 1
ID Enter : 12
Task Description : this is task management system\
Priority Level of this task : 1

Management System
1. Add New Task
2. View All Tasks
3. Remove Highest Priority Task From list
4. Remove Task by ID
5. Exit
Enter your choice:
```

```
--------
- Errors: 0
- Warnings: 0
- Output Filename: D:\C++\link_listed_operation.exe
- Output Size: 1.83708477020264 MiB
- Compilation Time: 6.28s
```