Assignment IV: Min-Heap Implementation

Student Name: Orlando Alvarado Vargas
Student ID: 2226968

Instructions:

You are required to implement a min-heap in C++ using Object-Oriented Programming (OOP) principles. The min-heap will be used to manage a priority queue for task scheduling based on task priority. Your program should:

- Implement a min-heap in C++ using a vector to manage a priority queue for task scheduling.
- Each task should have a priority (integer) and a description (string). The heap should always allow extraction of the task with the highest priority (lowest integer value).
- Your implementation must support insertion of new tasks, extraction (pop) of the highest-priority task, and heapify operations to maintain the heap property.
- Write a main program that demonstrates scheduling tasks by priority, showing the order in which tasks are executed.
- Include comments in your code to explain key sections and logic.
- Provide at least two test cases that show your heap correctly schedules and executes tasks based on priority.

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 are applied.
- Ensure your code is well-structured and follows best practices.

A Introduction

In this report I explain how I did the Task schedule based on task priority, using a min-heap with C++. With this system, the user can manage a list of tasks by adding, removing a task with most priority, and viewing all the schedule.

B Class Definitions

I did an assignment like this previously, so I only needed to change the previous list function with a heap one. First of all, I defined the classes for the task and the schedule with their own attributes and functions:

- **Task:** This structure defines an object for tasks with description and priority that will determine their position in the heap. The class provides methods to display the task's information and get the priority to ease the process.
- **Schedule:** This class manages the heap for the schedule. It provides some methods to reorganize all the heap, using heapify up and down. Also here are implemented the main functions of the system like adding, removing a task with most priority, and viewing all the schedule that are going to be called by the main function depending of the user selection.

```
// ---------- Definition of the Schedule class ----------
class Schedule{
    private:
        vector<Task> heap;
        void heapify_Up(int i);     // -- Move a node UP to its
           correct spot after insert --
        void heapify_Down(int i);   // -- Move a node DOWN to its
           correct spot after extract --
    public:
        void insert(int priority, string description);  // ---- Add a
           new task ----
        bool isEmpty(); // -- Check if heap is empty --
        Task extract(); // ---- Remove a task ----
        void display(); // ---- Display all the agenda ----
};
```

C Method Implementations

This section describes the implementation of the methods and functions of the system's classes. The Task class is pretty basic, so I'm going to emphasize in the schedule function. First of all, I defined the vector that stores the elements. Then I implemented functions that helps to re-organize the heap automatically:

- **Heapify UP:** When a new element is inserted, it is storage at the end of the array. So the method identify if the element is less than his parent, if it is true they swap until the element is in place.
- **Heapify DOWN:** When an element is deleted, the last element is moved to the first place. This function identify if the element is greater than the child and swap them until the element is in place.

Then there are the public methods which the user is going to interact with. There is a function besides the insert, delete and display function that checks if the heap is empty. This helps the delete and display functions to notify that it is, actually, empty.

```
class Schedule{
    private:
        vector<Task> heap;
        // -- Move a node UP to its correct spot after insert --
        void heapify_Up(int i){
            while (i > 0 && heap[(i-1)/2].getPriority() >
               heap[i].getPriority()){
                swap(heap[(i-1)/2], heap[i]);
                i = (i-1)/2;
            }
        }
        // -- Move a node DOWN to its correct spot after extract --
        void heapify_Down(int i){
            int n = heap.size();     // Total number of elements
            while (2*i + 1 < n){
                int j = 2*i + 1;     // Get left child index
                if (j + 1 < n && heap[j+1].getPriority() <
                   heap[j].getPriority()){
                    j++;
                }
                if (heap[i].getPriority() <= heap[j].getPriority()){
```

```
                    break;
                }
                swap(heap[i], heap[j]);
                i = j;
            }
        }
    public:
        // ---- Add a new task ----
        void insert(int priority, string description){
            heap.push_back(Task(priority, description));     // Add
               task to the very end of the vector
            heapify_Up(heap.size()-1);                       // Bubble
               it up to its correct position
        }
        // -- Check if heap is empty --
        bool isEmpty(){
            return heap.empty();
        }
        // ---- Remove a task ----
        Task extract(){
            Task minPriority = heap[0]; // Save the root (min value)
               to return later
            heap[0] = heap.back();      // Move the LAST element to
               the root
            heap.pop_back();            // Remove the last element
               (which is now duplicated at the root)
            heapify_Down(0);            // Bubble the new root down
               to its correct spot
            return minPriority;         // Return the task we saved
        }
        // ---- Display all the agenda ----
        void display(){
            for(int i = 0; i < heap.size(); i++){
                heap[i].display();
            }
        }
};
```

D Main Function

Finally, in the main function, I created a menu that operates within a do-while loop, which repeats until the user selects the exit option. I created an instance for the schedule heap and a display to show the options. The user selects any number between 0 and 3, and a switch statement handles the user's choice:

- **Exit (0):** This option allows the user to exit the system by breaking the do-while loop.
- **Add Task (1):** This option allows the user to insert a new task, asking for the description and the priority between some options. Then the program calls the insert function that include the task and re-organizes the heap automatically.
- **Remove Task (2):** This option checks if the heap is empty. If the heap has elements, the program removes the Max-Priority task and re-organizes the heap automatically.
- **See the Agenda (3):** This option checks if the heap is empty. If the heap has elements, the program shows all the current agenda with descriptions and priority.

- **Default:** This option is created to warn the user that a different character was selected.

```
int main(){
    Schedule task;  // Create heap instance
    int selection;
    do{ // Menu loop
        // Print menu options
        cout <<
           "\n\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" <<
                "0)␣Exit␣the␣System.\n";
        cout << "\nSelection:␣";
        cin >> selection;
        switch(selection){  // Handle selection
            case 0:{        // Exit
                cout << "\nHave␣a␣good␣day!!!\n";
                break;
            }
            case 1:{        // Add Task
                int priority;
                string description;
                cout << "\n---------------NEW␣TASK---------------\n";
                cout << "\nEnter␣the␣new␣Task:␣";
                getline(cin >> ws, description);    // Read full line
                   (skip whitespace)
                cout << "Select␣the␣task␣priority:\n" <<
                        "1)␣Due␣Today\n" <<
                        "2)␣Due␣Tomorrow\n" <<
                        "3␣->␣7)␣Due␣this␣week\n" <<
                        "8␣->␣...)␣Other␣priority\n";
                cout << "\nPriority:␣";
                cin >> priority;
                cin.ignore();   // Clear newline from input
                task.insert(priority, description); // Call heap
                   insert
                cout <<
                   "\n-----------------------------------------\n";
                break;
            }
            case 2:{        // Remove Task
                cout << "\n---------------REMOVE␣TASK-------------\n";
                if (task.isEmpty()){    // Check if empty
                    cout << "\nThere␣is␣nothing␣in␣the␣Schedule␣D:\n";
                    cout <<
                       "\n-----------------------------------\n";
                    break;
                }
                else{
                    Task removed = task.extract();  // Call heap
                       extract
                    cout << "\n...Removing␣Max-Priority␣Task...\n";
                    cout << "\nTask␣->␣";
                    removed.display();  // Show what was removed
```

```
                    cout << "Successfully removed :D\n";
                    cout <<
                       "\n----------------------------------------\n";
                    break;
                }
            }
            case 3:{        // Display Agenda
                cout <<
                   "\n-------------------AGENDA-------------------\n";
                if (task.isEmpty()){
                    cout << "\nThere is nothing in the Schedule D:\n";
                    cout <<
                       "\n----------------------------------------\n";
                    break;
                }
                else{
                    task.display();
                    cout <<
                       "\n----------------------------------------\n";
                    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 selects 0
    return 0;
}
```

E Test Cases

Here are some examples of how the menu looks.

Test Case: Main Menu

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

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

Selection:
```

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

Enter the new Task: Hacer la tarea
Select the task priority:
1) Due Today
2) Due Tomorrow
3 -> 7) Due this week
8 -> ...) Other priority

Priority: 2

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

Insert New Task

```
----------------------AGENDA----------------------
1  -  Hacer de comer
2  -  Hacer la tarea
4  -  Ir a comprar despensa

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

See the Schedule

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

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

Selection: 2

---------------------REMOVE TASK--------------------

...Removing Max-Priority Task...

Task -> 1  -  Hacer de comer
Successfully removed :D

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

Remove Task

Figure 1: Screenshots of different library operations.