* Design
  + I had a double linked list design with no tail pointer
  + Each node has a firstName, lastName, value, and pointers to previous and next nodes
  + This was the most simple and effective design for me
  + The addCricketer function was difficult for me and a major challenge
    - I used several different approaches that lead me astray, such as trying to sort the list after adding the nodes and using too many unnecessary case tests to figure out where to add the nodes
  + The CricketType caused be a lot of confusion when testing my project
* Pseudocode

CricketTeam::CricketTeam(const CricketTeam& copy){

Set head to nullptr if copy is empty

else {

Create new node and assign values

newNode is head, original set to copy of node

while (ogNode->next != nullptr) {

Assign values to new Node

Update which node is pointed to, to iterate until the end

}

Set next to nullptr

}

}

CricketTeam::~CricketTeam(){ //destructor

Create new node and set it to head

while (p != nullptr){

Create temp node after and delete previous node

}

}

const CricketTeam&CricketTeam:: operator=(const CricketTeam& copy){ //assignment operator

Check if lists are the same and return pointer to this object

Use destructor

Use deep copy

return pointer to this object

}

int CricketTeam::cricketerCount() const{

Iterate through loop while increasing count until iterator reachers nullptr

Return count

}

bool CricketTeam::addCricketer(const std::string& firstName, const std::string& lastName, const CricketType& value){

// case where person is already on list

Iterate through to find, and return false

// empty list case

Create new node, initialize variables, and set to head

else {

Create new node and initialize

if (i->lastName == lastName){ //same last name to first name comparison

if (i->firstName > firstName){

if (i->prev == nullptr){ // front check

Fit new node before original head

Return true;

}

Fit new node in between nodes and update values

}

else if (i->next == nullptr){ // end case

Fit new node after original end

return true;

}

}

else if (i->lastName > lastName){

if (i->prev == nullptr){ // front check

Fit new node before original head

}

Fit new node in between nodes and update values

}

else if (i->next == nullptr){ // end case

Fit new node after original end

return true;

}

Iterator moves to next node;

}

}

return true;

}

bool CricketTeam::substituteCricketer(const std::string& firstName, const std::string& lastName, const CricketType & value){

Create fullname string

Node\* p = head;

while (p != nullptr){

Find where fullname is

return true;

}

Iterate to next

}

return false;

}

bool CricketTeam::releaseCricketer(const std::string& firstName, const std::string& lastName){

std::string fullName = firstName + lastName;

Node\* p = head;

while (p != nullptr) {

Delete and update values, depending on whether cricketer is at beginning, middle, or end

return true;

}

Iterate through

}

return false;

}

bool CricketTeam::searchForCricketer(const std::string& firstName, const std::string& lastName, CricketType& value) const{

Create fullname string

Iterate through to find fullname and change val

return true;

}

Iterate to next node

}

return false;

}

bool mergeCricketers(const CricketTeam & odOne, const CricketTeam & odTwo, CricketTeam & odJoined){

Initialize status, first name, last name, value

Add everything in odOne to Joined

Add everything from odTwo that’s not in Joined

Change status to false if value is same but name different when doing the comparison

}

}

return status;

}

void checkCricketers (const std::string& fsearch, const std::string& lsearch, const CricketTeam& odOne, CricketTeam& odResult){

Initialize status, first name, last name, value

Create two match variables

First match only true when wild card entered, or first name true

Last match only true when wild card entered, or last match true

Add cricketer if both matches true

**Test Cases**

**CricketTeam GOAT;**

**assert(GOAT.cricketerCount() == 0); //test count**

**assert(GOAT.noTeam()); //test size**

**CricketTeam GOAT2; //copy constructor**

**GOAT2.addCricketer("James", "Cobb", 24);**

**GOAT2.addCricketer("Paul", "Chou", 73);**

**CricketTeam copy(GOAT2);**

**assert(copy.rosteredOnCricketTeam("James", "Cobb"));**

**assert(copy.rosteredOnCricketTeam("Paul", "Chou"));**

**assert (GOAT2.cricketerCount() == 2); // test count again**

**CricketTeam GOAT3; //assignment operator**

**GOAT3 = GOAT2;**

**assert(GOAT3.rosteredOnCricketTeam("James", "Cobb"));**

**assert(GOAT3.rosteredOnCricketTeam("Paul", "Chou"));**

**CricketTeam GOAT4;**

**// Add cricketer test**

**GOAT4.addCricketer("Paul", "Chou", 73);**

**GOAT4.addCricketer("Madi", "Young", 56);**

**GOAT4.addCricketer("Ethan", "Hui", 8);**

**// same last name but different first names**

**GOAT4.addCricketer("Adam", "Jones", 15);**

**GOAT4.addCricketer("Jim", "Jones", 57);**

**GOAT4.addCricketer("Ryan", "Jones", 72);**

**for (int n = 0; n < GOAT4.cricketerCount(); n++)**

**{**

**string first;**

**string last;**

**int val;**

**GOAT4.checkTeamForCricketer (n, first, last, val);**

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

**}**

**//substituteCricketer test**

**GOAT4.substituteCricketer("Madi", "Young", 100);**

**int val1 = 100;**

**assert(GOAT4.searchForCricketer("Madi", "Young", val1));**

**// addorsubstitute test**

**GOAT4.addOrSubstitute("Cody", "Lejang", 30);**

**int val2 = 30;**

**assert(GOAT4.searchForCricketer("Cody", "Lejang", val2));**

**GOAT4.addOrSubstitute("Cody", "Lejang", 100);**

**int val3 = 100;**

**assert(GOAT4.searchForCricketer("Cody", "Lejang", val3));**

**// releaseCricketer test**

**GOAT4.releaseCricketer("Cody", "Lejang");**

**assert(GOAT4.rosteredOnCricketTeam("Cody", "Lejang")==false);**

**// rostered on cricket team test**

**assert(GOAT4.rosteredOnCricketTeam("Cody", "Lejang")==false);**

**assert(GOAT4.rosteredOnCricketTeam("Adam", "Jones"));**

**assert(GOAT4.rosteredOnCricketTeam("Baba", "Booey")==false);**

**// search for cricketer**

**int val4 = 30;**

**assert(GOAT4.searchForCricketer("Bobby", "Jones", val4) == false);**

**assert(GOAT4.searchForCricketer("Madi", "Young", val1));**

**// checkTeamForCricketer**

**CricketTeam GOAT5;**

**GOAT5.addCricketer("James", "Cobb", 75);**

**GOAT5.addCricketer("Paul", "Chou", 89);**

**GOAT5.addCricketer("Cody", "Lejang", 64);**

**string a1;**

**string a2;**

**int val5;**

**GOAT5.checkTeamForCricketer(2, a1, a2, val5);**

**string temp1 = "Cody";**

**string temp2 = "Lejang";**

**int temp3 = 64;**

**assert(GOAT5.searchForCricketer(temp1, temp2, temp3));**

**// check swap**

**GOAT5.tradeCricketTeams(GOAT2);**

**assert(GOAT5.rosteredOnCricketTeam("James", "Cobb"));**

**assert(GOAT5.rosteredOnCricketTeam("Paul", "Chou"));**

**// merge test**

**CricketTeam GOAT7;**

**CricketTeam GOAT8;**

**GOAT7.addCricketer("Billy", "Bob", 64);**

**mergeCricketers(GOAT5, GOAT7, GOAT8);**

**assert(GOAT8.rosteredOnCricketTeam("Billy", "Bob"));**

**assert(GOAT8.rosteredOnCricketTeam("Paul", "Chou"));**

**// checkCricketers test**

**CricketTeam GOAT9;**

**checkCricketers("Adam", "Jones", GOAT4, GOAT9);**

**assert(GOAT9.rosteredOnCricketTeam("Adam", "Jones"));**

**cout << "Passed all tests" << endl;**

**return 0;**

**}**