Report: Programming Project 2

1. My doubly linked list implementation is as follows:

My doubly linked list is circular, such that the last element is linked to the first through a dummy node. The empty sequence is such that a head and a tail, are the same and the dummy node. It also has no elements in it.

A picture containing text, whiteboard

Description automatically generated

1. Pseudocode:

int Sequence::insert(int pos, const ItemType &value)

Check if the sequence is empty.   
If yes, create a new node and give it the value.   
Link it appropriately to the dummy nodes, so that the list only has one element in the sequence.

Check if we are trying to insert at the back of the sequence. Create the new node, assign it the value and accordingly adjust the pointers, so that the new node goes after the last node in the sequence but before the dummy.

If inserting anywhere else in the sequence, create a new node and assign it the value. Then, appropriately reassign the pointers.

In every case above, return the position and increment the number of items in sequence by 1.

int Sequence::insert(const ItemType& value)

Check if the sequence is empty. If yes, then allocate a new node and assign it the value. Reassign the pointers so that it is pointing to the dummy node both previously and after it.

If the sequence is not empty, traverse through the sequence to find the position where the item is less than value.

Then insert the value by creating a new node and assigning it the value. Rearrange the pointers accordingly to include this addition in the list.

In both cases, return the position where the value is inserted.

bool Sequence::erase(int pos)

Check if the sequence is empty.

If yes, then return false as we cannot erase anything from an empty sequence.

If not, then traverse through the list using the for loop.

Upon reaching the position, delete the item from the position and rearrange the pointers accordingly.

Decrement the number of items in the sequence by 1 upon deleting the node.

int Sequence::remove(const ItemType& value)

Set the counter to zero.

Traverse the sequence starting at the node next to dummy.

Move along and find all instances where the item in node is equal to the value.

Delete these nodes and rearrange the pointers of the linked list.

Increment counter for each iteration of deletion in the loop.

Decrement number of items in the sequence for each iteration of deletion.

bool Sequence::set(int pos, const ItemType& value)

Traverse through the sequence starting at the node after dummy.

Also start a counter x, so that it increments every time we successively pass through a position.

When the counter equals the position, copy the value to the item of the node, and return true.

bool Sequence::get(int pos, ItemType& value) const

Traverse through the sequence starting at the node after dummy.

Also start a counter x, so that it increments every time we successively pass through a position.

When the counter equals the position, copy the item of the node to the value, and return true.

int Sequence::find(const ItemType& value) const

Traverse the sequence starting at the node next to dummy.

Also, start a counter starting at zero so that it increments every time the pointer moves to the next position.

Whenever the item at the node equals value, return the position of the node.

If there’s no match in the sequence, return -1.

void Sequence::swap(Sequence& other)

Swap the size of the sequence using a temp variable

swap the dummy variables of the sequences using a temp node

int Sequence::interleave(const Sequence& seq1, const Sequence& seq2)

Set variables equal to the size of all sequences, i.e. seq1, se2, and result.

If one of the sequences is empty, loop through the other sequence inserting its value in the result.

If none of the sequences are empty, but one of the sequences is longer than the other, loop through the elements of the sequences up to the level of the shorter sequence.

Then add the remaining members of the longer sequence in the end.

Use the erase function to remove all the initial elements of the result.

This gives result only the value of the interleaved sequences.

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

Start counters for both the sequences.

When the element in one sequence matches the element in the other sequence we increment the counter and record the position.

Get the next element of both sequences.

If it doesn’t match, break loop and return counter for position.

If it matches, continue the loop until no subsequent items match, we break the loop and return the counter for position. s

1. Test Cases

void test(){

Sequence seq; //to check for constructor

Sequence seq2;

Sequence result;

Sequence seq3;

Sequence seq4;

assert(seq.empty()); //for implementation of empty function

assert(seq.find("boola") == -1); //to check if the find function is working; return -1 if it cannot find

assert(seq.insert("boola") == 0); //to check if the insert function is working

assert(seq.size() == 1 && seq.find("boola") == 0); //to run the size function again after insertion and checking the if the find function is working

seq.insert(0, "boola");

seq.insert(0, "hoola");

seq.insert(2, "hoola");

seq.insert(4, "boola");

seq.insert(4, "loola");

seq.insert(3, "doula");

seq.insert(3, "boula");

assert(seq.remove(“hoola”) == 2); //to check if the remove function is working

assert(seq.size() == 6);

assert(seq.erase(0) == true); //to check if the erase function is working

string x;

assert(seq.get(0, x) && x == “boola”); //to check if the get function is working

assert(seq.set(0, “foola”)); //to check if the set function is working

assert(seq.get(0, x) && x == “foola”);

seq2.insert(0, "poodle");

seq2.insert(1, "noodle");

seq2.insert(2, "doodle");

seq2.insert(3, "oodle");

seq.2insert(4, "coodle");

seq2.insert(5, "moodle");

seq2.swap(seq); //to check if the swap function is working

string p;

assert(seq.get(0, p) && p == “poodle”);

assert(seq2.get(4, x) && x == “loola”);

result.insert(0, "lol");

result.insert(1, "cute");

result.insert(2, "you");

result.insert(3, "are");

seq.interleave (seq, seq2, result); //to check if the interleaving function works

assert(result.get(0, x) && x == “foola”);

seq3.insert(0, "10");

seq3.insert(1, "20");

seq3.insert(2, "30");

seq3.insert(3, "40");

seq4.insert(0, "10");

seq4.insert(1, "20");

seq4.insert(2, "30");

seq4.insert(3, "50");

assert( seq3.subsequence(seq3, seq4) == 0); //to check if the subsequence function

}