Raymond Lin

304937942

Project 4 Report

1. The myHash class is not completely implemented properly. The getNumItems function is incorrect. It also does not associate keys to values properly.
2. Class MyHash  
    This class uses an array of linked lists. It has a private member variable of a pointer to a Node pointer. Each node has a key and value member variable and a next pointer. The constructor initializes the array to all nullptrs. The destructor deletes all of the items in the hash table by looping through the array, and if it points to something, it traverses the linked list and deletes every Node. The reset function works similarly by combining the destructor and the constructor. The associate function works by looking for the key in the hash table. Once found, if it is a nullptr, it will add a single node as the start of a linked list. If it is not, it will add to the front of the linked list. If it exceeds the maximum load factor, it will allocate a new array that is twice the size. The find function works the same way but simply returns the value that is associated with the given key.  
     
   Class Tokenizer  
    For the tokenize function, a vector of strings is used to store each token. It loops through the input string and finds any separators. Then it adds the words between separators to the vector.   
     
   Class WordList  
    The wordToPattern function is a helper function that uses a myHash data structure. It loops through the letters of the input string. If it is a letter, then it will add it to myHash and associate it with a letter in the alphabet. If it is not, it will associate it with itself. Then it retrieves the association and returns the pattern of the input word. The loadwordlist function transforms every word into a pattern and then associates it in myHash. The contains function finds the input word using its pattern. It traverses the linked list to find it. The findCandidates function uses a vector of strings to store the candidate words that match the input word’s pattern. It will loop through the letters of the candidate and check to see if the corresponding letter in the current translation of the input word is either the same or a question mark. If it is not, there are no candidates.   
     
   Class Translator  
    This class uses a vector of unordered maps, because it must store the current mapping based on the candidate words. The vector is used because it is implemented like a stack. There is also an unordered map to store the current mapping of of translations. The constructor places one mapping into the vector that associates each character in the alphabet to a question mark. The pushmapping function associates each letter in the ciphertext word to the corresponding letter in the plaintext word. It also checks to see if any values are repeated in the current mapping. If it is, it must be wrong. The popmapping function simply pops the back element of the vector of maps, if it has more than 1 element. The getTranslation function loops through each letter of the ciphertext string and gets the translation of that letter and adds it to a string.  
     
   Class Decryptor  
    The crack function has a helper function known as crackHelper. This function first tokenizes the input string and finds the word with the most unknown characters. It then gets the current translation of that word and finds the candidates that it may translate to. If there are no candidates, the mapping table is thrown away. Then it loops through the candidates. For each candidate, it first loops through the current translation and if there is anything that conflicts with the candidate and the current translation, the mapping is wrong. If nothing conflicts, it pushes a new mapping. Then it translates the entire cipherword string based on the mapping it just pushes. Then each newly translated word is checked to see if it is a valid word in the wordlist. If it is not, it pops the mapping table. If it exists, and there are still question marks in the current translation, we recurse the crack function. If there are no question marks, a viable solution is found.
3. Class MyHash  
   MyHash(): satisfied  
   reset(): satisfied  
   associate(): satisfied  
   find(): satisfied  
   getnumitems(): satisfied  
   getloadfactor(): satisfied  
     
   Class Tokenizer  
   tokenizer(): O(1), only assigns string to private member variable  
   tokenize(): satisfied  
     
   Class WordList  
   loadWordList(): satisfield  
   contains(): satisfied  
   findCandidates(): satisfied  
     
   Class Translator  
   pushMapping(): satisfied  
   Popmapping(): satisfied  
     
   Class Decryptor  
   load(): satisfied