Assignment II: Task Scheduler Implementation

Student Name: Orlando Alvarado Vargas
Student ID: 2226968

Instructions:

You are required to implement a Task Scheduler Implementation using a queue data structure in C++.

- Implement a **Queue** class that manages tasks.
- Each task should be represented by a simulated execution time (e.g., an integer or float).
- Provide methods to add tasks to the queue, remove tasks from the queue, and display the current queue of tasks.
- Demonstrate the scheduling and processing of tasks in your report. Simulate the execution of at least 5 tasks by printing their execution times.
- Explain how the queue data structure is used to manage task scheduling.

Submission Requirements:

- Include your well-commented source code in the report.
- Provide sample test cases and their outputs.
- Briefly explain your design choices and how OOP principles and data structures are applied.
- Ensure your code is well-structured and follows best practices.

A Introduction

In this document I explain how I did the Task Scheduler Implementation for the assignment in C++. With this system, the user can manage a list of tasks by adding, removing, viewing, and clearing them.

B Class Definitions

First thing I did was to define the class that I used with their own attributes and functions:

- **Task:** With a description, estimated time, and a number in the queue. The class provides methods to display the task's information and to return its attributes, ensuring that tasks can be easily identified and listed.
- **Queue:** Manages the list of tasks and implements the system's main functions: adding new tasks, removing tasks by their number, displaying the entire agenda, and clearing all tasks from the list.

B.1 Task Class

```
// Represents a single scheduled task with a name and time
class Task{
private:
    string taskname;     // Task description
    float time;          // Estimated time in minutes
    int tasknum;         // Sequential number assigned to each task
    static int nextnum; // Static counter to assign task numbers
public:
    Task (string taskname, float time); // Constructor
    void display() const;               // Display Task information
    string getname() const;             // Return task name
    float gettime() const;              // Return estimated time in
       minutes
};
```

B.2 Queue Class

```
// Manages a list of Task
class Queue{
private:
    //---------- Definition of vector ----------
    vector<Task> list;
public:
    void addtask();      // Add a new task
    void removetask();   // Remove a task
    void agenda() const;// Display all the agenda
    void clear();        //Clear all tasks from the agenda
};
```

C Method Implementations

This section describes the implementation of the constructors, methods, and accessors of the system's classes. A static counter was used in the Task class to automatically assign incremental IDs, while the Queue class includes error handling to prevent invalid operations, such as removing a task from an empty list or selecting task numbers that are out of range.

C.1 Task Class

```
int Task::nextnum = 1;  // Initialize static counter for Task
   numbering
    // Constructor
Task::Task (string taskname, float time){
    this -> taskname = taskname;
    this -> time = time;
    tasknum = nextnum++;  // Assign number and then increment the
       counter
}
void Task::display() const{
    cout << tasknum << "␣␣␣␣" << taskname << "␣␣␣␣Time:␣" << time <<
       "␣minutes␣\n";
}
string Task::getname() const{
    return taskname;
}
float Task::gettime() const{
    return time;
}
```

C.2 Queue Class

```
void Queue::addtask(){
    string name;
    float time;
    cout << "\n----------------------NEW␣TASK----------------------\n";
    cout << "\nWhat␣are␣you␣going␣to␣schedule?:␣";
    getline(cin >> ws,name);          // Read a full line (skips leading whitespace)
    cout << "What␣is␣the␣estimated␣time␣in␣minutes?:␣";
    cin >> time;                      // Read estimated time in minutes
    cin.ignore();                     // Clear newline from input buffer
    list.push_back(Task(name,time));  // Append new Task to the vector
    cout << "\n----------------------------------------------------\n";
}

void Queue::removetask(){
    int num;
    cout << "\n---------------------REMOVE␣TASK--------------------\n";
    if (list.empty()){
        cout << "\n############␣␣There␣is␣no␣task␣to␣remove␣:o␣␣############\n";
        return;                       // Early exit if there are no tasks
    }
    cout << "\n␣Enter␣the␣task␣number␣you␣want␣to␣remove:␣";
    cin >> num;                       // Read the task number
    cin.ignore();                     // Clear newline from input buffer

    if (num>0 && num<=list.size()){
        // Convert from 1-based input to 0-based index and erase
        list.erase(list.begin()+(num-1));
        cout << "\nTask␣" << num << "␣successfully␣removed␣:D";
        cout << "\n----------------------------------------------------\n";
    }
    else{
        // Guard against out-of-range numbers
        cout << "\nPlease␣enter␣a␣number␣between␣1␣and␣" << list.size() << "\n";
        return;
    }
}
void Queue::agenda() const{
    cout << "\n-----------------------AGENDA-----------------------\n";
    if (list.empty()){
        cout << "\n############␣␣There␣is␣no␣task␣to␣do␣:)␣␣############\n";
        return;                       // Early exit if nothing to list
    }

    // Displays each task in the list
    for (int i = 0; i < list.size(); i++){
        list[i].display();
    }
    cout <<
       "\n----------------------------------------------------\n";
}
void Queue::clear(){
    cout <<
       "\n----------------------------------------------------\n";
    if (list.empty()){
        cout << "\n############␣␣There␣is␣no␣task␣to␣clear␣␣
           ############\n";
        return;                         // Early exit if already empty
    }

    list.clear();                       // Erase all elements from the
       vector
    cout <<
       "\n####################################################\n";
    cout << "\n================␣CLEANING␣THE␣AGENDA␣
       ===============\n";
    cout <<
       "\n####################################################\n";
    cout <<
       "\n----------------------------------------------------\n";
}
```

D Main Function

In the main function, I only created an instance fo the Queue class and build a simple menu to interact with the system. The menu allows the user to add, remove, show the agenda and clear all the tasks. A switch statement handles the user's choice, and the program keeps running until the exit option is selected.

```
int main (){
    Queue task;             // Instance managing all tasks
    int selection;
    do{
        cout <<
           "\n----------------------SCHEDULE----------------------\n";
        cout << "\nWhat␣would␣you␣like␣to␣do?\n" <<
                "1)␣Add␣a␣Task.\n" <<
                "2)␣Remove␣a␣Task.\n" <<
                "3)␣See␣the␣Agenda.\n" <<
                "4)␣Clear␣the␣Agenda.\n" <<
                "0)␣Exit␣the␣System.\n";
        cin >> selection;

        // Handle user selection
        switch (selection){
            case 0:{
                cout << "\nHave␣a␣good␣day!!!\n";
                break;
            }
            case 1:{
                task.addtask();
                break;
            }
            case 2:{
                task.removetask();
                break;
            }
            case 3:{
                task.agenda();
                break;
            }
            case 4:{
                task.clear();
                break;
            }
            default:{       // Invalid input guard for numbers
               outside the expected range
                cout << "\n############␣␣␣␣␣Please␣select␣a␣number␣
                   between␣0␣and␣4␣␣>:(␣␣␣␣␣############\n";
            }
        }
    } while (selection != 0);  // Repeat until user select exit
    return 0;
}
```

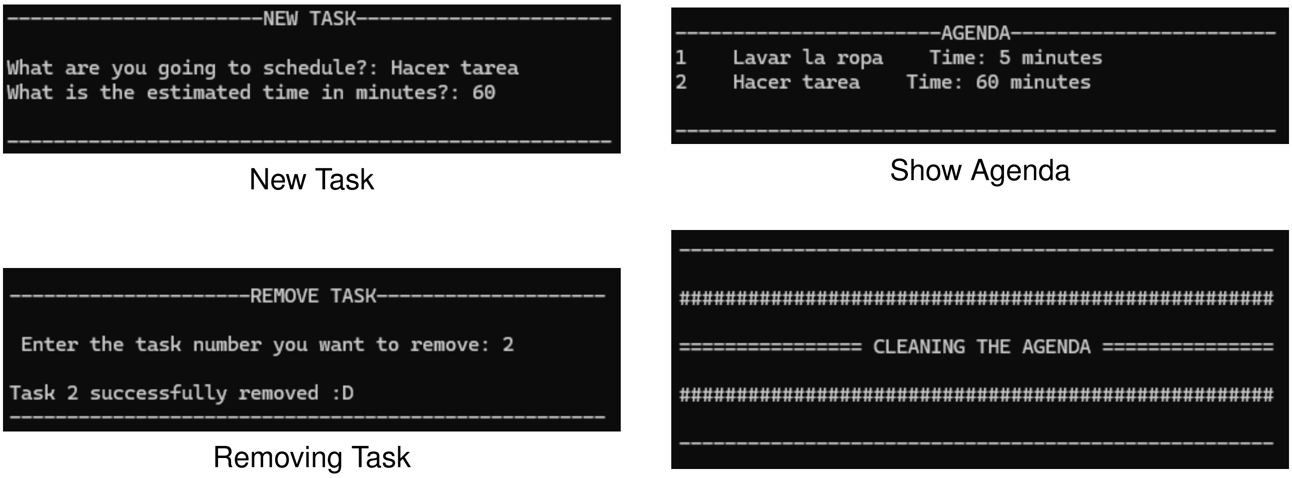
E Test Cases

Here are some examples of how the code looks.

Test Case: Main Menu

```
----------------------SCHEDULE----------------------

What would you like to do?
1) Add a Task.
2) Remove a Task.
3) See the Agenda.
4) Clear the Agenda.
0) Exit the System.
```



```
---------------------NEW TASK---------------------

What are you going to schedule?: Hacer tarea
What is the estimated time in minutes?: 60

--------------------------------------------------
```

New Task

```
----------------------AGENDA----------------------
1    Lavar la ropa    Time: 5 minutes
2    Hacer tarea     Time: 60 minutes

--------------------------------------------------
```

Show Agenda

```
--------------------REMOVE TASK--------------------

 Enter the task number you want to remove: 2

Task 2 successfully removed :D
---------------------------------------------------
```

Removing Task

```
--------------------------------------------------

##################################################

=============== CLEANING THE AGENDA ===============

##################################################

--------------------------------------------------
```

Clearing the Agenda

Figure 1: Screenshots of different operations.