Aseem Sane

Project 4 Report

1)

There are no known bugs at this time.

2)

For myHash, I chose to make my own Node struct and create a dynamically allocated array of Node pointers for my buckets. The Node struct was a singly linked list element, with one value for the key and one value for the value. For dynamically resizing the hash table, I created a new private member function named createNewArray, which would create a temporary dynamically allocated array of double size, transfer the previous table’s contents to it, and then delete the previous table and replace it with the bigger one.

For Tokenizer, I included a string which would contain all of the separators. Then, to separate a string into a vector, I would go through the initial string until I hit one of the elements of the separator string, and push back each string when I hit a separator.

For Translator, I chose to go with two doubly-linked lists with head and tail pointers, which had as their values myHashes. The first linked list corresponded to the stack of ciphertext letters to plaintext letters hashes, while the second list represented the stack of plaintext to ciphertext letters maps. When pushing a map, I would first check to see whether the letters to be associated already were associated to other letters in the map, both in the forward and reverse direction (plaintext->ciphertext and vice versa). Only if there was no such conflict, I would push a new map on both stacks. To get a translation, I would use the ciphertext to plaintext letters map to find plaintext letters that matched, and I would return a translation of the given ciphertext string, paying attention to the case of the letters in the ciphertext string.

For WordList, I chose to use two myHashes – one which associated patterns to a vector of strings that fit the pattern, and another which only had a list of words and had a dummy value as their value. When attempting to find a word in the list, I used the map with the actual words as the key to make it faster, whereas for findCandidates I used the pattern of the given ciphertext string as the key to find words values that matched the pattern. After finding these eligible words, I checked whether each one fit the constraints of having correct alphabetical letters and apostrophes in the right places.

For Decrypter, I had WordList, Translator, and Tokenizer member variables, and I used the given algorithm in the spec in my “cracker” helper function, which had 3 parameters: the full length ciphertext message, a vector of ciphertext strings, and a vector of plaintext strings, possibly translated in some form.

3)

All of my methods meet the big-O requirements of the spec.