Assignment 1: Vectors, Lists, and Generic Programming

CS361: Advanced Data Structures and Algorithms

BACKGROUND

In competitive games such as Chess, Checkers, Go, and even Tic-Tac-Toe, there are 3 results to a given game: either player wins, or the game is drawn. There are different style tournaments that can be played, each supporting different numbers of players. For example,

- Round-robin tournaments will have each player play every other player
- Elimination tournaments will have players play another player, with the winner moving on to the next round, halving the field of competitors each round.
- Swiss tournaments are non-elimination tournaments that pair players according to their tournament results and pre-tournament rating.

Given a list of players and a list of game results, we can calculate the scores for each player in any of these tournament types.

PROBLEM

You are provided an almost-complete program that will read a list of players and a list of game results and produce the final standings. Your task is to finish the Tournament class such that it reads in the 2 input files and writes the output file in the proper order.

REQUIREMENTS

1. Implement the Tournament::Tournament(…) constructor provided such that it loads the data from the players input file into the m_Players vector and data from the games input file into the m_Games list **without using any user-defined loops**.
2. Implement the Tournament::GenerateResults() function such that it writes the list of players with their scores in descending order by player score **without using any user-defined loops**.
3. Submit your updated tournament.h file (and only the tournament.h file) to Blackboard.

HINTS

- Notice that GenerateResults is a const function. This means you cannot modify the data in the Tournament class within it. Take note of the necessary copy-constructors.
- You will need to use std::sort. We will not go over the details of this function until later in the semester. For now, all you need to know is that when you pass in the proper iterators and use std::greater<Player>() for your comparison function, it will sort your data properly.
- For this assignment, the IDs of the players will match their position in the array. That is, an ID of 0 will correspond to m_Players[0].