Implementation:

I chose to create a typical doubly linked list for my project, typical as in not circular. I had two dummy nodes, one for the head of the list and one for the tail, to reduce the need for more special case codes. My node consisted of two pointers to reference the previous and the next node in the list, as well as three data points. The data points were two strings to represent the first and last name, and also a special type OnlineType to store the value each node was connected to. In the given code, the typedef is defined as a string but this can be changed to integer if needed as shown in some test cases.

Notable Obstacles:

As someone who has had a hard time understanding pointers, this project was definitely a struggle. My first notable obstacle was deciding what to put inside my struct. The given code for the header defined typedef as a string, but the following test cases had an integer in the space of the OnlineType so I was really confused as to which type I should define it as. Luckily, someone on Piazza asked the question before I needed to, so I knew to change it for the appropriate test cases. While coding the multiple functions, I came across lots of errors of incorrect comparisons like a string to a string. It made sense in my head and theoretically should have worked, but I ultimately had to look online for different methods to get my program working. For most of my obstacles figuring out how to logically code the functions, watching lectures and YouTube videos helped the most. My biggest obstacle I was not able to overcome was time due to other commitments/classes. I only wish I had more time to focus on the project to present my full potential.

Pseudocode:

blockPreviousMatch

if (list is empty)

return false;

set Node ptr as the head

for (entirety of list)

if (first == firstName && last == lastName)

//Case 1: Item to be deleted is first in list

if (i == 0){

set head as next node

delete node

subtract number of items in list

return true;

//Case 2: Item to be deleted is last in list

set head and tail as nullptr

delete ptr

subtract number of items in list

return true;

//Case 3: Item to be deleted is not first or last in list

Delete surrounding references

Delete ptr

Subract number of items from list

Return true;

Else

// Name does not match any in list

Return false

Test Cases:

// For Default Constructor

OnlineDating glee;

//--------For an empty list--------

// Test Size

assert(glee.howManyMatches()==0);

// Test Empty

assert(glee.noMatches());

// Test no match to block

assert(!glee.blockPreviousMatch("Finn", "Hudson"));

// For Copy Constructor

OnlineDating glee;

glee.makeMatch("Rachel", "Berry", 12);

OnlineDating fox = glee;

assert(glee.howManyMatches()==1);

// Test if items copy over

assert(fox.howManyMatches()==1);

// No Matches

OnlineDating glee;

// Check list has no items

assert(glee.noMatches());

glee.makeMatch("Blaine", "Anderson", 100);

// Check list has match

assert(!glee.noMatches());

// how Many Matches

OnlineDating glee;

// No Matches

assert(glee.howManyMatches()==0);

glee.makeMatch("Kurt", "Hummel", 9);

glee.makeMatch("Sam", "Evans", 67);

// Check how many items

assert(glee.howManyMatches()==2);

glee.blockPreviousMatch("Kurt", "Hummel");

// Check item count if removal

assert(glee.howManyMatches()==1);

// make Matches

OnlineDating glee;

glee.makeMatch("Santana", "Lopez", 11);

// Cannot add duplicates

assert(!glee.makeMatch("Santana", "Lopez", 11)==**true**);

// Add same Last Name

glee.makeMatch("Matt", "Lopez", 12);

assert(glee.howManyMatches()==2);

// ! Will fail alphabetical test cases, code can't sort !

//Transform Match

OnlineDating glee;

glee.makeMatch("Tina", "Chen", 23);

// Check if working

assert(glee.transformMatch("Tina", "Chen", 5)==**true**);

// Check if no matching name

assert(glee.transformMatch("Artie", "Abrams", 21)==**false**);

//make or Transform

OnlineDating glee;

// Add if empty list

assert(glee.makeOrTransform("Jake", "Puckerman", 3)==**true**);

assert(glee.howManyMatches()==1);

// Transform number if same name

assert(glee.makeOrTransform("Jake", "Puckerman", 5)==**true**);

// block Previous Match

OnlineDating glee;

// blocks matching name

glee.makeMatch("Quinn", "Fabray", 42);

glee.makeMatch("Mercedes", "Jones", 85);

glee.blockPreviousMatch("Quinn", "Fabray");

assert(glee.howManyMatches()==1);

// No change if name does not match

glee.blockPreviousMatch("Puckerman", "Puck");

assert(glee.howManyMatches()==1);

// someone Among Matches

OnlineDating glee;

glee.makeMatch("Will", "Schuester", 543);

glee.makeMatch("Sebastian", "Smythe", 314);

glee.makeMatch("Kitty", "Wilde", 312);

// True if name matches

assert(glee.someoneAmongMatches("Kitty", "Wilde")==**true**);

// False if name does not match

assert(glee.someoneAmongMatches("Mike", "Chang")==**false**);

// look at Matches

OnlineDating glee;

**int** h= 99;

glee.makeMatch("Becky", "Jackson", 90);

glee.makeMatch("Sugar", "Motta", 45);

// False if no name match

assert(glee.lookAtMatches("Joe", "Hart", h)==**false**);

// True if name match

assert(glee.lookAtMatches("Sugar", "Motta", h)==**true**);

// trade Matches

OnlineDating glee;

OnlineDating awkward;

awkward.makeMatch("Jenna", "Hamilton", 5);

glee.tradeMatches(awkward);

// Switching lists

assert(glee.howManyMatches()==1);

// Was not able to finish confirmMatch, mergeMatches, authenticateMatches