Programming II (420-2P6-AB)  
Winter 2025 – Section 1

**Assignment # 4 [Worth 4%]**

**Part 1 (Algorithm and Pseudo-Code): Due Tuesday, Mar 18 at midnight on Lea**

**Part 2 (Confirmation of Git & Testing Plan): Due Thursday, Mar, 20 at midnight on Lea**

**Part 3 (Code – Working and Tested): Due Sunday, Mar 23 at midnight on Lea**

*Deductions for late submissions is 10%/day, up to a maximum of 3 days including weekends.*

*30% penalty if submission is missing appropriate Revision History.  
Cheating or Plagiarism including any autogenerated code/AI tools, will result in 0%   
and the student(s) will be automatically reported to Registrar.*

# Topics Covered & Objectives:

* Write algorithm & pseudo-code.
* Create proper functional decomposition with method reuse
* Store related data in structs
* Use lists to store multiple structs
* Insert elements into a list in sorted order
* Save data (in list of structs) to a file in comma-separated-value (CSV) format
* Load csv data from a file into a list of structs
* Display strings using appropriate formatting
* Add proper comments
* Use of Git
* Perform proper testing
* Respect coding guidelines: respect the Do’s and Don’ts.

# Instructions

* Submit one Visual Studio .Net Core Console Project
* All programs should have pause at the very end: **Console.ReadLine();**
* Proper coding conventions will be graded: avoiding magic numbers, using CONSTANTS, proper naming convention (camelCase, PascalCase), code efficiency.
* **Comment your code** (worth 10%):
  + Add comment before each method header, explaining the algorithm of that method. Your comments must be as detailed as the example on the last page.
  + Also add comments inside your code: at the end of every line of every declared variable and any line that is considered important to the understanding of the algorithmic logic. Ask the teacher if unsure.
* For each question, in comments write the tester name and the relationship to you. Example:

**/\* Problem #1: Program Title   
What the program does in your own words.   
Tester: Bob Doe (student in class). \*/**

* C# program header comments:

/\*

\* Programming 2 - Assignment 4 – Winter 2025

\* Created by: YOUR NAME and ID

\* Tested by: TESTER NAME

\* Relationship: colleague/father/mother/etc

\* Date: TODAY'S DATE

\*

\* Description: The goal of this program is to do the \* following…

\*/

* Create 1 Visual Studio project, output the following header.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Welcome to "Programming 2 - Assignment 4 – Winter 2025"

Created by <YOUR NAME> (<ID>) on <DATE>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**↓ Read Next page ↓**

# Leaderboard

## Objective:

Create a leaderboard that stores information about the winners of a game in order of their score. Save the leaderboard to a file and load the leaderboard from a file to support persistent storage. Display the leaderboard after loading, as well as redisplay the leaderboard when it changes.

## Requirements:

### 1. Leaderboard setup

- Create a struct that contains the fields on information you will store for a winner on the leaderboard. The struct must store a player name, a player score, and a game ending time (stored as a DateTime). It must also store at least two other pieces of information of your choice. (i.e., minimum of 5 pieces of information per leaderboard entry).

- Use a list to store the leaderboard data for all winners.

- Display the leaderboard after each change is made to it.

### 2. Leaderboard rules

- A new winner should be inserted into the list in order of decreasing score. In other words, the first element in the list should always be the player with the highest score, the second element should be the second highest, and the last element should have the lowest score on the leaderboard.

- A player should only appear once on the leaderboard, and their entry should represent the highest score they have had. A player is identified uniquely by their name.

- If a player gets the same score as someone already on the leaderboard, then they will appear after that player on the leaderboard. For example, if you tie the high score, you won’t become first in the list, but you will still be added to the list (most probably in second place unless that player also has the same score, in which case third or later as appropriate).

- The user can remove an entry from the leaderboard.

- The Leaderboard will have no fixed size.

### 3. Input

- All information about winners will be provided by the user.

- Ensure user input is valid based on sensible rules that you will define. For example, valid names, legal score values, valid values for other fields you have identified.

- Every field must have some type of validation.

- All input provided by the user for a particular winner should be stored in a struct.

### 4. Display Leaderboard

- Display the Leaderboard on the console in a nicely formatted manner.

- Use appropriate formatting for times

- The highest score entry should be visibly different than all other entries to highlight that it is the high score.

- At all times, the latest leaderboard should be displayed.

### 5. Save to/Load from File

- Provide the capability to save the current leaderboard to a file in comma-separated value (CSV) format

- The leaderboard entries should be saved in the proper sorted order.

- The user can specify the name of the file to use

- Provide the capability to load a saved leaderboard from a file.

- When a leaderboard is loaded, it replaces any prior leaderboard in memory.

Note: Take care when saving and loading the DateTime field value – you must convert to a string and then convert back to a DateTime.

### 5. Leaderboard Operations

- In its top-level menu (i.e., in Main), the program should give the user the option to:

1. Add a winner to the leaderboard

2. Delete an entry from the leaderboard

3. Save the leaderboard to a file

4. Load the leaderboard from a file

5. Clear the leaderboard

6. Quit

- Clearing the leaderboard only clears the leaderboard in memory. It does not clear any saved files.

- If the user decides to quit, the program should ask them for confirmation if there are any unsaved changes to the leaderboard.

- When submitting your code, Include a saved file containing a saved leaderboard so that it is easy to test the load feature.

## Bonus Features (Optional)

### Optional: Auto Save/Load (5%)

- When the program starts, automatically load the leaderboard that was active when the user last quit the program. (Hint: Use a specific filename)

### Optional: Dynamic Leaderboard (5%)

- Make the leaderboard dynamic in size by applying the following rule: The lowest score in the leaderboard must be bigger than half the value of the highest score. All previous winners whose scores are too low should be removed from the leaderboard when a new winner is added.

# Innovation Hub Variation

If you are participating in an Innovation Hub project, then instead of creating a leaderboard containing winners, create a data storage capability for the type of data that your innovation hub project is using. This may be data related to users, transactions, content, etc. Pick a single key type of data that contains multiple field values.

Your data must include a minimum of 5 fields of information, including one string, one number and one DateTime.

You must decide on appropriate validation rules for the field values.

You must decide what feature of the data will be unique. In other words, use the field value to ensure that a duplicate entry is never made. For example, username may be unique, if storing user accounts.

The data should be maintained in sorted order according to a data field that makes sense for your data. This may involve an alphabetical sort on username, for example, if storing user accounts.

Display the data that you have stored in a meaningful way. The first element in the list should be highlighted somehow to distinguish it from the others.

When submitting your code, include a saved file containing some data so that it is easy to test the load feature.

Auto save/load bonus is the same

For the dynamic data bonus, pick a rule that makes sense in the context of your data. Use that rule to automatically prune the data list when appropriate. Clearly document the rule you have chosen.

## Submission Guidelines

- For Part 1, in any visual format, submit a block diagram showing the high-level algorithm. Make sure to capture key requirements, functional decomposition and function reuse.

- For Part 1, in any written format, also submit pseudo-code for all functions in your program. Pseudo-code should clearly indicate key control structures (loops, ifs, etc.), key storage and key computations needed.

- For Part 2, make sure to save your testing plan inside your project and commit/push it to GitHub. In addition, on Lea, submit a zip file contain (a) Your project files, (b) A screenshot showing your commit history, and (c) A README.txt containing a link to your git repository. Make sure to share your repository with git user “tshussain”. (Or, you can make your repository public).

- For Part 3, make sure all code and documents are saved in your project and committed/pushed to GitHub. You should have made multiple commits while working on your project (e.g., when completing each capability, and/or after fixing key bugs). In addition, on Lea, submit a zip file contain (a) Your project files, (b) A screenshot showing your commit history, and (c) A Readme containing a link to your git repository.

- NO GLOBAL LIST or ARRAY VARIABLES. Send everything to methods as parameters. (You may use appropriate Global variables for Random, sizes, etc.). Note: The reason for this restriction is to make sure you demonstrate understanding of how to use lists/arrays as parameters, and how to make changes to those lists/arrays within methods.

- Add well commented code.

# Submission Instructions:

1. Send one Visual Studio project with all code in Program.cs
2. Git/GitHub Setup with appropriate commits (10%)
   1. The teacher needs to be able to access your GitHub repository to confirm git usage. Make sure to share your repository with user “tshussain”.
   2. You should have multiple meaningful commits. Recommendation is to commit (at a minimum) after completing each key requirement (e.g., after each method).
   3. Don’t forget to push your final changes before submitting!
   4. Include the GitHub link in a README.txt file in your project folder.
3. Testing (worth 10%)
4. Create an appropriate testing plan (as per instructions in Assignment 2)
5. Get your program tested by someone else (i.e., not just yourself).
6. DO NOT let your tester have access or look at your source code.   
   That is cheating/plagiarism. Instead, let your tester run your program and interact with the executable file.
7. **Comment your code** (worth 10%):
   1. Marking of comments will be stricter for this assignment. Make sure to comment in detail! Every uncommented method will be penalized.
   2. Have the following commentsin the program’s header of every class:

/\*

\* Programming 2 - Assignment X – Winter 202X

\* Created by: YOUR NAME and ID

\* Tested by: TESTER NAME

\* Relationship: colleague/father/mother/etc

\* Date: TODAY'S DATE

\*

\* Description: The goal of this program is to do the \* following…

\*/

* 1. Add comment before each method header, explaining the algorithm of that method, the parameters, return values and any exception thrown, and the detailed pseudo-code. Optionally, you can also include examples of how to call the method. Your comments must be as detailed as this example. If the pseudo-code for a method is included in a separate document, then you do not need to include it in the header comments for that method.

/\*\*

\* Computes the average of a list of numerical values.

\*

\* @method This method calculates the average by following the formula:

\* Average = (Sum of all values in the list) / (Number of values in the list)

\*

\* @param values A list of numerical values (integers or floats) for which the average needs to be computed.

\* The list must contain at least one value. If the list is empty, the method will throw an exception.

\*

\* @returns The computed average of the values in the list. The average is calculated by summing all the elements

\* in the list and dividing the sum by the number of elements.

\*

\* @exception ArgumentException Thrown when the list is empty, as dividing by zero is undefined.

\* @exception InvalidOperationException Thrown if any element in the list is not a numerical value.

\*

\* @example

\*

\* // Example 1

\* var result = computeAverage(new List<double> { 1, 2, 3, 4, 5 });

\* Console.WriteLine(result); // Output: 3.0

\*

\* // Example 2

\* var result = computeAverage(new List<int> { 10, 20, 30 });

\* Console.WriteLine(result); // Output: 20.0

\*

\* // Example 3 (Throws ArgumentException)

\* var result = computeAverage(new List<int>());

\* // Throws: ArgumentException: The list cannot be empty.

\*

\* @pseudocode

\* 1. If the list of values is empty:

\* - Throw an exception: "The list cannot be empty."

\*

\* 2. Initialize a variable `sum` to 0.

\*

\* 3. Loop through each value in the list:

\* - Add the value to `sum`.

\* - If a non-numerical value is encountered, throw an exception: "All elements in the list must be numerical values."

\*

\* 4. Calculate the average by dividing `sum` by the number of elements in the list.

\*

\* 5. Return the calculated average.

\*/

public double ComputeAverage(List<double> values)

* 1. Also add comments inside your code: at the end of every line of every declared variable and any line that is considered important to the understanding of the algorithmic logic.   
     Ask the teacher if unsure. Examples:

// array of row sums

// index of row with highSum

// calculate row sum

// if current row sum is highest, update highSum, highIndex

// create a temporary array the size of the columns

1. Make sure to include algorithm design (worth 10%)
2. Make sure to include pseudo-code for each key capability (worth 10%). If this pseudo-code is included in a separate document, then you do not need to include it in the header comments for the methods.
3. Make sure to follow the Do’s and Don’ts of programming (as shared in class and on Teams). (Penalized if not followed, up to 20%)
4. All programs should have pause at the very end, in Main: Console.ReadLine();
5. Make sure your program doesn’t crash right away (30% penalty). You should always submit code that works to some meaningful degree.
6. **Clean your project before submitting**: Visual Studio 🡺 Build 🡺 Clean Solution
7. Make sure the Revision History folder is included in the zip file (and contains the history files capturing your coding progress). Note: .vshistory for VS 2022 (PC), and .history for VS Code (Mac). (30% penalty if missing or vacuous)
8. Make sure code follows the requirements and uses good functional decomposition (worth remaining 50%)
9. Make sure to include a saved file containing a saved leaderboard (or saved data for Innovation Hub students) so that it is easy to test the load feature.
10. Compress the whole Visual Studio project into 1 Zip file (NO other format, not RAR) and submit it on LEA.