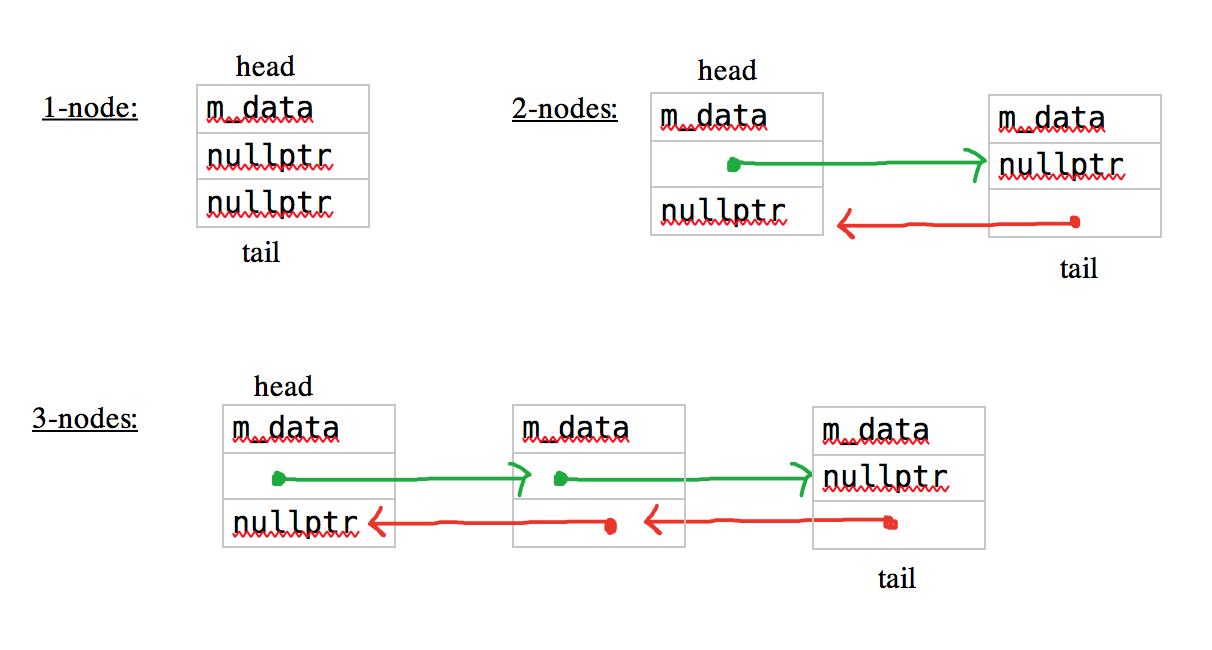
CS32: Project 2

Doubly-linked list implementation: My list is formed a struct Node, which has a prev and next pointers of type Node, and m\_data of type ItemType as its member variables. The list utilizes a head node to represent the first element in the list, and a tail node that represents the last node in the list. The head’s prev and the tail’s next pointers are nullptrs, as the head and tail are the edges of the list. The nodes in between link to each other by their prev and next pointers, such that the prev pointer of a node points to the node before it, and the next pointer points to the node after it.



Pseudocode of non-trivial algorithms

bool insert(const ItemType& value)

{  
 //If empty list is empty, insert first element.

//If not, search list for value. if found, return false

//if not found, create new node.

//adjust what the node points to

//adjust pointers to the new node

//at each insert, increment size.

}

bool erase(const ItemType& value);

{

//if empty, return false.

//search list. If found, create temporary node pointer to point to it.

//adjust pointers, delete temporary pointer.

}

void unite(const Set& s1, const Set& s2, Set& result)

{

//put s1 in result

//loop through s2. If element in s2 is not already in result, insert it.

}

void subtract(const Set& s1, const Set& s2, Set& result {

//put s1 in result. Loop trhough s2.

//If element is already in result, erase it }

Test cases for Set public functions

    Set a;

    assert(a.empty()==true);

    assert(a.size()==0);

    assert(a.insert("hello")==true);    //inserting one node

    assert(a.contains("hello")==true);

    assert(a.size()==1);

    assert(a.insert("hello")==false);   //repeated value will not insert.

    assert(a.size()==1);                //size will remain the same.

    assert(a.empty()==false);           //head is not nullptr, set is not empty

    assert(a.insert("hola")==true);     //inserting second node

    assert(a.contains("hola")==true);

    assert(a.size()==2);

    assert(a.insert("hola")==false);   //repeated value will not insert

    assert(a.size()==2);               //size will remain the same

    assert(a.empty()==false);          //head is not nullptr, set is not empty

    assert(a.insert("aloha")==true);    //inserting value that will come into first space in the set

    assert(a.contains("aloha")==true);

    assert(a.size()==3);

    assert(a.insert("aloha")==false);   //repeated value will not insert

    assert(a.size()==3);                //size will remain the same

    assert(a.empty()==false);           //head is not nullptr, set is not empty

    assert(a.insert("konichiwa")==true); //inserting node at the end of set

    assert(a.contains("konichiwa")==true);

    assert(a.size()==4);

    assert(a.insert("konichiwa")==false);   //repeated value will not insert

    assert(a.size()==4);                    //size will remain the same

    assert(a.empty()==false);               //head is not nullptr, set is not empty

    assert(a.insert("nihao")==true);        //inserting node at end of the set

    assert(a.contains("nihao")==true);

    assert(a.size()==5);

    assert(a.insert("nihao")==false);       //repeated value will not insert

    assert(a.size()==5);                    //size will remain the same

    assert(a.empty()==false);               //head is not nullptr, set is not empty

    assert(a.insert("bonjour")==true);      //inserting node in the middle of set

    assert(a.contains("bonjour")==true);

    assert(a.size()==6);

    assert(a.insert("bonjour")==false);  //repeated value will not insert

    assert(a.size()==6);                 //size will remain the same

    assert(a.empty()==false);            //head is not nullptr, set is not empty

//////////////////////////////////////////////////////////////////////////////////////////////////////////

//get function

    a.dump();                           //sorted in order? yep.

    ItemType beginning;                 //get first element in list (sorted in order, lexicon)

    a.get(0, beginning);

    assert(beginning=="aloha");

    assert(beginning!="bonjour");

    assert(beginning!="nihao");

    ItemType second;

    a.get(1, second);

    assert(second=="bonjour");   //get second element

    assert(second!="aloha");

    assert(second!="nihao");

    ItemType third;

    a.get(2, third);

    assert(third=="hello");     //get third element

    assert(third!="aloha");

    assert(third!="nihao");

    ItemType fourth;

    a.get(3, fourth);

    assert(fourth=="hola");     //get fourth element

    assert(fourth!="aloha");

    assert(fourth!="nihao");

    ItemType fifth;

    a.get(4, fifth);

    assert(fifth=="konichiwa"); //get fifth element

    assert(fifth!="aloha");

    assert(fifth!="nihao");

    ItemType end;

    a.get(5, end);

    assert(end == "nihao");     //get last element

    assert(end!="aloha");

    assert(end!="konichiwa");

    assert(a.empty()==false);   //get is const, shouldn't change anything about the set

    assert(a.size()==6);

    a.dump();

//////////////////////////////////////////////////////////////////////////////////////////////////////////

//erase function

    assert(a.erase("nihao")==true);       //erase node at the end of the list

    assert(a.contains("nihao")==false);

    assert(a.size()==5);                  //decrement size

    assert(a.empty()==false);             //head is not nullptr, set is not empty

    a.dump();                             //last element should no longer be in the list

    ItemType newEnd;

    a.get(4, newEnd);

    assert(newEnd=="konichiwa");

    assert(a.erase("aloha")==true);       //erase node at beginning of the list

    assert(a.contains("aloha")==false);

    assert(a.size()==4);                  //decrement size

    assert(a.empty()==false);             //head is not nullptr, set is not empty

    a.dump();                             //element that was at beginning shold no longer be there

    ItemType newBegin;

    a.get(0, newBegin);

    assert(newBegin == "bonjour");

    assert(a.erase("hola")==true);        //erase node in the middle of the list

    assert(a.contains("hola")==false);

    assert(a.size()==3);                  //decrement size

    assert(a.empty()==false);             //head is not nullptr, set is not empty

    a.dump();                             //"hola" no longer in the list

    assert(a.erase("notThere")==false);   //cannot erase something that is not there

    //testing if deleting all elements, size will return 0 and empty will return true (head is nullptr)

    a.erase("hello");

    assert(a.size()==2);

    a.erase("bonjour");

    assert(a.size()==1);

    assert(a.erase("konichiwa")==true);    //erasing one element

    assert(a.size()==0);                   //size is 0 because empty

    assert(a.empty()==true);               //head IS nullptr, the set IS empty

//////////////////////////////////////////////////////////////////////////////////////////////////////////

//swap function

    //swapping one element lists

    Set exchange1;

    exchange1.insert("donut");

    assert(exchange1.size()==1);

    Set exchange2;

    exchange2.insert("cereal");

    assert(exchange2.size()==1);

    exchange1.swap(exchange2);

    assert(exchange1.contains("cereal")==true);

    assert(exchange1.contains("donut")==false);

    std::cerr<< "this is first set NOW:" << std::endl;

    exchange1.dump();           //exchange1 should have "cereal" //works!

    exchange2.dump();           //exchange2 should have "donut"  //works!

    assert(exchange2.contains("cereal")==false);

    assert(exchange2.contains("donut")==true);

    std::cerr<<"this is second set NOW:" << std::endl;

    exchange2.dump();

    assert(exchange1.size()==1 && exchange2.size()==1);

    //////////////////////////////////////////////////

    //swapping lists with one with more elements than the other

    Set exchange3;

    exchange3.insert("paper");

    assert(exchange3.contains("paper")==true);

    assert(exchange3.size()==1);

    Set exchange4;

    exchange4.insert("pencil");

    exchange4.insert("pen");

    assert(exchange4.contains("pencil")==true && exchange4.contains("pen")==true);

    assert(exchange4.size()==2);

    exchange3.swap(exchange4);  //swap!

    assert(exchange3.contains("pencil")==true && exchange3.contains("pen")==true);

    assert(exchange3.contains("paper")==false);

    assert(exchange3.size()==2);    //sizes switched

    assert(exchange4.size()==1);    //sizes switched

    assert(exchange4.contains("paper")==true);

    assert(exchange4.contains("pencil")==false && exchange4.contains("pen")==false);

    exchange3.dump();       //elements switched

    exchange4.dump();

    exchange4.swap(exchange3);  //return to original states

    assert(exchange3.contains("paper")==true);

    assert(exchange3.size()==1);

    exchange4.insert("pencil");

    exchange4.insert("pen");

    assert(exchange4.contains("pencil")==true && exchange4.contains("pen")==true);

    assert(exchange4.size()==2);

    /////////////////////////////////////////////////////////

    //swapping same size list that have more than one element

    Set exchange5;

    exchange5.insert("ideal");

    exchange5.insert("gas");

    assert(exchange5.size()==2);

    assert(exchange5.contains("ideal") && exchange5.contains("gas"));

    Set exchange6;

    exchange6.insert("green");

    exchange6.insert("binder");

    assert(exchange6.size()==2);

    assert(exchange6.contains("green") && exchange6.contains("binder"));

    exchange5.swap(exchange6);                      //swap!

    assert(exchange5.size()==2);

    assert(exchange5.contains("green") && exchange5.contains("binder"));

    assert(exchange6.size()==2);

    assert(exchange6.contains("ideal") && exchange6.contains("gas"));

    /////////////////////////////

    //test for aliasing

    //     ???????

    exchange5.swap(exchange5);

    assert(exchange5.size()==2);

    assert(exchange5.contains("green") && exchange5.contains("binder"));

    exchange6.swap(exchange6);

    assert(exchange6.size()==2);

    assert(exchange6.contains("ideal") && exchange6.contains("gas"));

///////////////////////////////////////////////////////////////////////////////////////////

//spec test cases

    Set ss1;

    ss1.insert("bing");

    Set ss2;

    ss2.insert("matzo");

    ss2.insert("pita");

    ss1.swap(ss2);

    assert(ss1.size() == 2  &&  ss1.contains("matzo")  &&  ss1.contains("pita")  &&

           ss2.size() == 1  &&  ss2.contains("bing"));

    Set s;

    assert(s.empty());

    ItemType b = "arepa";

    assert( !s.get(42, b)  &&  b == "arepa"); // x unchanged by get failure

    s.insert("chapati");

    assert(s.size() == 1);

    assert(s.get(0, b)  &&  b == "chapati");

    Set ss;

    // For an empty set:

    assert(ss.size() == 0);        // test size

    assert(ss.empty());            // test empty

    assert(!ss.erase("roti"));     // nothing to remove

    ss.insert("lavash");

    ss.insert("roti");

    ss.insert("chapati");

    ss.insert("injera");

    ss.insert("roti");

    ss.insert("matzo");

    ss.insert("injera");

    assert(ss.size() == 5);  // duplicate "roti" and "injera" were not added

    string x;

    ss.get(0, x);

    assert(x == "chapati");  // "chapati" is greater than exactly 0 items in ss

    ss.get(4, x);

    assert(x == "roti");  // "roti" is greater than exactly 4 items in ss

    ss.get(2, x);

    assert(x == "lavash");  // "lavash" is greater than exactly 2 items in ss

    Set sss;

    sss.insert("dosa");

    assert(!sss.contains(""));

    sss.insert("tortilla");

    sss.insert("");

    sss.insert("focaccia");

    assert(sss.contains(""));

    sss.erase("dosa");

    assert(sss.size() == 3  &&  sss.contains("focaccia")  &&  sss.contains("tortilla")  &&

           sss.contains(""));

    string v;

    assert(sss.get(1, v)  &&  v == "focaccia");

    assert(sss.get(0, v)  &&  v == "");

    Set ssss;

    assert(ssss.insert("roti"));

    assert(ssss.insert("pita"));

    assert(ssss.size() == 2);

    assert(ssss.contains("pita"));

    ItemType c = "bing";

    assert(ssss.get(0, c)  &&  c == "pita");

    assert(ssss.get(1, c)  &&  c == "roti");

    cerr << "Passed all tests! yay!" << endl;