**Description of the the Design of the Doubly-Linked List Implementation**

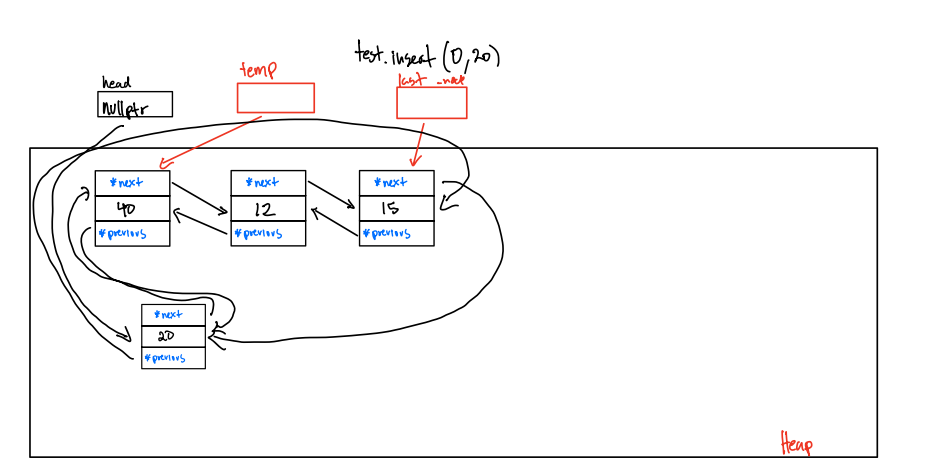
When a sequence class is constructed, head, a pointer to node, is created and is set to nullptr. Once the insert function is called, it checks for three cases. If the head pointer is pointing to a null pointer and is of size 0, the function would create a new node, assign it a value, and point the next and previous pointers to the newNode as this is the only node. I designed the program so that it was circular, and the tail pointer could be found by calling the previous pointer of the node.

The second case is when the program inserts a node at the end of the node. The program finds the end of the list by calling previous on head, and the program would swap the pointers to make the newNode next pointer point to the first node and make the firstNode previous pointer point to the newNode.

The third case is if the position we are trying to insert is in the middle of the linked list. The program would first find the pointer’s position through a loop and swap the pointers accordingly.

Finally, if we are trying to insert at the beginning of a linked list that is not empty, we would assign create a new node. The next and previous pointers would be swapped so that the newly created node would would fit into the circular doubly-linked list.

Figure 1: Example of diagram I drew while implementing insert at beginning of sequence



**Pseudocode**

Sequence::~Sequence()

If m\_size is not 0:

delete every item in the sequence

Set head to null pointer and m\_size to 0

Sequence::Sequence(const Sequence& other)

If the other linked list is empty, set the new sequence as empty too

Otherwise swap the values:

Set head to null pointer and m\_size to 0

Create Nodes for the current Sequence

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

If statement to prevent aliasing

Copy the values into temp using the copy constructor

Swap the values with temp

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

If position < 0, or position > size, or size == DEFAULT\_MAX\_ITEMS

return -1

Call unchecked Insert at position pos with value;

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

Create a new node

If the head pointer is pointing to a null pointer and is of size 0, execute (1). This means that the linked list is empty:

(1)

Assign First Node

If the position we are trying to insert in is equal to m\_size, execute (2). This means that we are inserting at the end of the list.

(2)

Find the end of the list

Change the pointers

If the position we are trying to insert is in the middle of the linked list, execute (3)

(3)

Find the previous pointer's position

Swap the pointers

If we are trying to inert at the beginning of a linked list that is not empty, execute (4)

(4)

Swap pointers

Increment m\_size by 1

int Sequence::insert(const ItemType& value)

Case 1

If linked list is empty

insert value at 0

return pos

Case 2

If linked list is not empty

Traverse through linked list

if value of temp == value

break

increment by 1

insert value at pos

return pos

bool Sequence::erase(int pos)

If position is less than 0 and position is equal to or greater than size or the size == 0 or the head is a nullptr

return false

If pos == 0 and head ==nullptr

return false

//Case 1: Erasing last node

If pos == 0 && m\_size == 1

erasing the last node and set the head to nullptr

//Case 2: Erase the node at position 0 if the size is greater than 0

else If pos == 0 && m\_size > 0

erase the node at the beginning of the list and connect the pointers accordingly

//Case 3: Erasing the node at the end of the linked list

else if (pos == end of the list and m\_size != 0

erase the node at the beginning of the list and swap the previous and next pointers

//Case 4: Erasing the node in the middle of the list

find the previous pointer’s position;

Swap the before pointer and after pointer;

delete the current pointer;

int Sequence::remove(const ItemType& value)

If (find(value) == -1

return 0;

int counter 0

loop through the sequence until value does not equal -1

erase the value passed into function

increment counter by 1

return the counter

bool Sequence::get(int pos, ItemType& value) const

if pos is less than 0 or pos is greater than or equal to size

return false

Node\* temp = head

Loop through the sequence until value found

Let temp point to the next value

Assign temp to value variable

return true

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

If pos is less than 0 or pos is greater than or equal to size

return false

Node\* temp = head

Loop through the sequence until value found

Let temp point to the next value

assign temp->value the value passed in by the function

return true

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

If head is a nullptr

return -1

Otherwise, loop through the list

If the linked list’s value == value

return p

Otherwise, go to next node

return -1

void Sequence::swap(Sequence& other)

Temporarily store the pointers of head to temp\_head\_this

Swap the actual heads

Temporarily store the other size to temp\_size

Swap the sizes

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

if the seq1 size >= seq2 size or seq size > 0 or seq2 size > 0

loop throught sequence 1

Ensure that the size of sequence 2 is bigger than size of sequence 1 to prevent undefined behavior.l If so, break

get i from sequence 1 and assign to item1

get 0 from sequence 2 ans assign to item2

If (item1 == item2)

create another loop

get i + k from sequence 1 and assign it to item1

get k from sequence 2 and assign it to item2

If item 1 != item2

break

otherwise, check the next item

If the iteration gets to the end, we know that they are equal, therefore, we can return i.

If no such k exists or if seq2 is empty, the function returns -1

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

If seq1 and seq2 are empty,

then when this function returns, result must be empty

Sequence temp;

If seq1 is empty and seq2 size is greater than 0,

loop through the seq2

reverse the order of seq2 and assign it to temp

If seq2 is empty and seq1 size is greater than 0

loop through the seq1

reverse the order of seq1 and assign it to temp

Otherwise, concatenate Reverse the sequence

loop through the sequence

reverse the order of seq1 and assign it to temp

loop through the seq1

reverse the order of seq1 and assign it to temp

result = temp

**Test Cases**

//Test Case 1: Subsequence (Same Sequence)

Sequence s1;

s1.insert(0, 30);

s1.insert(1, 21);

s1.insert(2, 63);

s1.insert(3, 42);

s1.insert(4, 17);

s1.insert(5, 63);

s1.insert(6, 17);

s1.insert(7, 29);

s1.insert(8, 8);

s1.insert(9, 32);

Sequence s2;

s2.insert(0, 30);

s2.insert(1, 21);

s2.insert(2, 63);

s2.insert(3, 42);

s2.insert(4, 17);

s2.insert(5, 63);

s2.insert(6, 17);

s2.insert(7, 29);

s2.insert(8, 8);

s2.insert(9, 32);

assert(subsequence(s1, s2) == 0);

//Test Case 2: ConcatReverse

Sequence test;

test.insert(0, 40);

test.insert(1, 12);

test.insert(2, 15);

test.insert(3, 17);

test.insert(4, 20);

test.insert(40);

test.insert(15);

test.insert(20);

test.remove(100);

cout << endl;

Sequence test1;

test1.insert(0, 504);

test1.insert(1, 24);

test1.insert(2, 89);

test1.insert(3, 18);

test1.insert(4, 100);

test1.insert(30);

test1.insert(15);

test1.insert(20);

Sequence r1;

concatReverse(test, test1, r1);

ItemType t;

//Test

assert(test.get(0, t) && t == 40);

assert(test.get(1, t) && t == 40);

assert(test.get(2, t) && t == 12);

assert(test.get(3, t) && t == 15);

assert(test.get(4, t) && t == 15);

assert(test.get(5, t) && t == 17);

assert(test.get(6, t) && t == 20);

assert(test.get(7, t) && t == 20);

//Test1

assert(test1.get(0, t) && t == 504);

assert(test1.get(1, t) && t == 24);

assert(test1.get(2, t) && t == 89);

assert(test1.get(3, t) && t == 18);

assert(test1.get(4, t) && t == 100);

assert(test1.get(5, t) && t == 30);

assert(test1.get(6, t) && t == 15);

assert(test1.get(7, t) && t == 20);

//r1

assert(r1.get(0, t) && t == 20);

assert(r1.get(1, t) && t == 20);

assert(r1.get(2, t) && t == 17);

assert(r1.get(3, t) && t == 15);

assert(r1.get(4, t) && t == 15);

assert(r1.get(5, t) && t == 12);

assert(r1.get(6, t) && t == 40);

assert(r1.get(7, t) && t == 40);

assert(r1.get(8, t) && t == 20);

assert(r1.get(9, t) && t == 15);

assert(r1.get(10, t) && t == 30);

assert(r1.get(11, t) && t == 100);

assert(r1.get(12, t) && t == 18);

assert(r1.get(13, t) && t == 89);

assert(r1.get(14, t) && t == 24);

assert(r1.get(15, t) && t == 504);

cout << "All test cases passed!";