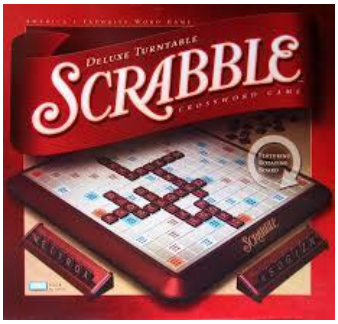
CS 341
Programming Languages
Fall 2017



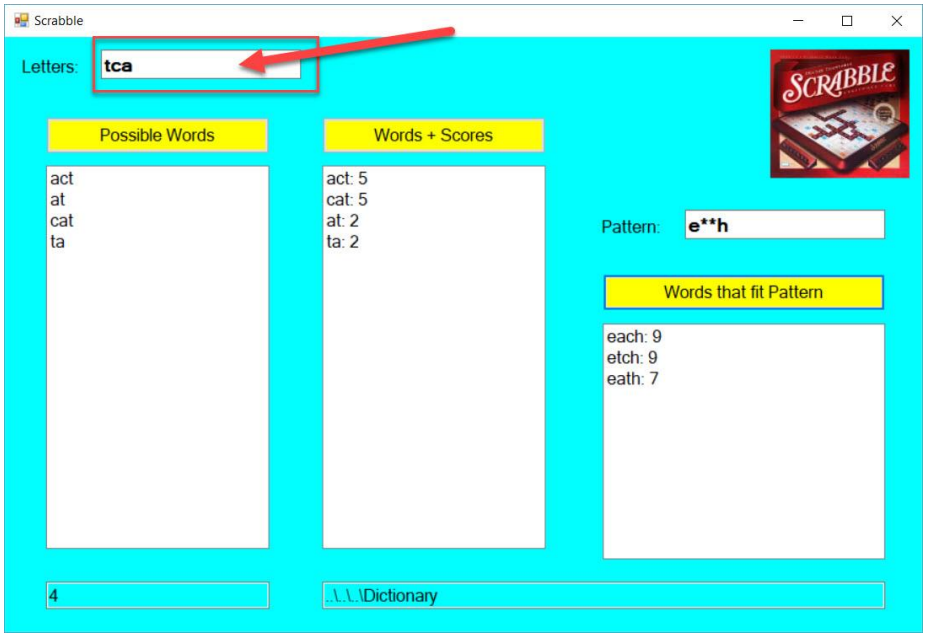
Project #05: Scrabble in F#

Complete By: Monday, Nov. 6th @ 11:59pm
Assignment: Scrabble GUI app in F#
Policy: Individual work only, late work *is* accepted
Submission: Blackboard (see "assignments", "P05 Scrabble")

Assignment

Scrabble is a popular game where players spell words based on a set of letters hidden in their hand. The assignment is to build out a GUI app that takes a collection of letters, and performs 3 operations:

1. Generates all possible words from these letters, where each letter is used at most once.
2. For each possible word, computes the score based on the scrabble value of each letter.
3. Given a pattern of blanks and additional letters, generates all possible new words that can be generated which fit this pattern.



There is an associated "dictionary.txt" file that contains the list of official Scrabble words; for the purposes of this assignment, this dictionary file defines the set of legal words. The screenshot above shows the words that are possible from the letters "tca". Notice the possible words are listed in alphabetical order, and then when scored, they are listed in descending order by score --- if 2 words have the same score, these words are then listed in alphabetical order (e.g. "act" then "cat").

A *pattern* consists of 0 or more explicit characters and 0 or more asterisks (*); an asterisk denotes any single letter a-z. Position matters, so for example the pattern "e**h" matches 4-letter words that start with 'e' and end with 'h'. The * are matched with letters from the "letters" text box. For example, in the screenshot above, the letters are "tca". As a result, the pattern successfully spells 3 Scrabble words: "each", "etch", and "eath".

Assignment Details

The app consists of two parts: a front-end C# GUI and a back-end F# library. The front-end has been written for you; your job is to complete the back-end. In particular, you need to implement the following three F# functions:

```
//
// possibleWords:
//
// Finds all Scrabble words in the Scrabble dictionary that can be
// spelled with the given letters.  The words are returned as a list
// in alphabetical order.
//
// Example:  letters = "tca" returns the list
//   ["act"; "at"; "cat"; "ta"]
//
let possibleWords letters =
  [ “?”; “?”; “?” ]


//
// wordsWithScores:
//
// Finds all Scrabble words in the Scrabble dictionary that can be
// spelled with the given letters.  The words are then "scored"
// based on the value of each letter, and the results returned as
// a list of tuples in the form (word, score).  The list is ordered
// in descending order by score; if 2 words have the same score,
// they are ordered in alphabetical order.
//
// Example:  letters = "tca" returns the list
//   [("act",5); ("cat",5); ("at",2); ("ta",2)]
//
let wordsWithScores letters =
  [ ("?", -1); ("?", -2); ("?", -3) ]


//
// wordsThatFitPattern:
//
// Finds all Scrabble words in the Scrabble dictionary that can be
// spelled with the given letters + the letters in the pattern, such
// that those words all fit into the pattern.  The results are
// returned as a list of tuples (word, score), in descending order by
// score (with secondary sort on the word in alphabetical order).
//
// Example:  letters = "tca" and pattern = "e**h" returns the list
//   [("each",9); ("etch",9); ("eath",7)]
//
let wordsThatFitPattern letters pattern =
  [ ("?", -1); ("?", -2); ("?", -3) ]
```

You are free to implement these functions are you see fit, although you may *not* use mutable variables. All code you write must be functional in nature, i.e. free from side-effects. Also, do not change the design of these functions. The API --- function names, parameters, and types --- must remain the same. You will be submitting only the F# code for grading, and so your functions must integrate with our testing framework.

The provided F# code includes an "Initialization" function that is called from the GUI at program startup. The purpose of this function is to load the Scrabble dictionary into a list to aid with word generation / validation:

```
//
// Initialize:
//
// This function is called ONCE at program startup to initialize any
// data structures in the library.  We use this function to input the
// Scrabble dictionary and build a list of legal Scrabble words.
//
let mutable private WordList = []

let Initialize folderPath =
  let alphabetical = System.IO.Path.Combine(folderPath, "alphabetical.txt")
  WordList <- [ for line in System.IO.File.ReadAllLines(alphabetical) -> line ]
  printfn "%A" (List.length WordList)
```

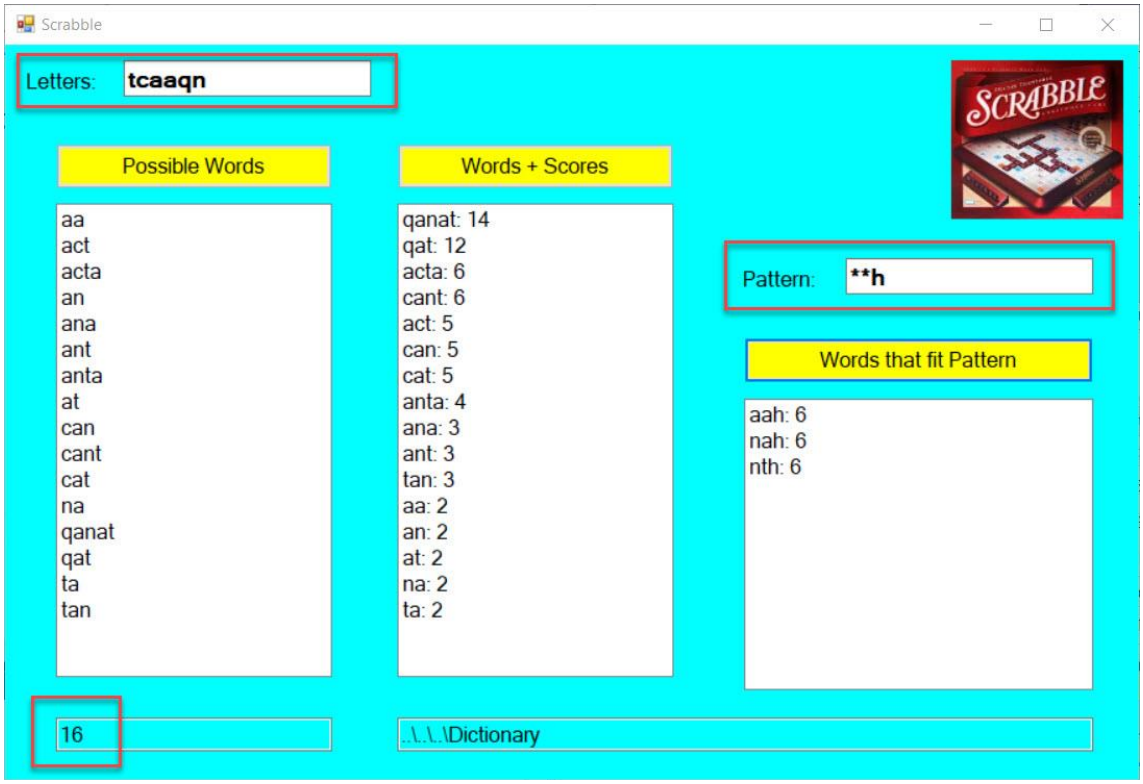
Think of **WordList** as a module-level global variable that you can refer to from your functions as needed. You may assume WordList is in alphabetical order (since "dictionary.txt" is in alphabetical order). If you would like to experiment with the binary search tree code discussed in class (for faster word lookup), note that a randomized version of the dictionary file is provided in "random.txt". This will allow you to build a binary search tree that is roughly balanced since the words are input in random order. This is not required, but provided for those who may want to experiment with trees in F#.

When scoring words, here are the official Scrabble values for each letter:

a: 1	j: 8	s: 1
b: 3	k: 5	t: 1
c: 3	l: 1	u: 1
d: 2	m: 3	v: 4
e: 1	n: 1	w: 4
f: 4	o: 1	x: 8
g: 2	p: 3	y: 4
h: 4	q: 10	z: 10
i: 1	r: 1	

Another Example

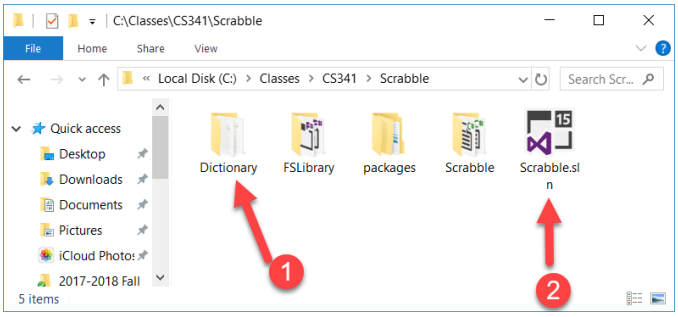
Here's another example. Note that if "letters" contains 2 of the same letter --- e.g. 'a' appears twice --- then words may contain at most 2 a's. For example, "acta" and "qanat". Given the pattern "**h", we can form 3 legal Scrabble words given the letters, the additional 'h', and the pattern: "aah", "nah", and "nth".



Download

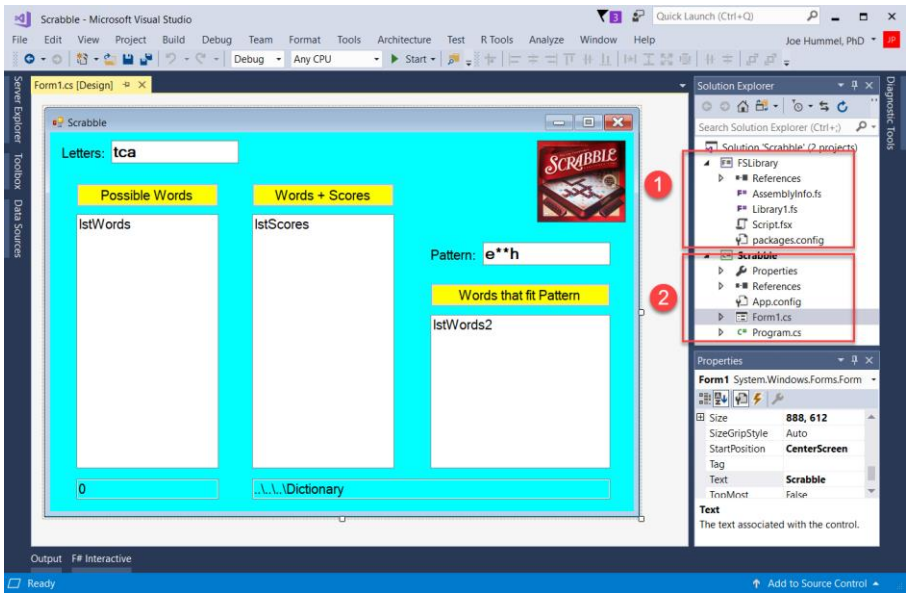
The GUI app is provided as a Visual Studio solution. You can download and open as follows:

1. **Download Scrabble.zip**: on the course web page, open "Projects", and then "project05-files". Download Scrabble.zip to your desktop.
2. **Extract VS files**: double-click the .zip file to open, and extract the **Scrabble** folder to your desktop. Close and delete the .zip file to avoid using it by accident. If you want, move the Scrabble 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 Scrabble folder, and notice 2 things. First, there is a folder named "Dictionary" --- this folder contains the dictionary files. Second, notice the VS Solution (.sln) file --- double-click on this solution file to open the program in Visual Studio.



<< continued on next page >>

4. **Program layout**: as in the previous project, the Visual Studio solution contains 2 projects: a front-end GUI written in C#, and a back-end library written in F#. You'll want to open the "Library1.fs" file and focus your efforts there. You will ultimately submit this "Library1.fs" file for grading.



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 scrabble word generation.
//
// YOUR NAME
// U. of Illinois, Chicago
// CS 341, Fall 2017
// Project #05
//
```

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 "P05 Scrabble".

Policy

Late work *is* accepted for this 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 .