Name: Muhammad Ammar

Reg No: Sp23-bse-019

Assignment No: 01

Subject: DS

Explanation

Task Structure:

Each task is represented as a node containing an ID, description, priority, and a pointer to the next task.

TaskManager Class:

Contains a pointer to the head of the linked list.

Methods include:

addTask(): Adds a task in order of priority.

RemoveHighestPriorityTask():

Removes the task at the head of the list (highest priority).

RemoveTaskById():

Removes a task based on its ID.

ViewTasks():

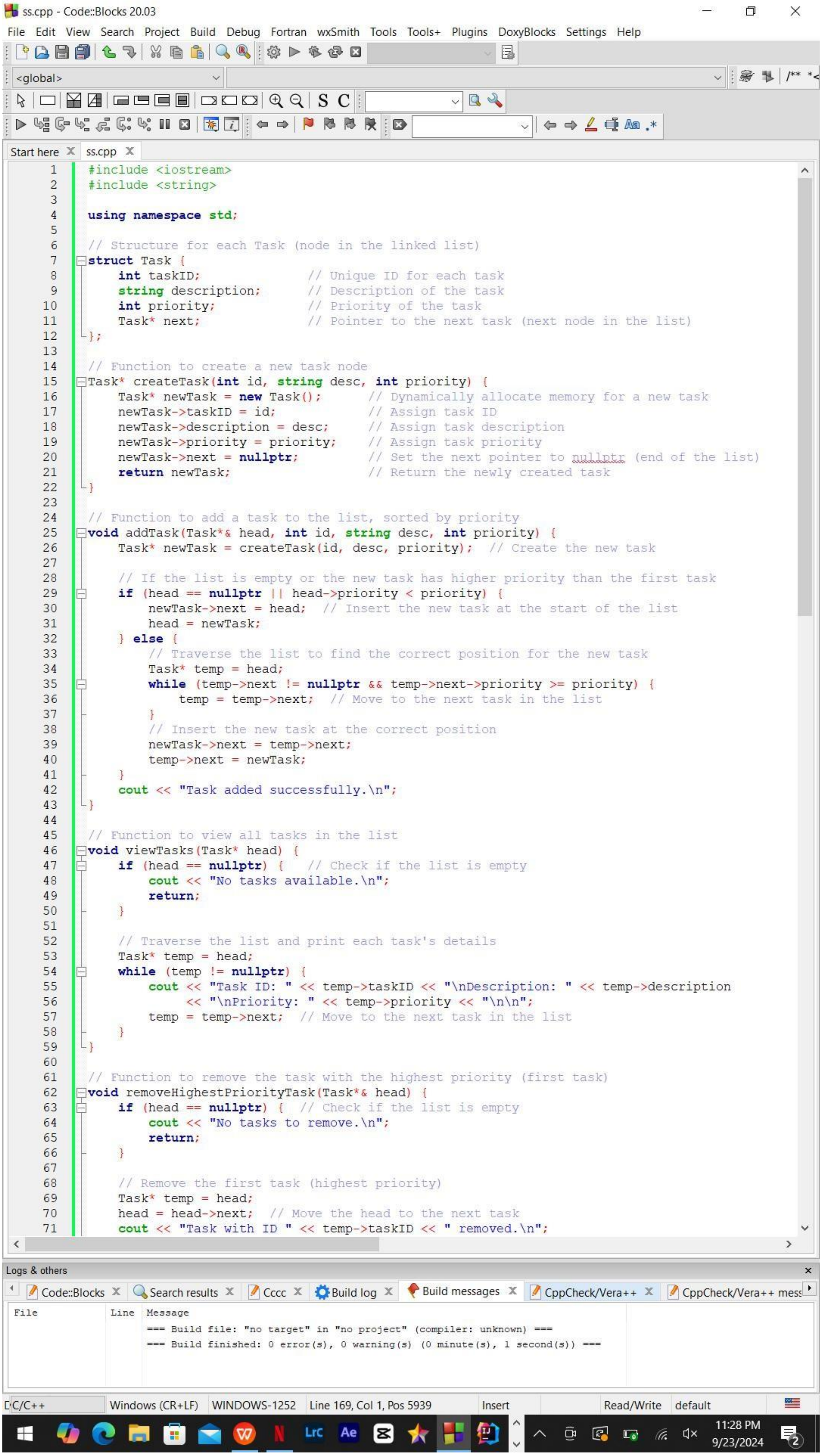
Displays all tasks.

Menu Loop:

A console-based menu allows the user to interact with the task manager.

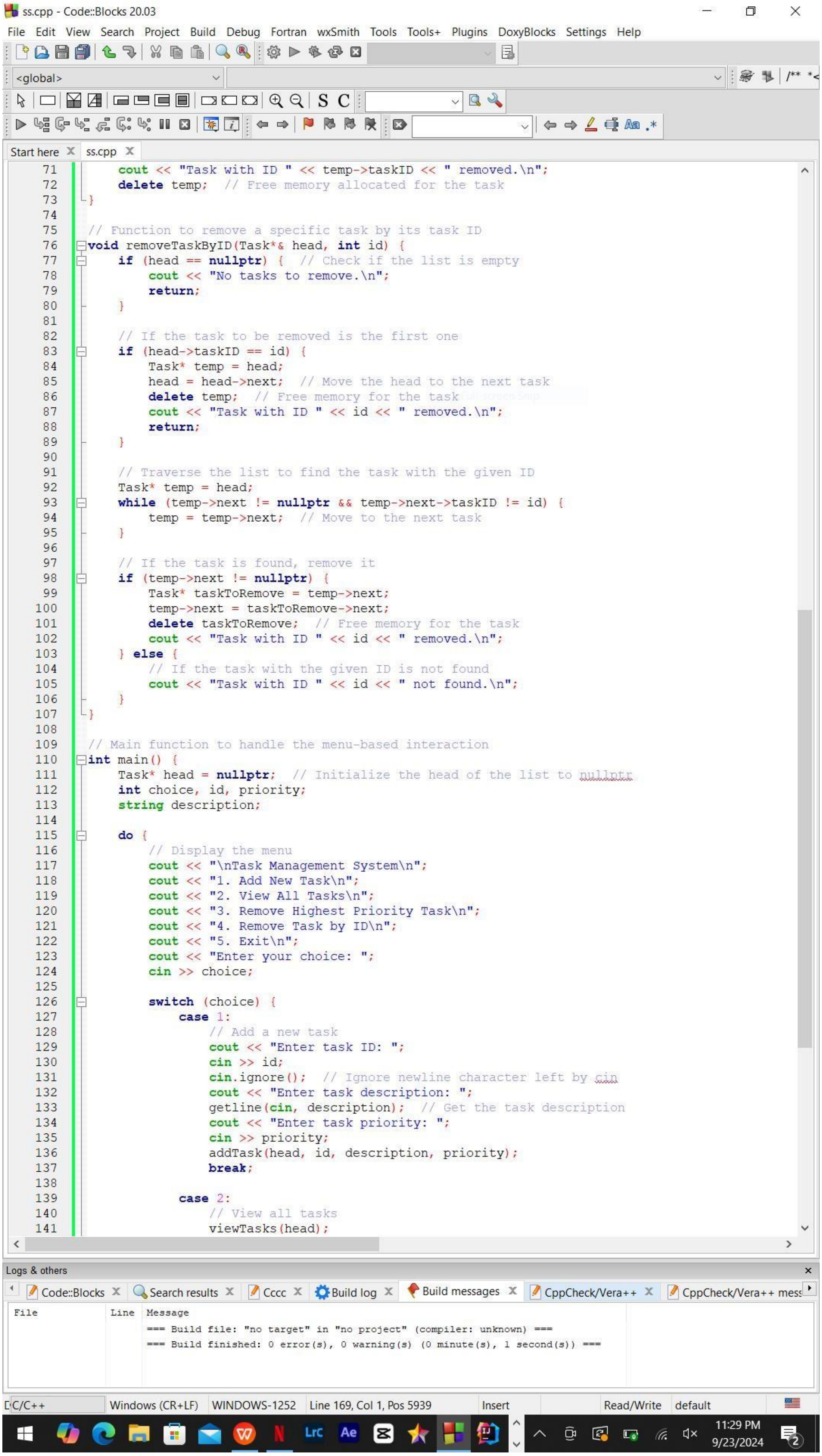
Usage:

Compile the code and run it. You can add tasks, view them, remove the highest priority task, or remove a task by its ID. The task list maintains order based on priority, with higher priority tasks appearing first.



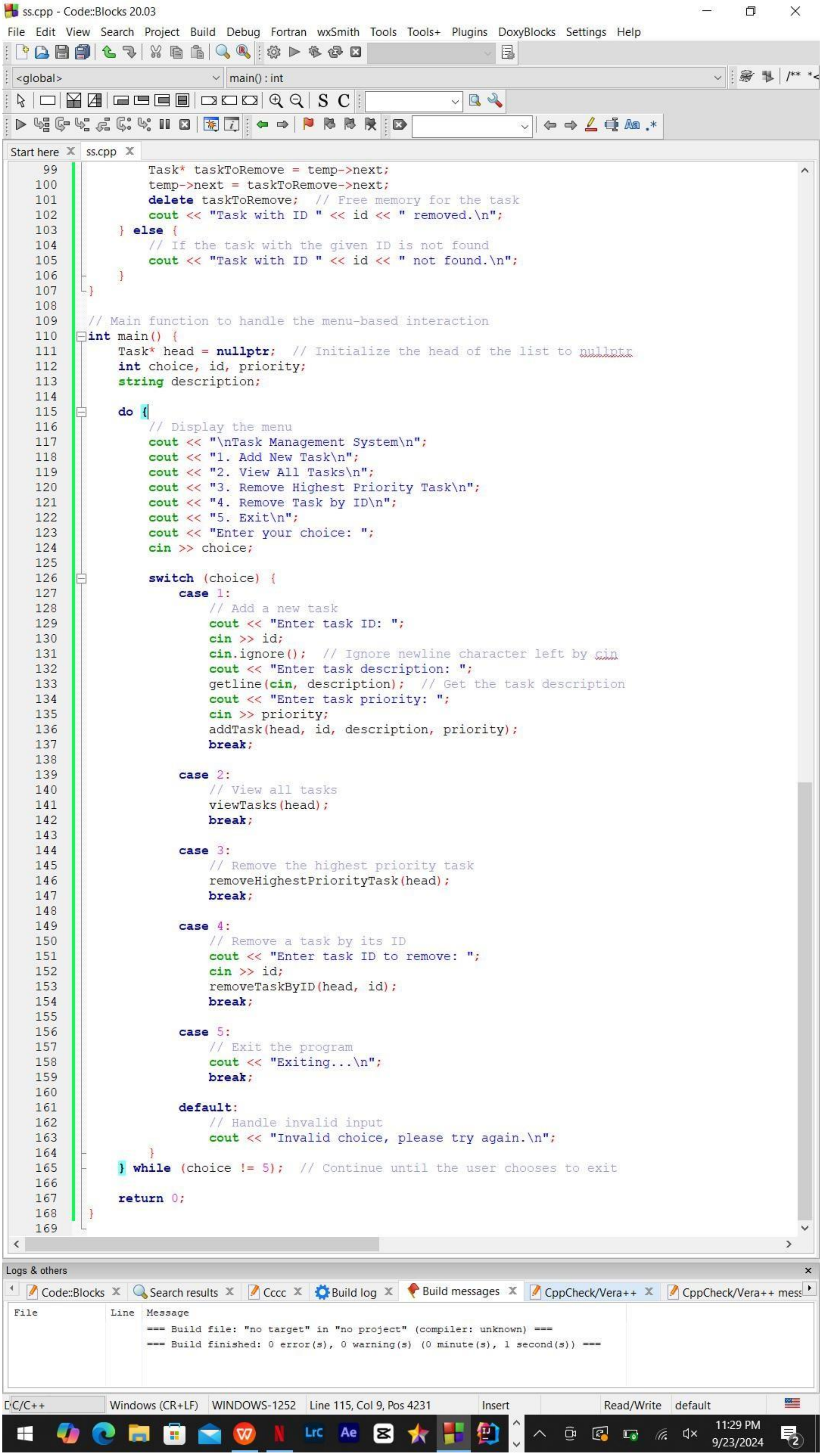
```cpp
#include <iostream>
#include <string>

using namespace std;

// Structure for each Task (node in the linked list)
struct Task {
    int taskID;              // Unique ID for each task
    string description;      // Description of the task
    int priority;            // Priority of the task
    Task* next;              // Pointer to the next task (next node in the list)
};

// Function to create a new task node
Task* createTask(int id, string desc, int priority) {
    Task* newTask = new Task();       // Dynamically allocate memory for a new task
    newTask->taskID = id;             // Assign task ID
    newTask->description = desc;      // Assign task description
    newTask->priority = priority;     // Assign task priority
    newTask->next = nullptr;          // Set the next pointer to nullptr (end of the list)
    return newTask;                   // Return the newly created task
}

// Function to add a task to the list, sorted by priority
void addTask(Task*& head, int id, string desc, int priority) {
    Task* newTask = createTask(id, desc, priority);  // Create the new task

    // If the list is empty or the new task has higher priority than the first task
    if (head == nullptr || head->priority < priority) {
        newTask->next = head;  // Insert the new task at the start of the list
        head = newTask;
    } else {
        // Traverse the list to find the correct position for the new task
        Task* temp = head;
        while (temp->next != nullptr && temp->next->priority >= priority) {
            temp = temp->next;  // Move to the next task in the list
        }
        // Insert the new task at the correct position
        newTask->next = temp->next;
        temp->next = newTask;
    }
    cout << "Task added successfully.\n";
}

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

    // Traverse the list and print each task's details
    Task* temp = head;
    while (temp != nullptr) {
        cout << "Task ID: " << temp->taskID << "\nDescription: " << temp->description
             << "\nPriority: " << temp->priority << "\n\n";
        temp = temp->next;  // Move to the next task in the list
    }
}

// Function to remove the task with the highest priority (first task)
void removeHighestPriorityTask(Task*& head) {
    if (head == nullptr) {  // Check if the list is empty
        cout << "No tasks to remove.\n";
        return;
    }

    // Remove the first task (highest priority)
    Task* temp = head;
    head = head->next;  // Move the head to the next task
    cout << "Task with ID " << temp->taskID << " removed.\n";
```

```
File          Line  Message
                    === Build file: "no target" in "no project" (compiler: unknown) ===
                    === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ===
```



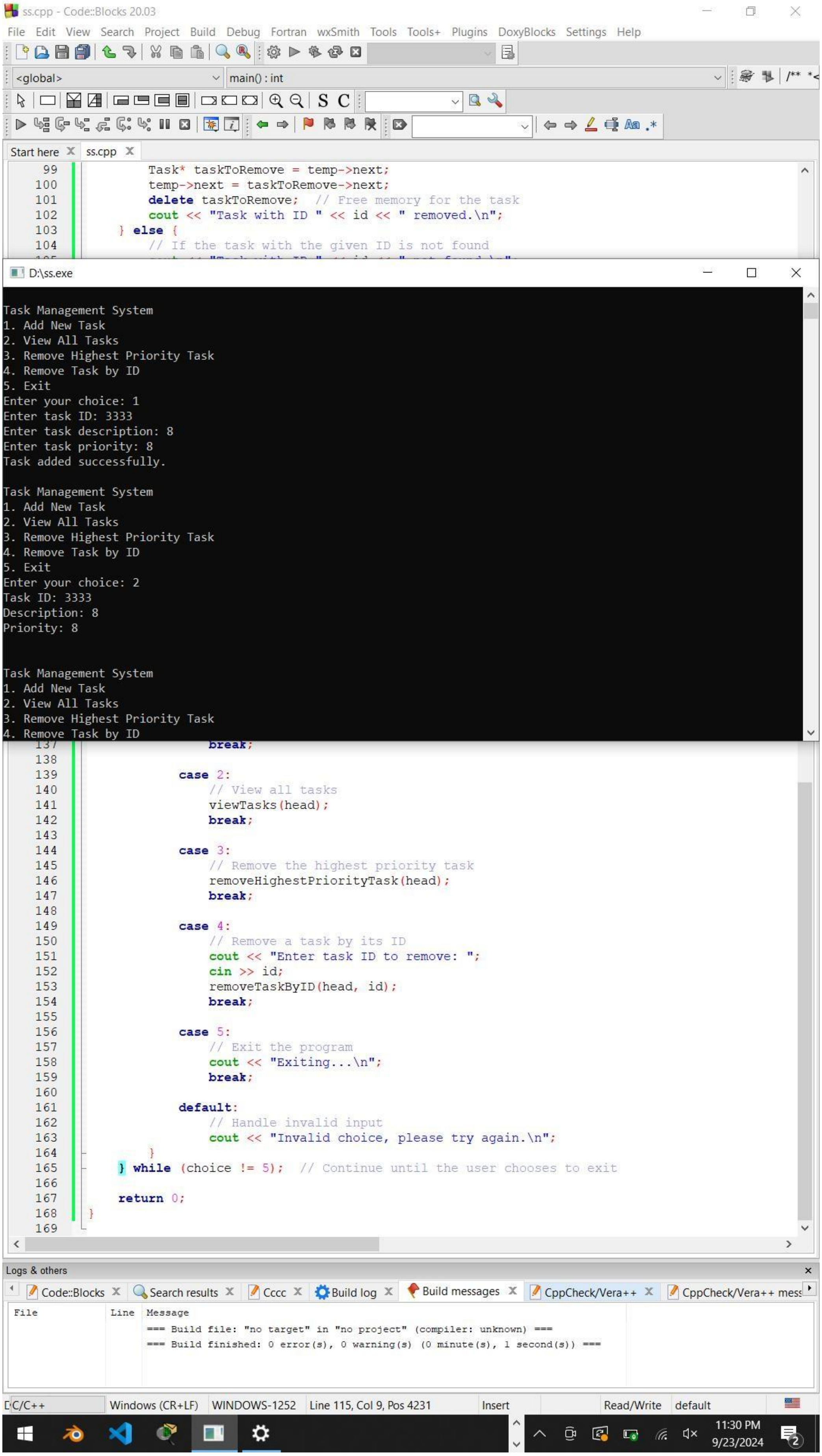
```cpp
    cout << "Task with ID " << temp->taskID << " removed.\n";
    delete temp;  // Free memory allocated for the task
}

// Function to remove a specific task by its task ID
void removeTaskByID(Task*& head, int id) {
    if (head == nullptr) {  // Check if the list is empty
        cout << "No tasks to remove.\n";
        return;
    }

    // If the task to be removed is the first one
    if (head->taskID == id) {
        Task* temp = head;
        head = head->next;  // Move the head to the next task
        delete temp;  // Free memory for the task
        cout << "Task with ID " << id << " removed.\n";
        return;
    }

    // Traverse the list to find the task with the given ID
    Task* temp = head;
    while (temp->next != nullptr && temp->next->taskID != id) {
        temp = temp->next;  // Move to the next task
    }

    // If the task is found, remove it
    if (temp->next != nullptr) {
        Task* taskToRemove = temp->next;
        temp->next = taskToRemove->next;
        delete taskToRemove;  // Free memory for the task
        cout << "Task with ID " << id << " removed.\n";
    } else {
        // If the task with the given ID is not found
        cout << "Task with ID " << id << " not found.\n";
    }
}

// Main function to handle the menu-based interaction
int main() {
    Task* head = nullptr;  // Initialize the head of the list to nullptr
    int choice, id, priority;
    string description;

    do {
        // Display the menu
        cout << "\nTask Management System\n";
        cout << "1. Add New Task\n";
        cout << "2. View All Tasks\n";
        cout << "3. Remove Highest Priority Task\n";
        cout << "4. Remove Task by ID\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                // Add a new task
                cout << "Enter task ID: ";
                cin >> id;
                cin.ignore();  // Ignore newline character left by cin
                cout << "Enter task description: ";
                getline(cin, description);  // Get the task description
                cout << "Enter task priority: ";
                cin >> priority;
                addTask(head, id, description, priority);
                break;

            case 2:
                // View all tasks
                viewTasks(head);
```

```
File            Line  Message
                      === Build file: "no target" in "no project" (compiler: unknown) ===
                      === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ===
```



```cpp
        Task* taskToRemove = temp->next;
        temp->next = taskToRemove->next;
        delete taskToRemove;  // Free memory for the task
        cout << "Task with ID " << id << " removed.\n";
    } else {
        // If the task with the given ID is not found
        cout << "Task with ID " << id << " not found.\n";
    }
}

// Main function to handle the menu-based interaction
int main() {
    Task* head = nullptr;  // Initialize the head of the list to nullptr
    int choice, id, priority;
    string description;

    do {
        // Display the menu
        cout << "\nTask Management System\n";
        cout << "1. Add New Task\n";
        cout << "2. View All Tasks\n";
        cout << "3. Remove Highest Priority Task\n";
        cout << "4. Remove Task by ID\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                // Add a new task
                cout << "Enter task ID: ";
                cin >> id;
                cin.ignore();  // Ignore newline character left by cin
                cout << "Enter task description: ";
                getline(cin, description);  // Get the task description
                cout << "Enter task priority: ";
                cin >> priority;
                addTask(head, id, description, priority);
                break;

            case 2:
                // View all tasks
                viewTasks(head);
                break;

            case 3:
                // Remove the highest priority task
                removeHighestPriorityTask(head);
                break;

            case 4:
                // Remove a task by its ID
                cout << "Enter task ID to remove: ";
                cin >> id;
                removeTaskByID(head, id);
                break;

            case 5:
                // Exit the program
                cout << "Exiting...\n";
                break;

            default:
                // Handle invalid input
                cout << "Invalid choice, please try again.\n";
        }
    } while (choice != 5);  // Continue until the user chooses to exit

    return 0;
}

```

```
File            Line  Message
                      === Build file: "no target" in "no project" (compiler: unknown) ===
                      === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ===
```



```cpp
        Task* taskToRemove = temp->next;
        temp->next = taskToRemove->next;
        delete taskToRemove;  // Free memory for the task
        cout << "Task with ID " << id << " removed.\n";
    } else {
        // If the task with the given ID is not found
```

```
Task Management System
1. Add New Task
2. View All Tasks
3. Remove Highest Priority Task
4. Remove Task by ID
5. Exit
Enter your choice: 1
Enter task ID: 3333
Enter task description: 8
Enter task priority: 8
Task added successfully.

Task Management System
1. Add New Task
2. View All Tasks
3. Remove Highest Priority Task
4. Remove Task by ID
5. Exit
Enter your choice: 2
Task ID: 3333
Description: 8
Priority: 8


Task Management System
1. Add New Task
2. View All Tasks
3. Remove Highest Priority Task
4. Remove Task by ID
```

```cpp
                break;

            case 2:
                // View all tasks
                viewTasks(head);
                break;

            case 3:
                // Remove the highest priority task
                removeHighestPriorityTask(head);
                break;

            case 4:
                // Remove a task by its ID
                cout << "Enter task ID to remove: ";
                cin >> id;
                removeTaskByID(head, id);
                break;

            case 5:
                // Exit the program
                cout << "Exiting...\n";
                break;

            default:
                // Handle invalid input
                cout << "Invalid choice, please try again.\n";
        }
    } while (choice != 5);  // Continue until the user chooses to exit

    return 0;
}
```

```
File            Line  Message
                      === Build file: "no target" in "no project" (compiler: unknown) ===
                      === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ===
```