1. This doubly-linked list is a regular linked list with a head pointer and a tail pointer. It does not contain a dummy node and is not circular either. In every node, it contains an ItemType variable which stores value, a pointer to the previous node, and a pointer to the next node.

Typical:

Value: 300

prev

next

Value: 100

prev

next

Value: 200

prev

next

Empty:

1. //destructor

Sequence::~Sequence()

Create node p pointing to head

While p isn’t a nullptr

Create node n point to p->next

Delete p

Assign n to p

//copy constructor

Sequence::Sequence(const Sequence& s)

Point head and tail to nullptr

Set size to s’s size

Return if s is empty

Create node p pointing to s’s head

Create new node n

Copy and set head pointer

Create node temp pointing to n

P points to its next

While p isn’t nullptr

Copy the nodes and set p and temp to their next

Set tail pointer

//assigning operator

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

Return \*this if s and this are the same

Delete all the original nodes in this sequence

/\*this part of code is the same as copy constructor\*/

Return \*this

//insert

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

Return -1 if pos is not a valid number

If pos is 0, add value to front

Else if pos is equal to size, add value to rear

Else add to specific position

Increase the size by 1

Return pos

Int Sequence::insert(const ItemType& value)

Create node p, point it to head

Create integer pos, set it to 0

While p doesn’t point to nullptr, and p’s value is less than value

Point p to its next

Increase pos by 1

Return pass pos and value to insert()

//erase

Bool Sequence::erase(int pos)

Return false if pos is invalid number

If pos is equal to zero

Create node p, point it to head

Point head to p’s next

Point tail to nullptr if head is nullptr

Else point head’s prev to nullptr

Delete p

Else if pos is equal to size – 1

Create node p, point it to tail

Point p’s prev’s next to nullptr

Point tail to p’s prev

Delete p

Else

Create node p, point it to head

Find the node at that position

Point p’s prev’s next to p’s next

Point p’s next’s prev to p’s prev

Delete p

Decrease size by 1

Return true

//remove

Int Sequence::remove(const ItemType& value)

Create integer count, set to 0

Create node p, point it to head

Go through every node in sequence

Break if p is nullptr

Create node temp, point it to p’s next

Erase if p’s value is equal to value, increase count by 1

Set p to temp

Return count

//get

Bool Sequence::get(int pos, ItemType& value)

Return false if pos is invalid number

Set value to head’s value if pos is equal to 0

Set value to tail’s value if pos is equal to size – 1

Else, go through every node and set value to value at that position

Return true

//set

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

Return false if pos is invalid number

Set head’s value to value if pos is equal to 0

Set tail’s value to value if pos is equal to size – 1

Else, go through every node and set value at that position to value

Return true

//swap

Void Sequence::swap(Sequence& other)

Create sequence temp, copying other

Assign \*this to other

Assign temp to \*this

//add to front

void Sequence::addtofront(const ItemType& value)

Create new node n

Set n’s value, prev and next

Set head’s prev to n if it isn’t nullptr

Set head to n

Set tail to n if the sequence is empty

//add to rear

void Sequence::addtorear(const ItemType& value)

create new node n

set n’s value, prev and next

set tail’s next to n

set tail to n

//add item

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

create new node n and set its value

go through the sequence and find the node at that position

set n’s next to p

set n’s prev to p’s prev

set p’s prev’s next to n

set p’s prev to n

//subsequence

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

return -1 if seq1’s size is less than seq2’s or seq2 is empty

create ItmeType tem1 and temp2, integer i

go through every node in seq1

get seq1’s node at i

get seq2’s node at 0

comparing following node if they are identical

once find the difference, break the loop

return i if all nodes in seq2 have been examined

return -1

//interleave

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

create sequence sq1, sq2, and rs by copying seq1, seq2, and result

delete all the nodes in rs

set result to rs and return if sq1 and sq2 are empty

interleave nodes in sq1 and sq2 to rs until one sequence runs out of its nodes

copying the rest of nodes in that larger sequence to rs

set result to rs

1. //tests were performed on a sequence of unsigned long

//default constructor

Sequence a;

//test empty()

assert(a.empty() == 1);

//test get() and insert()

a.insert(10);

a.insert(30);

a.insert(20);

a.insert(0, 0);

a.insert(4, 50);

a.insert(4, 40);

ItemType x = 0;

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

assert(a.get(1, x) && x == 10);

assert(a.get(2, x) && x == 20);

assert(a.get(3, x) && x == 30);

assert(a.get(4, x) && x == 40);

assert(a.get(5, x) && x == 50);

//test copy constructor

Sequence b = a;

assert(b.get(4, x) && x == 40); //now b is the same as a

//test assigning operator

Sequence c;

c = a;

assert(c.get(4, x) && x == 40); //c is the same as a

//test erase()

assert(a.erase(0) && a.get(0, x) && x == 10);

assert(a.erase(4) && a.get(3, x) && x == 40);

assert(a.erase(2) && a.get(1, x) && x == 20); //now a is left with 10, 20, 40

//test remove

Sequence d;

for (int i = 0; i < 4; i++)

d.insert(10);

d.insert(20); //now d is sequence with 10, 10, 10, 10, 20

int n = d.remove(10);

assert(n == 4 && d.get(0, x) && x == 20); //removing four 10

n = d.remove(3);

assert(n == 0); //there is no 3 in the sequence, thus n is zero

//test set()

assert(a.set(0, 100) && a.get(0, x) && x == 100);

assert(a.set(1, 200) && a.get(1, x) && x == 200);

assert(a.set(2, 400) && a.get(2, x) && x == 400);

assert(!(a.set(3, 300))); //position 3 is invalid since there are only three members in a

//test find()

int f = a.find(100);

assert(f == 0); //100 is ta position 0 in a

f = a.find(1000);

assert(f == -1); //there is no 100 in a

//test swap()

Sequence e;

e.insert(10);

e.insert(20);

e.insert(40); //e is sequence 10, 20, 400

e.swap(a);

assert(e.get(2, x) && x == 400); //now e is 100, 200, 400

assert(a.get(2, x) && x == 40); //a is 10, 20, 40

//test subsequence()

Sequence sq1, sq2, sq3;

for (int i = 1; i < 6; i++)

sq1.insert(i); //sq1 is 1, 2, 3, 4, 5

for (int i = 3; i < 5; i++)

sq2.insert(i); //sq2 is 3, 4

assert(subsequence(sq1, sq2) == 2); //starting at position 2, sq2 becomes sq1’s subsequence

//test interleave

interleave(sq1, sq2, sq1); //seq1 and result refer to same sequence

assert(sq1.get(0, x) && x == 1);

assert(sq1.get(1, x) && x == 3);

assert(sq1.get(2, x) && x == 2);

assert(sq1.get(3, x) && x == 4); //sq1 now becomes 1 3 2 4 3 4 5