Cracked

1. Whether any of your classes have known bugs or other problems that we should know about.

I finished all the implementations of the five classes. No known bugs so far.

1. A high-level description of what data structures and algorithms you chose for each of your classes’ non-trivial methods and data structures.
2. **MyHash**

I used the data structure of pointers to dynamically allocated array for keep tracking of my hash table. I used this structure because we need to dynamically resizing the array if the load factor gets bigger than the maximum load factor. Also the hash table is an open hash table, so I used linked list to link the hash nodes with the same hash value. Each node of the linked list has 3 members, the key, the value and a pointer to the next node. The class is using template parameters for key and value. This allows us to create a class template whose functionality can be adapted to more than one type or class.

**//This is my helper function for inserting or updating a node**

void insert(const KeyType& key, const ValueType& value)

{

If the key is already in the hash table

Update the corresponding value of the key

Return

get the hash value

if the table of the hash value is not a null pointer

make a new node with key, value and stored and pointer points to the bucket position during construction.

insert to the beginning of the bucket

make the bucket point to the new node

increase the number of items in the hash table by 1

else

make a new node in the hash value position with key and data

increase the number of items in the hash table by 1

}

void associate(const KeyType &key, const ValueType &value)

{ if key cannot be found in the hash table

if insertion would cause the load factor to be over the maximum load factor

resize()

insert(key, value)

}

1. **Tokenizer Class**

I used a dynamically allocated character array to store the separators.

I use this structure because the number of the separators are unknown beforehand.

vector<string> tokenize(const std::string& s) const

{

Declare an empty vector container

Declare an empty string temp

Iterating through the string char by char

Iterating through the char array of the separators

If a character of the string is not one of the separator

Append to the temp string

If reach to the end of the string

Push into the vector

Else if temp is not empty

Push temp into the vector

}

1. **WordList Class**

I use the hash table to store the list of words that match to the pattern. The pattern is the key and the vector of string is the value. After loading the word list file, I use a helper function to convert each word to its pattern. And then the words with the same patterns are put in a vector. Next, map the index and the words in the hash table.

vector<string> WordListImpl::findCandidates(string cipherWord, string currTranslation) const

{

Declare an empty vector

Index cipherWord

If the index of the cipherWord is not in the hashtable

Return empty vector

Otherwise

Get the list of words that match the index

Iterating through each word, reduce the collection of words by eliminating those tat are inconsistent with currTranslation.

Return the reduced collection of words.

}

1. **Translator Class**

For this class, I used a hash table to store the mapping between ciphertext and plaintext. Since there can be a collection of mapping tables, the stack is used as a container to hold the different mapping tables. Whenever a new valid mapping appears, put the current table into the stack. The top mapping table will be discard when popmapping has been called. I implemented a stack using array structure to store the mapping tables. The stack is a template class which can work with many data types. In this class I used the stack to store characters.

bool TranslatorImpl::pushMapping(string ciphertext, string plaintext)

{

If ciphertext size not equal to plaintext size

return false;

if there are conflicts between the cipher text and plaintext

return false

push current map to stack

copy current map's content to a new map

add the new mappings to the new map

make the new map as current map

increase the number of successful pushmapping by 1

return true;

}

1. **Decrypter Class**

I used a WordList class type member to load the contents of the file to be used during decrypter process. There is a vector to hold the final solution we get from crack() function. I have two private helper functions and a non-member function. The private member function pickAWord helps to pick an unused word that has the most ‘?’s in it. Another private member function recursive() helps to recursively call from step2 to step7 described in the algorithm. The non-member function compareNoCase compares two strings’ alphabetical ordering.

vector<string> DecrypterImpl::crack(const string& ciphertext)

{

1. Start with an empty mapping by declaring a translator class instance.

Tokenize the ciphertext message into separate words

Recursively call from step2 to step7

2. pick a ciphertext word w from the message that (a) has not yet been chosen and (b) has the most ciphertext letters for which we don’t have any translation.

3. translate w using the current mapping table to get a partial decrypted translation.

4. get a collection C of all words in the word list that could possibly match the ciphertext word compatibly with the partially decrypted version of the word

5. if C is empty, return to the recursive call

6. check each candidate word in C with w, get a temporary mapping table.

If the mapping table is compatible with current mapping, then keep calling the recursive function

If a fully translated word is not in the word list, check the next candidate in C

If cipher words are fully translated and in the word list, then save as a solution and throw away the current mapping

7. if all candidates have been check

Throw away the current mapping go back to the recursive call.

sort solutions in alphabetical order.

return solutions;

}

1. Whether or not each method satisfies our big-O requirements, and if not, what you did instead and what the big-O is for your version. All the methods satisfy the big-O requirements.