Vanessa Lee

504766034

Project 4

1. Both my WordList and Decrypter classes are incomplete. They only compile. WorldList is half implemented, but implemented incorrectly. In WordListImpl::loadWorldList(), I was going to try to index each word with a letter pattern in a map but did not finish implementing this, so I did the bare minimum of just storing the words. WordListImpl::contains() does not follow the big-O requirement. WordListImpl::findCandidates() only performs a check for mismatching parameter string lengths, otherwise, it does not do what it is supposed to do. DecrypterImpl::crack() is not implemented.
2. High level description of data structures and algorithms:
   1. ***class MyHash***
      1. For my private members, I declared a struct Node that holds a next pointer, KeyType key and ValueType value. To implement my open hash table, I used a dynamically allocated array of Node pointers. I used a dynamically allocated array instead of a regular array because the size of buckets can change if the load factor exceeds the max load factor.
      2. For reset(), I deleted every single node in the bucket, then I deleted the the memory that m\_hashTable pointed to. I dynamically allocated a new array of Node\* pointers, m\_buckets large and initiated every bucket to a nullptr.
      3. For associate(), I called the hashFunc() to get the bucket number for the passed in key. If the bucket was empty, I just link up the new node. If it was not empty, I check if the key already exists. If it does, then I replace the value. Otherwise, I traverse to the end of the list and link up the new node.
      4. For find(), I call the hashFunc() to find the appropriate bucket number for the parameter key, then I go directly to that bucket. If the bucket is empty, the key does not exist. Otherwise, I traverse through the nodes to find the key.
   2. ***class Tokenizer***
      1. For my private data members, I used a vector of char to hold the separators. I used a vector because it is resizable because the number of separators passed in during construction is unknown. Accessing an item in a vector is also easy.
      2. For the constructor, I added the separators to my vector.
      3. For tokenize(), I created a vector of strings called tokens. I created an empty string temp. I loop through the string parameter and inside, I loop through the separator vector. If the current character in the string is a letter, I append it to temp, otherwise if the character is a separator, I check if temp is empty. If not, I push back temp to tokens and reset temp to empty string and break out of the loop (to go to next character in string). Otherwise if temp is empty, I just break out of the loop.
   3. ***class Translator***
      1. For my private data members, I have a map of char and char called m\_table. This holds the current mapping of letters. I used a map because it maps something to something else, and I needed to map a letter to its decoded letter. I have vector of map of char and char called m\_oldMaps. This stores the previous maps and acts like a stack. I used a vector because it allows for easy access to the end item, which is needed because a stack only access the last item. I also have int m\_pushMapCall and int m\_popMapCall which keeps track of the number of times pushMapping() and popMapping() were called, respectively.
      2. For my constructor, I initialized m\_table to map every letter of the alphabet to a ‘?’.
      3. For pushMapping(string ciphertext, string plaintext), I first checked if the parameters had equal stirng length. If not, the function returns false. Otherwise, I push back the current m\_table to m\_oldMaps to store it. I transform ciphertext and plaintext to all uppercase letters so there is no problem of different cases. I loop through ciphertext. If there are any non-alpha characters in either ciphertext or plaintext, the function returns false, since we do not map non-alpha characters. Otherwise, I call the STL find() function to find the appropriate letter and update the mapping. I increment m\_pushMapCall by one to keep track of how many times the function was called.
      4. For popMapping(), I check if the number m\_popMapCall is equal to m\_pushMapCall, if so return false. Otherwise, I go to the end of the vector m\_oldMaps and set m\_table to point to it. Then I delete the last map in m\_oldMaps (‘pop’ the last item).
      5. For getTranslation(const string& ciphertext), I initialize string result to empty. I also initialize a string temp to ciphertext. I transform the cases of string temp to all uppercase(to avoid the trouble of case sensitivity, for now). I loop through string temp. I call the find() function to find the character in m\_table. Then I check the original ciphertext for case and turn it back to lowercase if necessary, then I append the character in the mapping to result. If the character is not a letter, I append it to result.
   4. ***class WordList***
      1. For my private data members, I used a vector of strings m\_listOfWords to store the list of words.
      2. My constructor does nothing.
      3. For loadWordList(string filename), I push each word in the text file onto my vector.
      4. For contains(string word), loop through m\_listOfWords to find if a word is in the vector, if so return true. Otherwise return false.
      5. For findCandidates(string ciphertext, string currTranslation), it will return an empty vector if the string parameters are not the same. Otherwise, it is not implemented.
   5. ***class Decrypter***
      1. For load(string filename), it initializes a variable of type WordList and calls loadWordList.
      2. crack(const string& ciphertext) is not implemented.