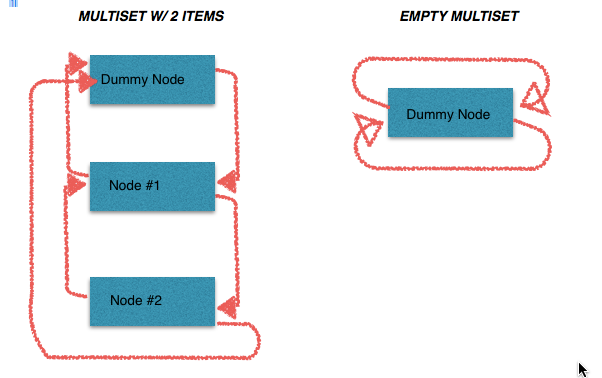
I used a circularly, doubly linked list with a dummy node created in the constructor. The dummy node helps me keep track of the beginning of the list. If the list is empty, the dummy node points to itself in both ways. Otherwise, the dummy node points forward to the first node in the list. Each node consists of a *value* of *ItemType* and an *int number.*

**

* **Basic Constructor**
  + Seat values of unique size and total size to 0, and head to nullptr.
  + Created a dummy node at head. It points forward and back to itself.
* **Deconstructor**
  + created 2 pointers to nodes.
  + Go through each node from the end.
    - set pointer y to the prev of pointer x
    - delete the node at pointer x
    - set pointer x to y
  + Delete the last node(which is .the dummy node)
* **Copy Constructor:**
  + set head to nullptr, unique size to the other objects unique size and total size to the other objects total size.
  + transverse through the other objects list.
    - create a new node and set its values to the same as the node in the other list
    - point it back to the last node and forward to the dummy node
* **Copy Assignment Operator:**
  + check for aliasing
  + Use copy construct to create a temp object identical to the one to be copied
  + swap the temp object with the object that needs to be changed
* **Empty:**
  + If total size is 0 return true otherwise return false
* **Size:**
  + return the total number of item in the list including all instances of each item
* **UniqueSize:**
  + return the number of unique items in the list
* **Insert:**
  + Using a for loop transverse through the existing list
  + if the value already exists, increment its number by one & the total size by 1
  + else, create a new node and set its value to the value and number to one
  + point it forward to the dummy node and back to the last node & increment unique size and total size by 1.
  + return true (item can always be inserted)
* **Erase:**
  + Using a for loop transverse through the existing list
  + If the value exists in the list, decrement its number by one and the total size by 1
  + if the number is now 0, point the previous node to the next node and the next node back to the previous node
  + delete the current node, decrement unique size by one, and return 1
  + if these cases don’t run, return 0: aka no items deleted
* **EraseAll:**
  + use a for loop to transverse the existing list and find a value if it’s in the list.
  + record the values number of instances
  + point it forward to the dummy node and back to the last node
  + delete the current node
  + decrement unique size by one and the total size by the number previously recorded
  + return the number previously recorded
  + if these cases don’t run, return 0: aka no items deleted
* **Contains:**
  + use a for loop to transverse the existing list and find a value if it’s in the list.
  + if so, return the number of instances of that value.
  + if the value doesn't exist in the list, return 0
* **Get:**
  + if the list is empty, return 0
  + if the int parameter is between 0 and one less than the unique size inclusive…
    - use a for loop to increment p to p->next, with each valid integer producing a different node of the list.
    - if p == head, increment it by one more node.
    - set the value parameter to the value of p
    - return the number of instances of p
  + return 0 if these cases don’t run.
* **Swap:**
  + create a temporary node to point to head of the object.
  + set the head to point to the head of the other object.
  + set the head of the other object to the temp node created previously.
  + create temp total size, and unique size integers of the object
  + set the object’s total size and unique size to the other object’s respective values
  + set the other objects total size and unique size to the temp total size and unique size created previously.
* **Combine:**
  + use a copy constructor to create a temporary object of ms1.
  + use a for loop of the number of instances in ms2.
    - use the get function to return all the different values of ms2.
    - use a for loop of each value’s number of instances
      * insert each value the correct number of times into ms1 using the insert function
  + set the result object to the temp object.
* **Subtract**
  + use a copy constructor to create a temporary object of ms1.
  + use a for loop of the number of instances in ms2.
    - use the get function to return all the different values of ms2.
    - use a for loop of each value’s number of instances
      * delete each value the correct number of times into ms1 using the erase function
  + set the result object to the temp object.

***TEST CASES***

#include "Multiset.h"

#include <iostream>

#include <cassert>

using namespace std;

int main(){

Multiset t1;

**//Empty Function works**

assert(t1.empty());

assert(t1.size() == 0);

assert(t1.uniqueSize() == 0);

assert(t1.erase(123) == 0);

assert(t1.eraseAll(123) == 0);

**//Insert Function works**

assert(t1.insert(1));

assert(t1.insert(2));

assert(t1.insert(1));

**//check size & uniqueSize & count functions**

assert(!t1.empty());

assert(t1.size() == 3);

assert(t1.uniqueSize() == 2);

assert(t1.count(1)==2 && t1.count(2) == 1);

**//check erase Function**

assert(t1.erase(1) == 1);

assert(t1.size() == 2);

assert(t1.uniqueSize() == 2);

assert(t1.erase(1) == 1);

assert(t1.size() == 1);

assert(t1.uniqueSize() ==1);

assert(t1.eraseAll(2) == 1);

**//check eraseAll function**

assert(t1.insert(1));

assert(t1.insert(2));

assert(t1.insert(1));

assert(t1.eraseAll(1) == 2);

assert(t1.size() == 1);

assert(t1.uniqueSize() ==1);

t1.eraseAll(2);

t1.insert(1);

t1.insert(5);

t1.insert(6);

t1.insert(5);

t1.insert(3);

t1.insert(3);

t1.insert(5);

t1.insert(1);

assert(t1.uniqueSize() == 4 && t1.size() == 8);

ItemType value1, value2;

**//Check Get Function & Copy Constructor & Count Function**

Multiset temp1(t1);

for(int i = 0; i < t1.uniqueSize();i++){

assert((t1.get(i, value1) == t1.count(value1)) == (temp1.get(i, value2) == temp1.count(value2)));

value1 = value2;

}

assert(t1.size()==temp1.size() && t1.uniqueSize()==temp1.uniqueSize());

**//Check Copy Assingment Operator**

Multiset t2;

t2.insert(23);

t2.insert(33);

t2.insert(88);

t2.insert(99);

t2.insert(24);

t2.insert(2993);

t2.insert(99);

t2.insert(99);

t2.insert(23);

t2.insert(27);

t2.insert(21);

t2.insert(11);

t2 = t1;

for(int i = 0; i < t1.uniqueSize();i++){

assert((t1.get(i, value1) == t1.count(value1)) == (temp1.get(i, value2) == temp1.count(value2)));

value1 = value2;

}

assert(t1.size()==temp1.size() && t1.uniqueSize()==temp1.uniqueSize());

**//Check for aliasing on copy constructor**

t1 = t1;

for(int i = 0; i < t1.uniqueSize();i++){

assert((t1.get(i, value1) == t1.count(value1)) == (t1.get(i, value2) == t1.count(value2)));

value1 = value2;

}

Multiset t3;

t3.insert(23);

t3.insert(33);

t3.insert(88);

t3.insert(99);

t3.insert(24);

t3.insert(2993);

t3.insert(99);

t3.insert(99);

t3.insert(23);

t3.insert(27);

t3.insert(21);

t3.insert(11);

Multiset temp3(t3);

**//Check swap function (t1 = temp1; t3 = temp3)**

t1.swap(t3); **//so t1 should = temp3. t3 should = temp1**

for(int i = 0; i < temp3.uniqueSize();i++){

assert((t1.get(i, value1) == t1.count(value1)) == (temp3.get(i, value2) == temp3.count(value2)));

value1 = value2;

}

assert(t1.size()==temp3.size() && t1.uniqueSize()==temp3.uniqueSize());

for(int i = 0; i < temp1.uniqueSize();i++){

assert((t3.get(i, value1) == t3.count(value1)) == (temp1.get(i, value2) == temp1.count(value2)));

value1 = value2;

}

assert(t3.size()==temp1.size() && t3.uniqueSize()==temp1.uniqueSize());

**//Check Combine Function**

Multiset t4, t5, t6, temp6;

temp6.insert(11);

temp6.insert(22);

temp6.insert(33);

temp6.insert(44);

temp6.insert(55);

temp6.insert(66);

temp6.insert(77);

t4.insert(11);

t4.insert(22);

t4.insert(33);

t4.insert(44);

t5.insert(55);

t5.insert(66);

t5.insert(77);

combine(t4, t5, t6);

for(int i = 0; i < temp6.uniqueSize();i++){

assert(t6.get(i, value1) == t6.count(value1) == temp6.get(i, value2) == temp6.count(value2));

value1 = value2;

}

**//Check Subtract Function**

Multiset t7, t8, t9, temp9;

temp9.insert(22);

temp9.insert(33);

t7.insert(11);

t7.insert(22);

t7.insert(22);

t7.insert(33);

t8.insert(11);

t8.insert(22);

t8.insert(47);

t8.insert(11);

t8.insert(11);

subtract(t7, t8, t9);

for(int i = 0; i < temp9.uniqueSize();i++){

assert(t9.get(i, value1) == t9.count(value1) == temp9.get(i, value2) == temp9.count(value2));

value1 = value2;

}

**//Check for Aliasing in both subtract & combine**

Multiset temp10, temp11;

**//temp 10: combination of 2 temp9’s**

temp10.insert(22);

temp10.insert(22);

temp10.insert(33);

temp10.insert(33);

combine(temp9, temp9, temp9);

for(int i = 0; i < temp9.uniqueSize();i++){

assert((temp9.get(i, value1) == temp9.count(value1)) == (temp10.get(i, value2) == temp10.count(value2)));

value1 = value2;

}

**//temp9 should be empty at the end**

subtract(temp9, temp9, temp9);

for(int i = 0; i < temp9.uniqueSize();i++){

assert((temp9.get(i, value1) == temp9.count(value1)) == (temp11.get(i, value2) == temp11.count(value2)));

value1 = value2;

}

cout << "tests successful!!" << endl;

}