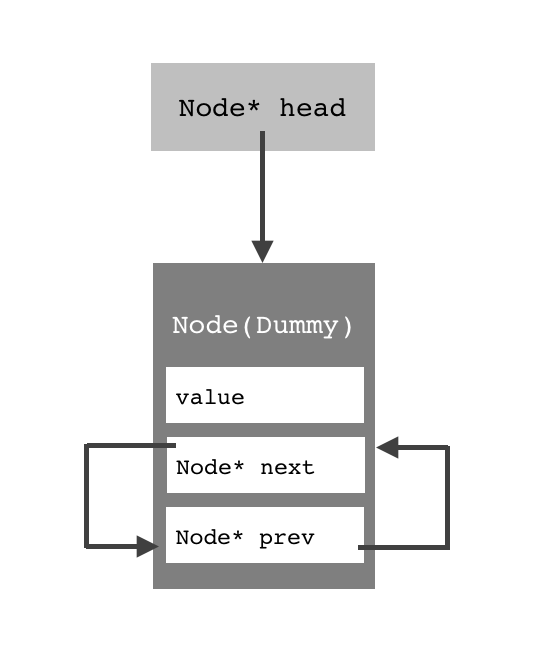
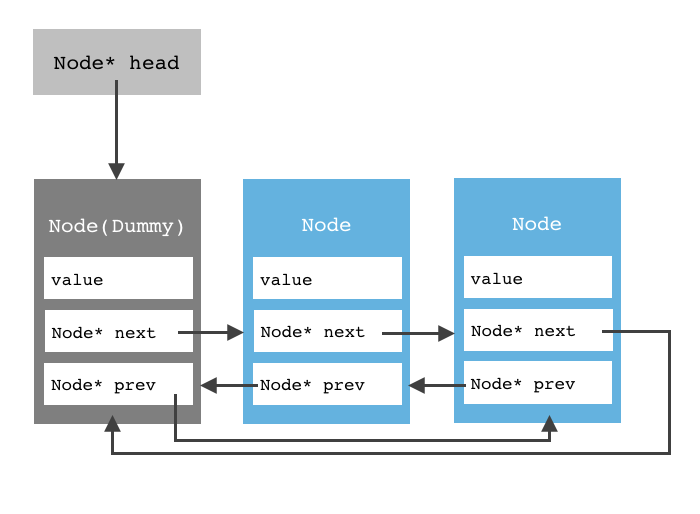
# Linked List Design

I implemented a circular doubly-linked list with a dummy node as head. Below is a schematic of an empty list:

Each node struct has 3 member variables: its value being stored, a pointer to the next node in the list, and a pointer to the previous node. An empty list consists of the dummy node, whose pointers all point to itself, and a head pointer pointing to the dummy node in order to identify the list’s position in the heap. After inserting some nodes, the list eventually becomes:

Note that the head pointer always points to the dummy node, whose prev pointer points to the last node in the list. The next pointer of the last node points to the dummy node, making the list circular.

Besides the linked list itself, each sequence instance also holds an integer member variable which stores its current size.

# Algorithms

Sequence::Sequence()

set size to 0;

point the head to a new node;

let the node’s prev and next pointers point to itself;

Sequence::Sequence(const Sequence& other)(the copy constructor)

set size to 0;

point the head to a new node;

let the node’s prev and next pointers point to itself;

repeatedly:

get the value for each node of the other sequence;

insert new nodes to self, with that value;

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

return self if rhs & lhs of the operator has the same reference;

else:

use copy constructor to create a new sequence temp which is identical to other;

swap self and temp;

return self;

Sequence::~Sequence()(the destructor)

create a pointer curNode pointing to the first node (the one after the dummy node);

repeatedly:

delete the node which curNode is pointing at;

let curNode point to the next node, until it points to the dummy;

delete the dummy node;

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

return false if position is invalid;

create a pointer nextNode that points to the first node;

move nextNode until it points to the node at position = pos;

(forward if pos is in the first half of the list, backward if pos is in the second half of the list)

create a new node;

let nextNode->prev->next point to the new node;

let new node->prev point to nextNode->prev;

let nextNode->prev point to the new node;

let new node->next point to nextNode;

increment size by 1;

return true;

int Sequence::insert(const ItemType &value)

create a pointer nextNode that points to the first node;

move nextNode forward until the value of the node it points to > value;

record the position;

call the insert function above to insert a new node at this position with value;

bool Sequence::erase(int pos)

return false if position is invalid;

create a pointer curNode that points to the first node;

move curNode until it points to the node at position = pos;

(forward if pos is in the first half of the list, backward if pos is in the second half of the list)

let curNode->prev->next = curNode->next;

let curNode->next->prev = curNode->prev;

delete the node which curNode points to;

decrement size by 1;

return true;

int Sequence::remove(const ItemType &value)

create a counter that counts the number of nodes removed;

create a pointer curNode that points to the first node;

move curNode forward;

if curNode points to a node with desired value:

call erase function on that position;

increment counter by 1;

return counter;

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

return false if position is invalid;

create a pointer curNode that points to the first node;

move curNode until it points to the node at position = pos;

(forward if pos is in the first half of the list, backward if pos is in the second half of the list)

set value to the value of the node pointed to by curNode;

return true;

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

return false if position is invalid;

create a pointer curNode that points to the first node;

move curNode until it points to the node at position = pos;

(forward if pos is in the first half of the list, backward if pos is in the second half of the list)

set the value of the node pointed to by curNode to value;

return true;

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

create an integer p = -1;

create a pointer curNode that points to the first node;

move curNode forward repeatedly:

if curNode points to a node with desired value:

set p to the position of the node;

break out of the loop;

return p;

void Sequence::swap(Sequence &other)

exchange the address held by the head pointers of self and other;

exchange the size of self and other;

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

return -1 if seq2 is longer than seq1, or if seq2 is empty;

find in seq1 the first occurrence of the first value of seq2;

let that position be i;

for i from i + 1 to seq1.size - seq2.size:

create a bool variable named match, whose default is true;

for j from 0 to seq2.size:

if i-th item of seq1 != j-th item of seq2:

set match = false;

break to the outer loop;

if match:

return i;

return -1;

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

create a new seqence newseq which is a copy of seq1;

for i from 0 to seq2.size:

insert i-th item of seq2 to (i \* 2 + 1)-th position of newseq, if (i \* 2 + 1) is not out of bounds of newseq;

else, insert i-th item of seq2 to the end of newseq;

result = newseq;

return result;

# Test Cases

