Project #04: Exam Analysis in F#

Complete By: ~~Monday October 23rd~~ Tuesday October 24th @ 11:59pm
Assignment: Exam Analysis GUI app in F#
Policy: Individual work only, late work *is* accepted
Submission: Blackboard (see "assignments", "P04 Exam Analysis")

Before you get started…

Before you get started, let's make sure your install of Visual Studio is ready to go. Please check the following:

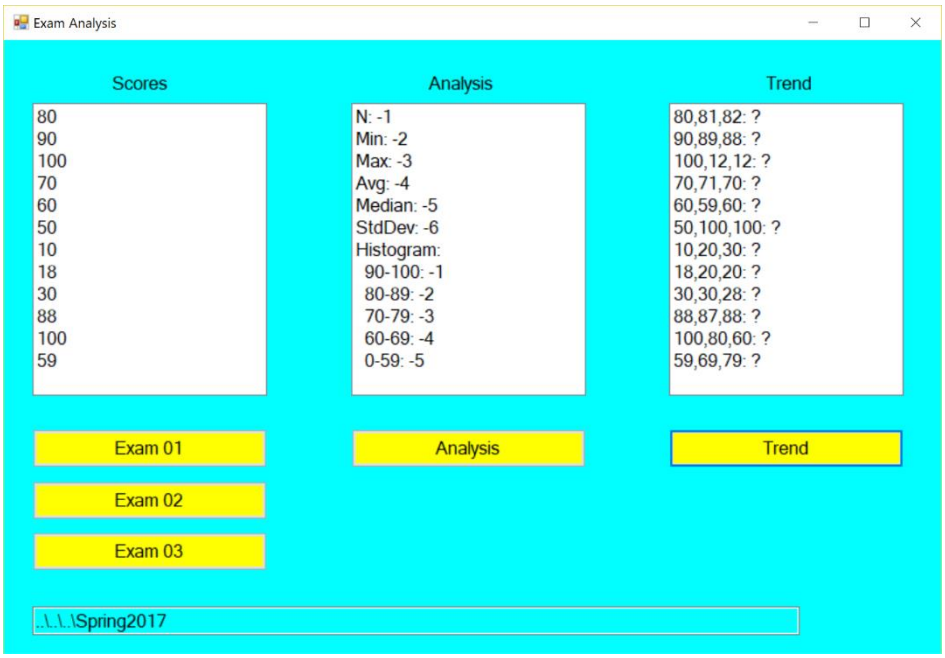
1. **.NET Desktop Development is installed**: the assignment requires that C#, F# and GUI development is installed. If you aren't sure, run "visual studio installer", and see if the workspace ".NET Desktop Development" is checked. If not, check the item and install.

2. **Tabs to Spaces**: F# doesn't like the tab key :-) What I do is configure VS to automatically convert tabs to spaces: Tools menu, Options, scroll to Text Editor, expand, find All Languages, expand, click on Tabs, click radio button that says **Insert spaces**. Indent size of 2?

Assignment

The assignment is to build out a GUI app that reads exam scores from a set of input files, and does some simple analysis: average, min, max, etc. Here's a screenshot:

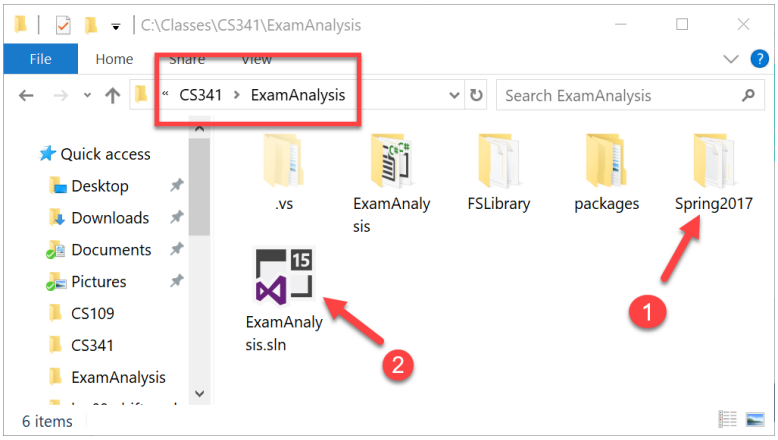
The GUI is provided for you, your assignment is to write the back-end F# library that performs the analysis.



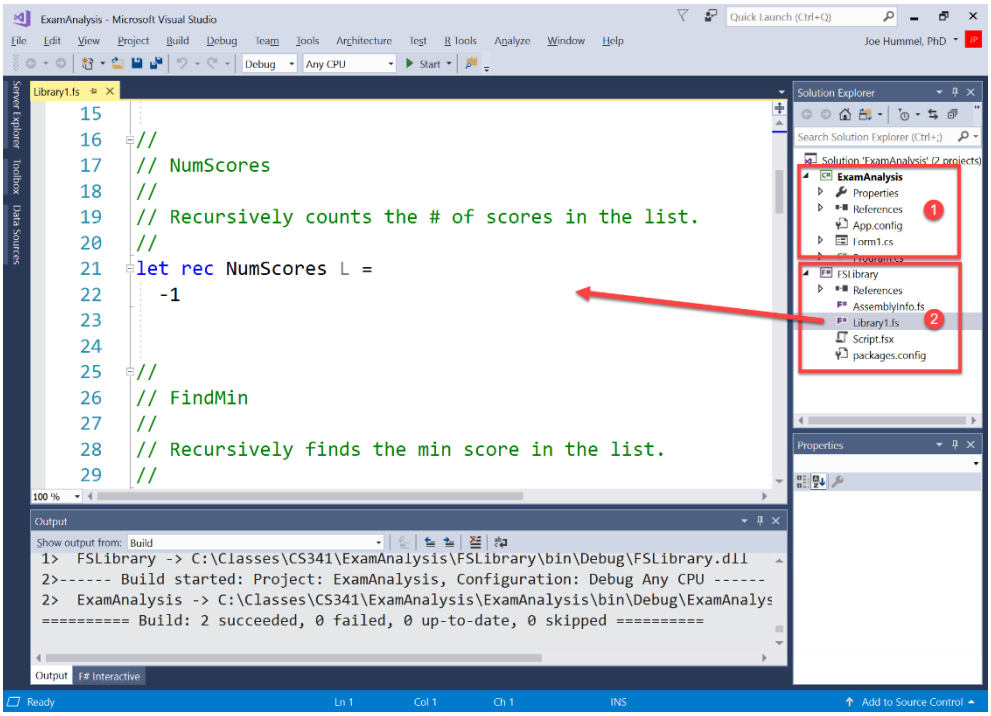
Download

The skeleton of the app is provided. You can download and open as follows:

1. **Download ExamAnalysis.zip**: on the course web page, open "Projects", and then "project04-files". Download ExamAnalysis.zip to your desktop.
2. **Extract VS files**: double-click the .zip file to open, and extract the ExamAnalysis folder to your desktop. Close and delete the .zip file to avoid using it by accident. If you want, move the ExamAnalysis folder off your desktop to another location --- but avoid moving to a cloud-based storage location since Visual Studio's background compiler often doesn't play well with dropbox, onedrive, etc.
3. **Open in Visual Studio**: open the ExamAnalysis folder, and notice 2 things. First, there is a folder named "Spring2017" --- these are an initial set of input files to test against. Second, notice the VS Solution (.sln) file --- double-click on that solution file to open the program in Visual Studio.



4. **Program layout**: the Visual Studio program contains 2 parts: the front-end GUI code written in C#, and the back-end analysis code written in F#. The C# code is written for you, your job is to write the F# code to do the required analysis. Go ahead and open the F# library file "Library1.fs" in the editor to see what is provided:



MyLibrary in F#

Your assignment is to implement all the functions in MyLibrary ("library1.fs"), except the first function **InputScores** (this one is complete as given). The functions are already called by the front-end GUI code, so as you complete each function, you can simply run (F5) and click the buttons to test.

Suggestion #1: after your write an F# function, don't run the app right away. First BUILD the F# code to eliminate any errors --- the F# code is easier to fix if you compile it separately (vs. running the app which will build the entire program, and you'll get a mix of C# and F# error messages). To build the F# library separately, have the F# code open in the editor, and then in the BUILD menu, select "**Build FSLibrary**". Don't run / test until the library builds successfully.

Suggestion #2: you can also test your F# code via the F# Interactive window (View, Other Windows, F# Interactive). Select your code in the editor, right-click, and "Execute in Interactive". Now you can call your functions and test them vs. needing to run the entire app --- don't forget to end your function calls with ";;".

Here's a copy of the "library1.fs" file, the header comments explain what each function is supposed to do. The functions are in order of difficulty, the idea being that the first few functions are good practice for learning F#. That also means some of these functions are easily written by simply calling a built-in library function: this is not allowed, as noted below, and will be graded as incorrect if you don't adhere to the requirements:

NumScores L: write this recursively, do not use higher-order and do not call built-in functions. Tail-recursion not required; if you need helper function(s), add them.

FindMin L: write this recursively, do not use higher-order and do not call built-in functions. Tail-recursion not required; if you need helper function(s), add them.

FindMax L: write this recursively, do not use higher-order and do not call built-in functions. Tail-recursion not required; if you need helper function(s), add them.

Average L: write this recursively, do not use higher-order and do not call built-in functions. Tail-recursion not required; if you need helper function(s), add them.

Median L: use the built-in sort to sort the data, then return the median. Remember that if |L| is even, then the median is the average of the middle elements. Access the middle element(s) however you want.

StdDev L: solve this however you want. Recall that we talked about standard deviation in class on Monday 10/2 (Week 06, Day 15). We are computing the standard deviation where L is considered a complete population.

Histogram L: solve however you want.

Trend L1 L2 L3: solve however you want.

Helpful function: to convert an integer x to a real number, one approach is to use the **float(x)** function.

```
module MyLibrary

#light

//
// InputScores
//
// Given the complete filepath to a text file of exam scores,
// inputs the scores and returns them as a list of integers.
//
let InputScores filepath =
  let L = [ for line in System.IO.File.ReadAllLines(filepath) -> line ]
  List.map (fun score -> System.Int32.Parse(score)) L


//
// NumScores
//
// Recursively counts the # of scores in the list.
//
let rec NumScores L =
  -1


//
// FindMin
//
// Recursively finds the min score in the list.
//
let rec FindMin L =
  -2


//
// FindMax
//
// Recursively finds the max score in the list.
//
let rec FindMax L =
  -3


//
// Average
//
// Computes the average of a non-empty list of integers;
// the result is a real number (not an integer).
//
let Average L =
  -4.0
```

```
//
// Median
//
// Computes the median of a non-empty list of integers;
// the result is a real number (not an integer) since the
// median may be the average of 2 scores if the # of scores
// is even.
//
let Median L =
  -5.0


//
// StdDev
//
// Computes the standard deviation of a complete population
// defined by the integer list L.  Returns a real number.
//
let StdDev L =
  -6.0


//
// Histogram
//
// Returns a list containing exactly 5 integers: [A;B;C;D;F].
// The integer A denotes the # of scores in L that fell in the
// range 90-100, inclusive.  B is the # of scores that fell in
// the range 80-89, inclusive.  C is the range 70-79, D is the
// range 60-69, and F is the range 0-59.
//
let Histogram L =
  [-1; -2; -3; -4; -5]


//
// Trend
//
// Trend is given 3 lists of integer scores:  L1, L2, L3.  The lists are
// non-empty, and |L1| = |L2| = |L3|.  L1 are the scores for exam 01, L2
// are the scores for exam 02, and L3 are the scores for exam 03.  The
// lists are in "parallel", which means student i has their scores at
// position i in each list.  Example: the first exam in each list denote
// the exams for student 0.
//
// Trend returns a new list R such that for each student, R contains a '+'
// if the exam scores were score1 < score2 < score3 --- i.e. the scores
// are trending upward.  R contains a '-' if score1 > score2 > score3, i.e.
// the scores are trending downward.  Otherwise R contains '=' (e.g. if
// score1 < score2 but then score2 > score3).
//
let Trend L1 L2 L3 =
  [ for score in L1 -> '?']
```

Electronic Submission

You need only submit your final "library1.fs" file. Before you submit, be sure the top of the file contains a brief description, along with your name, etc. For example:

```
//
// My F# library for exam score analysis.
//
// YOUR NAME
// U. of Illinois, Chicago
// CS 341, Fall 2017
// Project #04
//
```

Your code should be readable with proper indentation and naming conventions; commenting is expected where appropriate.

When you're ready, submit your "library1.fs" file on Blackboard: look under "Assignments", and submit using the link "P04 Exam Analysis".

Policy

Late work *is* accepted for part 2 of the assignment. You may submit as late as 24 hours after the deadline for a penalty of 25%. After 24 hours, no submissions will be accepted.

All work is to be done individually — group work is not allowed. While I encourage you to talk to your peers and learn from them (e.g. via Piazza), this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is described here:

http://www.uic.edu/depts/dos/docs/Student%20Disciplinary%20Policy.pdf .

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases are handled via the official student conduct process described at http://www.uic.edu/depts/dos/studentconductprocess.shtml .