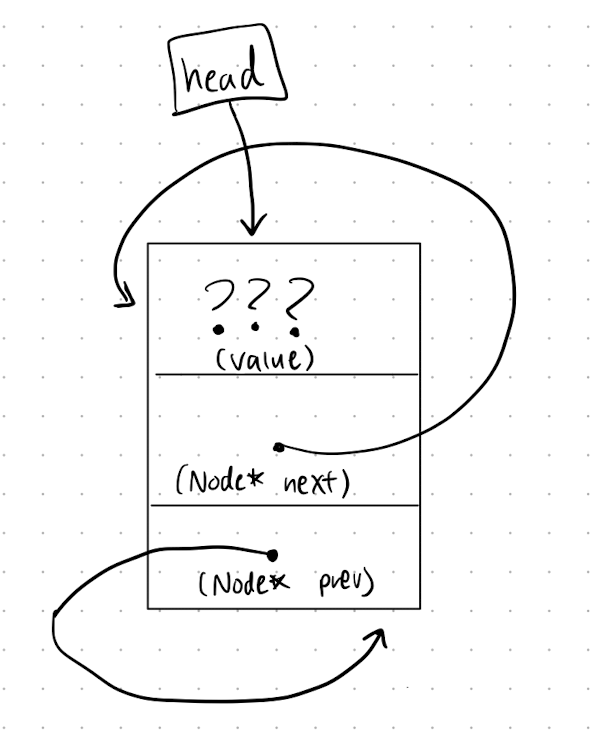
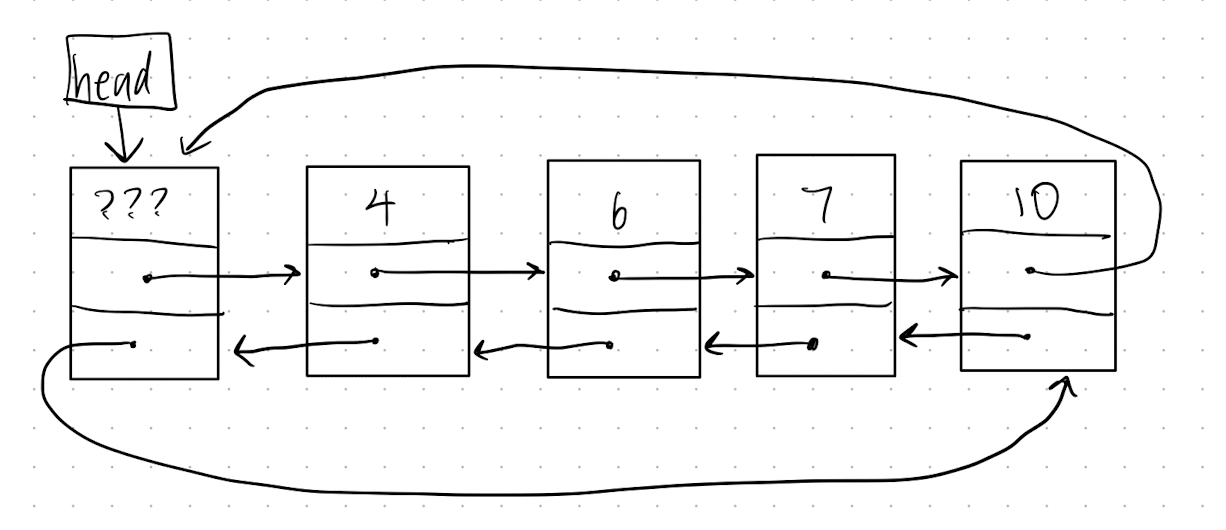
Project 2 Report - Joyce Chen

Implementation

I used a doubly-linked list that was also circularly linked. For each Set, I have a member variable that is a pointer (named “head”) to a dummy node. The dummy node contains “next” and “prev” Node pointers as well as a value field; however, it doesn’t have a name, nor is its value initialized. If the Set is empty, the dummy node’s “next” and “prev” pointers both point to itself. An empty Set looks like this:



A typical Set has the nodes ordered from least to greatest in terms of their value. Each node has a “prev” and “next” pointer that points to the previous node in the Set and the next node in the Set, respectively. Here’s a diagram of a typical Set:



Pseudocode

**copy constructor:**

set # of items to the # of items in the source Set

set head pointing to a new dummy node & circularly link it to itself

create a temp node to traverse through the source Set, initializing it to the first node

create a temp node to traverse through the Set, initializing it to head

while the source’s traversal node isn’t its head node:

create a new node that’ll be a copy of the source’s traversal node

store the Set’s current “next” node

link the current node’s “next” node to the new node

link the new node’s “next” to the old “next” node

link the new node’s “prev” to current node

set the current node to the new node

increment the source’s traversal node

**destructor:**

if the Set is empty,

delete the head node

otherwise,

create a node that traverses through the Set, starting at the first legit node

while the traversal node is not the head node:

save the location of the traversal node’s next node

delete the node pointed to by the traversal node ptr

set the traversal node to that saved node

delete the head node

**assignment operator:**

copy the source Set into a temporary Set

swap the Set with the temporary Set

return the Set (destructor on temporary Set will be called after this)

**insert function:**

if Set already contains the value to be inserted,

return false

create a node for the new inserted value

set the node’s value to the argument’s value

declare a temp pointer starting at the head that will traverse the Set

if the Set is not empty,

while temp’s next pointer is not the head pointer and the inserted value is greater

than temp’s next pointer’s value:

increment temp’s pointer by setting it to its “next” pointer

create a node to store temp’s “next” node (first value greater than the inserted node)

link the inserted node’s “next” to the above node

link that node’s “prev” to the inserted node

link the temp node’s “next” to the inserted node

link the inserted node’s “prev” to the temp node

increment # of items

return true

**erase function:**

create a temp node to traverse through the Set, initializing it to head

while temp’s next node is not head itself (aka while list isn’t empty):

if temp’s next node contains the value we want to erase,

create a killMe pointer to this node

link temp’s “next” to killMe’s next node (temp’s next next node)

link killMe’s next node’s “prev” to the temp node

delete the killMe node

decrement # of items

return true

set temp to its “next” node (continue traversing through the Set)

return false (if no value is found)

**contains function:**

create a temp node to traverse through the Set, initializing it to head

while the temp node’s “next” is not head itself:

increment the temp node by setting it to its “next” node

if the temp node’s value equals the value we want to check,

return true

return false (the Set doesn’t contain this value)

**get function:**

if 0 <= i < size of Set,

create a temp node to traverse through the Set, initializing it to the 1st legitimate node

repeatedly for i times:

set the temp node to its “next” node

now that temp is pointing to our wanted node, store its value into the argument’s value ref

return true

return false

**swap function:**

store the # of items in a temporary int variable

set the # of items to the # of items in the source Set

set the # of items in the source Set to the temporary int variable

store the head pointer in a temporary Node pointer

set the head pointer to the source Set’s head pointer

set the source Set’s head pointer to the temporary Node pointer

**unite function:**

assign the result Set to s1 (using assignment operator)

create an ItemType variable “value”

initialize a “count” variable to 0

while “count” is less than the size of s2:

get the value in s2 that’s greater than exactly “count” items

store this value into “value”

insert “value” into the result Set

increment “count”

**butNot function:**

assign the result Set to s1 (using assignment operator)

create an ItemType variable “value”

initialize a “count” variable

while “count” is less than the size of s2:

get the value in s2 that’s greater than exactly “count” items

if the result Set contains this value,

erase this value from the result Set

increment “count”

Test Cases

