SENG1000 – C/C++ PROGRAMMING

MAJOR ASSIGNMENT 4 - FILES AND STRING PARSING

OVERVIEW

Write a program to generate sport team win-loss records from game results in files.

NOTE: This is a harder assignment than the previous major assignments.

GENERAL COURSE OBJECTIVES ADDRESSED IN THIS ASSIGNMENT

- Use file I/O (text mode).
- Use C-style strings and string functions.
- Use best practices for magic numbers.
- Design computer programs for modularity and maintainability.
- Demonstrate the development cycle for C programs.
- Follow stated requirements.

ACADEMIC INTEGRITY AND LATE PENALTIES

- Link to Academic Integrity Information
- Link to Late Policy

EVALUATION

- The evaluation of this assignment will be done as in the Marking lecture from Week 2.

PREPARATION

- View Week 6, 10, and 11 content.

REQUIREMENTS

Data File Requirements

- With a text editor, you will create multiple files.
- The first file is teams.txt. teams.txt contains the names of other files, one file per line.
- You will also create files that correspond to the filenames found in teams.txt.
    - Each of these files will contain the results of a game, one game result per line.
    - The format of the game result line is:
        - opposingTeamName,yourScore-opponentScore

- All files will be in the "current directory". If you are running from within Visual Studio, that directory is accessible through right-clicking on the solution in the Solution Explorer and choosing *Open Folder in File Explorer.*
- An example of the contents of teams.txt might be:

```
Toronto Maple Leafs.txt
NewYorkYankees.txt
Manchester United.txt
Chennai Super Kings.txt
```

- An example of the contents of *Toronto Maple Leafs.txt* might be:

```
Arizona,3-2
St. Louis,2-2
New York Rangers,4-5
```

  In this example, the Toronto Maple Leafs beat Arizona 3-2, tied St. Louis 2-2, and lost to New York Rangers 5-4 (note that the score appears backwards since Toronto's goal value is always first).

  - In this example, "Toronto Maple Leafs" would be considered the "primary team". "Arizona", "St. Louis", and "New York Rangers" would be considered the "opponent team".
  - The name of the "primary team" is obtained by removing any extension from the appropriate filename. So, for example, the team name for "Toronto Maple Leafs.txt" is "Toronto Maple Leafs".
- Do not create any lines that are greater than 40 characters long. Your submission will not be tested with lines that are longer than that.
- Filenames found in the teams.txt file will have extensions (this is not necessarily the case for the Bonus, mentioned below).

Calculation Requirements

- In main(), you will get filenames from teams.txt. Each of those filenames contain game results for the team named in the filename (before the extension). For each of the filenames that you get, you will get each game result, indicating how that team did in their games. In processGames() (which is a function mentioned later), you will total all wins, losses, and ties for that team and then display a final winning percentage (calculated as (2 * wins + ties) /(2 * (wins + losses + ties))).
  - If there was at least one valid game, display the final winning percentage.
    - The text must be similar to (accounting for differences in test data) "Season result for Toronto Maple Leafs: 0.500 (1-1-1)"
    - In this output example, the "(1-1-1)" represents wins first, next losses, and lastly ties.

        - The calculation of the final winning percentage must be displayed with exactly three decimal places using what was covered in the "printf() revisited" lecture.
    - If you are unclear about this, please create some data files as above and run the sample executable.

Game Result Display Requirements

- Before processing the game result file, display "Processing " followed by the game result filename and a colon and '\n'.
    - e,g, "Processing Toronto Maple Leafs.txt:"
- For each game line being processed, display one of the following three outcomes (taken from the previous example):
    - "Toronto Maple Leafs beat Arizona 3-2"
    - "Toronto Maple Leafs and St. Louis tied at 2"
    - "Toronto Maple Leafs lost to New York Rangers 5-4"
    - Since the above are examples, it is hoped that you would realize that the names and scores would vary depending on your data files.
- Precede the outcome output with a TAB character ('\t') so they are indented.

Modular Function Requirements:

- You must process each team game result file within a function called processGames().
    - processGames() takes exactly one parameter, which is the filename for the team's game result file.
    - processGames() returns an int that is a status. If the processing of the file went perfectly OK, you can return a status that indicates that. Otherwise, you can return a status indicating a problem of some type.
        - What those statuses are is up to you. You absolutely have to distinguish between "it worked perfectly" and "it didn't work perfectly".
    - In this function, you will total all wins, losses, and ties for that team and then display a final winning percentage.
    - The object of this is to demonstrate that you should have this code **outside** of main() and in a function of its own. This function is called from main().
- You must parse (or extract information from) each game result line using a function called parseLine().
    - parseLine() takes exactly four parameters:
        - the array containing the game result string
        - an array that you will fill in with the opponent's name (found in the game result string)
        - a pointer to the score for the primary team
        - a pointer to the score for the opponent team

- parseLine() will return a status indicating if the parsing of the game result worked perfectly or not.
- parseLine() is called from processGames().
- parseLine() must not have any output. Any errors must be handled by the calling function (processGames()) through the use of the status being returned from parseLine().
- Again, the object of this is to demonstrate that you should have this code **outside** of main() and **outside** of processGames() in a function of its own.

Error Checking Requirements

- Error checking must be done for **all** file I/O function calls. If there is such an error, display an appropriate error message.
    - If the error occurs in working with opening teams.txt, quit the program after displaying the error message. Otherwise, skip the failed file operation and continue with the rest of the program.
    - Although it would be normal to close the opened file(s) upon an error, it is not required in this assignment (for simplicity).
- There might be empty/blank lines in the teams.txt file. If that is the case, simply skip those lines and do not indicate any error.
- There might be empty/blank lines in the game results files. If that is the case, simply skip those lines and do not indicate any error.
- For any non-empty/blank lines in the game results files, you can assume that they will have proper data (i.e. a valid opponent name and score, with appropriate delimiters) **except** for the possibility of a missing comma or a missing dash.
    - If either the comma or dash is missing, display an appropriate error message and skip that game result line (do not quit the program).
- All messages that are displayed must be correct in both grammar and spelling. They must all end with a '\n'.

Other Requirements

- Display a blank line after each team's data file is processed.
    - If you are unclear about this, please create some data files as above and run the sample executable.
- You must use C-style null-terminated strings and not C++ strings.
- You must use printf() and not cout.
- You must use the string functions mentioned in the "Other Functions" lecture. This is extremely important.
    - Do not use strtok(). Using strtok() will result in a cap of 40 and will not allow for a resubmission.

    - Do not use any function that is related to scanf(). This includes sscanf(), fscanf(), sscanf_s(), and fscanf_s(). Using any such function will result in a cap of 40 and will not allow for a resubmission.
- Use best practices with respect to Magic Numbers. This would especially be important with respect to array sizes and status constants.
- As usual, you are responsible for the normal course requirements for assignments (e.g. no global variables, initialize variables when you declare them, etc.).

BONUS REQUIREMENTS

- There are not many error checking requirements in this assignment compared to what would normally be expected in industry. If you want to do more extensive error checking, there is a possibility for a 50 mark bonus.
    - In order to attempt to qualify for the bonus, you must put the following comment immediately after the file header comment in your code:
        - // more error checking
    - If you indicate that you will be doing more error checking, your submission will be tested with more non-compliant data. Specifically, **anything** could be wrong with the input data **except** that the input lines will not be longer than specified previously.
    - A crash will invalidate the bonus.

CHECKLIST REQUIREMENTS

- Create a requirements checklist. This should contain the specific requirements from this assignment as well as any relevant requirements that have been covered in lecture or that are found in the SET Coding Standards or SET Submission Standards. Do it in whatever form you wish. Hand in your completed checklist in PDF form as checklist.pdf. Not having this checklist will result in a cap of 80 on your mark.

GIT REQUIREMENTS

- You must use GitHub under GitHub Classroom for revision control of your source file. The details were mentioned in an eConestoga announcement earlier. You must click on the link for the Major 4 Assignment in GitHub Classroom (mentioned in an eConestoga Announcement).
- Make git commits every time that you have something working or otherwise substantial.
- It is expected that you will make **regular** git commits with **meaningful** commit comments (describing the changes that you successfully made since the last commit).
    - On an assignment of this size, I would expect at least 10 distinct and meaningful commits. They should be done as you work on your code, so that they are separated by a reasonable amount of time (e.g. it would not be likely that you would complete this assignment in 20 minutes).
    - Take this requirement seriously.

- Failing to use GitHub Classroom will cap your mark at 60.

FILE NAMING REQUIREMENTS (BESIDES THOSE MENTIONED ABOVE)

- You must call your source file m4.cpp.
- You must call your checklist checklist.pdf.
- You should submit the data files that you created. However, these are not required and will not be marked **but** it will benefit you to provide them for me if your submission doesn't work as it should (so I can test with them to see what you saw). In general, though, I will test with my own test files.

SUBMISSION REQUIREMENTS

- Do not hand in any other files besides those mentioned in the File Naming Requirements.
- Follow the instructions in the SET Submission Standards and the lecture on Submitting Assignments to submit your program. Submit both files to the correct Assignment folder.
- Once you have submitted your files, make sure that you've received the eConestoga e-mail confirming your submission. Do not submit that e-mail (simply keep it for your own records until you get your mark).

HINTS

- Don't forget that fgets() often puts a '\n' at the end of the input line. This would be significant when dealing with filenames in teams.txt.
- Don't forget that text files do **not** necessarily have .txt extensions. Other than for “teams.txt”, your source code should **not** have ".txt" anywhere in it.
- The sample executable is your friend.
    - The sample executable took about 200 lines of source code.
    - The sample executable has a bit more error checking than is required by this assignment but not as much as the bonus would require.
- Leaving this for the last few days before the due date would be a **big** mistake.
- This assignment has a special due date. Please consult the eConestoga assignment dropbox for the due date.