Anagram Finder

Computer Programming 2: mid-semester project

26 kwietnia 2019

Autor: Karol Latos

Anagram Finder

Computer Programming 2: mid-semester project

Problem

Assignment 15
Anagrams. The user enters a string. Write a program that checks a (given) dictionary for occurrences of anagrams of this string. For example, ANGLE and GALEN are anagrams. Use a dictionary for crosswords' solvers.

Approach

The approach to solve this problem I broke down into two main components:

1. algorithmic approach and
2. architectural approach.

Algorithmic approach

Our goal is to get a word from user and then check given dictionary whether it contains any anagram of that word. I excluded the user-provided word itself, because although it technically is its own anagram, it's a trivial one. The procedure is as follows:

- get a dictionary file name;
- open the dictionary file and load every word to a vector of strings;
- get a word from the client;
- compare the word given by the user with every word from the dictionary vector:
  - count how many times each letter a-z appears in the user-specified word and save that information to an int array;
  - count how many times each letter a-z appears in the dictionary word and save that information to another int array;
  - compare these two arrays – if they are identical, those two words are made up of the same amount of the same letters: they may differ only in order of those letters, therefore they are anagrams;
- check if the dictionary word is actually the user-given word – if yes, neglect it; if no, push that word into a vector of strings;
- return the vector of strings to the client and list them line by line.

Architectural approach

Three classes are present in the application:

1. *DataLoader*
   DataLoader class is responsible for acquiring a dictionary filename (constructor), opening the dictionary file, reading off every word from it and pushing them into a vector of strings and then returning the vector.
2. *AnagramSolver*
   This class holds two int arrays as private fields. One of them is immediately set to match the user-provided word, while the second one is used for the dictionary word currently being investigated. *AnagramSolver* converts both words to lowercase, fills the consecutive arrays, checks if these arrays are identical – if yes, it adds the word to the vector of strings and returns this vector as a set of anagrams to the given word (given they are different than the word itself).
3. *Program*
   The highest level is abstraction hierarchy is occupied by the *Program* class. It holds two variables – input word and dictionary name – and one data structure; a string vector of anagrams. It commands all of the other classes and methods to work properly:
   - it sets the dictionary name with the filename specified by the client,
   - it gets the word, for which we want to find anagrams, from the user,
   - it runs the *AnagramSolver* class methods in order to posses a vector of anagrams and
   - it lists all of the anagrams in the console.

Difficulties

Apart from the technical/syntax problems characteristic to C++-based object-oriented projects (like having to include every needed library and every needed header file every time we want to use them in a specific file), there were little to none difficulties in writing this program.