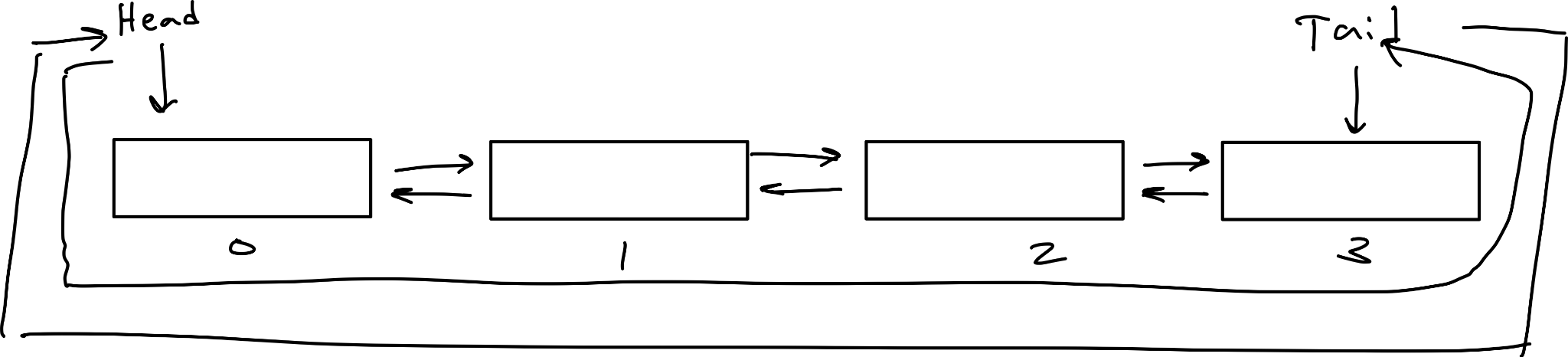
Drew Wan

305751876

Description:

It is a doubly linked list that is circular with no dummy node.



Pseudo Code:

**Insert Function (Taking in Position and value)**

Check invalid parameters:

Return -1

Use a newNode node set to the value

when inserted first:

if its empty:

Set head and tails to the newNode

if there is more than one:

Set the head and tails prev and next to the newNode

when inserted last:

Set the head and tails prev and next to the newNode

when inserted in the middle:

Create a temp node

Loop through the linked list until the position before the actual insert position (achieved with for loop)

Set the head to temp

Set the newNode and temps accordingly

Increase the size and return position

**Insert Function (Passing in only Value)**

Check if list is empty:

Use other insert function to insert at pos 0

Return

Repeatedly go through the list checking if value inserted is less than the values in the lists:

Increase position place

If its the tail and value imputed is greater, insert after and return the next position

Otherwise, insert at the position and return position

**Erase**

Type Bool and takes in position

Test for invalid positions or size:

Return false;

(

If at first position:

Set temp node (called killNode in my code) to head

Check if only one element in size:

Set head and tail to null

Otherwise:

Reassign head and tail to skip original head node.

Check if at the end of list:

Set temp node to tail

Set tail to the previous tail

Check if removing from middle:

Loop through to find until it finds the node to erase

Set the temp node and rearrange to point the prev and next skipping the one its pointing to

)

Delete the temp pointer and decrease the size and return true

**Remove**

Check if the size is not empty:

Repeatedly:

Check if the value in the list = the value wanting to remove:

If it is, erase at that position:

Increase counter

If not:

Move on to next position and next item in list

Check if tail’s value = value wanting to remove since while loop above only checks up to tail:

Erase the tail and increase counter

Return the amount removed

**Get**

Check for invalid parameters(pos less 0 and pos greater or equal to size):

Return false

Otherwise:

Repeatedly:

Move through the loop until the position

Copy value at that position to value passed through

Return true

**Set**

Check for invalid parameters(pos less 0 and pos greater or equal to size):

Return false

Otherwise:

Repeatedly:

Move through the loop until the position

Reverse of Get, we set the value passed through into the value already in the list

Return true

**Find**

Repeatedly:

Move through the list until position of value is found:

Return position

Increase position and move on in list

After loop, check tail:

If the value = value passed through:

Return the position

Otherwise return -1

**Swap**

Store head tail and size in a temp

Set head tail and size to other

Set the others to the temps

Basically (Swap the head and tail)

**Subsequence**

Repeatedly:

Go through seq1:

Go through seq2:

Get value at seq1

Get value at seq2

Check if not equal to each other:

Otherwise:

Increase size

If the size and count equal, return i or the index

If size of i is larger than seq1 - seq return -1

**concatReverse**

Repeatedly:

Loop backwards by starting at sequence size to 0

Get values in sequence 1 and insert into temp

Repeatedly:

Work backwards by starting at sequence size to 0.

Get values in sequence 2 and insert into temp after the reversed list from sequence 1

Set result to temp.

Test Cases

| //The tests were performed on a sequence of strings  Sequence a;  a.insert(0,"a"); //Checking inserts  assert(a.insert(1, "2") == 1);  assert(a.insert(2, "3") == 2); //checking with some asserts  a.insert(3, "4"); //using dump to check  assert(a.insert(4, "5") == 4);  assert(a.set(2, "6") == true); //Checking set  a.dump();  string x;  assert(a.get(0, x) && x == "a"); //Testing Get  assert(a.find("2") == 1); //testing find  Sequence s1;  s1.insert(0, "paratha");  s1.insert(0, "focaccia");  Sequence s2;  s2.insert(0, "roti");  s2.insert(0, "bruh");  s2.remove("bruh");  s1.swap(s2);  //testing insert swap and remove  assert(s1.size() == 1); //size should be 1 since swapped  assert(s1.find("roti") == 0); //roti found at pos 1  assert(s1.find("bruh") == -1); //bruh not in list  assert(s2.size() == 2); // size should switch  assert(s2.find("focaccia") == 0); //both of these should be found in swap  assert(s2.find("paratha") == 1);  s1.dump();  Sequence s3 = Sequence(a); //testing copy constructor  s2 = s1; //testing operator  a.erase(5); //testing erase from end  a.erase(0); // from front  a.erase(3); //from middle  a.dump(); //used dump to verify  Sequence b;  b.insert(0,"a");  b.insert(1, "b");  b.insert(2, "c");  b.insert(3, "d");  b.insert(4, "e");  b.erase(0); //retesting erase  assert(b.find("a") == -1); //testing find and verfying a is gone  b.insert(0,"a"); //testing insert and putting a back  assert(b.find("a") == 0); // verfying being put back  b.erase(1); //testing erase in the middle  assert(b.find("b") == -1);  b.insert(1,"b");  assert(b.find("b") == 1); // find at pos 1  b.erase(4); //testing erase in the end  assert(b.find("e") == -1);  b.dump();  Sequence c1;  c1.insert("sup"); //testing insert  c1.dump();  assert(c1.erase(0) == true); //testing erase again  //testing normal insert  c1.insert("1");  c1.insert("3");  c1.insert("5");  c1.insert("4");  c1.insert("2");  c1.insert("0");  cerr << "test c1" << endl;  c1.dump(); //testing via dump. should be 012345  assert(b.find("d") == 3); //testing find d should be at 3  b.remove("d"); //testing remove  b.dump();  assert(b.find("d")== -1); //return -1 cuz not there  b.remove("e"); //remove 3  assert(b.find("e") == -1); //return -1 cuz not there  Sequence a1;  a1.insert("1");  a1.insert("2");  a1.insert("3");  Sequence a2;  a2.insert("4");  a2.insert("5");  a2.insert("6");  Sequence a3;  //testing concatReverse via dump with an empty 3rd sequence  concatReverse(a1,a2,a3);  a3.dump();  Sequence b1;  b1.insert(0,"hi");  b1.insert(1, "bruh");  b1.insert(2, "hola");  b1.insert(3, "suppo");  Sequence b2;  b2.insert(0, "bruh");  b2.insert(1, "hola");  b2.insert(2, "suppo");  //testing subsequnce  assert(subsequence(b1, b2) == 1);  b2.dump();  Sequence d1;  d1.insert(0,"1");  d1.insert(1,"3"); //testing if the 3 comes first but the subsequence is after  d1.insert(2,"2");  d1.insert(3,"3");  d1.insert(4,"4");  d1.insert(5,"5");  Sequence d2;  d2.insert("3");  d2.insert("4");  d2.insert("5");  //testing subsequnce  cerr << subsequence(d1,d2) << endl;  assert(subsequence(d1,d2) == 3); //should start at position 3     //The tests were performed on a sequence of unsigned long  a.insert(0,1);  a.insert(1, 2);  a.insert(2, 3);  a.insert(3, 4);  a.insert(4, 5);  a.set(2, 6);  unsigned long x;  assert(a.get(0, x) && x == 1);  assert(a.find(2) == 1);  Sequence s1;  s1.insert(0, 1234);  s1.insert(0, 34);  Sequence s2;  s2.insert(0, 56);  s1.swap(s2);  assert(s1.size() == 1);  assert(s1.find(56) == 0);  assert(s1.find(235) == -1);  assert(s2.size() == 2);  assert(s2.find(34) == 0);  assert(s2.find(1234) == 1);  Sequence s3 = Sequence(a);  s2 = s1;  a.erase(5);  a.erase(0);  a.erase(3);  Sequence b;  b.insert(0,1);  b.insert(1, 2);  b.insert(2, 3);  b.insert(3, 4);  b.insert(4, 5);  b.erase(0);  assert(b.find(1) == -1);  b.insert(0,1);  b.erase(1);  assert(b.find(2) == -1);  b.insert(1,2);  b.erase(4);  assert(b.find(5) == -1);  b.dump();  Sequence c1;  c1.insert(1);  c1.dump();  assert(c1.erase(0) == true);  assert(b.find(4) == 3);  b.remove(4);  b.dump();  assert(b.find(4)== -1);  b.remove(5);  assert(b.find(5) == -1);  Sequence a1;  a1.insert(1);  a1.insert(2);  a1.insert(3);  Sequence a2;  a2.insert(4);  a2.insert(5);  a2.insert(6);  Sequence a3;  concatReverse(a1,a2,a3);  a3.dump();  Sequence b1;  b1.insert(0,0);  b1.insert(1, 1);  b1.insert(2, 2);  b1.insert(3, 3);  Sequence b2;  b2.insert(0, 1);  b2.insert(1, 2);  b2.insert(2, 3);  assert(subsequence(b1, b2) == 1);  b2.dump(); |
| --- |
| Sequence e; |
| assert(e.empty() == true); // testing if empty |

Same test cases above just in case it didn’t format or show up correctly

The tests were performed on a sequence of unsigned long

Sequence a;

a.insert(0,"a"); //Checking inserts

assert(a.insert(1, "2") == 1);

assert(a.insert(2, "3") == 2); //checking with some asserts

a.insert(3, "4"); //using dump to check

assert(a.insert(4, "5") == 4);

assert(a.set(2, "6") == **true**); //Checking set

a.dump();

string x;

assert(a.get(0, x) && x == "a"); //Testing Get

assert(a.find("2") == 1); //testing find

Sequence s1;

s1.insert(0, "paratha");

s1.insert(0, "focaccia");

Sequence s2;

s2.insert(0, "roti");

s2.insert(0, "bruh");

s2.remove("bruh");

s1.swap(s2);

//testing insert swap and remove

assert(s1.size() == 1); //size should be 1 since swapped

assert(s1.find("roti") == 0); //roti found at pos 1

assert(s1.find("bruh") == -1); //bruh not in list

assert(s2.size() == 2); // size should switch

assert(s2.find("focaccia") == 0); //both of these should be found in swap

assert(s2.find("paratha") == 1);

s1.dump();

Sequence s3 = Sequence(a); //testing copy constructor

s2 = s1; //testing operator

a.erase(5); //testing erase from end

a.erase(0); // from front

a.erase(3); //from middle

a.dump(); //used dump to verify

Sequence b;

b.insert(0,"a");

b.insert(1, "b");

b.insert(2, "c");

b.insert(3, "d");

b.insert(4, "e");

b.erase(0); //retesting erase

assert(b.find("a") == -1); //testing find and verfying a is gone

b.insert(0,"a"); //testing insert and putting a back

assert(b.find("a") == 0); // verfying being put back

b.erase(1); //testing erase in the middle

assert(b.find("b") == -1);

b.insert(1,"b");

assert(b.find("b") == 1); // find at pos 1

b.erase(4); //testing erase in the end

assert(b.find("e") == -1);

b.dump();

Sequence c1;

c1.insert("sup"); //testing insert

c1.dump();

assert(c1.erase(0) == **true**); //testing erase again

//testing normal insert

c1.insert("1");

c1.insert("3");

c1.insert("5");

c1.insert("4");

c1.insert("2");

c1.insert("0");

cerr << "test c1" << endl;

c1.dump(); //testing via dump. should be 012345

assert(b.find("d") == 3); //testing find d should be at 3

b.remove("d"); //testing remove

b.dump();

assert(b.find("d")== -1); //return -1 cuz not there

b.remove("e"); //remove 3

assert(b.find("e") == -1); //return -1 cuz not there

Sequence a1;

a1.insert("1");

a1.insert("2");

a1.insert("3");

Sequence a2;

a2.insert("4");

a2.insert("5");

a2.insert("6");

Sequence a3;

//testing concatReverse via dump with an empty 3rd sequence

concatReverse(a1,a2,a3);

a3.dump();

Sequence b1;

b1.insert(0,"hi");

b1.insert(1, "bruh");

b1.insert(2, "hola");

b1.insert(3, "suppo");

Sequence b2;

b2.insert(0, "bruh");

b2.insert(1, "hola");

b2.insert(2, "suppo");

//testing subsequnce

assert(subsequence(b1, b2) == 1);

b2.dump();

Sequence d1;

d1.insert(0,"1");

d1.insert(1,"3"); //testing if the 3 comes first but the subsequence is after

d1.insert(2,"2");

d1.insert(3,"3");

d1.insert(4,"4");

d1.insert(5,"5");

Sequence d2;

d2.insert("3");

d2.insert("4");

d2.insert("5");

//testing subsequnce

cerr << subsequence(d1,d2) << endl;

assert(subsequence(d1,d2) == 3); //should start at position 3

The tests were performed on a sequence of unsigned long

a.insert(0,1);

a.insert(1, 2);

a.insert(2, 3);

a.insert(3, 4);

a.insert(4, 5);

a.set(2, 6);

**unsigned** **long** x;

assert(a.get(0, x) && x == 1);

assert(a.find(2) == 1);

Sequence s1;

s1.insert(0, 1234);

s1.insert(0, 34);

Sequence s2;

s2.insert(0, 56);

s1.swap(s2);

assert(s1.size() == 1);

assert(s1.find(56) == 0);

assert(s1.find(235) == -1);

assert(s2.size() == 2);

assert(s2.find(34) == 0);

assert(s2.find(1234) == 1);

Sequence s3 = Sequence(a);

s2 = s1;

a.erase(5);

a.erase(0);

a.erase(3);

Sequence b;

b.insert(0,1);

b.insert(1, 2);

b.insert(2, 3);

b.insert(3, 4);

b.insert(4, 5);

b.erase(0);

assert(b.find(1) == -1);

b.insert(0,1);

b.erase(1);

assert(b.find(2) == -1);

b.insert(1,2);

b.erase(4);

assert(b.find(5) == -1);

b.dump();

Sequence c1;

c1.insert(1);

c1.dump();

assert(c1.erase(0) == **true**);

assert(b.find(4) == 3);

b.remove(4);

b.dump();

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

b.remove(5);

assert(b.find(5) == -1);

Sequence a1;

a1.insert(1);

a1.insert(2);

a1.insert(3);

Sequence a2;

a2.insert(4);

a2.insert(5);

a2.insert(6);

Sequence a3;

concatReverse(a1,a2,a3);

a3.dump();

Sequence b1;

b1.insert(0,0);

b1.insert(1, 1);

b1.insert(2, 2);

b1.insert(3, 3);

Sequence b2;

b2.insert(0, 1);

b2.insert(1, 2);

b2.insert(2, 3);

assert(subsequence(b1, b2) == 1);

b2.dump();