While working on pset5 “Speller” I was Interested in how pointers work so for my final project for CS50x I took “Speller” as a starting point.

I wanted to re-implement the functionality of the parts of “Speller” that I had programmed (primarily the trie for the dictionary), and I wanted to create a program that was based on words and letters, so I came up with the idea of a program that creates anagrams from a data set of letters provided by the user.

In the end it didn’t work out quite as I imagined because the dictionary trie wasn’t needed.

The algorithm that checked ***every*** combination of the letters that a user could input (I’d decided on a maximum of 12) against a dictionary had an unacceptably long run-time, so instead I turned it around to check every word in a dictionary against the pot of letters that the user has entered.

This process uses two linked lists; one to store the letters entered by the user and one to store dictionary words that can be made from the user’s letters.

Using these linked lists was a great learning experience for me because I did not implement linked lists in any of the other CS50x programs.

Some of the challenges I encountered during the creation of my “anagram” program:

- Linked list creation. I wanted to use a linked list to store the letters that the user had entered. I did not create a linked list whilst studying CS50x so the process of creating and using a linked list was new to me.
- Using only the alphabetic characters that the user entered via their keyboard input. I needed the program to ignore any non-alpha characters as these would add unnecessary time to the “check” function.
- For my linked list I created a node (by “typedef”ing a struct) which I named ‘node’. This caused me big problems because the existing dictionary trie already has a node named ‘node’. It took a while for the penny to drop and realise that having two nodes with the same name is not a good idea!
- The biggest problem was that I initially tried to check all permutations of the user-inputted letters against a dictionary. I spent many hours trying to find a way to iterate over all the user-inputted letters, to get every possible combination to check against the dictionary. Eventually I took a step back and tried to see if there was another way. I looked at it from the opposite direction, i.e. taking each word from a dictionary and checking to see if it could be made from the letters available. I achieved this by setting a flag in the node structure of the “letters” linked list which flagged when the node had been used by the current word that was being checked. This was how I created a second linked list which contains all the words that can be made from the available letters.

- Once I got into the main loop of the program (where every permutation of the available words is tested to see if it is an anagram of the set of letters that the user entered), I realised after a while that there was too many sub-loops on top of each other, which made it difficult to see what was going on in the code and difficult to follow. So I broke the program up into smaller chunks; each chunk being a separate function doing a particular job. This made it much easier to “walk through” the main loop because instead of seeing lines and lines of code I was now only seeing a function name and what is passed to it. It’s really important to give functions useful, informative names, even if that makes the function name slightly verbose. Also, it’s easy to re-use functions as and when required.
- I experienced one niggling Segmentation Fault which took me a little while to resolve. After deploying CS50 IDE’s resident debugger (debug50) I was able to see that the problem lay in a function that I was calling from main, rather than in the code in main where I thought the issue lay. A quick look at the function and the penny dropped – I had forgotten to reset the linked list pointer to the root node at the start of the function. Once I’d corrected that… no more SegFault!