Daxuan Shu

204853061

CS 32

Winter 2017

**Project 2 Report**

In this project. My double-linked list begins with a dummy node “m\_head” and end with the last node in the sequence. Each node has two pointers point to the adjacent node except the head and the last node. The m\_prev pointer of the first node(m\_head) is set to nullptr and the “m\_next” pointer of the last node (I did not use “tail” ) is set to nullptr. Thus, the list is not circular.

inline functions:

bool empty() const;

return (size() == 0);

int size() const;

return m\_size;

Constructor

Sequence::Sequence():m\_size(0)

{

    m\_head = nullptr;

}

Destructor

Sequence::~Sequence()

{

    Node\* p;

    p = m\_head;

        while(p != nullptr)

            {

                Node\* n = p->m\_next;

                delete p;

                p = n;

            }

}

Copy constructor

Sequence::Sequence(const Sequence& other): m\_size(other.m\_size)

{

    m\_head = new Node;

    m\_head->m\_value = other.m\_head->m\_value;

    m\_head->m\_prev = nullptr;

    Node \*m = m\_head;

    Node \*n = other.m\_head->m\_next;

    while (n != nullptr)

    {

        m->m\_next = new Node;

        m->m\_next->m\_value = n->m\_value;

        m->m\_next->m\_prev = n;

        n = n->m\_next;

        m = m->m\_next;

    }

    m->m\_next = nullptr;

   }

Assignment operator

Sequence& Sequence::operator=(const Sequence &src)

{

    if (this == &src)

        return \*this;

    Sequence temp = src;

    swap(temp);

    return \*this;

}

**Pseudocode:**

private member function

void Sequence::uncheckedInsert(int pos, const ItemType &value)

Create a new Node n for value.

if sequence is empty, make this node to be the head.

if pos == 0 (customer wants to insert in the first position),

then replace the head with Node n. Move all other Nodes one after.

if want to add at the rear

else (in the middle)

bool Sequence::erase(int pos)

if pos is invalid, return false

if erase head, make m\_head point to next node.

delete

connect the two Nodes before and after pos

delete Node at pos.

size--

int Sequence::remove(const ItemType& value)

int count = 0;

if find(value) = -1 ,return -1.

if find(value >=0 ),

erase( pos = find(value))

increment count

keep searching

return count

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

search from head to end.

void Sequence::swap(Sequence& other)

swap size and head

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

create an instance temp\_result

for the nums of Nodes that both seq1 and seq2 are not end.

in a for loop to loop the size of the smaller sequence times from 0 to small size.

in each loop, get seq1 and seq2’s value and insert into temp\_result.

Note seq1’s position in 0,2,4,6,8.…

seq2’s position in 1,3,5,7,9,…

for the remaining big size, for loop remaining times to insert value in to temp\_result.

result.swap(temp\_result);

Test Cases

 Sequence s;

    assert(s.insert(0, "lavash"));

    assert(s.insert(0, "tortilla"));

    assert(s.size() == 2);

    assert(s.insert(2, "apple"));

    assert(s.size() == 3);

    ItemType x = "injera";

    assert(s.get(0, x)  &&  x == "tortilla");

    assert(s.get(1, x)  &&  x == "lavash");

    assert(s.insert(1, "banana") && s.size() == 4);

    ItemType y = "trump";

    assert(s.get(2, y) && y == "lavash");

    assert(s.insert(0, "is header?") && s.size() == 5);

    assert(!s.insert(6, "out of size"));

    assert(s.erase(0) && s.size() == 4);

    assert(s.insert("cat") == 0);

    assert(s.erase(1));

    assert(s.insert("hahaha") == 2);

    assert(s.insert("banana") == 0);

    assert(s.insert("banana") == 0);

    assert(s.size() == 7);

    assert(s.remove("banana") == 3);

    assert(s.set(2, "banana"));

    assert(s.find("apple") == 3);

    assert(s.find("cat") == 0);

    assert(s.find("trump") == -1);

    Sequence s2;

    s2.insert(0, "annie");

    s2.insert(1, "bot");

    s2.insert("carot");

    s.swap(s2);

    assert(s.size() == 3);

    Sequence s4;

Sequence s3;

    interleave(s4, s2, s3);