My doubly-linked list is pretty straightforward; I gave Multiset a head variable which points to the first node in the list which has an Itemtype, a count, a previous pointer, and a next pointer. This next pointer points to the second node which points to the third and continues in a linear, not circular, order with no dummy nodes. The list is only ordered in the order the values were inserted.

Empty Multiset: Typical Multiset with 2 Nodes:

nullptr

Node \*start

int Bigness

int UniqueBigness

Node\* start;

int Bigness;

int UniqueBigness;

Node 1:

Node \*prev

Node \*next

ItemType item

int count

nullptr

**PseudoCode:**

**Multiset::erase**

*…*

*establish tracker node*

*repeatedly:*

*check if item at node = value*

*decrease count by 1*

*check if count is 0 connect prev node to next node*

Node 2:

Node \*prev

Node \*next

ItemType item

int count

*delete this node*

nullptr

*return 1 instance of removal*

*update tracker node*

*return 0*

…

**subtract**

…

*repeatedly:*

*get ith element of m1 and count of it*

*get count of this element in m2*

*subtract counts*

*check if difference is positive*

*insert element into new multiset difference amount of times*

…

**More PseudoCode:**

**Multiset:: insert**

*check if list is empty*

*create new node with value parameter*

*update private counter variables*

*return true;*

*repeatedly*

*check if parameter value matches value already in set*

*add count to that value*

*update size variable*

*return true;*

*if the tracker node is not the last node*

*update the tracker to the next node*

*create a new node with value parameter and count one*

*assign tracker variable to new node*

*update counters*

*return true*

**Multiset::eraseAll**

*repeatedly*

*check if tracker node’s value matches parameter value*

*de-link node from list*

*update counter variables*

*remove node*

*return count*

*if tracker node is not null*

*move tracker node to next node*

*return 0*

**Multiset::get**

*check if parameter i does not identify an index within the list*

*return false*

*repeatedly*

*update tracker node until it as at ith node*

*change string parameter to value at tracker node*

*return count at tracker node*

**Multiset::getMostFrequentValue**

*repeatedly*

*check if count at tracker node is higher than previous counts*

*assign highest count variable to count at tracker node*

*if tracker node is not null*

*move tracker to next node*

*reset tracker node*

*repeatedly*

*if tracker node count matches highest count*

*increment number of modes by 1*

*assign value at tracker node to mode variable*

*if tracker node is not null*

*move tracker to next node*

*if there is only one mode*

*change string parameter to mode*

*return true*

*return false*

(follow similar process for getLargestValue and getSecondLargestValue)

**Multiset::replace**

*repeatedly*

*check if item at tracker node matches original value*

*repeatedly check if new value is already in the list*

*de link original node*

*add original node count to new value node*

*remove original node*

*return true*

*set tracker node value to new value*

*return true*

*if tracker node is not null*

*move tracker node to next node*

*return false*

**Multiset::countIf**

*if the parameter char operator is not valid*

*return -1*

*if the parameter is ‘=‘*

*add count of value to counter variable*

*if the parameter is ‘<‘*

*repeatedly*

*if tracker node’s value < parameter value*

*add tracker node’s count to count variable*

*if tracker node is not null*

*move tracker node to next node*

*if the parameter is ‘>‘*

*repeatedly*

*if tracker node’s value > parameter value*

*add tracker node’s count to count variable*

*if tracker node is not null*

*move tracker node to next node*

*return counter variable*

**Multiset::swap**

*trade head pointers by creating a temporary pointer*

*trade sizes and unique sizes using temporary variables*

**combine**

*if the passed result Multiset parameter is not empty*

*repeatedly*

*remove values of Multiset using eraseAll function*

*cycle through Multiset*

*use copyIntoOtherMultiset to fill ms1 values in result Multiset*

*use copyIntoOtherMultiset to fill ms2 values in result Multiset*

**Test Cases**

1. Multiset ms;

ms.insert("cinnamon");

ms.insert("galangal");

ms.insert("cinnamon");

string s1;

int n1 = ms.get(1, s1);

assert((s1 == "cinnamon" && n1 == 2) || (s1 == "galangal" && n1 == 1));

string s2;

int n2 = ms.get(1, s2);

assert(s2 == s1 && n2 == n1);

2. Multiset ms;

ms.insert("a");

ms.insert("a");

ms.insert("b");

ms.insert(“c");

ms.insert(“ “);

for (int k = 0; k < ms.uniqueSize(); k++)

{

string x;

int n = ms.get(k, x);

cout << x << " occurs " << n << " times." << endl;

}

3. ItemType value;

Multiset ms1, ms2;

assert(ms2.getLargestValue(value) == false );

assert(ms2.getSecondLargestValue(value) == false );

assert(ms2.getMostFrequentValue(value) == false );

ms1.insert("ABC"); ms1.insert("ABC");

ms1.insert("XYZ"); ms1.insert("GDP");

ms2.insert("Hello"); ms2.insert("ABC");

ms2.insert("XYZ"); ms2.insert("PPP");

assert(ms1.getLargestValue(value) == true && value == "XYZ");

assert(ms1.getSecondLargestValue(value) == true && value == "GDP");

assert(ms1.getMostFrequentValue(value) == true && value == “ABC");

4. Multiset ms1;

ItemType x = “a”;

assert(ms1.empty());

ms1.insert(“hello”); ms1.insert(“world”); ms1.insert(“goodbye”);

assert(!(ms1.empty()));

assert(ms1.size() == ms1.UniqueSize() == 3);

assert(ms1.eraseAll(“hello”) == 1);

assert(ms1.erase(“hello”) == 0);

assert(!(ms1.contains(“hello”));

assert(ms1.count(“world”) == ms1.count(“goodbye”) != ms1.count(“hello”));

assert(!(ms1.getMostFrequentValue(x)) && x == “a”);

assert(ms1.getLargestValue(x) && x == “world”);

assert(ms1.getSecondLargestValue(x) && x == “goodbye”);

5. ItemType value;

Multiset ms1, ms2;

ms1.insert("ABC"); ms1.insert("ABC");

ms1.insert("XYZ"); ms1.insert("GDP");

ms2.insert("Hello"); ms2.insert("ABC");

ms2.insert("XYZ"); ms2.insert("PPP");

ms1.replace("ABC","GDP");

assert(ms1.uniqueSize() == 2 && ms1.size() == 4);

ms2.replace("Hello","abc");

assert(ms2.uniqueSize() == 4 && ms2.size() == 4 );

6. Multiset ms1;

ms1.insert("cumin");

ms1.insert("cumin");

ms1.insert("cumin");

ms1.insert("turmeric");

Multiset ms2;

ms2.insert("coriander");

ms2.insert("cumin");

ms2.insert("cardamom");

ms1.swap(ms2); // exchange contents of ms1 and ms2

assert(ms1.size() == 3 && ms1.count("coriander") == 1 &&

ms1.count("cumin") == 1 && ms1.count("cardamom") == 1);

assert(ms2.size() == 4 && ms2.count("cumin") == 3 &&

ms2.count("turmeric") == 1);

7. ItemType value;

Multiset ms1, ms2;

ms1.insert("ABC"); ms1.insert("ABC");

ms1.insert("XYZ"); ms1.insert("GDP");

ms2.insert("Hello"); ms2.insert("ABC");

ms2.insert("XYZ"); ms2.insert("PPP");

assert( ms2.countIf('=',"ABC") == 1 );

ms1.copyIntoOtherMultiset(ms2);

assert(ms2.uniqueSize() == 5 && ms2.size() == 8 );

assert( ms2.countIf('=',"ABC") == 3 );

assert( ms2.countIf('>',"ABC") == 5 );

assert( ms2.countIf('<',"ABC") == 0 );

8. ItemType value;

Multiset ms1;

ms1.insert("ABC"); ms1.insert("ABC");

ms1.insert("XYZ"); ms1.insert("GDP");

Multiset ms2(ms1);

assert(ms2.getLargestValue(value) == true && value == "XYZ");

assert(ms2.getSecondLargestValue(value) == true && value == "GDP");

assert(ms2.getMostFrequentValue(value) == true && value == "ABC");

9. ItemType value;

Multiset ms1, ms2;

ms1.insert("ABC"); ms1.insert("ABC");

ms1.insert("XYZ"); ms1.insert("GDP");

ms2 = ms1;

assert(ms2.getLargestValue(value) == true && value == "XYZ");

assert(ms2.getSecondLargestValue(value) == true && value == "GDP");

assert(ms2.getMostFrequentValue(value) == true && value == "ABC")

10. void test()

{

cerr << "test A" << endl;

Multiset ms;

cerr << "test B" << endl;

ms.insert(IntWrapper(10));

cerr << "test C" << endl;

ms.insert(IntWrapper(20));

cerr << "test D" << endl;

Multiset ms2;

cerr << "test E" << endl;

ms2.insert(IntWrapper(30));

cerr << "test F" << endl;

ms2 = ms;

cerr << "test G" << endl;

ms2.insert(IntWrapper(40));

cerr << "test H" << endl;

}

int main()

{

test();

cerr << "DONE" << endl;

}

***Test Case Explanations***

1. This case is designed to make sure that successive calls to the get function with the same parameter return the same value
2. This case ensures the get function iterates over all elements of the Multiset. It prints out each element with their corresponding count.
3. Tests the getFrequentValue, getLargestValue, & getSecondLargestValue functions. The first few assert statements test these functions to return false when there is no appropriate value to return and the latter assert statements make sure they work when there is.
4. Runs the empty(), size(), uniqueSize(), insert(), erase(), eraseAll(), and count() functions as well as the get value functions.
5. Tests the replace function to make sure that when a value is replaced, if the new value is already in the set, it only affects the count and does not create a new node.
6. Makes sure swap function works and swaps all values, including their counts.
7. Tests the copyIntoOtherMultiset() and countIf() functions.
8. This tests the copy constructor to ensure that all the same values are copied into the new Multiset.
9. Same template as the copy constructor but I used the assignment operator to fill in ms2 to make sure it receives the same values as ms1.
10. I used the code on the class website to implement IntWrapper and made sure the output of this function matched the output listed on the site. There were a few discrepancies but I attributed those to the differences in approach between my project and the one used by the site.