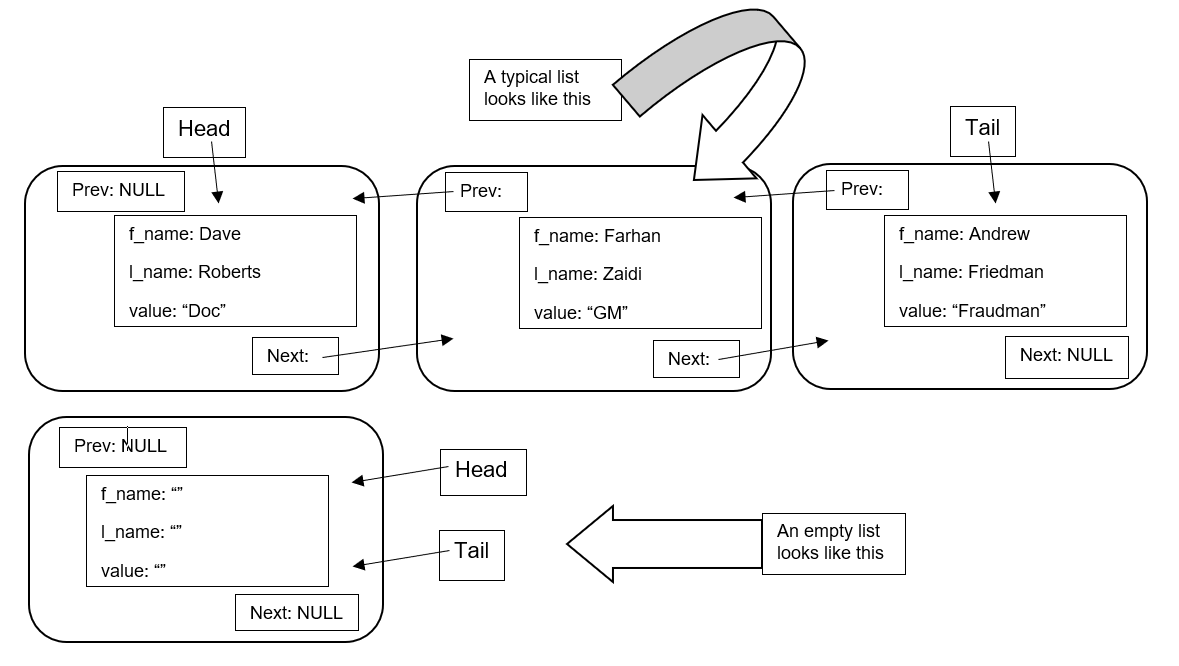
Kevin Hahn

CS 32: Project 2

Report

Implementation:

I decided to use head and tail pointers to the first and last nodes respectively. Each node has a previous and next pointer and the only NULL pointers are the head’s previous pointer and the tail’s next pointer. An empty node would have head and tail both equal to nullptr, The private data members f\_name, l\_name, and value would all be the empty string because that’s a string’s default value when it is declared.



Empty list looks like this

Tail: NULL

Head: NULL

Prev: NULL

Head:

List with one item looks like this

f\_name: Dave

l\_name: Roberts

value: “45”

Tail:

Next: NULL

Obstacles:

Figuring out how to implement the copy constructor, assignment operator, and swapping the contents of 2 doubly linked lists were the most difficult problems to address. These 3 functions were essential to implementing the remaining functions and I had a lot of trouble making sure the copy constructor and assignment operator didn’t create a memory link or put the object in an invalid state. The 2 non-member public functions were another area of difficulty due to the limitations imposed by using only the public interface of CoachingStaff because figuring out how to indirectly obtain the private data members of each node in CoachingStaff was tricky.

Pseudocode:

mergeStaffs function:

1. check if csMerged is an empty list. If not, then empty the contents.
2. If csOne is empty AND csTwo is also empty:
   1. Just return true and leave csMerged as is because csMerged is an empty list right now.
3. If csOne is not empty but csTwo is:
   1. Set csMerged = to csOne and return true.
4. If csOne is empty but csTwo is NOT:
   1. Set csMerged = to csTwo and return true.
5. Declare a boolean variable called **sameNameButDifferentValue** and set it to ***false***.
6. Create copies of csOne and csTwo and declare 2 sets of 3 variables (a string to hold the first name, a string to hold the last name and an IType variable to hold the value)
   1. 1 set for each copy as you iterate through the list and temporarily store the private data members of the current node in these variables
7. Iterate through one of the lists and use whichCoach function to copy the values of the private data members (*firstName, lastName, value*) into one of the sets.
   1. Then use coachOnStaff function on the other list to check if it has a coach with that same first name and last name
8. If there’s a coach with the same full name and same value in both lists, then delete the coach from only one of the lists.
9. If there’s a coach with same full name but different values in both lists, then delete the coach from both lists.
   1. Set the bool variable **sameNameButDifferentValue** = to ***true***.
10. Set csMerged equal to one of the lists, and traverse the copy of the other list, access each node’s private data members with whichCoach function and then call hireCoach function on csMerged with those private data members passed in as the arguments.
11. If **sameNameButDifferentValue == true,** then return false. Otherwise, return true.

fireCoach function:

1. Call the noCoaches function to see if the list is empty
   1. If the list is empty: then return false
2. Call the coachOnStaff function to see if the name is even in the list
   1. If the name is not on the list: then return false
3. Now we know the name is on the list, so we know for sure that we are ultimately going to delete a pointer and return true
4. Traverse through the list until you reach the Node **p** whose first name and last name match the arguments passed in and then break out of the loop.
5. Check through 4 different cases:
   1. Case 1: The node is both the head and the tail (i.e. it’s the only item in the list)
   2. Case 2: The node is the head
   3. Case 3: The node is the tail
   4. Case 4: The node is somewhere in the middle of the list
6. Adjusts the previous and next nodes accordingly.
   1. Case 1: set head and tail equal to NULL
   2. Case 2: set head = head🡪next and head🡪prev = NULL
   3. Case 3: set tail = tail🡪prev and tail🡪next = NULL
   4. Case 4: set p🡪next 🡪prev = p🡪prev and p🡪prev🡪next = p🡪next
7. Delete the Node pointer **p** and return true.

searchStaff function:

1. Check if csOne and csResult are actually the same. If so, then return right away.
2. Check if the list is empty and empty it if needed
3. If fsearch and lsearch are both equal to “\*” then use assignment operator to set csResult = csOne and return
4. Else if fsearch is “\*” but lsearch is not, then iterate through csOne and access the private data members of its nodes using whichCoach and store those values into temporary variables. Call hireCoach on csResult with those values passed in (only if the value of the last name is == to lsearch).
5. Else if lsearch is “\*” but fsearch is not, then iterate through csOne and access the private data members of its nodes using whichCoach and store those values into temporary variables. Call hireCoach on csResult with those values passed in (only if the value of the first name is == to fsearch).
6. If both fsearch and lsearch are NOT equal to “\*”, then iterate through csOne and access the private data members whichCoach and store those values into temporary variables. Call hireCoach on csResult with those values passed in (only if the value of the first name is == to fsearch and last name == lsearch).

hireCoach:

1. Check if head and tail are both = to nullptr (i.e. list is empty)
   1. Insert a new node with the private data members equal to the values of the parameters passed in and return true
2. Use coachOnStaff to see if the last name is already in the list, and if so then return false
3. If there is not an existing node in the list with the same last name, then traverse the list until you reach the first node whose last name is > than the lastName of our new node.
4. Else if there is an already existing node with the same last name, then traverse the list until you encounter the first node whose last name is == to lastName and whose first name is > than firstName.
5. Insert the node accordingly and return true

Test Cases:

// default constructor

CoachingStaff dodgers;

// For an empty list:

assert(dodgers.numberOfCoaches() == 0); // an empty list

assert(dodgers.noCoaches() == true); // empty list

assert(!dodgers.fireCoach("Bob", "Geren")); // nothing to erase

CoachingStaff padres;

// swap two empty lists:

padres.changeStaff(dodgers);

assert(padres.noCoaches() && dodgers.noCoaches() && padres.numberOfCoaches() == 0 && dodgers.numberOfCoaches() == 0);

// testing mergeStaff and searchStaff on two empty lists:

CoachingStaff NL;

assert(mergeStaffs(padres, dodgers, NL) == true && NL.numberOfCoaches() == 0 && NL.noCoaches() == true);

searchStaff("\*", "\*", dodgers, NL);

searchStaff("Let's Go", "\*", dodgers, NL);

searchStaff("\*", "ITFDB", dodgers, NL);

assert(NL.noCoaches() == true && NL.numberOfCoaches() == 0);

// Testing a list with one node:

assert(dodgers.hireCoach("Dave", "Roberts", "45") == true);

assert(dodgers.numberOfCoaches() == 1);

assert(dodgers.noCoaches() == false);

IType v;

string firstName; string lastName;

assert(dodgers.findCoach("Dave", "Roberts", v) && v == "45");

assert(dodgers.coachOnStaff("Dave", "Roberts") == true);

assert(dodgers.whichCoach(0, firstName, lastName, v) && firstName == "Dave" && lastName == "Roberts" && v == "45");

// swap a list with one node and a list that's empty

CoachingStaff giants;

dodgers.changeStaff(giants);

// dodgers is now empty

assert(dodgers.noCoaches() == true && dodgers.numberOfCoaches() == 0);

assert(giants.noCoaches() == false && giants.numberOfCoaches() == 1);

string fName; string lName; IType val;

assert(giants.coachOnStaff("Dave", "Roberts") && giants.findCoach("Dave", "Roberts", val) && val == "45");

assert(giants.whichCoach(0, fName, lName, val) && fName == "Dave" && lName == "Roberts" && val == "45");

giants.hireOrRename("Bruce", "Bochy", "65"); // hires this coach

assert(giants.hireCoach("Bruce", "Bochy", "75") == false);

giants.hireOrRename("Bruce", "Bochy", "50"); // changes the value of Bruce Bochy to 50

assert(giants.whichCoach(0, fName, lName, val) && fName == "Bruce" && lName == "Bochy" && val == "50");

assert(giants.findCoach("Bruce", "Bochy", val) && val == "50");

assert(giants.coachOnStaff("Bruce", "Bochy") && giants.coachOnStaff("Dave", "Roberts"));

IType age = "66";

giants.renameCoach("Bruce", "Bochy", age);

assert(giants.whichCoach(0, fName, lName, val) && val == "66");

assert(giants.noCoaches() == false && giants.numberOfCoaches() == 2);

assert(dodgers.fireCoach("Dave", "Roberts") == false && giants.fireCoach("Dave", "Roberts") == true);

assert(giants.fireCoach("Bruce", "Bochy"));

giants.dump();

// dodgers and giants have been emptied

// For a typical list:

CoachingStaff lad;

lad.hireCoach("Manny", "Machado", "26");

lad.hireCoach("Cody", "Bellinger", "23");

lad.hireCoach("Corey", "Seager", "24");

lad.hireCoach("Kyle", "Seager", "30");

assert(lad.numberOfCoaches() == 4 && lad.noCoaches() == false);

assert(lad.whichCoach(3, fName, lName, val) && fName == "Kyle" && lName == "Seager" && val == "30");

lad.hireOrRename("Kyle", "Seager", "Corey's brother");

assert(lad.findCoach("Kyle", "Seager", val) && val == "Corey's brother");

assert(lad.hireCoach("Corey", "Seager", "25") == false);

CoachingStaff sfg;

sfg.changeStaff(lad);

assert(lad.noCoaches() == true && lad.numberOfCoaches() == 0);

assert(sfg.noCoaches() == false && sfg.numberOfCoaches() == 4);

//sfg.dump();

lad.changeStaff(sfg);

assert(sfg.noCoaches() == true && sfg.numberOfCoaches() == 0);

assert(lad.noCoaches() == false && lad.numberOfCoaches() == 4);

//lad.dump();

lad.hireOrRename("Cody", "Bellinger", "Bellibomb");

assert(lad.findCoach("Cody", "Bellinger", val) && val == "Bellibomb");

lad.renameCoach("Manny", "Machado", "Mannywood");

assert(lad.findCoach("Manny", "Machado", val) && val == "Mannywood");

lad.renameCoach("Corey", "Seager", "The Kid");

// Testing mergeStaff and searchStaff with one non-empty list and one empty list:

CoachingStaff combo;

CoachingStaff cali;

mergeStaffs(lad, sfg, cali);

cali.dump();

assert(cali.numberOfCoaches() == 4 && cali.noCoaches() == false);

searchStaff("\*", "\*", lad, combo);

combo.dump();

// after running searchStaff, combo should contain Kyle Seager and Corey Seager

searchStaff("\*", "Seager", lad, combo);

combo.dump();

// after searchStaff below, combo should contain Corey Dickerson and Corey Seager

lad.hireCoach("Corey", "Dickerson", "Pittsburgh");

searchStaff("Corey", "\*", lad, combo);

combo.dump();

// after running searchStaff below, combo should still be an empty list

CoachingStaff vets; // empty list

searchStaff("\*", "\*", vets, combo);

assert(combo.noCoaches() == true && combo.numberOfCoaches() == 0);

searchStaff("\*", "Kershaw", vets, combo);

assert(combo.noCoaches() == true && combo.numberOfCoaches() == 0);

searchStaff("Clayton", "", vets, combo);

assert(combo.noCoaches() == true && combo.numberOfCoaches() == 0);

// now testing mergeStaffs with two typical, non-empty lists

sfg.hireCoach("Cody", "Bellinger", "Bellibomb");

sfg.hireCoach("Buster", "Posey", "The Third");

sfg.hireCoach("Madison", "Bumgarner", "MadBum");

sfg.hireCoach("Brandon", "Crawford", "UCLA Alum");

assert(mergeStaffs(lad, sfg, cali) == true);

cali.dump();

// testing mergeStaffs where it should return false:

sfg.renameCoach("Cody", "Bellinger", "ROTY"); // there is a Cody Bellinger in the lad list but with a different value

assert(mergeStaffs(lad, sfg, cali) == false);

cali.dump();

// testing mergeStaffs on lists containing just one item each, with same full name and same value

// and testing searchStaff on a list with just one item

CoachingStaff sdp;

CoachingStaff ari;

CoachingStaff blend;

sdp.hireCoach("Andy", "Green", "40");

ari.hireCoach("Andy", "Green", "40");

searchStaff("\*", "\*", sdp, blend);

assert(blend.noCoaches() == false && blend.numberOfCoaches() == 1);

blend.dump();

searchStaff("\*", "Green", sdp, blend);

assert(blend.noCoaches() == false && blend.numberOfCoaches() == 1);

searchStaff("Andy", "\*", sdp, blend);

assert(blend.noCoaches() == false && blend.numberOfCoaches() == 1);

searchStaff("Bud", "Black", sdp, blend);

assert(blend.noCoaches() == true && blend.numberOfCoaches() == 0);

searchStaff("Lavar", "\*", sdp, blend);

assert(blend.noCoaches() == true && blend.numberOfCoaches() == 0);

searchStaff("\*", "Ball", sdp, blend);

assert(blend.noCoaches() == true && blend.numberOfCoaches() == 0);

mergeStaffs(sdp, ari, blend);

assert(blend.noCoaches() == false && blend.numberOfCoaches() == 1);

// now testing merge staffs with lists containing just one item each, with same full name but different value

// resulting list should be empty

sdp.renameCoach("Andy", "Green", "30");

mergeStaffs(sdp, ari, blend);

assert(blend.noCoaches() == true && blend.numberOfCoaches() == 0);