Ram Yadav

Cis 32:

I have designed a circular link list

in struct, three different data member (vale, next pointer, and previous pointer)

size of the linked list

and head that points each own next pointer and pervious pointer

So, I have head, and next node starts at index(position) Zero and size 1.

Each nodes has three different value namely: value (ItemType), next(point to next pointer) and previous(point to previous node).

Default constructors:

created a head node, its next and previous pointer points itself(head) with size 0

start with zero index

Copy constructors:

crated head node

increment loop by pointing to next pointers

increments loops until and copied until it found head

inside loop created new node each time it loop, where value, previous and next node from old node copy to new node

assignment operator:

if sequence equal to itself then return itself

else, temp copied other objects and swap with current objects, after swap current objects

retained the value of old and return current objects

Destructor:

go through loop as longest head’s next doesn’t point itself

as removed nodes, interchange its data with its adjacent nodes, finally delete node

once every nodes deleted delete head

Empty:

if size is zero, return true else return false

Size:

return size of sequence

insert:

check if position is negative or more than size

check if the po is greater than 0 and less than size

assigned head to new node

go through loop until index or pos is less than its size

create new node

iterate through each node

if the given position is equal to index I (which is position in other words)

exchange the value

when finished increase the size by one

points to next node

or if pos is equal to size

create new node and exchange the next 0r previous pointers with head

at the end incrate the size

insert:

if the list is empty

create new node and exchange next and previous pointers with the head

insert value into the newNode and finally call insert function to insert the value at given position

else, go through loop and find if given value is less or equal to nodes value

if yes, assigned position with the index I otherwise assigned position with its size

finally insert the position and value

and return the position as question has asked

erase:

Pos must be greater than 0 and less than size

going through the loop as loop iteration, if found target node, exchange the pointers with the one before deleting node with one after deleting node

finally delete node

and decrease the size of the link list

remove:

created node, store the value head to the created node

iteration as pointing pointers to the next node

if given value is equal to node value, delete that node

before delete node exchange the next and previous pointers value with targeted consecutives node

finally delete target node

increase the removed size and also size of the link list

finally returned removed item

get:

pos must be greater than 0 and less than size()

for size n position must be n-1

iterating through the loops

if given position is equal to index, get the value and return true

set:

pos must be greater than 0 and less than size()

for size n position must be n-1

iterating through the loops

if given position is equal to index, set the value return true

find:

starting with the next node at positon 0, iterate through all nodes in the link list until last node

as iterate through list and if it is last item then its next pointer must equal to head and breaks

if found the item, then store its position and return position

otherwise return -1

swap:

swap the size and head of current sequence with the other sequence

subsequence:

if the sequence is empty or sequence2 size is grater than sequence1 size then return -2

else, must loop until index of seq1 until size-1 of sequence and when found size of sequence2 must be zero by then

get the value of seq1 and aeq2

compare it and store the value

increase index of seq1 and seq2 and decrease the size of sequence 2

else, keep increasing index1 and reset index2 again zero and its size2 to its actual size

at the end return subsequence

interleave:

if seq1 and seq2 is empty then break out the loop

if seq1 size is zero then result will be just seq2

if seq2 size is 0 then result will be seq1

otherwise, position of seq1 is equal to its size-1 or position of sequence2 is equal to its size-1

using if statement for both sequence, store its values inside result sequence one after other

getting value of seq1, inserting its value and keeps increasing its index and position

getting value of seq2, inserting its value and keeps increasing its index and position

#include "Sequence.h"

#include <string>

#include <iostream>

#include <cassert>

#include <type\_traits>

using namespace std;

#define CHECKTYPE(f, t) { auto p = (t)(f); (void)p; }

static\_assert(std::is\_default\_constructible<Sequence>::value,

              "Sequence must be default-constructible.");

static\_assert(std::is\_copy\_constructible<Sequence>::value,

              "Sequence must be copy-constructible.");

void thisFunctionWillNeverBeCalled()

{

    CHECKTYPE(&Sequence::operator=,  Sequence& (Sequence::\*)(const ItemType&));

    CHECKTYPE(&Sequence::empty,      bool (Sequence::\*)() const);

    CHECKTYPE(&Sequence::size,       int  (Sequence::\*)() const);

    CHECKTYPE(&Sequence::insert,     bool (Sequence::\*)(int, const ItemType&));

    CHECKTYPE(&Sequence::insert,     int (Sequence::\*)(const ItemType&));

    CHECKTYPE(&Sequence::erase,      bool (Sequence::\*)(int));

    CHECKTYPE(&Sequence::remove,     int  (Sequence::\*)(const ItemType&));

    CHECKTYPE(&Sequence::get, bool (Sequence::\*)(int, ItemType&) const);

    CHECKTYPE(&Sequence::set, bool (Sequence::\*)(int, const ItemType&));

    CHECKTYPE(&Sequence::find,       int  (Sequence::\*)(const ItemType&) const);

    CHECKTYPE(&Sequence::swap,       void (Sequence::\*)(Sequence&));

    CHECKTYPE(subsequence, int  (\*)(const Sequence&, const Sequence&));

    CHECKTYPE(interleave,  void (\*)(const Sequence&, const Sequence&, Sequence&));

}

void test()

{

    // default constructor

    Sequence ok;

    // For an empty sequence:

    assert(ok.size() == 0);             // test size

    assert(ok.empty());                 // test empty

    assert(ok.remove("paratha") == 0);  // nothing to remove

    //INSERT

    Sequence s; // created a sequence object

    assert(s.insert(0, "lavash")); //insert value(string) at position 0

    assert(s.insert(0, "tortilla"));//insert another value(string) at position zero

    //SIZE

    assert(s.size() == 2); // makin sure size is two

    // GET

    //the value at position one went up by one

    ItemType x = "injera"; //define ItemType

    assert(s.get(0, x)  &&  x == "tortilla"); //after inserted making sure tortilla is at position 0

    assert(s.get(1, x)  &&  x == "lavash");// making sure lavash is at position 1

    Sequence k; // created another a sequence object

    assert(k.insert(0, "one")); //insert value(string) at position 0

    assert(k.insert(0, "two"));//insert another value(string) at position zero

    //SIZE

    assert(k.size() == 2); // makin sure size is two

    //GET

    ItemType  y = "sdfdsf"; //define ItemType

    assert(k.get(0, y)  &&  y == "two"); //after inserted making sure two is at position 0

    assert(k.get(1, y)  &&  y == "one");// making sure one is at position 1

    //FIND

    assert(k.find("one") == 1); //one is at position 1

    assert(k.find("two") == 0); // two is at position 0

    //SWAP

    k.swap(s); // swaping two function

    ItemType m = "injera"; //define ItemType

    assert(s.get(0, m)  &&  m == "two"); //after swaping s get k value at the position 0

    assert(s.get(1, m)  &&  m == "one");// after swaping s get k value at position 1

    //EMPTY

    assert(k.empty() == false); // checking if emtpy

    assert(s.empty() == false); // checking if empty

    //SET

    k.set(0, "three");

    assert(k.get(0, m)  &&  m == "three"); //after setting the value at position 0, changed value to three from two

    //REMOVE

    assert(k.remove("three") == 1); // removing the value

    //ERASE

    assert(k.erase(2) == false); // since k doesn't have position 2, returns false

    assert(s.erase(1) == true); // since s objects has items at postion 1, removed the item

    //INSERT

    assert(s.insert("zeo") == 1); // value is greater than item in sequence so it return 1

    //SUBSEQUENCE

    //making another sequence

    Sequence a;

    assert(a.insert(0, "a")); //insert value(string) at position 0

    assert(a.insert(1, "b"));//insert another value(string) at position 1

    assert(a.insert(2, "c")); //insert value(string) at position 2

    assert(a.insert(3, "d"));//insert another value(string) at position 3

    Sequence b;

    assert(b.insert(0, "c")); //insert value(string) at position 0

    assert(b.insert(1, "d"));//insert another value(string) at position 1

    assert(subsequence(a, b) == 2); // subsequence starts at position 2

    //INTERLEAVE

    Sequence result;

    interleave(a, b, result);

    ItemType z = "ninja";

    assert(result.get(0, z) && z == "a");//a from first sequence

    assert(result.get(1, z) && z == "c");//c from second sequence

    assert(result.get(2, z) && z == "b");// b from first sequence

    assert(result.get(3, z) && z == "d");//d from second sequence

    assert(result.get(4, z) && z == "c");//c from first sequence

    assert(result.get(5, z) && z == "d");//d from second sequence

}

int main()

{

    thisFunctionWillNeverBeCalled();

    test();

    cout << "Passed all tests" << endl;

}