John Yoo

304787909

Design: In the empty sequence, both the head and tail are set to the null pointer. In a sequence with one node, the prev and next Node are set to null pointers. In a sequence with multiple node, the head node’s prev node is nullptr while the tail node’s next node is set to nullptr.

Pseudocode

Sequence::~Sequence() {

while the head node is not set to nullptr:

Another node is set to the node that comes after the head node

The head node’s content is deleted

The other node becomes the new head node

}

bool insert(int pos, const ItemType& value) {

Inserts the value into the Sequence at the position indicated in the argument

If the value is successfully inserted

Return value is set to true

If the position passed is less than 0 or more than the Sequence’s current size

Return value is set to false

}

int insert(const ItemType& value) {

Inserts the value in alphabetical order

Returns the pos in the sequence in which the value was inserted

}

bool erase(int pos) {

Erases the value at the position indicated in the argument

If the pos indicated is more than the sequence’s size or less than 0 (can’t be found in the sequence)

Returns false

If the value is successfully removed

Returns true

}

int remove(const ItemType& value) {

If the value is found in the sequence:

The value is removed from the sequence

Writes down the number of times the value was found and removed from the sequence

}

void swap(Sequence& other) {

Swaps the head and tail nodes with another sequence’s head and tail nodes as well as the number of items contained in the sequence

}

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

Repeatedly:

if the 2nd sequence is found in the 1st sequence:

Writes the position in the 1st sequence where the 2nd sequence is found

else:

Writes -1

}

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

First value of seq1 is inserted to result and then the first value of seq2 is inserted into result

Seq1 and seq2 alternate inserting their values into result until all of their values have been inserted

}

Test cases:

Sequence s;

//Tests empty function

assert(s.empty());

//Tests the two-argument insert function

assert(s.insert(0, "a"));

assert(s.insert(1, "b"));

assert(s.insert(2, "c"));

assert(s.insert(2, "cc"));

assert(s.insert(3, "d"));

assert(s.insert(4, "e"));

assert(s.insert(4, "b"));

assert(s.insert(0, "aa"));

assert(s.insert(7, "ee"));

assert(s.insert(7, "b"));

//Tests the size function

assert(s.size() == 10);

//Tests the remove function

assert(s.remove("b") == 3);

//Tests the get function as well as the correctness of the remove function

ItemType x;

assert(s.get(0, x) && x == "aa");

assert(s.get(1, x) && x == "a");

assert(s.get(2, x) && x == "cc");

assert(s.get(3, x) && x == "d");

assert(s.get(4, x) && x == "e");

assert(s.get(5, x) && x == "ee");

assert(s.get(6, x) && x == "c");

//Tests 1 argument insert function

Sequence s2;

s2.insert(“d”);

s2.insert(“a”);

s2.insert(“z”);

s2.insert(“y”);

s2.insert(“r”);

s2.insert(“c”);

assert(s2.get(0, x) && x == "a");

assert(s2.get(1, x) && x == "c");

assert(s2.get(2, x) && x == "d");

assert(s2.get(3, x) && x == "r");

assert(s2.get(4, x) && x == "y");

assert(s2.get(5, x) && x == "z");

//Tests erase function

s2.erase(0);

s2.erase(5);

s2.erase(1);

assert(s2.get(0, x) && x == "c");

assert(s2.get(1, x) && x == "r");

assert(s2.get(2, x) && x == "y");

//Tests set and find functions

x = “x”;

assert(s2.set(0, x) && s2.find(“x”) == 0);

x = “x1”;

assert(s2.set(1, x) && s2.find(“x1”) == 1);

x = “x2”;

assert(s2.set(2, x) && s2.find(“x2”) == 2);

//Tests swap function

swap(s, s2);

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

assert(s.get(1, x) && x == "x1");

assert(s.get(2, x) && x == "x2");

assert(s2.get(0, x) && x == "aa");

assert(s2.get(1, x) && x == "a");

assert(s2.get(2, x) && x == "cc");

assert(s2.get(3, x) && x == "d");

assert(s2.get(4, x) && x == "e");

assert(s2.get(5, x) && x == "ee");

assert(s2.get(6, x) && x == "c");

//Tests operator overload and copy constructor

s2 = s;

assert(s2.get(0, x) && x == "x");

assert(s2.get(1, x) && x == "x1");

assert(s2.get(2, x) && x == "x2");

//Tests subsequence

Sequence s3;

assert(s3.insert(0, "g"));

assert(s3.insert(1, "i"));

assert(s3.insert(2, "m"));

assert(s3.insert(3, "g"));

assert(s3.insert(4, "g"));

assert(s3.insert(5, "i"));

assert(s3.insert(6, "k"));

Sequence s4;

assert(s4.insert(0, "g"));

assert(s4.insert(1, "i"));

assert(s4.insert(2, "k"));

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

s3.print1();

cerr << "Return true" << endl;

return 0;

//Tests interleave function

Sequence result;

interleave(s3, s4, result);

assert(s2.get(0, x) && x == "g");

assert(s2.get(1, x) && x == "g");

assert(s2.get(2, x) && x == "i");

assert(s2.get(3, x) && x == "i");

assert(s2.get(4, x) && x == "m”);

assert(s2.get(5, x) && x == "k");

assert(s2.get(6, x) && x == "g");

assert(s2.get(7, x) && x == "g");

assert(s2.get(8, x) && x == "i");

assert(s2.get(9, x) && x == "k");