**Project 4 report**

I was able to finish everything in the project to the best of my ability. I did not use any banned STL components. I do not have any bugs that I know of.

**PersonProfile Test Case:**

int main() {

PersonProfile person = PersonProfile("Jacky", "ykcaj@g.ucla.edu");

cout << person.GetName() << endl;

cout << person.GetEmail() << endl;

AttValPair a = AttValPair("job", "doctor");

AttValPair b = AttValPair("hobby", "hobby");

AttValPair c = AttValPair("hobby", "gaming");

AttValPair d = AttValPair("hobby", "poker");

AttValPair e = AttValPair("hobby", "improv");

AttValPair f = AttValPair("trait", "reactive");

AttValPair g = AttValPair("trait", "complacent");

AttValPair h = AttValPair("trait", "bewildered");

AttValPair i = AttValPair("hobby", "dancing");

AttValPair j = AttValPair("hobby", "dancing");

person.AddAttValPair(a);

person.AddAttValPair(b);

person.AddAttValPair(c);

person.AddAttValPair(d);

person.AddAttValPair(e);

person.AddAttValPair(f);

person.AddAttValPair(g);

person.AddAttValPair(h);

person.AddAttValPair(i);

person.AddAttValPair(j);

AttValPair print;

for(int i = 0; i < person.GetNumAttValPairs(); i++)

{

person.GetAttVal(i, print);

cout << print.attribute << ", " << print.value << endl;

}

}

This allows me to check if a valid PersonProfile was made with a name and an email address. Then I added attributes value pairs to the person and printed out every Attribute Value pair in the person to see if every worked correctly

**Attribute Translator:**

void listCompatiblePairs(const AttributeTranslator& translator)

{

AttValPair att("job", "salesperson");

std::vector<AttValPair> result =

translator.FindCompatibleAttValPairs(att);

if (!result.empty())

{

std::cout << "Compatible attributes and values:" << std::endl;

for (const auto& p: result)

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

}

}

int main()

{

AttributeTranslator tra = AttributeTranslator();

tra.Load(TRANSLATOR\_FILE);

std::cout << "----------" << std::endl;

listCompatiblePairs(tra);

}

Called load on the AttributeTranslator and got a vector of Compatible AttValPairs using the listCompatiblePairs. Printed out the compatible AttValPairs for ("job", "salesperson"), (“trait”,”critical”), and (“hobby”,”painting”). Checked with the translator.txt file. (using command f)

**MemberDatabase:**

void printEmails(const MemberDatabase& mdb)

{

AttValPair att("job", "salesperson");

std::vector<std::string> result = mdb.FindMatchingMembers(att);

if (!result.empty())

{

std::cout << "Compatible emails:" << std::endl;

for (const auto& p: result)

std::cout << p << std::endl;

}

}

void findMemberByEmail(const MemberDatabase& md, std::string member\_email)

{

const PersonProfile\* ptr = md.GetMemberByEmail(member\_email);

if (ptr != nullptr)

std::cout << "Found info for member: " << ptr->GetName() << std::endl;

else

std::cout << "No member has address " << member\_email << std::endl;

}

int main()

{

MemberDatabase mdb;

mdb.LoadDatabase(MEMBERS\_FILE);

printEmails(mdb);

findMemberByEmail(mdb, "TitusQuin0@me.com");

findMemberByEmail(mdb, "RPa6425@cox.net");

findMemberByEmail(mdb, "ScHob4657@juno.com");

findMemberByEmail(mdb, "AzariahSala558@zoho.com");

}

Called load on the MemberDatabase and got a vector of emails using the FindMatchingMembers. Printed out the matching emails for ("job", "salesperson"), (“trait”,”critical”), and (“hobby”,”painting”). Checked with the member.txt file. Then checked if findMemberByEmail() function worked by printing the name that is attached to the person that is mapped with the email. Tested with [TitusQuin0@me.com](mailto:TitusQuin0@me.com), [RPa6425@cox.net](mailto:RPa6425@cox.net), [ScHob4657@juno.com](mailto:ScHob4657@juno.com), and AzariahSala558@zoho.com .

**MatchMaker:**

int main()

{

MemberDatabase mdb;

if (!mdb.LoadDatabase(MEMBERS\_FILE))

{

std::cout << "Error loading " << MEMBERS\_FILE << std::endl;

return 1;

}

AttributeTranslator at;

if (!at.Load(TRANSLATOR\_FILE))

{

std::cout << "Error loading " << TRANSLATOR\_FILE << std::endl;

return 1;

}

while (findMatches(mdb, at))

;

std::cout << "Happy dating!" << std::endl;

}

Tested with email: [EmiliTy3@charter.net](mailto:EmiliTy3@charter.net) Threshold: 5

Tested with email: [CesPicke66968@outlook.com](mailto:CesPicke66968@outlook.com) Threshold: 4

Tested with email: [ScHob4657@juno.com](mailto:ScHob4657@juno.com) Threshold: 8

compared with the sample executable built :/usr/local/cs/bin/samplep4

**Radix Tree:**

int main()

{

RadixTree<int> rT;

rT.insert("shop", 5);

rT.insert("good", 2);

rT.insert("mad", 9);

rT.insert("cool", 10);

rT.insert("shell", 111);

rT.insert("sheep", 123);

rT.insert("she", 123333);

rT.insert("car", 2);

rT.insert("car", 1);

rT.insert("camp", 3);

rT.insert("cramp", 99);

rT.insert("carry", 960);

rT.insert("card", 3);

rT.insert("camp", 3);

rT.insert("car",90);

rT.insert("car",67);

rT.insert("cat",999);

rT.insert("car",9988);

rT.insert("she", 1);

rT.insert("sheep", 2);

std::cout << \*(rT.search("car")) << std::endl;

std::cout << \*(rT.search("card")) << std::endl;

std::cout << \*(rT.search("cat")) << std::endl;

std::cout << \*(rT.search("camp")) << std::endl;

std::cout << \*(rT.search("cramp")) << std::endl;

std::cout << \*(rT.search("carry")) << std::endl;

std::cout << \*(rT.search("shop")) << std::endl;

}