1. There is currently a bug with push maps that a map can be pushed where two ciphers point at the same plaintext but it doesn’t seem to affect decrypter
2. MyHash:

For MyHash class, I created the hashtable using an array of pointers to buckets that contained a node with a keytype, a valuetype, and and a pointer to the next node in that bucket should there be one.

The associate functions worked by first inserting a key and value into the array by hashing the key, going to the correct bucket and either inserting a new node or change the value of a current node if needed. It then checked the load factor of the table and if it exceeded the intended load factor, we create a new table with double the buckets and inserted every node from the old table by rehashing the key and putting them into the right buckets

Find simply hashes the key it is given to find, goes to that bucket and looks for the key in that linked list

Tokenizer:

The tokenizer class has no data structure itself, except if you consider the vector it returns for tokenize to be one of its data structures.

The main part of this class was the tokenize function. I made this function by creating a empty string before I looped through the string parameter. While the char at the index of the string parameter wasn’t a seperator, I added that character to the current word, once I ran into a seperator or the end of the string parameter, I added the word to the vector of tokens and reset the word to an empty string.

WordList

WordList has two different hash tables that are both MyHash data structures. One of them, m\_table, is a MyHash<string, string> where the key is a word and the value is that same word. This allows the user to search a word and see if that word is in the table quickly. The other table, m\_patternTable is a MyHash of string to a vector of strings. The keytype is the pattern of a word and the vector is a list of all words with that pattern. This allows a user to quickly get all words with that pattern.

The loadWordList function looked at a input file and for every word, added that word to m\_table, then found the words pattern and added that word to the vector of words with that pattern in m\_patternTable

Find candidates first found the pattern of the first string parameter and got the vector of all words with that pattern that looped through all of them to make sure they matched the current translation of the first string parameter.

Translator:

This class had two data structures, a map<char,char> current map which is the current mapping table for which we can translate words with the current mapping table. The other data structures was a vector of map<char,char>. This allowed us to store previous possible maps on a sort of “stack” so if the current map ever turned out to be a bad map in some way, we could go back through the vector to get a map we could use to try and find the correct translations.

The pushMapping function would look through the mapping the user wanted to push and make sure it was a valid map before combining it with out current map.

The popMapping function would get rid of the current map and make the new current map the top of the vector stack if there was a map in the vector

getTranslation used our current mapping table and looped through the string parameter, changing it to plaintext based of our known translation table in the current map

Decrypter:

This classes doesn’t have data structures itself, rather it relies on the objects previous classes it has to store a lot of the data and do many of the processing and looking through data.

This is only big algorithim here, the crack function which follows to the letter the algorithim given to us by the project spec for solving the cipher. It is a recursive function that looks for the current token that has the most question marks (we know the least letters in it) and based off of the pattern finds the candidates for this word. It attempts to push mapping tables based off of the candidates we have and is constantly discarding mapping tables that make the ciphertext into plaintext “words” that are not in our wordlist. The main recursion comes when it finds a good mapping table that translates some words but not all and it calls the crack function again until all words and translated into real words. It does this many times until it finds all possible completed phrases out cipher could be.

3. All of my functions follow the bigO requirements of the spec