Laser Tag

Due: Monday February 6th - pull request issued on Github.

Introduction

Help! Mustang Laser Tag's scoring system just experienced a fatal segmentation fault and they need you to implement a new system. The old system just couldn't handle the masses of young mustangs wanting to enjoy a harmless game of Laser Tag to pass the Thursday and Friday nights. They need you to write a piece of software that will take input from multiple files and then output various reports, depending on what the players want to see.

Your Task

You'll read in data from three different text files:

1. Team member names for team A, a Team Member Input File
2. Team member names for team B, a Team Member Input File
3. Laser Tag Match file giving information about what player successfully tagged another, a Match File

Your program will aggregate the data from the Match File to be used for output. Let's take a look at each input file type.

Team Member input file

- The first line of this file will contain the team name. The name of the team may contain spaces.
- The second line of this file will contain a positive integer 1 < n <= 10 representing the number of members on the team.
- The next n lines will contain team member's id number (a positive integer), a single space, and the team member's name. Names may contain spaces.

The name of the file will be provided via a command-line argument when the program is executed. As to be expected, each execution of the program will require two different Team Member input files.

Match File

- The first line of the file will contain a positive integer 1 < n <= 100000 representing the number of 'tags' that happened during a particular match
- The next n lines will contain 4 fields each separated by a single space:
    - Field 1: Tagger's ID number, a positive integer
    - Field 2: Target's ID number, a positive integer

- Field 3: Number of milliseconds since the match started, a positive integer
- Field 4: Location ID where target was hit (chest, back, shoulder, etc), a positive integer

For each execution of the program, only one Match File will be used.

Tag Scoring

The score received for a tag depends on the location the target was tagged. Locations have the following ID's and point values:

- 1 (Back): 5 points
- 2 (Chest): 8 points
- 3 (Shoulder): 10 points
- 4 (Laser gun): 15 points

Output File

All results should be written to an output file. Your program should have three different verbosity levels of output:

1. Low verbosity (vlow): Output the cumulative score of each team on a separate line and on line 3, output the winning team.
2. Medium Verbosity (vmed): For both teams, output the team name and then each team member's name followed by the number of successful tags they achieved (i.e., how many total times they successfully tagged another player). Then output the highest-ranking member for each team followed by the winning team.
3. High Verbosity (vhigh): Print the team name. Then, for each player on a team, list each player on the opposing team that they tagged and the number of tags achieved against that player. This should be followed by the total number of tags for that player. After all of the players for one team have been output, print the total score for that team. After player detail for both teams have been output, output the winning team.

General comments about output

- When multiple players for a team are being output, always order based on decreasing number of successful tags.
- When outputting teams, order alphabetically by team name.

Command-Line Parameters

The input data files and verbosity level will be passed to the program as command-line parameters. The parameters will have the following order, where verbosity is one of vlow, vmed, or vhigh mentioned above:

```
./a.out TeamA.txt TeamB.txt MatchFile.txt OutputFile.txt verbosity
```

Sample Input & Output

Sample Team A Input File (assume it is named cowboys.txt):

```
The Cowboys
3
1 Bob
2 Sally
3 Sam
```

Sample Team B Input File (assume it is named sharks.txt):

```
The Sharks
3
4 Jack
5 Jill
6 Kenny
```

Sample Match File (assume it is named match1.txt):

```
6
6 1 8388 2
3 5 10111 1
4 2 12123 3
2 4 13131 1
2 5 14155 2
3 5 15555 4
```

Sample Output File (with low verbosity):

```
The Cowboys: 33 points
The Sharks: 18 points
Overall Winners: The Cowboys
```

Sample Output File (with medium verbosity):

```
The Cowboys
      Sam had a total of 2 tags
      Sally had a total of 2 tags
      Bob had a total of 0 tags

The Sharks
      Kenny had a total of 1 tag
      Jack had a total of 1 tag
      Jill had a total of 0 tags

Best score from The Cowboys: Sam (20 points)
Best score from The Sharks: Jack (10 points)
The Cowboys: 33 points
The Sharks: 18 points
Overall Winners: The Cowboys
```

Sample Output File (with high verbosity):

```
The Cowboys
      Sam tagged Jack 0 times
      Sam tagged Jill 2 times
      Sam tagged Kenny 0 times
      Sam had a total of 2 tags
      Sally tagged Jack 1 time
      Sally tagged Jill 1 time
      Sally tagged Kenny 0 times
      Sally had a total of 2 tags
      Bob tagged Jack 0 times
      Bob tagged Jill 0 times
      Bob tagged Jenny 0 times
      Bob had a total of 0 tags
      The Cowboys: 33 points

The Sharks
      Jack tagged Bob 0 times
      Jack tagged Sally 1 time
      Jack tagged Sam 0 times
      Jack had a total of 1 tag
      Kenny tagged Bob 1 time
      Kenny tagged Sally 0 times
      Kenny tagged Sam 0 times
      Kenny had a total of 1 tag
      Jill tagged Bob 0 times
      Jill tagged Sally 0 times
      Jill tagged Sam 0 times
      Jill had a total of 0 tags
      The Sharks: 18 points

Winners: The Cowboys
```

Assumptions

- No IDs will be repeated between files for a team.
- All files will be properly formatted and contain valid data
- Players on the same team will not tag each other (system won't allow it, so don't worry)

Comments about this Project

This is your opportunity to make a great first impression on the Prof/TAs in CSE 2341. We will be looking for simple, elegant, well-designed solutions to this problem. Please do not try to "wow" us with your knowledge of random, esoteric programming practices. Here is a list of things that you might want to consider while tackling this project:

- Procedural vs Object Oriented Design
    - A seemingly infinite amount of software has been designed, implemented, deployed, maintained, updated, and redeployed using both of these paradigms. One could argue for days, or week even, about which is the "better" paradigm for modern software development. Regardless of which paradigm you choose to use, the most important thing is that you produce an elegant solution following solid software development practices.
- File input and output
    - It is so important to be able to read from and write to data files. Think about some software program that doesn't use files...
- Just the right amount of comments in just the right places
- Minimal amount of code in the driver method (main, in the case of C++)
    - The code in main should be minimal and only used to "get the ball rolling."
- Proper use of command-line arguments

Implementation Requirements

- You must use your custom implemented string class. You may not use the STL string class or any other available string class from the internet. Additionally, you may NOT use c-strings (null-terminated character arrays) except to implement your custom string class.

What to Submit

During lab in the coming weeks, you will cover a basic introduction to GitHub and how to use it to submit your projects. You will need to push and tag your project by 6am on February 6, 2017.

Grading

There are several ways to solve this problem. You should attempt to find the most “efficient” solution in both space usage and time. But, what does it mean for a solution to be efficient? How do you know your solution is more efficient than another solution? These are just things to think about.

Grading Rubric

	Points Possible	**Points Awarded**
string class implementation and tests	25	
Solution Design	20	
Source Code Formatting	20	
Source Code Comments	20	
Implementation and Functionality	40	