CS 211
Fundamentals of Computer Programming II
Winter 2023

Project C++ Extra-Credit (v1.0)

Assignment: multiset
Submission: Gradescope
Policy: individual work only, late work not accepted
Complete By: Saturday March 16th @ 11:59pm CST
Late submissions: this is an extra-credit project, no late submissions.

Pre-requisites: Project 08 (set)

Overview

In project 08 we implemented a **set**, which stored unique keys --- duplicates were not allowed. C++ also provides the concept of a **multiset**, where duplicates *are* allowed. Example:

```
multiset<int> S;

S.insert(1);
S.insert(2);
S.insert(2);
S.insert(3);

cout << S.size() << endl;
```

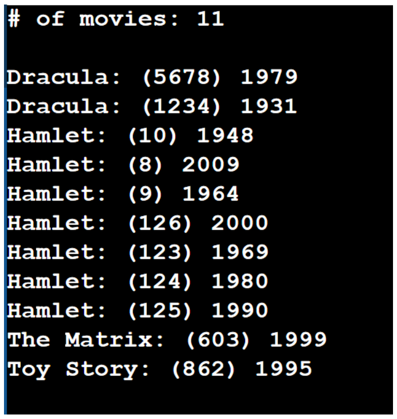
In this case S contains 4 elements: 1, 2, 2, and 3, with a size of 4. Here's a more interesting example, storing **Movie** objects in a multiset --- the duplicates result from movies being remade with the same title:

```
multiset<Movie> movies;

movies.insert( Movie(603, "The Matrix", 1999) );   // ID, Title, Year
movies.insert( Movie(862, "Toy Story", 1995) );
movies.insert( Movie(10, "Hamlet", 1948) );
movies.insert( Movie(5678, "Dracula", 1979) );
movies.insert( Movie(1234, "Dracula", 1931) );
movies.insert( Movie(8, "Hamlet", 2009) );
movies.insert( Movie(9, "Hamlet", 1964) );
movies.insert( Movie(126, "Hamlet", 2000) );
movies.insert( Movie(123, "Hamlet", 1969) );
movies.insert( Movie(124, "Hamlet", 1980) );
movies.insert( Movie(125, "Hamlet", 1990) );

cout << "# of movies: " << movies.size() << endl;
cout << endl;

for (const Movie& m : movies) {
   cout << m.Title << ": ("
        << m.ID << ") "
        << m.Year << endl;
}
```



```
# of movies: 11

Dracula: (5678) 1979
Dracula: (1234) 1931
Hamlet: (10) 1948
Hamlet: (8) 2009
Hamlet: (9) 1964
Hamlet: (126) 2000
Hamlet: (123) 1969
Hamlet: (124) 1980
Hamlet: (125) 1990
The Matrix: (603) 1999
Toy Story: (862) 1995
```

Notice that *foreach* iterates through the multiset in order, in this case alphabetically by Title. In the case of duplicates, the duplicates are output in order of insertion. For example, notice there are 7 movies named "Hamlet", output in the order they were inserted. This implies the **iterator** for multiset, in particular the iterator's **++** operator, must iterate in this order.

How can objects be inserted into containers like set and multiset? In C++, as long as you provide a way for the others to be compared with **<**, they can be inserted into a container. A common approach is to overload the < operator in the class. For example, here's the **Movie** class with < overloaded to order by Title:

```
class Movie {

public:
  int ID;
  string Title;
  int Year;

  Movie(int id, string title, int year)
   : ID(id), Title(title), Year(year)
   { }

  bool operator<(const Movie& other) const
  {
    if (this->Title < other.Title)
      return true;
    else
      return false;
  }
};
```

Finally, if you review your solution to Project 08 (set), notice that *set::contains( )* and *set::insert( )* methods are written to search the tree using only the < operator, nothing else. Two objects a and b are equal if !(a<b) and !(b<a).

Extra-credit assignment

The extra-credit assignment is to implement a multiset class, based on your solution to Project 08. We are not releasing a solution to project 08, so you must have a solution before continuing --- and this must be YOUR solution, not someone else's. The multiset must be an extension of project 08, using a threaded binary search tree for the underlying implementation, and passing all existing unit tests for project 08 and set. You'll then modify and extend as outlined in the next section.

The extra-credit project is worth 50 points, which can be used to bolster project scores (C or C++) that are < 100; no project score may exceed 100 points. These points cannot be used for any other purpose. Notice the due date, and that there is no late period --- it's the end of the quarter, we cannot extend. Also, don't expect Gradescope to open until the last day, this is extra work for the staff / myself and so testing is up to you. Don't ask on Piazza when Gradescope will open, otherwise we'll just delay it further.

Assignment details

A **multiset** stores duplicates whereas a **set** does not, that's the crucial difference. This implies that in a multiset, the insert( ) function must be changed to insert duplicate keys. But how?

The approach used by C++ (and other languages) is to insert duplicate keys into the tree as separate nodes. If a key is < root you insert to the left, otherwise you insert to the right --- this implies that each time you insert a duplicate, it is inserted to the right. This is a good technique, other than the size of the tree grows since each duplicate yields a new node. But with rebalancing, operations remain O(lgN), so this approach in fact works well.

However, we're going to take a different approach. The NODE class is going to be extended to include a vector, and duplicate keys are stored in this vector. This way, all duplicates can be immediately found by searching for the key (e.g. "Dracula" or "Hamlet") and then iterating through the vector. This is a good approach when you are expecting lots of duplicates, or might be deleting keys and their duplicates. [*Why not a list / linked-list instead of the array-based vector? Vectors allow efficient access to any element, where list / linked-list only allows efficient access to the first and last elements. This will likely matter when it comes time to implement the iterator.*]

Here are the steps / functionality you need to implement. Note that your unit tests are considered as important as your implementation of multiset. You are required to write good tests for your submission to be accepted:

1. Create a new folder to work in, and copy over your files from project 08
2. Rename "set.h" to "multiset.h"
3. Rewrite tests to refer to "multiset.h" and "multiset"
4. Build and run, all existing tests should pass

5. Add the vector to NODE class; add getters and setters as needed by future steps. You are required to use a vector-based approach, no other approach will be accepted.
6. Change insert( ) to insert duplicates into the vector.
7. Build and run --- you may need to fix any tests that insert duplicates, which would have failed in previous tests but now they succeed. Adjust tests as appropriate.
    1. *Since the NODE class contains a vector, does this impact how we free memory in multiset's destructor? No, you only have to worry about memory that is allocated with new. Since we don't*

allocate the vector with new --- instead it's part of the NODE itself --- when we delete the NODE we'll also delete the vector.

8. Leave toPairs( ) as is, this function is for testing the threads.
9. Rewrite toVector( ) to also include the duplicates; duplicates are added in order of insertion.
10. Add a count(TKey key) method that returns the # of keys in the multiset; return 0 if the key does not exist. Example: in the earlier movies example, if m is a Movie object whose Title is "Hamlet", then count(m) returns 7.
11. Test; you must have your own tests for submission.

12. Think about how you are going to iterate through a multiset…
13. Modify the iterator class, and the begin( ) and end( ) methods as necessary.
14. The find( ) method should now return an iterator such that if duplicate keys exist, the iterator denotes the first key inserted.
15. Test; you must have your own tests for submission, including tests of foreach-based iteration over multisets containing objects.

16. Add an erase(TKey key) that deletes a key --- and all duplicates --- from a multiset. We didn't talk about this, but implement like insert using *prev* and *cur* pointers. If found, you'll unlink and delete the node and its duplicates. There are 5 cases (we strongly recommend you draw before and after diagrams of each case, since you need to maintain the threads as well). Cases 1 – 4 are "easy" in that they only require adjusting a pointer or two. Case 5 is more complicated in that it requires a search for a suitable replacement:
    1. *Deletion of a leaf node (no children), to left of parent.*
    2. *Deletion of a leaf node (no children) to right of parent.*
    3. *Deletion of an interior node with only one sub-tree, to the left.*
    4. *Deletion of an interior node with only one sub-tree, to the right.*
    5. *In this case you are deleting a node N with two sub-trees. Find the smallest key K in the right sub-tree (or the largest key in the left sub-tree, your choice). Unlink the node containing K --- this is easier because it will fall into cases 1 – 4. Now copy the contents of node K into node N, and delete node K. We are essentially replacing N with K; convince yourself this works with a diagram.*
17. Test, test, test.

Grading and Electronic Submission

Submit your multiset.h and tests.cpp files to Gradescope for evaluation. To submit from the EECS computers, run the following command (you must run this command from inside your **release** directory):

```
/home/cs211/w2024/tools/project08xc  submit  multiset.h  tests.cpp
```

Gradescope submissions will open the last day; you'll have 4 submissions.

By default, Gradescope records the score of your **LAST SUBMISSION** (not your highest submission score). If you want us to grade a different submission, activate a different submission via your "Submission History".

This must be done before the due date.

You MUST have test cases that insert duplicates, test toVector( ) and count( ) and erase( ), and test your foreach-based iteration with duplicates. And you need to test with non-trivial objects like the Movie class. Failure to write your own good set of tests will invalidate your submission.

Academic Conduct Policy

Northwestern publishes a basic guide to academic integrity, which can be found here. In summary, here are NU's eight cardinal rules of academic integrity:

1. *Know your rights*
2. *Acknowledge your sources*
3. *Protect your work*
4. *Avoid suspicion*
5. *Do you own work*
6. *Never falsify a record or permit another person to do so*
7. *Never fabricate data, citations, or experimental results*
8. *Always tell the truth when discussing your work with your instructor*

School policies and more information can be found on NU's academic integrity website. With regards to CS 211, unless stated otherwise, all work submitted for grading *must* be done individually. While we encourage you to talk and learn from the course staff, peers, and others, this interaction must be superficial with regards to all work submitted for grading. This means you cannot work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The use of AI (ChatGPT, Co-pilot, etc.) is currently forbidden.

Examples of what is not allowed? Downloading work from a github repository and submitting it as your own, whether partial or complete. Downloading answers from StackOverflow and submitting them as your own. Emailing your work to another student, or receiving work from another student. Sharing your screen with another student so they can see your work (and potentially submit it as their own). Participating in a screen share and taking screenshots or photos of someone else's work, and then using that as a guide to submit your own work. Copying answers posted to Piazza, making a few simple changes, and then submitting as your own work. Allowing someone else to write / type the answer for you. Using AI (ChatGPT, Co-pilot, etc.) to generate code for you which you then submit as your own.

Okay, so what is allowed? Talking to the instructor or course staff, and getting insights --- but not direct answers --- to help you solve the assignment. Talking to other students about the assignment, using diagrams / natural language / pseudo-code to convey ideas. Searching the internet for guidance, and using that guidance to help you form your own solution. If you do receive help / guidance, it's always best to cite your source by name or URL just in case there is a question as to where the work came from.