design of my linked list: I used a circular doubly linked list. Each node has a previous and a next. If the list is empty there is only a head where it’s next and previous is equal to itself. If the list is not empty he first item’s (the head) next item is the last time, and the last item’s previous is the head.

PSEUDOCODE

Sequence()

set head next and previous to point to itself

~Sequence()

point to second item in list

while not back at the start

delete item

delete the first item

Sequence(const Sequence& other)

if other Sequence is empty

create new head node that points to itself

else

create new head node that points itself

repeatedly while less than other sequences size

move along other Sequence

add to back of this sequence

Sequence& operator=(const Sequence& other)

if the this is the same as the other

return this sequence

set this size to other’s size

repeatedly

insert data from other sequence

swap the two so this has the data from other

return this sequence

int insert(int pos, const ItemType& value)

if position is not valid

return -1

if position is the front or the sequence is empty

add to the front

return the first position (0)

if the position is the same as the size of the sequence

add to the back and return the size

else

create new memory for a node pointer

set it equal to the first item

walk through the sequence until it gets to position

set the new node’s data

set the new node’s previous to the position’s previous

set the new node’s next to the position

set the next of the previous of position to n

set the prevb of position to n

increase the size

return the position

int insert(const ItemType value)

if Sequence is empty

add to the front

if found a value in the sequence that is greater than or equal to value

found a match

else

move onto the next item

if no valid value is found in the list

add value to the back

if position is the front

add to the front

otherwise

create new memory for a new node

set the new node’s data

set the new node’s previous to the position’s previous

set the new node’s next to the position

set the next of the previous of position to n

increase size

bool erase(int pos)

if position isnt valid

return false

walk through the list until at the position

if the position is the first item

set head to the next item (because that head will be removed)

set the position’s previous’ next equal to the position’s next

set the position’s next’s previous equal to the position’s previous

delete the item at the position

decrease size

int remove(const ItemType& value)

walk through the list until a matching value is found

if value matches

set the position’s previous’ next equal to the position’s next

set the position’s next’s previous equal to the position’s previous

if position is the first

set head to the next item (because that head will be removed)

create pointer to the item to be deleted (trash)

move the position forward

delete the item at the trash pointer

increase the number found

decrease size

move back one since the size has been decreased

else

move to next item

return amount found

bool get(int pos, ItemType& value)

if position isnt valid

return false

if position is first item

set to value of first item

otherwise

walk through the list until the position

set value equal to the data at that position

bool set(int pos, const ItemType& value)

if position isn’t valid

return false

otherwise walk through the list until at desired position

set the data to the inputted value

int find(const ItemType& value)

if the list is empty there is no value to be found so

return -1

otherwise

walk through the list until desired value is found

if value is found

break

update position

void swap(Sequence& other)

swap the pointers to the first item

swap sizes

void dump()

walk through list

print value at iteration

if the list is empty

set first item to value

else

create new memory

set the data to the value

set the new node’s next to the current first item

set the new node’s previous to the first item’s previous

set the last item’s next to the new node

set the first item’s previous to the new node

update so the new item is now the first item

increase size

void addToBack(ItemType value)

if the list is empty

add to the front

else

walk through the list until at the last item

create new memory

set the data to the value

set the last value’s next equal to the new node

set the new nodes previous node equal to the last node

set the new node’s next to the first item

set the first item’s previous to the new node

increase size

int subsequence(const Sequence& seq1, const Sequence& seq2)

if the sub sequence is empty there is nothing to compare

return -1

if the sub sequence is bigger than the comparing string

return -1

loop through the comparing string until the first value of the substring is found

if a matching value is found

check that the sub sequence is not longer than where it starts on the comparing string

check that each subsequent item in the comparing string is equal to the end of the sub sequence

if at any point it is not equal

return -1

void interleave(const Sequence& seq1, const Sequence& seq2, result)

if either of the input is empty

set it to the other

else

while each source sequence has items

alternate adding items

TEST CASES

////to test string

//to test std::string

//Sequence ss; // ItemType is std::string

//assert(ss.insert(0, "aaa") == 0);

//assert(ss.insert(1, "bbb") == 1);

//assert(ss.insert(2, "ccc") == 2);

//ItemType x = "xxx";

//assert(!ss.get(3, x) && x == "xxx"); // x is unchanged

//assert(ss.get(1, x) && x == "bbb");

////to test int

//int i = 0;

//Sequence s;

//to test the insert and empty function

//assert(s.empty());

//assert(s.size() == 0);

//assert(s.insert(0, 1) == 0);

//assert(s.insert(1, 2) == 1);

//assert(s.insert(2, 3) == 2);

//assert(s.insert(3, 4) == 3);

//assert(s.insert(4, 5) == 4);

//assert(s.insert(5, 6) == 5);

//assert(s.insert(6, 7) == 6);

//assert(s.insert(7, 8) == 7);

//assert(s.insert(4) == 3);

//

//to test insert, find and erase

//Sequence s1;

//assert(s1.insert(4) == 0);

//assert(s1.insert(64) == 1);

//s1.dump();

//assert(s1.insert(23) == 1);

//s1.dump();

//assert(s1.insert(84) == 3);

//s1.dump();

//assert(s1.insert(23) == 1);

//s1.dump();

//assert(s1.insert(16) == 1);

//s1.dump();

//assert(s1.insert(74) == 5);

//s1.dump();

//assert(s1.insert(86) == 7);

//s1.dump();

//assert(s1.find(16) == 1);

//assert(s1.find(4) == 0);

//assert(s1.find(74) == 5);

//assert(s1.erase(5) == true);

//assert(s1.get(5, i) && i == 84);

//assert(s1.erase(9) == false);

//s1.dump();

//assert(s1.remove(23) == 2);

//s1.dump();

//to test get and set

//assert(s1.get(2, i));

//assert(i == 64);

//assert(s1.set(2, 30) == true);

//s1.dump();

//assert(s1.set(s1.size() - 1, 1) == true);

//s1.dump();

//to test the swap function

//Sequence s13;

//s13.insert(17);

//s13.insert(26);

//s13.insert(2);

//s13.dump();

//assert(s13.set(2, 27) == true);

//assert(s13.set(-1, 27) == false);

//assert(s13.set(5, 27) == false);

//s13.dump();

//s1.swap(s);

//assert(s.find(30) == 2);

//to test subsequence

//Sequence s2;

//assert(s2.insert(6) == 0);

//s2.dump();

//assert(s2.insert(17) == 1);

//s2.dump();

//assert(s2.insert(43) == 2);

//s2.dump();

//assert(s2.insert(43) == 2);

//s2.dump();

//assert(s2.insert(17) == 1);

//s2.dump();

//assert(s2.insert(6) == 0);

//s2.dump();

//assert(s2.insert(90) == 6);

//s2.dump();

//assert(s2.insert(16) == 2);

//s2.dump();

//assert(s2.insert(6) == 0);

//s2.dump();

//assert(s2.insert(43) == 6);

//s2.dump();

//assert(s2.insert(17) == 4);

//s2.dump();

//assert(s2.insert(9) == 3);

//assert(s2.find(90) == 11);

//assert(s2.find(17) == 5);

//assert(s2.find(43) == 8);

//assert(s2.remove(6) == 3);

//s2.dump();

//assert(s2.size() == 9);

//assert(s2.remove(90) == 1);

//assert(s2.size() == 8);

//s2.dump();

//assert(s2.remove(17) == 3);

//Sequence s3, s4, s5, s6, s7;

//s3.insert(0, 5);

//s3.insert(1, 23);

//s3.insert(2, 6);

//s3.insert(3, 4);

//s3.insert(4, 2);

//s3.insert(5, 64);

//s3.insert(6, 234);

//s3.insert(7, 63);

//s3.insert(8, 73);

//s3.insert(9, 53);

//s3.insert(10, 7);

//s3.insert(11, 42);

//assert(s3.size() == 12);

//s3.dump();

//s4.insert(0, 2);

//s4.insert(1, 64);

//s4.insert(2, 234);

//s4.insert(3, 63);

//s4.dump();

//s5.insert(0, 4);

//s5.insert(1, 42);

//s5.insert(2, 3);

//s5.insert(3, 7);

//s5.dump();

//s6.insert(0, 1);

//s6.insert(1, 2);

//s6.insert(2, 3);

//s6.insert(3, 4);

//s6.insert(4, 5);

//s6.insert(5, 6);

//s6.insert(6, 7);

//s6.insert(7, 8);

//s6.insert(8, 9);

//s6.insert(9, 10);

//s6.insert(10, 11);

//s6.insert(11, 12);

//s6.insert(12, 13);

//assert(s6.size() == 13);

//s6.dump();

//s7.insert(0, 4);

//s7.insert(1, 2);

//s7.insert(2, 64);

//s7.insert(3, 234);

//s7.insert(4, 63);

//s7.insert(5, 73);

//s7.insert(6, 53);

//s7.insert(7, 7);

//s7.insert(8, 42);

//s7.insert(9, 4);

//s7.insert(10, 4);

//s7.insert(11, 2);

//s7.insert(12, 1);

//s7.insert(13, 4);

//assert(subsequence(s3, s4) == 4);

//assert(subsequence(s3, s5) == -1);

//assert(subsequence(s3, s6) == -1);

//assert(subsequence(s3, s7) == -1);

//Sequence s11, s12;

//s11.insert(0, 4);

//s11.insert(1, 2);

//s11.insert(2, 64);

//s12.insert(0, 2);

//s12.insert(1, 64);

//assert(subsequence(s11, s12) == 1);

//Sequence s14;

//assert(s14.empty() == true);

//assert(s14.size() == 0);

//assert(s14.insert(0, 5) == 0);

//s14.dump();

//assert(s14.insert(1, 9) == 1);

//s14.dump();

//assert(s14.insert(2, 3) == 2);

//s14.dump();

//assert(s14.insert(1, 5) == 1);

//s14.dump();

//assert(s14.insert(4, 20) == 4);

//s14.dump();

//assert(s14.insert(19) == 4);

//assert(s14.size() == 6);

//assert(s14.empty() == false);

//s14.dump();

//assert(s14.erase(0) == true);

//s14.dump();

//assert(s14.size() == 5);

//assert(s14.erase(3) == true); //size = 4

//s14.dump();

//size = 4;

//assert(s14.insert(0, 5) == 0); //size = 4

//s14.dump();

//assert(s14.remove(5) == 2);

//s14.dump();

//assert(s14.remove(20) == 1);

//s14.dump();

//assert(s14.remove(9) == 1);

//s14.dump();

//to test interleave

//Sequence s8, s9, s10;

//s8.insert(0, 6);

//s8.insert(1, 3);

//s8.insert(2, 8);

//s9.insert(0, 12);

//s9.insert(1, 7);

//s9.insert(2, 3);

//interleave(s8, s9, s10);

//assert(s10.get(0, i) && i == 6);

//assert(s10.get(1, i) && i == 12);

//assert(s10.size() == 6);

//Sequence sEmpty;

//assert(sEmpty.empty());

//interleave(sEmpty, s8, s10);

//Sequence s15 = Sequence(s8);

//assert(s15.size() == 3);

//cout << "passed!";

return 0;