Project 2(BirthdayParty) Report

Student: Xinyi Li

Professor: E. Ambrosio

Date: July 16, 2020

Implementation:

The structure of the doubly linked list is combined with struct nodes, head pointer and tail pointer. A struct node consist the first name, last name, value, previous node pointer and next node pointer. An empty linked list consists a head pointer and a tail pointer that are both pointed to “nullptr”, and it does not have a dummy node. The reason I chose not to have a dummy node is that it is easier to implement with having the simple linked list structure from homework 1. Each node can be traversed either starting with the head pointer or the tail pointer. Each node can access to its previous node or next node’s pointer.

Obstacles that I overcame:

There are a few difficulties in this project. First the doubly linked list is more complex than the simple linked list to implement. Second, adding a person to the list in the right position that their names are sorted in orders takes some algorithms. Last, needing to consider aliasing in non-member functions makes the problem harder.

I overcame these obstacles by drawing out the structures of linked list and algorithms, breaking down a bigger problem into smaller pieces and checking piazza and lecture notes.

Before I write code, I will first draw the algorithm or just some simple examples on a piece of paper so that I can have a better understanding of the task. It makes things clearer when they are drawn out and easier to come up with pseudocode. By breaking down a bigger problem into smaller pieces, those big functions can be solved. In the addInvitee function, I first wrote down adding one node to the front first. With that done, it got easier to implement different cases when nodes need to be inserted. When I have doubts, I go to piazza and check if anyone have same problem I do. Most of the time, I can get some hint on my questions.

Pseudocode:

bool addInvitee(const std::string& firstName, const std::string& lastName, const BirthdayType& value){

If the list is empty: then insert a node at the front of the list.

Else: while pointer p is not nullptr,

if lastname is p->last:

if firstname == p->first: return false;

else if firstname < p->first: insert the node before p and return true;

else: either go to next or insert after the last node and return true;

else if lastname < p->last:

add this node before p and return true;

else:

go to the next node or insert at the last node;

}

bool BirthdayParty::dropFromGuestList(const string& firstName, const string& lastName)

{

Using p to traverse through the list;

If (firstName == p->first && lastName == p->last):

// deleting concept:

// first link the p’s next’s prev to p’s prev

// then link p’s prev’s next to p’s next;

// then change head and tail accordingly

// last delete p;

if (p->prev == nullptr && p->next == nullptr): delete this node and return true;

else if (p->prev == nullptr): delete this node and return true;

else if (p->next == nullptr): delete this node and return true;

else: delete this node and return true;

else: p goto next node;

return false;

}

bool combineGuestLists(const BirthdayParty & bpOne, const BirthdayParty & bpTwo, BirthdayParty & bpJoined)

{

if bpjoined is not empty and bpjoined is not aliasing with bpone or bptwo: delete all nodes in bpjoined;

**if** bpjoined is bpone or bpjoned is just itself: copy bpone to bpjoined

for every node in bptwo:

if have person but value different: delete the person from bpjoined , the return will be false;

else if don’t have this person: add this person to bpjoined with the value it has in bptwo;

**else if** bpjoined is aliasing with bptwo:

do everything did in above but copy bpone’s nodes to bpjoined;

}

void verifyGuestList (const string& fsearch, const string& lsearch, const BirthdayParty& bpOne, BirthdayParty& bpResult)

{

If bpresult not empty and bpresult not equal to bpone: delete all nodes in bpresult;

If bpresult aliasing with bpone:

BirthdayParty new\_party;

Case: two “\*” : new\_party = bpone;

Case: first name use “\*” : traverse through bpone and only check on last name, put them in new\_party;

Case: last name use “\*”: traverse through bpone and only check on first name, put them in new\_party;

Case: else: traverse bpone; if fsearch == first && lsearch == last: add this person to bpresult;

Bpresult = new\_party;

Else // no aliasing case:

Do the above but no need new\_party to hold list;

}

Test cases:

#include "BirthdayParty.h"

#include <string>

#include <iostream>

#include <cassert>

using namespace std;

int main() {

BirthdayParty theLastDance;

theLastDance.addInvitee ("Michael", "Jordan", 23);

theLastDance.addInvitee ("Scottie", "Pippen", 33);

theLastDance.addInvitee ("Dennis", "Rodman", 91);

theLastDance.addInvitee ("Luc", "Longley", 13);

theLastDance.addInvitee ("Ron", "Harper", 9);

BirthdayParty b1;

b1.addInvitee("Michael", "Jordan", 23);

b1.addInvitee("Scottie", "Pippen", 33);

b1.addInvitee("xinyi", "Li", 25);

b1.addInvitee("eli", "Li", 33);

BirthdayParty comb;

bool check\_comb = combineGuestLists(theLastDance, b1,b1);

cout << check\_comb << "\*\*\*\*\*" << endl;

cout << "-----------------------" << endl;

BirthdayParty result;

result.addInvitee("eli", "Li", 29);

verifyGuestList("\*", "\*", b1, b1);

}

cout << theLastDance.whosOnTheGuestList() << endl;

int count = theLastDance.whosOnTheGuestList();

for (int i = 0 ; i < count; i++){

cout << "-----"<< i << endl;

string first;

string last;

BirthdayType val;

theLastDance.selectInvitee (0, first, last, val);

cout << first << " " << last << " " << val << endl;

cout << theLastDance.dropFromGuestList(first, last) << endl;

}

cout << "-----------------------" << endl;

for (int i = 0 ; i < theLastDance.whosOnTheGuestList(); i++){

string first;

string last;

BirthdayType val;

theLastDance.selectInvitee (i, first, last, val);

cout << first << " " << last << " " << val << endl;

//theLastDance.dropFromGuestList(first, last);

}

BirthdayParty b1;

bool m1 = b1.modifyInvitee("aaon", "harper", 222);

cout << m1 << endl;

cout << ".";

cout << "." << endl;

b1.addOrModify("xinyi","li",25);

BirthdayType t1 = 99;

cout << t1 << endl;

bool m2 = b1.checkGuestList("xinyi", "li", t1);

cout << t1 << endl;

cout << m2 << endl;

string s1;

string s2;

BirthdayType t2;

b1.selectInvitee(1, s1, s2, t2);

cout << s1 << " " << s2 << " " << t2 << endl;

b1.changeGuestList(theLastDance);

BirthdayParty dodgers;

dodgers.addInvitee("Clayton", "Kershaw", 31.0);

dodgers.addInvitee("Cody", "Bellinger", 11.5);

assert(!dodgers.personOnGuestList ("",""));

dodgers.addInvitee("Mookie", "Betts", 27.0);

dodgers.addInvitee("", "", 0.57);

dodgers.addInvitee("Justin", "Turner", 20.0);

assert(dodgers.personOnGuestList ("", ""));

dodgers.dropFromGuestList("Mookie", "Betts");

assert(dodgers.whosOnTheGuestList() == 4

&& dodgers.personOnGuestList("Clayton", "Kershaw")

&& dodgers.personOnGuestList ("Cody", "Bellinger")

&& dodgers.personOnGuestList ("Justin", "Turner")

&& dodgers.personOnGuestList ("", ""));

return 0;

}