Duncan Di Mauro

UID #805163177

3/15/2022

CS32 Project 4 Report – Unhinged

**1. Parts of the project I was unable to finish**

I was able to finish all parts of the project, although I have a memory leak error which will be discussed in section 3.

**2. Banned STL components**

I did not use any banned STL components in any of my classes. For the MemberDatabase class, it did not say that we mustn’t use STL set. However, it did not explicitly state that we may use STL set. I utilized set to check for duplicates, which drastically sped up the time it took for my database to load. However, I could have easily used STL list, but this would make the database take much longer to load. If I wanted to use list instead of set, I would simply

1. Change all instances of the word “set” in MemberDatabase.h or MemberDatabase.cpp to “list”
2. I would replace the following code in my MemberDatabase.cpp file:

if (!allEmails.insert(email).second)

return false;

with this code:

pointerToEmailSet->push\_back(email); //insert the email

pointerToEmailSet->sort(); //sort the list

int size1 = pointerToEmailSet->size(); //get initial size

pointerToEmailSet->unique(); //remove duplicates

if (pointerToEmailSet->size() != size1) //check if a duplicate removed

duplicate = true; //if false, attribute has this email already

**3. Bugs I haven’t been able to find/fix**

My current implementation of my RadixTree class results in a lot of memory leaks *after* the main routine is run. When I test my classes using a dummy version of RadixTree using map, there are no memory leaks. I am not exactly sure what is/isn’t getting deleted when my RadixTree destructs – I have implemented a cleanup() helper function in RadixTree which appears to be able to successfully destruct all nodes, however the values associated with those nodes are not able to be properly destructed. There are no other bugs for my project that I know of.

**4. Test cases**

For my **RadixTree**, for testing purposes only (this function is now removed), I created a function called PrintAllWords which prints all the words in the tree. For example, if I inserted the words “Car” and “Carey” into the tree, the function would print:

root->Car

root->Car->ey

This not only allowed me to confirm that the words were in the tree, but it also let me see where each of the substrings were stored and if my RadixTree was working properly. For testing the search function, I inserted various elements into the tree with specific values, and then searched for a specific string and printed out its value to make sure it was correct.

For **MemberDatabase**, I created custom members.txt documents in order to test my functions. In order to test my LoadDatabase function, I would load various members.txt files and print out all the attValues stored by my class, as well as all emails, and would cross-check with the text file to make sure everything was there and there were no duplicates. I also tested what would happen if, for example, one user’s information was not completely filled out. In order to test the FindMatchingMembers function, I did the same thing as the LoadDatabase function. I would make custom members.txt files and then print out all matching members to make sure that all members with a certain attribute were found, and that even if a member had that attribute written twice, their email would only be stored once. For my GetMemberByEmail function, I used the same method as previously described.

For **PersonProfile**, some test cases I used are shown below. I examined the cases manually using print statements. Testing for GetNumAttValPairs and GetEmail were simple and involved the same method.

PersonProfile Bob = PersonProfile("Bob", "bob@gmail.com");

Bob.AddAttValPair(AttValPair("hobby", "biking"));

Bob.AddAttValPair(AttValPair("hobby", "biking"));

Bob.AddAttValPair(AttValPair("hobby", "shoe eating"));

Bob.AddAttValPair(AttValPair("job", "memes"));

cout << endl << "Number of Pairs is: " << Bob.GetNumAttValPairs() << endl << endl;

cout << "Bob's hobbies are: " << endl;

//Bob.printValuesOfThisAttribute("hobby");

//This is a custom function I made which

//no longer exists, but it prints all the values

//associated with the specific attribute

cout << endl << "Bob's jobs are: " << endl;

//Bob.printValuesOfThisAttribute("job");

cout << endl;

PersonProfile pp("Carey Nachenberg", "climberkip@gmail.com");

pp.AddAttValPair(AttValPair("hobby", "biking"));

pp.AddAttValPair(AttValPair("hobby", "biking"));

pp.AddAttValPair(AttValPair("hobbyy", "biking"));

pp.AddAttValPair(AttValPair("hoobbyy", "biking"));

pp.AddAttValPair(AttValPair("hobby", "shoe eating"));

pp.AddAttValPair(AttValPair("job", "memes"));

pp.AddAttValPair(AttValPair("job", "memes"));

pp.AddAttValPair(AttValPair("job", "memes"));

pp.AddAttValPair(AttValPair("job", "memeLord"));

pp.AddAttValPair(AttValPair("job", "yay"));

pp.AddAttValPair(AttValPair("job", "sad"));

pp.AddAttValPair(AttValPair("electric", "memes"));

pp.AddAttValPair(AttValPair("electric", "no u"));

for (int k = 0; k != pp.GetNumAttValPairs(); k++)

{

AttValPair av;

pp.GetAttVal(k, av);

std::cout << av.attribute << " -> " << av.value << std::endl;

}

cout << endl << "pp's hobbies are: " << endl;

//pp.printValuesOfThisAttribute("hobby");

cout << endl << "pp's jobs are: " << endl;

//pp.printValuesOfThisAttribute("job");

cout << endl;

For **AttributeTranslator**, I followed the exact same method used to test the MemberDatabase class. I made custom translator text files and printed out the data received by FindCompatibleAttValPairs in a readable manner to make sure that it was working correctly.

For **MatchMaker**, I tested this function last. I used the main routine provided to us, along with custom translator and member text files in order to test that the proper matches were being made. For example, if I knew that johnnyT@gmail.com had 2 matches with bob@ucla.org and 5 matches with helga@theBigO.gov (I know this based on the custom text files I created), then if I input johnnyT@gmail.com for the email and asked to identify matches for users with 2 or more matches, then helga@theBigO.gov should print first, showing that they have 5 matching attributes, followed by bob@ucla.org, showing that they have 2 matching attributes. I also tested for cases where the print order is determined by the alphabetical ranking of the email and other edge cases.