EC327 - Spring 2025 - Homework 5

Background

This homework will reinforce key concepts, including standard template libraries, templates, and algorithms. All of the problems on this assignment will require solutions to be implemented in C++ using C++17.

Important

Please make sure your code compiles and runs as intended on the engineering grid. Code that does not compile will NOT be graded and will receive a 0.

Note on Collaboration

You are welcome to talk to your classmates about the assignment and discuss high level ideas. However, all code that you submit must be your own. We will run code similarity tools against your submissions and will reach out with questions if anything is flagged as suspicious. The closed book exams for this class will mostly cover material very similar to what you do on the homework assignments. The best way to prepare for the exams is to do the assignments on your own and internalize the learnings from that process. Cheating on the assignments will very likely result in you not doing well on the exams.

Problem 1 - Leaderboard Ranking System (Big-O, STL map) (50 points)

Submission Instructions

Your solution to this problem should contribute a single .h file, .cpp file, and a .txt file to your overall hw5 zip. The .h file should follow the following format: **Leaderboard.h**
The .cpp file should follow the following format: **Leaderboard.cpp**
The .txt file should follow the following format: **Leaderboard.txt**

Actual Problem

You are building a **Leaderboard** system for an online game. Each player has a **username** (string) and a **score** (int). The leaderboard should store each player as a key-value pair in a **map** or **unordered_map**, where the **key is the username** (a unique string) and the **value is the score** (an integer). In your Leaderboard.cpp file, implement a class that:

- Allows adding/updating a player's score
- Returns the **top N players** sorted by highest score
- Removes a player

Here is the Leaderboard.h file started for you:

```
#ifndef LEADERBOARD_H

#define LEADERBOARD_H



#include <string>

#include <vector>

#include <utility>

class Leaderboard {

public:

   // Adds a new player or updates the score if the player already exists. If the player already exists, this function should properly update the player with the correct score.

   void addOrUpdatePlayer(const std::string& username, int score);

   // Removes a player from the leaderboard

   void removePlayer(const std::string& username);
```

```
    // Returns the top N players sorted by highest score (descending order)
    std::vector<std::pair<std::string, int>> getTopN(int N) const;

private:
    // Choose appropriate STL container(s) for storage
    // Example: std::map<std::string, int> or std::unordered_map<std::string, int>
};

#endif
```

Requirements:

- Use `std::map<string, int>` or `std::unordered_map<string, int>` for storage.
- Provide a method `getTopN(int N)` that returns a `vector<pair<string, int>>` of top players.
- Analyze the **time complexity** of adding, removing, and retrieving top N players.

This is an example of what could be returned for the **`getTopN()`** function:

```
std::vector<std::pair<std::string, int>> top3 = {
    {"Bob", 2000},
    {"Carol", 1800},
    {"Dave", 1700}
};
```

In your **Leaderboard.txt** file, answer the following questions:

1. What are the trade-offs between using `std::map` and `std::unordered_map` for storing player scores? Which is better for this application and why?

2. What is the time complexity of the following operations in your implementation?
    - `addOrUpdatePlayer()`
    - `removePlayer()`
    - `getTopN()`

Problem 2 - Reverse Polish Notation (STL stack, templates) (50 points)

Submission Instructions

Your solution to this problem should contribute a single .h file and a single .cpp file to your overall hw5 zip. The .h file should follow the following format: **PolishNotation.h**
The .cpp file should follow the following format: **PolishNotation.cpp**

Actual Problem

You are given either a string or a vector of strings called **tokens** that represents an arithmetic expression in a Reverse Polish Notation. Evaluate the expression and return an int that represents the value of the expression.
Note:

- The string expression will be separated by white spaces.
- The valid operators are '+', '-', '*', and '/'.
- Each operand may be an integer or another expression.
- The division between two integers always truncates toward zero.
    - For example, 5 / 2 = 2

Examples:

expression = ["2","1","+","3","*"] (vector of strings)
answer = ((2 + 1) * 3) = 9

expression = “4 13 5 / +” (string)
answer = 6

Here is the PolishNotation.h file started for you:

```
#ifndef POLISHNOTATION_H
#define POLISHNOTATION_H

#include <vector>
#include <string>
#include <stack>

template <typename InputType>
class RPNCalculator {
   public:
       // Evaluates the given reverse polish notation. You should be checking whether
the expression is a string or a vector of strings.
       int evaluate(const InputType& expression);

   private:
       // Tokenizes a string into vector<string> if the input is a string.
       std::vector<std::string> tokenize(const std::string& expr);

       // Choose appropriate STL container(s) for storage
       // Example: std::stack<std::string>
       std::stack<std::string> stack;
};

#endif // POLISHNOTATION_H
```

Since we are using templates, it is typical convention to use a .tpp file. However, in this case we only want this calculator to work with strings and vector of strings so we can use explicit template instantiation at the bottom of the .cpp file.

Here is the PolishNotation.cpp file started for you:

```
#include "PolishNotation.h"

#include <vector>
#include <string>

template <typename InputType>
int RPNCalculator<InputType>::evaluate(const InputType& expression) {

}

template <typename InputType>
std::vector<std::string> RPNCalculator<InputType>::tokenize(const std::string& expr) {

}

// Explicit template instantiations
template class RPNCalculator<std::string>;
template class RPNCalculator<std::vector<std::string>>;
```

Requirements:

- Use *`std::stack`*
- Use `templates` to allow the calculator to accept both strings and vector of strings