Description

My program implements a doubly linked list. The sequence contains a pointer to a head node only that is initialized to nullptr and a size that is set to 0. My nodes contain a value variable of type itemtype and pointers to the previous and next node. The previous node of the first node is nullptr and the next node of the last is nullptr. For the nodes inbetween the next points to the next node and the previous to the previous node. There is no tail node nor dummy node and it is not circular. The size is increment and decrement as nodes are deleted and inserted.

Non-Trivial Functions

Insert(position, value)

-b

Check if position is not valid

Return false

Check if first value in list

Create new node

sets p's value and pointers

if not first node set next node's prev to new

Increment size

Return true

Check if last value

Traverse through and find last

Create new node

Set last nodes values and pointers

Increment and return

Else

Find node in position desired

Set found node to next node of new node

Increment size

Return

-e

insert(pos)

Set count to count position

Check if head is nullptr

Insert in first position

Run through node while not nullptr

Add to counter

Check if value is less than current node or if last node

break

Increase pointer

If next pointer is null

Insert in last pos

Else

Insert in count

bool erase(pos)

if it is empty, return false

if you're erasing first value

Set temp pointer to head

Set head to next

If there is a node after temp

Set next node prev to null

Delete node and decrease size

If pos invalid

Return false

Traverse through the list to find the node in pos

If node’s next is not null

Link the previous and next node

Delete node

Decrease size and return

Else

Set the prev next to nullptr

Delete the node

End

Int remove(value)

B

Set counters to 0

Create traversing pointer

Loop through till nullptr

Check if values are equal

Move to next node

erase current pos

Decrease pos

Increase total no. deleted

Else

Move onto next node

increase pos

Return total deleted

E

sequence(other sequence)

B

Set the sizes equal

If head not nullptr

Set temp node ptr to other head

Create new node

Set the head to new node

Loop through temp till next not = nullptr

Set values for temp and new equal

Create new node

Set first new node\_next to newest node

Set newest node\_prev to older new node

Move new node and temp node to next node

Set the value for the last temp and new equal

Set the new next to nullptr

E

subsequence(seq1,seq2,seq3)

B

Check if valid sizes for seq1 and seq2

Set counter and temp variables

Loop through the first sequence(j)

Get val in pos j for seq1,2

Check if val are equal

Increase count

Loop through remaining no. of seq2 vals

Get val in pos j for seq1,2

Check if val are equal

count ++

If count is equal to size(they’re all equal)

Return j, which was pos of beginning of sequence

Else

Reset count

Return -1 if non found

E

interleave(seq1,seq2,result)

B

Create tempResult with result

erase all potential values in tempResult

check for aliasing

If either empty, set the tempResult to other

Find the bigger sequence and set big and big\_seq to it and small to other

Loop through the bigger sequence

If the value is less than the biggest value of small seq

Insert value from bigger seq

Insert value from small seq

Else

Insert value from bigger

Set result equal to tempresult(avoid aliasing)

e

TEST CASES

Sequence a;//checking default constructor

//checking insert works at end and end when end is beginning

a.insert(a.size(), 2);

a.insert(a.size(), 3);

a.insert(a.size(), 4);

a.insert(a.size(), 5);

a.insert(a.size(), 7);

//checking that inserts before previous ten

assert(a.insert(5)==3);

//checking insert into last works

assert(a.insert(9)==6);

string test1 = "2345579";

//checking each is in the correct order

for (int i = 0; i < a.size(); i++)

{

ItemType y;

a.get(i, y);

assert(y + 48 == test1[i]);

}

assert(a.remove(5)==2);

assert(a.set(0, 8));

assert(a.set(a.size()-1, 7));

//checking remove works and order is correct

//checking set functions work

string test2 = "83477";

for (int i = 0; i < a.size(); i++)

{

ItemType y;

a.get(i, y);

assert(y + 48 == test2[i]);

}

//testing out of bounds

ItemType g;

assert(a.insert(7, 4)==false);

assert(a.find(11)==-1);

assert(a.get(7, g)==false);

Sequence b;

Sequence c;

Sequence d;

//check it works for null;

b=c;

assert(b.get(0, g)==false);

b.insert(1);

b.insert(2);

b.insert(3);

//check interleave works for empty sequence

interleave(b, c, d);

for (int i = 0; i < d.size(); i++)

{

ItemType y;

ItemType z;

b.get(i, y);

d.get(i, z);

assert(y == z);

}

//checking copy constructor for empty sequence

Sequence e(c);

assert(e.size()==0);

Sequence seq1;

seq1.insert(0,30); // Inserting into the seqeuence

seq1.insert(1,21);

seq1.insert(2,63);

seq1.insert(3,42);

seq1.insert(4,17);

seq1.insert(5,63);

seq1.insert(6,17);

seq1.insert(7,29);

seq1.insert(8,8);

seq1.insert(9,32);

seq1.insert(10,63);

seq1.insert(11,42);

seq1.insert(12,17);

Sequence seq2;

seq2.insert(0, 63); // inserting into the second sequence

seq2.insert(1, 99);

seq2.insert(2, 17);

assert(subsequence(seq1, seq2)==-1);// tests subsequence works

assert(subsequence(seq1, seq1) == 0);// checking works for aliasing

seq2.set(1, 42);

//checking finds position and not the second position

assert(subsequence(seq1, seq2)==2);

Sequence p;

assert(p.empty()==true);//checking empty function works

assert(p.insert(0, 10)==true);//checking normal insert works for first node

assert(p.insert(0, 20)==true);//checking insert works inserting in first position

assert(p.size() == 2);//checking size functions work

ItemType x;

assert(p.get(0, x) && x == 20);//checking get function works and inserted correctly

assert(p.get(1, x) && x == 10);//checking insert and get again

assert (p.insert(15) == 0);//checking other insert function works correctly and inserted in correct spot

assert (p.insert(6, 4) == false);//checking can't insert out of bounds

//p=(15, 20, 10)

assert(p.remove(15)==1);// checking remove works

assert(p.get(0, x) && x == 20 && p.size()==2);//checking item has been removed and size decremented

assert(p.insert(10)==0);//checking new insert works

assert(p.insert(0, 10)==true);

assert(p.remove(10) == 3);//checking insert and remove again

assert(p.insert(6)==0);

assert(p.insert(7)==1);

p.insert(0, 15);

//p = (15, 6, 7, 20)

p.erase(0);//checking erase works

assert(p.get(0, x) && x == 6 && p.size()==3);

assert(p.erase(4)==false);//checking you can't erase past size

p.erase(1);//checking erase works for nodes in the middle

assert(p.get(1, x) && x == 20 && p.size()==2);

//checks insert again

assert(p.insert(10)==1);

assert(p.insert(5)==0);

assert(p.get(2, x) && x == 10 && p.size()==4);

assert(p.insert(25)==4 && p.size()==5);

assert(p.insert(25)==4 && p.size()==6);

assert(p.insert(21)==4 && p.size()==7);

//p = (5, 6, 10, 20, 21, 25, 25)

assert(p.remove(25)==2 && p.size()==5); // checks remove works for two variables and reduces size

p.erase(4);

p.erase(3);

assert(p.get(2, x) && x == 10 && p.size()==3);

//checking remaining elements

assert(p.remove(6)==1 && p.get(1, x) && x == 10 && p.size()==2);

//s = (5, 10)

//checking ordering of the sequence

assert(p.insert(2, 15) && p.get(2, x) && x == 15 && p.size()==3);

assert(p.insert(12)==2 && p.size()==4);

assert(p.insert(15)==3);

assert(p.insert(15)==3);

assert(p.remove(15)==3 && p.size()==3);

for (int i = 0; i < p.size(); i++)

{

ItemType y;

p.get(i, y);

cerr << y << " ";

}

cout << " ???=== 5 10 12" << endl;

//should print out (5, 10, 12)

//checking new inserts and the positions in lists

assert(p.set(2, 24)==true && p.size()==3);

assert(p.set(3, 31)==false);

p.set(0, 0);

p.insert(3, 10);

//checking find function works too

assert(p.find(10)==1);

assert(p.find(24)==2);

assert(p.find(31)==-1);

assert(p.find(0)==0);

for (int i = 0; i < p.size(); i++)

{

ItemType y;

p.get(i, y);

cerr << y << " ";

}

cout << " ???=== 0 10 24 10" << endl;

//should print out (0, 10, 24, 10)

//checking if swap works for empty

Sequence m;

m.swap(p);

assert(p.empty()==true && m.empty()==false);

assert(m.get(2, x) && x == 24 && m.size()==4);

//checking swap for aliasing

m.swap(m);

assert(m.get(2, x) && x == 24 && m.size()==4);

//checking inserts in the correct order

p.insert(0, 1);

p.insert(10);

p.insert(20);

p.insert(30);

p.insert(40);

//checking the assingment operator

p = m;

assert(p.get(2, x) && x == 24 && p.size()==4);

assert(m.get(2, x) && x == 24 && m.size()==4);

assert(p.get(5, x)==false && x == 24);

//checking the inverse of assignment operator

Sequence w = p;

assert(w.get(0, x) && x == 0 && w.size()==4);

assert(w.get(3, x) && x == 10 && w.size()==4);

//checking the interleave function

Sequence t;

t.insert(0, 1);

t.insert(1, 3);

t.insert(2, 5); //(1, 3, 5,)

Sequence k;

k.insert(0, 2);

k.insert(1, 4);

k.insert(2, 6); //(2, 4, 6)

Sequence q;

q.insert(0, 999);

interleave(t, k, q);

//checking that the interleaved function is correct

assert(q.get(0, x) && x == 1 && q.size()==6);

assert(q.get(1, x) && x == 2);

assert(q.get(2, x) && x == 3);

assert(q.get(3, x) && x == 4);

assert(q.get(4, x) && x == 5);

assert(q.get(5, x) && x == 6);

q.insert(7);

q.insert(8);

q.insert(9);

interleave(t, q, k);

assert(q.get(0, x) && x == 1 && q.size()==9);

assert(q.get(1, x) && x == 2);

assert(q.get(2, x) && x == 3);

assert(q.get(3, x) && x == 4);

assert(q.get(4, x) && x == 5);

assert(q.get(5, x) && x == 6);

assert(q.get(6, x) && x == 7);

assert(q.get(7, x) && x == 8);

assert(q.get(8, x) && x == 9);

//checking the order of the new sequence

string test = "113253456789";

for (int i = 0; i < k.size(); i++)

{

ItemType y;

k.get(i, y);

assert(y + 48 == test[i]);

}

Sequence allSev;

//cehcking all values are deleted

for(int j =0; j<10; j++)

{

allSev.insert(7);

}

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

assert(allSev.remove(7)==10);

assert(allSev.size()==0);