Rohith Venkatesh

**Project 4 Writeup**

**Program Issues**

* Parts I was Unable to Finish
  + I was able to finish all classes and methods in the project
* Parts that use banned STL components
  + I was able to finish all parts of the project without using any banned STL components
* Bugs that I was unable to find/fix
  + When trying to find compatible people with a threshold of 3 or less compatible attributes, my program throws an exception. This is most likely due to a vector being constructed that is too large and adding onto the other vectors in the programs and causing memory to run out

**Testing Explanation**

I initially had my radix tree class implemented as an unordered\_map so I could test out the other classes. After making sure I was able to code the other classes, I implemented my Radix tree. When testing my other classes, I did not start off my using the “member.txt” and “translator.txt” files that were provided. I instead chose to use a smaller version of both files (that had the same syntax) so that I did not have to worry about long debugging times. This allowed me to fix the logic of the other classes before implementing them on a larger scale.

When testing my PersonProfile class, I construvcted a PersonProfile object with a name and email and checked to make sure that I could insert AttValPairs and duplicate AttValPairs would not be inserted. I then created an AttributeTranslator object and made sure it was able to load information from a file and return a vector of compatible AttValPairs based on an AttValPair input. I also created a MemberDatabase object and made sure it was capable of inserting information from a file and find a PersonProfile based on an email and find all matching members based on an AttValPair input. Using the PersonProfile, MemberDatabase, and AttributeTranslator classes, I created a MatchMaker object and tested it to make sure that it can return a vector of the email addresses of conmpatible people.

I tested my RadixTree Class with the following test cases (ValueType is int in these cases)

Insert("Biology",7)->Insert("Bio",3)  
Insert("Biology",7)->Insert("biology,7)  // case sensitive  
Insert("Bio",3)->Insert("Biology",7)  
Insert("Biology",7)->Insert("Zoology",7)  
Insert("Bio",3)->Insert("Biology,7)->Insert("Biolo",5)  
Insert("Biology",7)->Insert("Bio",7)->Insert("Biography",9)  
Insert("Biography,9)->Insert("Biology",7)->Insert("Bio",3)  
Insert("Biology",7)->Insert("Bio",7)->Insert("Biolophy",8)  
Insert("B",1)->Insert("Bio")->Insert("Biology",7)  
Insert("Bio",3)->Insert("B")->Insert("Biology",7)  
Insert("Bio",3)->Insert("",0)  
Insert("",0)->Insert("Bio",3)

I then proceeded to use the search method to search for the keys that I inserted above, as well as to search for keys not in the RadixTree (eg. “Hello”).

After making sure that my radix tree implementation worked, I proceeded to test it with the methods from the other classes. Since I had already tested the other classes with a RadixTree implemented as an unordered map, I already knew that the syntax and logic of the other classes was correct. Like I did before, I again started off by testing the whole program with smaller versions of the given “member.txt” and “translator.txt” files before testing the program on a larger scale.

I included a header file named “utility.h” and “utility.cpp” that included the method, std::string attValPairtoString(const AttValPair& source). This method returns a string that is created by concantenating the attribute string and value string in the attValPair and it is used to create a radix tree that requires the key to be an AttValPair.