CSE 3150 — C++ Essentials — Yufeng Wu — Fall 2024
Programming Assignment 5. Due: 11 pm @ 10/4/2024.

---

There are objectives in this assignments.

1. Get familiar with STL: vector, set and map, and common algorithms like std::sort.
2. Learn how to write programs with low complexity.

# 1 Background

In a sports tournament (e.g., US Open Tennis), individual players entering the tournament may have different credentials and backgrounds. There are also some other considerations in organizing a tournament. In particular, here are the more specifics of how a tournament is supposed to run.

1. Ranking: each player has a rank (i.e, different from any other player). For example, in men's professional tennis, every player has a so-called ATP ranking, where 1 is the highest (the best). Rank is in the form an integer. Note: a player with certain rank may not participate in a tournament (that is, say the best player in ATP ranking may decide not to compete in this tournament).
2. Country of origin: a player comes from some country, which is coded as an integer. Sometimes, a tournament may enforce a restriction that no country can have more than two players from that country so as to promote diversity.
3. The number of players in a tournament is assumed to be a power of 2, and has at least 16 players participating.
4. The top ranked players are the seeded players. For simplicity, we fix the number of seeds to be 8. That is, the top 8 players from the tournaments are seeded players. If there are ties in ranking, there may be different ways of designing the seeded players. We simply allow an arbitrary one from these possible sets of seeded players.
5. A tournament has a draw, which is an **ordered** list of players and specifies the "order of play". This list is divided into pairs of adjacent players. During each round, the two players in each pair play a match and the winner advances and the loser is eliminated. For example, suppose the list of players is $a, b, c, d, e, f, g, h$. Then, the tournament will run three rounds: (i) during the first rounds, $a$ plays with $b$, $c$ with $d$, $e$ with $f$ and $g$ with $h$. (ii) In the second round, the winners (say $a, c, e, g$ sequentially) will play again: $a$ with $c$ and $e$ with $g$, (iii) in the third round, the two remaining winners say $a$ and $e$ are in the final match (the championship match).

In this assignment, you are implement some functions that will examine a *given* draw to determine whether the draw is valid or not according to a certain set of rules. Note: all these rules are in effect by default *unless* we invoke the corresponding function to disable the rule for more flexibility. There are some variations in different sports. For example, in professional tennis, ranking is distinct; but some other sports may allow ties in ranking. In order to make your code flexible, you need to support the following configuration functions.

1. Seeded players are evenly distributed in the draw so that the seeded players won't compete with each other during early rounds. This is perhaps enforced in most tournament. To be specific, the top two players must be located in different halves of the draw. For example, suppose the list of players is $a, b, c, d, e, f, g, h$ and the top two players are $a$ and $d$. Then this

is invalid because $a$ and $d$ belong to the same half. But if $b$ and $g$ are the top two players, then this draw is valid. Then, the top four players must be in different quarters. Then top eight players must be in different eighth.

2. No duplicate ranking.

3. No country can have more than two players from that country.

4. No two players from the same country will play during the first two rounds. That is, no two players from the same country are located in the same pair or in the pair that is to be played in the second round. This may make matches more interesting to watch.

## 2 Interface functions to implement

1. *ECCheckTournamentDraw*: inputs are two vectors of integers of the same length, one for the rank and one for the country of origin. These two vectors are the draw: at each position $i$, the two vectors give the rank and the country of the $i$-th player in the draw. Return true if the draw is valid, and false otherwise.

2. Configuration of rules (to disable certain rules) by calling:

   (a) A function that allows ties in ranking.

   (b) Allow more than 2 players from the same country.

   (c) Allow players from the same country to compete in any round.

   (d) Allow seeded players to be located in any position of the draw (i.e., two seeded players can play very early; that is not good, but..)

To understand the requirements, please read the provided starter code and also the test cases.

## 3 Code quality

In this assignment, in addition to check functionalities, we will also use autograder to check the code quality. That is, there will be test cases that automatically analyze your program; if your code is too complex, you won't pass those test cases. So why are we measuring the code quality here? Writing code with low complexity helps to improve code **readability** and **maintainability**, which might be even more important when code generation using say AI becomes more popular.

Please note: your code must achieve basic functionalities before we can talk about code quality.

1. Try to avoid long functions

2. Try to reduce nested loops; only use double loops when necessary; avoid triple loops

3. Try to reduce conditional statements

4. My code has 1 double loop, 5 single loops, 4 if-conditional, longest function has 20 line of code (LOC), total LOC is 132 (including comments). Try to write your code that has similar complexity.

To get 100 in this PA, your code should not only pass functionality tests, but also have low complexity:

1. Use no more than 11 functions.

2. The number of lines per function is at most 60.

3. The so-called Cyclomatic complexity (computed by Gradescope) is at most 5 (the code I wrote has Cyclomatic complexity of 4). You will get partial credits for this particular aspect if your code has Cyclomatic complexity of no more than 7. The Cyclomatic complexity measures the complexity of the control flow of your code. If you use a nested loops, for example, Cyclomatic complexity would increase.

### Tips for how to reduce the complexity of code

1. Use the existing code/data structure provided by C++ STL as much as you can. Don't reinvent the wheel. The STL containers set/vector/map/... are quite powerful. Also the provided algorithms such as sort in STL are also quite useful. Using these can greatly simplify your code.

2. Create a common interface for functions with similar interfaces (input/output) and purposes. In this assignment, the interface functions to configure the rules fall into this category. Then you can use a table (array) to store function pointers to these functions and invoke them in a uniform way. This can avoid long if-conditional statements. In fact, this is somewhat related to object orientation (OO). In some old C programs, function pointers are used to achieve some aspects of object orientation. We haven't covered OO yet. But working with function pointers this way can let you get a taste of OO.