Description:

My sequence class is based on a doubly-linked list and an int member variable that tracks the size of the list. An empty list would contain a head pointer that is set to nullptr and a size variable that equals to 0. A usual list would consist of a size variable that equal to the number of nodes currently in that list. A struct Node is declared in the header file. Each Node stores a value of type ItemType, a prev pointer that points to the previous Node in the list, and a next pointer that points to the next pointer in the list. The first Node has a prev pointer of nullptr, while the last Node has a next pointer of nullptr. We traverse the list by running loops that follows the next pointer of each individual Node.

Pseudocode:

bool insert(int pos, const std::string& value)

if pos is out of bound

return false

if list is empty

create new node storing value

add new node to front of list

size of list++

return true

if pos = size of list

create new node string value

add new node to the end of list

size of list++

return true

loop through the list until we found the node at pos

create new node storing value

insert new node between the node we found and the node before the node we found

size of list++

return true

int insert(const std::string& value)

loop through the list until a node with an item <= value is found

record the position

if no item in list is <= value

create a new node that stores value

add new node to the end of the list

size of list++

return position of node

else

create a new node that stores value

insert the new node between the node we found and the node before the node we found

size of list++

return position of node

bool erase(int pos)

if pos is out of bound

return false

if list is empty

return false

loop through the list until node at pos is found

set the pointers of the nodes before and after current node to connect to each other

delete current node

size of list--

return true

int remove(const std::string& value)

loop through every node in the list

if the item stored at current node equal to value

call erase function on this position

keep track of number of items removed

return number of items removed

bool get(int pos, std::string& value)

If the pos is out of bound

Return false

If the list is empty

Return false

Loop through the list until the node at position pos

Set value to the item stored at pos

Return true;

bool set(int pos, const std::string& value)

if pos is out of bound

return false

if list is empty

return false

loop through the list until the node at pos is found

set the item stored in the node at pos to value

return true

int find(const std::string& value)

loops through the list using the head pointer

keep track of the current position in the list

checks if the current node stores an equivalent of value

return current position

void swap(Sequence& other)

swaps the size variables of the two sequences

swaps the head pointer of the two sequences

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

find the longer list

set result equal to the longer list

loop through the shorter list

create new node storing value at current position in shorter list

interleave the new node into desired position in result

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

if seq2 is shorter than seq1

return -1

if either list is empty

return -1

loop through seq1

if current node is seq1 = first node of seq2

record position

if the number of nodes left in seq1 < number of total nodes in seq2

return -1

loop through all nodes in seq2

if all items in seq2 are found consequtively in seq1 starting at recorded location

return recorded location

Test code:

int main()

{

Sequence s;

Sequence s2;

//const func test: empty, size, get, find

assert(s.empty());

assert(s.find(3) == -1);

ItemType testArray[7] = { 34, 5436, 4356, 890, 23, 43, 6 };

load(testArray, 7, s);

cout << "done loading" << endl;

printDat(s);

assert(s.size() == 7 && s.find(5436) == 1);

ItemType test;

s.get(3, test);

assert(test == 890);

assert(s.find(23) == 4);

cout << "Passed all const" << endl;

//non-const func test: erase, remove

Sequence x;

Sequence x2;

ItemType remove[10] = { 1,2,4,6,7,8,9,65,4,4 };

ItemType erase[9] = { 1,2,3,4,5,6,7,8,9 };

load(remove, 10, x);

load(erase, 9, x2);

cout << "x:" << endl;

printDat(x);

cout << "x2:" << endl;

printDat(x2);

x2.erase(3);

ItemType result;

x2.get(3, result);

assert(result != 4);

cout << "x2 size: " << x2.size() << endl;

for (int i = 3; i < x2.size(); i++) {

x2.get(i, result);

assert(result == i + 2);

}

cout << "erase passed" << endl;

x.remove(4);

assert(x.find(4) == -1);

cout << "erase and remove passed" << endl;

//swap test

ItemType testArray2[7] = { 5, 3, 1, 7, 6, 2, 4 };

for (int i = 0; i < 7; i++) {

s2.insert(testArray2[i]);

}

cout << endl << "//////////////swap test//////////////" << endl;

cout << "originals:" << endl << "s:" << endl;

printDat(s);

cout << "s2:" << endl;

printDat(s2);

cout << endl;

cout << "swapped:" << endl;

s.swap(s2);

cout << "s:" << endl;

printDat(s);

cout << "s2:" << endl;

printDat(s2);

cout << "s2 copy constructor:" << endl;

Sequence s2copy = s2;

printDat(s2);

cout << "s2 assignment:" << endl;

s = s2;

printDat(s);

cout << "testing interleave and subsequence" << endl;

Sequence inter1;

Sequence inter2;

Sequence sub1;

Sequence sub2;

ItemType inter\_1[4] = { 1,3,5,7 };

ItemType inter\_2[3] = { 2,4,6 };

ItemType sub\_1[6] = { 2, 4, 6, 8, 10, 12 };

ItemType sub\_2[2] = { 4, 6 };

load(inter\_1, 4, inter1);

load(inter\_2, 3, inter2);

load(sub\_1, 6, sub1);

load(sub\_2, 2, sub2);

int subResult = subsequence(sub1, sub2);

assert(subResult == 1);

cout << "subPassed" << endl;

Sequence interResult;

interleave(inter1, inter2, interResult);

printDat(interResult);

cout << "interPassed" << endl;

}