Lara Chunko

Kelley Kelley

CSCI 2270 Final Project Write Up

**Data Structure(s) used:**

From our initial write up, we (Kelley and Lara) stated we would be using a trie for keywords and a tree for responses. Once we began, we determined a tree wouldn’t help us with responses and would only complicate things. It was too complex to create a tree for each response and be able to pick the correct one based on different keywords. In challenges, we wrote about our difficulty implementing the computer’s response.

In the end, we chose to implement a trie, deque, and vector.

For the trie, we created the keywords trie. This trie consists of structs called letterNode which store information about the letters in the word. We chose to implement a trie as the backbone of our data, because it is the most efficient given a large data set of words. It makes searching, adding, and deleting words much faster than if we had chosen any other data structure.

For responses, we decided to use deque because we needed to be able to put the user responses in an array such that we could add to the front easily. A deque is a double ended queue so it made it really easy enqueue into the front that way the sentence order was correct.

I used the vector when I needed to add to one end and didn’t know how large the data would be. Otherwise if I knew exactly how large it would be, I used an array.

**References:**

Reference for the trie:

<https://www.geeksforgeeks.org/trie-insert-and-search/>

Reference for adding to a text file:

<https://stackoverflow.com/questions/2393345/how-to-append-text-to-a-text-file-in-c>

References for deque and deque vs vector:

<http://interactivepython.org/runestone/static/pythonds/BasicDS/WhatIsaDeque.html>

<http://www.cplusplus.com/forum/beginner/60348/>

<http://www.cplusplus.com/reference/deque/deque/>

References for words:

<https://www.ef.com/wwen/english-resources/english-vocabulary/top-50-verbs/>

<https://www.paperrater.com/page/lists-of-adjectives>

<https://gist.github.com/deekayen/4148741>

<https://www.mobap.edu/wp-content/uploads/2013/01/list_of_adverbs.pdf>

Reference for sentence structure:

<https://academicguides.waldenu.edu/writingcenter/grammar/sentencestructure>

**Task Division:**

* Together
  + As a team, we decided the data structures of our project, the format of the data, and how we would use the data in our input and output analysis.
* Kelley
  + Kelley worked on taking in user input and formatting the user interface. (Main.cpp and Responses.cpp)
* Lara
  + Lara worked on Keywords.hpp and Keywords.cpp. She implemented all of the functions in Keywords.cpp and building the trie

**Methodology:**

We began this project by creating a few basic c++ files such as Main.cpp and Keywords.cpp. We later realized it would be better to create .hpp files and add more c++ files for readability and distinction between each layer of the chatbot.

In Keywords.hpp, we created all of the methods we believed we would need and commented on how we would be using them.

In the main, we decided what the user should be able to do and how Keywords.cpp ties into the main. Depending on the user input will determine which functions are called from Keywords.cpp.

In Responding.cpp, Kelley created cases for the different types of grammatical English sentences. He figured out how to differentiate between whether the user was saying a statement or asking a question. As he learned more cases and more keywords, he added the cases to Responding.cpp and the keywords to keywords.txt.

**Challenges:**

As we worked through the project, our biggest challenge was determining how to respond to user input. Given this, Kelley came up with a way to classify responses, but we did not have enough time to code the responses. Instead, Kelley commented each response case. This makes it possible to understand where we wanted to go with the chatbot but ultimately did not have enough time to implement these goals. Note that the response cases and how the bot handles user input were built around the bot being obsessed with fruit.

Another challenge we encountered was the problem with removing a word from the keywords.txt. The only way we could find to remove a specific word from a list of words in a text file is to save all of the words in the file and then rewrite a whole new file simply without the word we wish to remove. This would take an extremely long amount of time due to the amount of words we desired to have in the keywords.txt. To circumvent this challenge, Lara implemented removeWord in Keywords.cpp by finding the end letter of the word and setting it’s isEnd boolean to false. This establishes that the word is no longer a “complete word” and cannot be located in findWord, but it also doesn’t cause problems if the word was a subset of a larger word. For example, if I had the word snow and snowman, if I removed the letters in the word snow, the trie would not be able to find snowman given that the letters s, n, o, and w are no longer present in the trie.

**Results:**

Refer to output.txt (it is quite long)