Mudith Mallajosyula -- ID# 404937201 -- Project 4 report

1. As far as I know, there are no major bugs in my code, but it does take a long time to decrypt some phrases (for example, the John Kennedy quote from the examples).

**2. High-Level Descriptions**

MyHash.h:

The primary data structure within the MyHash class was a template-based hash table. This hash table was comprised of a dynamically allocated array of Buckets, which were structures containing pointers to the head of a linked list. The Nodes of this linked list contained the key-value pair data for the hash table.

The reset function deleted all the dynamically allocated Buckets in the hash array, and then allocated a new array of length 100. The associate function checked if the given key was in the array, updated it if it was, and inserted a new node into a bucket if it wasn’t. If this addition would have caused the table to surpass its max load factor, it reallocated a new array of double the length. The find function searched for a key in the hash table and returned a pointer the the value it mapped to if it was in the table.

Tokenizer.cpp:

The tokenizer class contained only a vector of characters to contain its separators as a data membe.

The tokenize function iterated through a string and, when it encountered a separator character, added the substring from the last position to the character to a vector of strings. When it finished going through the string, it returned the vector of strings.

Translator.cpp:

The translator class’s data members consisted of a stack of strings that served as mapping tables and an integer to keep track of the number of mapping tables.

The constructor initialized a string with 26 ‘?’ characters to serve as a base. The pushMapping function took in a string of ciphertext and one of plaintext, and attempted to create a mapping of ciphertext to plaintext. If as it iterated through the characters it encountered a mapping that conflicted with the current one, it returned false and left the stack unchanged. Otherwise, it pushed the new map onto the stack. The popMapping function popped the top of the mapping table stack if possible. The getTranslation function iterated through the characters of the input string and found their mappings in the mapping table.

WordList.cpp:

The wordlist class contained a MyHash style hash table mapping strings to vectors of strings. The strings that served as keys were based on the patterns of the words and the vectors were the words from the list that fit these patterns.

The loadWordList function read each line of the input text, performed a hashing algorithm based on the letter pattern it contained, and pushed the values into the hash table. The contains function checked based on pattern and a generated hash whether or not the word list contained a word. The findCandidates function took in a cipher word and its partial translation, and identified based on letter pattern words that might prove full translations. It did this by following the hashing algorithm mentioned above and iterating through a linked list of string vectors with candidates.

The getLetterPattern and matches functions were helper functions that aided in the categorization of strings by their respective patterns.

The toLowerCase function iterated through each character of a string and changed it to a lowercase version.

**3. Big-O requirements**

All of the functions I implemented followed the big-O requirements specified in the spec, with the exception of the tokenize function, which may have run faster. However, some of these functions involved calls to others (STL functions, etc) that may have contributed to an increased big-O.