Calvin Liu

Report

The technique I used to implement the code is a circularly, doubly linked list that used a dummy node to keep track of the beginning. The circularly linked list will have the last node pointing to the beginning and the beginning pointing to the end of the linked list. The doubly linked list will make it easier to traverse through the linked list.

Multiset Constructor:

Initialize all ints to 0

Create the first node that will act as the dummy node

The next pointer will point to itself

The previous pointer will point to itself

Multiset Destructor:

Create a temporary pointer that points to the first real node of the linked list which is after the dummy node (A).

Go through the whole linked list until you reach the dummy node

For each node, the head pointer will skip the one it’s pointing to (A) and will point to the one after it (B)

Delete the temporary pointer and the thing it is pointing to

Move the temporary pointer to the next Node to get ready to be deleted

Delete the dummy node

Multiset Copy Constructor:

Copy the integer members from the initialized multiset to the noninitialized

Create a new Node that is not initialized (dummy Node)

Make its next pointer point to itself

Make its previous pointer point to itself

Create a pointer to the head of the uninitialized Multiset

Create a pointer (X) to the Node after the head pointer of the initialized Node

Loop X through until it reaches the head again

Make the uninitialized and lonely Node, make a new node and point to it

That Node’s data will be the data of the Node after X

That Node’s count will be the count of the Node after X

Make that Node’s next point to the head of the uninitialized set

Move up the pointer of the initialized set

Move up the pointer of the uninitialized set

After that is done, make the pointer of the uninitialized set point to its head again

Make the head’s previous of that set point to the end

Multiset operator:

Do the same thing as the copy constructor just check for aliasing first

Multiset size:

Return the number of items in the list which should be the sum of the counts

Multiset uniqueSize:

Return the number of unique items in the list

Multiset empty:

The size should be 0 and if it is return true, if not then return false

Multiset insert:

Go through the multiset and see if the value was already inputted

If you find the same value, then just increase the count and the total size

Create the first node besides the dummy one

Its next should be the head, its previous should be the head

The previous head should point to the first node

The next head should point to the first node

Increase the count of the value by 1

Put in the value

Increase the uniquesize

After inputting the first node, follow different rules

Make a new node, input the data and increase the count to 1

The new node next points to the head

The new thing’s previous should be the head’s previous, the ending node

The heads previous (ending node) should point to the new one

The ending node should point to the previous last one

Increase the size and the uniquesize

Multiset erase:

Loop through the thing and try to find the value

If you did then decrease the count

Decrease the size

If the count is 0 then it needs to be deleted

Make a new storing pointer that points to the pointer being looped

Move the pointer being looped previous next arrow to the pointer being looped next

Move the pointer being looped next previous arrow to the pointer being looped previous

Delete the storage pointer that we created

Decrease the uniquesize

Multiset eraseAll:

Loop through the list and try to find the value

If you do, store the count to return later

Have a pointer that points to the node where the value was found

Make the pointers previous next arrow point to the pointer being looped next

Make the pointers next previous arrow point to the pointer being looped previous

Move the ptr to the next node

Delete the storage pointer

Decrease the uniquesize

Decrease the size by the count that was stored earlier

Return the number of instances removed

Multiset contains:

Make a pointer that points to the node after the dummy

Run through the linked list until you get back to the dummy node

If the pointer is pointing to the right value

Then it found it and return true

If not then update the pointer to the next Node

If it goes through the whole thing without finding it and getting back to the dummy node, then return false

Multiset count:

Create a pointer that points to the node after the dummy

Run through the linked list until you get back to the dummy node

If you find the right Node with the right value then return its count

Update the pointer

Multiset get

Create a pointer pointing to the node after the dummy

The position for that node is 0 because it doesn’t really count

If i is valid, then

Go through all the nodes until you reach the last one

Multiset swap

Make a temporary variable

Store the size of Multiset 1 in to the variable

Store the size of multiset 2 into the size of multiset 1

Make the size of multiset 2 the size of the temporary

Function combine

Override result with one of the multiset

Go through the other multiset, inserting as many times as the count

Use the get function to get the copied value

The copied value will be used in determining the count

Function subtract

Override result with one of the multiset (it has to be m1)

Go through the other multiset the amount of items in m1

Go until as many as there are count and erase

Create a temporary pointer

Make head of multiset 1 point to it

Make head of multiset 2 point to multiset 1

Make temp head of multiset 1 point to multiset 2

Delete the temporary pointer

Test Cases:

What if the list is empty? Check all the functions

What if the list has multiple elements in it? Check the functions

What if the list has multiple elements that are not adjacent to one another? Check the functions

What if the list only has one node?

What if the list only has two nodes?

Are there any concerns in inserting from the front?

Are there any concerns in inserting from the back?

Here is some of my code to check these cases used in different combinations:

Linked lists that are not equal length

void print();

//void Multiset::print()

//{

// for(Node\* ptr = m\_head->next; ptr != m\_head; ptr = ptr->next)

// {

// cout << ptr->data << endl;

// }

//}

//REMEMBER TO DELETE THIS ====================================================

//int main()

//{

// Multiset a;

// Multiset b;

// Multiset c;

// b.insert(10);

// b.insert(11);

// b.insert(12);

// b.insert(1);

// a.insert(1);

// a.insert(2);

// a.insert(2);

// a.insert(3);

// combine(a, b, c);

// cout << a.uniqueSize() << endl;

// a.erase(1);

// a.insert(4);

// a.insert(5);

// a.insert(5);

// a.insert(5);

// a.insert(7);

// a.eraseAll(2);

// swap(a,b);

// a.print();

// ItemType x;

// for(int i = 0; i < b.uniqueSize(); i++)

// {

// for(int k = 0; k < b.get(i, x); k++)

// {

// cout << x;

// }

// }

// b.print();

// subtract(a, b, c);

// c.print();

// if(a.contains(3))

// cout << "contains worked" << endl;

// cout << "count says " << a.count(2) << endl;

// cout << "size()" << a.size() << endl;

// cout << "uniqueSize()" << a.uniqueSize() << endl;

// unsigned long x;

// a.get(1, x);

// cout << x << endl;

// cout << a.get(1, x) << endl;

// if(a.empty())

// cout << "empty is working fine" << endl;

//}