20510102: Computer Programming Language
Fall 2014

Problem Set 9

Due: 12/21/2014

Introduction

This is a group project. The goal of the project is twofold: 1)practice how to read a file and process strings; 2) learn how to use git, the world's most popular distributed version control system, and get a taste of group-working experience.

The git website is http://www.git-scm.com/. The introduction of git can be found at http://www.git-scm.com/doc. You may take the initiative to teach yourself the basic concept of version control and how to use git.

Group policy: each group has no more than five students. Each student needs to solve at least one of the problems listed in the rest of the introduction and then use the git system to merge all the codes. All the codes should compile together and successfully produce an executable file. Each group elects a project-lead, who is responsible for coordinating your work and submitting the final source code. Please indicate each student's specific role in the group submission. All members of the group receive the same grade.

In this problem set, you'll implement a wordgame-Scrabble. Scrabble is described as follows: Letters are dealt to players, who then construct one or more words out of their letters. Each valid word receives a score, based on the length of the word and the letters in that word.

The rules of the game are as follows:

Dealing

- A player is dealt a hand of n letters chosen at random (assume $n = 7$ for now).
- The player arranges the hand into a set of words using each letter at most once.
- Some letters may remain unused (these won't be scored).

Scoring

- The score for the hand is the sum of the score for the words.
- The score for a word is the sum of the points for letters in the word, plus 50 points if all n letters are used on the first go.
- Letters are scored as follows: A is worth 1, B is worth 3, C is worth 3, D is worth 2, E is worth 1, and so on. We have defined the array SCRABBLE_LETTER_VALUES that maps each lowercase letter to its Scrabble letter value.
- For example, 'weed' would be worth 8 points (4+1+1+2=8), as long as the hand actually has 1 'w', 2 'e's, and 1 'd'.

- As another example, if $n = 7$ and you get ‘waybill’ on the first go, it would be worth 65 points (4+1+4+3+1+1+1=15, +50 for the ’bingo’ bonus of using all seven letters).

Getting Started

Download and save these files into the same folder.

- `ps9.c`: the skeleton you’ll fill in for Problem 1-5
- `words.txt`: the list of valid words (all words acceptable in North American Scrabble up to 10 letters long)
- `Scrabble.exe`: a version of the Scrabble game with hand size 12. You may play the game to see how it works.

Unit Testting

This problem set is structured so that you will write a number of modular functions and then glue them together to form the complete word playing game. Instead of waiting until the entire game is *ready*, you should test each function you write, individually, before moving on. This approach is known as unit testing, and it will help you debug your code.

Problem 1: Load Words

The first step is to implement function `load_words` to load the list of valid words from the file `words.txt` to `word_list` array. Fill in the code for `load_words` in `ps9.c`.

Problem 2: Word Scores

The second step is to implement some code that allows us to calculate the score for a single word. The function `get_word_score` should accept a string of lowercase letters as input (a word) and return the integer score for that word, using the game’s scoring rules. Fill in the code for `get_word_score` in `ps9.c`.

Problem 3: Dealing with Hands

Representing hands

A hand is the set of letters held by a player during the game. The player is initially dealt a set of random letters. For example, the player could start out with the following hand:

`a, q, l, m, u, i, l`

A straightforward way to represent a hand in C is as a string. However, we’ll represent the hand in a different way, because it simplifies the code we’ll need in the `update_hand` and `is_valid_word` functions. (In general, there are many ways to represent, in code, various concepts –some are better suited to certain operations than others).

In our program, a hand will be represented by two arrays. The first array represents the unique characters. The second represents the number of times the particular letter is repeated in that hand. For example, the above hand would be represented as:

```
hand[]="aqlmui";
handCount[]={1, 1, 2, 1, 1, 1};
```

the string `hand` stores (lowercase) letters and the array `handCount` stores the number of times the particular letter is repeated in that hand. Notice how the repeated letter ’l’ is represented.

Displaying a hand

We want to display it in a user-friendly way. We have provided the implementation for this in the the `display_hand` function. Make sure you read through this carefully and understand what it does and how it works.

Generating a random hand

The hand a player is dealt is a set of letters chosen at random. We now need a function that generates this random hand. We have to be careful when randomly picking a hand. We need to ensure that there are enough `VOWELS` in the hand to allow the player to spell some words.

We use function `deal_hand` to generate random letters. We require at least one third of the letters in the hand should be `VOWELS`.

Please implement function `deal_hand`.

Testing: Make sure `deal_hand` generates and save random letters appropriately. You may also want to test your implementation of `deal_hand` with some reasonable inputs.

Removing letters from a hand

The player starts with a hand, a set of letters. As the player spells out words, letters from this set are used up. For example, the player could start out with the following hand:

```
a, q, l, m, u, i, l
```

The player could choose to spell the word `quail`. This would leave the following letters in the player's hand:

```
l, m
```

In our implementation, we use function `update_hand` to remove letters from a hand. The code is already provided. Make sure you read through this carefully and understand what it does and how it works.

Problem 4: Valid Words

At this point, we have written code to generate a random hand and display that hand to the user. However, at this point we have not written any code to verify that a word given by a player obeys the rules of the game.

A *valid* word is: in the word list; and it is composed entirely of letters from the current hand.

Implement the `is_valid_word` function.

Testing: You may want to test your implementation by calling it multiple times on the same hand - what should the correct behavior be?

Problem 5: Playing a Hand

We are now ready to begin writing the code that interacts with the player.

Implement the `play_hand` function. This function allows the user to play out a single hand.

Testing: Try out your implementation as if you were playing the game.

Note: Do not assume that there will always be 7 letters in a hand! The global symbolic name `HAND_SIZE` represents this value. Here is some example output of `play_hand` (your output may differ, depending on what messages you print out):

```
Current Hand: a c i h m m z
Enter word, or a . to indicate that you are finished: him
him earned 8 points. Total: 8 points
Current Hand: a c m z
Enter word, or a . to indicate that you are finished: cam
cam earned 7 points. Total: 15 points
Current Hand: z
Enter word, or a . to indicate that you are finished: .
Total score: 15 points.
```

Playing a Game

A game consists of playing multiple hands. We've implemented the code for you.

Uncomment the code that implements the `play_game` function. Read through and make sure you understand what this code does and how it works. There is no coding for this question - the only "work" you have to do here is actually just uncommenting some lines and test it.

Testing: Try out this implementation as if you were playing the game.