



## CPSC 350: Data Structures and Algorithms Spring 2025

### Programming Assignment 5: Scare Games

Due: See Canvas for the due date

You may work in groups of at most 2 students for PA5 (if you'd like). Be sure to include both names in the heading, README, and header comments.

### The Assignment

#### Background

In this assignment, you'll be implementing a **Tournament Tree** for a game called *Scare Games*. This is a tree structure used to track the winners and losers as competitors face off in rounds until there is a final champion. You'll have two types of tournaments to implement:

1. **Single-Elimination Tournament:** Competitors are eliminated from the tournament after losing once.
2. **Double-Elimination Tournament:** Competitors have a chance to compete in the "Losers' Bracket" after their first loss, giving them another opportunity to reach the finals.

#### Learning Objectives

1. **Understand and Implement Tournament Trees:** Learn how tournament trees work in both single and double-elimination formats.
2. **Work with Vectors:** Use `std::vector` to manage dynamically sized lists of competitors.
3. **Generate DOT Files:** Output the structure of your tournament tree as a DOT file, which can be visualized as a graph.

### Tournament Trees

A **Tournament Tree** is a binary tree where each match between competitors is represented as a node, and the winner of each match progresses up the tree. In a single-elimination tournament:

- Every competitor starts in the initial round.
- Competitors face off, and each match has a winner and a loser.
- The winner progresses to the next round, and the process repeats until one competitor remains.

In a **Double-Elimination Tournament**, competitors move to a "Losers' Bracket" after their first loss, allowing them to still reach the finals if they keep winning in the Losers' Bracket.

## Introduction to Vectors in C++

A `std::vector` is a container in C++ that allows you to store a dynamic array of elements. Unlike regular arrays, vectors can resize themselves automatically as new elements are added or removed, making them more flexible for many programming scenarios.


### Basic Vector Operations:

- **Declaration:** `std::vector<int> numbers;` (creates an empty vector of integers)
- **Add an Element:** `numbers.push_back(5);` (adds 5 to the end of `numbers`)
- **Access Elements:** `numbers[0]` (accesses the first element)
- **Size:** `numbers.size()` returns the number of elements in the vector.
- **Iterate:** Use a for-loop or a range-based for-loop to access elements in the vector.

## DOT Files

A **DOT file** is a plain-text file format used to describe graphs. In this assignment, you'll use DOT files to visually represent your tournament trees. Each node and connection in the tree is represented in the DOT file format, and by visualizing this file, you'll be able to see the structure of your tournament.

You can visualize your DOT files using the [Graphviz Online tool](#):

1. Open the website.
2. Paste the content of your DOT file into the editor. You can use one of the sample output dot files given to test this (  `ScareGames_SampleFiles` )
3. The tool will render a graph that visually represents your tournament structure.

Attached ( [DOTFileGeneratorMethods.cpp](#) ) is the code to generate a DOT file from a tournament tree. You will need to integrate this code into your project to save the structure of your tree. The `saveTreeAsDot` function writes the tree structure to a DOT file.

## Design

- **Monster Class:**
  - Represents each competitor in the tournament.

- Attributes: Each Monster object has a name and a scream power level.
- Implement comparison operators in this class to compare monsters based on their scream power levels, which will be essential for determining match outcomes in the tournament.
- **TournamentNode Class:**
  - Represents a single match node within the tournament tree.
  - Attributes:
    - **winner:** A pointer to the competitor who won this match.
    - **left** and **right** pointers: These link to the child nodes, representing the competitors in this match.
  - This class will be crucial for organizing and tracking each match within the tournament.
- **TournamentTree Class:**
  - Manages the tournament process and tracks the progression of winners and losers.
  - **Single-Elimination Tournament Logic:**
    - Competitors are eliminated after one loss.
    - Only one bracket is used, and the final champion is the winner of the last match in this bracket.
  - **Double-Elimination Tournament Logic:**
    - Competitors are eliminated after two losses.
    - Competitors who lose in the main bracket are moved to the "Losers' Bracket."
      - Add all losers to a losers vector as they lose in the winner's bracket. Then, once the winner's tree is done, create the losers' tree based on the losers vector. The losers' tree will contain all monsters except the winner of the winners' tree.
      - The ultimate winner of the losers' tree plays against the ultimate winner of the winners' tree.
    - The final match is between the winners of the main bracket and the losers' bracket.
  - **saveTreeAsDot Method:** This method generates a DOT file from the tournament tree, which can be used to visualize the tournament. **We've provided the code for this functionality**; implementing the tournament logic will allow you to see the match structure. [DOTFileGeneratorMethods.cpp](#)
  - If there is an uneven number of competitors in a round, one competitor will advance to the next round without a match. This is common in tournaments with non-power-of-two participant numbers (e.g., 5, 7, 9).
    - In your implementation, when there's an odd number of competitors in either the winners or losers rounds, the last competitor in the round without a matching opponent is added directly to the next round.
    - **Example Scenario** if there are 5 monsters in a round:
      - Monsters 1 and 2 compete, and one advances.
      - Monsters 3 and 4 compete, and one advances.

- Monster 5 has no opponent, so it automatically moves to the next round.
  - You can add the monsters from the vector in any order you like. Just keep in mind that the order can change the results and the structure of the tree. In the sample input, they were added in the order they appear in the vector. So if you want to check your work against the sample output, it's best to add them in that same order.
- **RunScoreGame Class:**
  - Handles input/output operations and the setup of the tournament.
  - Create the `runTournament()` method in this class to initiate the tournament. This method will be called from `main` with parameters specifying the tournament format, input file, and output file.
- **Main File:** will contain a main method which will be relatively straightforward since its primary role is to handle command-line arguments and start the tournament simulation by calling the `RunScoreGame` class.
  - Should get the input file name and whether the tournament should be double or single as command line arguments
  - Sample run command: `./monster.exe monsters.txt double`

## Sample Inputs & Outputs

Sample files used and generated when running the following command:

```
./monster.exe monsters.txt double
```

📁 ScareGames\_SampleFiles

File containing dot file methods: [DOTFileGeneratorMethods.cpp](#)

## Due Date

This assignment is due at 11:59 pm on see Canvas for the due date. Submit all your commented code as a zip file to Canvas. The name of the zip file should be `LastName_FirstInitial_A5.zip`

## Grading

Please fill out [this form](#) if you have any inquiries on your PA grades! This will go to our grader. If you have a personal question, please message your instructor first and they'll let you know how to proceed.

Grades will be based on correctness, adherence to the guidelines, and code quality (including the presence of meaningful comments). An elegant OO solution will receive much more credit than procedural spaghetti code. I assume you are familiar with the standard style guide for C++, which you should follow. (See the course page on Canvas for a C++ style guide and Coding Documentation Requirements.)

Code that does not follow the specification EXACTLY will receive an automatic 25% deduction.  
Code that does not compile will receive an automatic 50% deduction.

## Readme & References

More info on Canvas!

- **README file:** All source code will be accompanied by a plain text README file. We encourage students to take advantage of markdown technology if they are inclined to do so. This file will contain:
  - The following identifying information:
    - Full name
    - Student ID
    - Chapman email
    - Course number and section
    - Assignment or exercise number
  - A list of all source files submitted for the assignment
  - A description of any known compile or runtime errors, code limitations, or deviations from the assignment specification (if applicable)
  - A list of all references used to complete the assignment, including peers (if applicable)
  - Instructions for running the assignment. (Typically applicable in advanced courses using build systems or third-party libraries) for us, it would be something like this:
    - To compile: `g++ *.cpp -o A1.exe`
    - To run: `./A1.exe input.txt output.html`
- **In-code citations:** You are allowed to use small, isolated lines of code from external sources in your programming assignments as long as they are appropriately cited. Any time you are including code that you did not write yourself, it must be cited. This includes sources from StackOverflow and similar forums, other current or previous students, tutors, books, tutorials, etc. You should wrap the copied code in a comment denoting the start and end of said code like this:

```
# code that is not copied (you wrote this)
someCode = 0;
print("hi I wrote this!")

""""BEGIN CODE FROM SOURCE: link/name of source""""
print("I did not write this");
""""END OF CODE FROM SOURCE: link/name of source""""

""""BEGIN CODE FROM CHAT GPT, PROMPT ASKED: how do you ...?
""""
print ("I did not write this");
""""END OF CODE FROM CHAT GPT""""

# back to code that you personally wrote
```

```
codeIWrote = 10;
```

- **Info in code files:** Additionally, all source files will start with a header comment containing the following items (one per line):
  - Full name
  - Student ID
  - Chapman email
  - Course number and section
  - Assignment or exercise number