Assignment 2: Evil Hangman

Due Friday, December 7, 11:59 PM

Introduction

It's time to pull together the full scope of what we've covered this quarter to build an incredible piece of software – **Evil Hangman**. Your task is to construct a computer program that dupes mortals like you and I into think we're playing a game of hangman while in reality we're facing the full power of a silicon foe.

The Basic Idea

Normally, a game of hangman goes like this:

1. One player chooses a secret word, then writes out a number of dashes equal to the word length.
2. The other players begin guessing letters. If a player guesses a letter that's in the word, the first player reveals all instances of that letter in the word. Otherwise, the guess is incorrect.
3. The game ends when either all the letters in the word have been revealed or when the guessers have run out of guesses.

Fundamental to the game is the fact the first player actually picks a word and accurately represents it when the other players are guessing. That way, when the other players guess letters, the player who chose the hidden word can tell whether that letter is in the word. But what happens if the player doesn't actually choose a word? What if instead that player just comes up with a list of every possible word that could fit in the space, then starts eliminating words whenever the other players start guessing? Provided that the player creating the word can do this quickly enough, the other players would have no idea what was going on.

Let's see exactly how this might work in practice. Suppose that you are choosing a hangman word of length four. Rather than choosing a word, instead you build a list of all of the four-letter words in the English language. For simplicity, let's assume this is your word list:

ALLY BETA COOL DEAL ELSE FLEW GOOD HOPE IBEX JAZZ KING

Now, whenever other players guess letters, you simply eliminate all words containing that letter from your word list and tell the player that the letter isn't in the word. So, for example, if someone guesses E, then you'd end up with a word list that looks like this:

ALLY COOL GOOD JAZZ KING

And if they then guessed I, the word list would become

ALLY COOL GOOD JAZZ

Theoretically, you can keep this up for a long time, constantly telling the guessing players that the letters they're guessing aren't in the word. If your word list is big enough (say, the full contents of the CS106X `Lexicon`), then you might be able to completely stump a guesser.

Practically speaking, however, you will almost certainly reach a point where the letter the uses guesses is in every single remaining word. Suppose, for example, that you have the following word list:

MOOD FOLD TOOL PROD GOLD SORE

If the user guesses O at this point, since all the words here contain the letter O, we can't simply throw out all the words containing O and tell the user that their guess was wrong. Instead, we'll have to be a bit clever about what we do. Let's take a look at all of the remaining words, highlighting the positions of the letter O in each word:

M**OO**D F**O**LD T**OO**L PR**O**D G**O**LD S**O**RE

If you'll notice, all of the above words fall into one of three "word patterns:" -OO-, -O--, and --O-. That is, if you take any of the above words and consider the positions of the o's in that word, you'll get one of the three patterns. We see that there are two words in the family -OO-, three words in the family -O--, and one word in the family --O-. Since -O-- is the most common family, we'll throw out all words that don't match this pattern, leaving the group of words

FOLD GOLD SORE

And will then tell the user that they correctly guessed that there was an O at the second letter of the word. Now, we're left with the largest possible set of words to continue play with, and we can still maintain the illusion that we're playing a fair game.

After play progresses, one of two things will happen. First, the user might be smart enough to pare the word list down to one word and then guess what that word is. In this case, we'll just print out a congratulatory message and say that they guessed correctly. Second, by far the most common case, the player will be completely stumped and will run out of guesses. When this happens, we'll just pick a random word out of the word list and tell them that that was the word they were guessing at all along. The irony is that the user would have no way of knowing that we were dodging guesses the whole time – it looks like we simply picked an unusual word and stuck with it the whole way.

The Assignment

Using only standard C++, you are to write an implementation of the evil hangman algorithm. Unlike the previous two assignments, for this assignment you will be building the entire program from scratch. Feel free to use any code off of the course website, especially for user input validation.

Your program should use the file `dictionary.txt`, downloadable from the CS106L website, as its master word list. Do not read the file multiple times – it's over one megabyte, which makes for some rather lengthy load times. Your program should then:

1. Prompt the user for a word length, reprompting as necessary until the user enters a number so that there's at least one word that's exactly that long.
2. Prompt the user for a number of guesses, which must be positive. We won't worry about the case where the user enters ridiculously large numbers, since guesses above 26 can't make a difference.
3. Prompt the user for whether she wants to have a running total of the number of words remaining in the word list. This partially defeats the purpose of the game, but is useful for testing.
4. Play a game of hangman using the evil hangman algorithm, as described below:
    1. Prompt the user for a single letter guess, reprompting until the user enters a letter that she hasn't guessed yet. Make sure that the input is exactly one letter.

2. If there is at least one word left in the word list that doesn't contain that letter, remove all words from the word list that contain that letter and mark the guess as incorrect.
3. Otherwise, find the most common "word family" in the remaining words, remove all words from the word list that aren't in that family, and report the position of the letters to the user.
4. If the player has run out of guesses, pick a random word from the word list and display it as the word that the computer initially "chose."
5. If the player correctly guesses the word, congratulate her.
5. Ask if the user wants to play again and loop accordingly.

Advice, Tips, and Tricks

Since you're building this project from scratch, you'll need to do a bit of planning to figure out what the best data structures are for the program. Try to leverage off of the STL as much as possible – have algorithms and the `<functional>` library do the heavy lifting. See if you can unify code using functional programming techniques, and of course take some time to pick the right container classes for the job. There is no "right way" to go about writing this program, but by trying to use preexisting code as much as possible you can greatly reduce the amount of code you'll need to write. As always, feel free to email me with questions and comments – I'm more than willing to help out!

Deliverables

Please submit your solution for this assignment into the **Assignment 2** folder of the **CS106L** subdirectory of the CS106X **Submissions** folder. Include your name at the top of any files you submit.