CSCI-GA.3033-017 Special Topic: Multicore Programming
Lab 0 Assignment

The purpose of this lab is twofold. First, it is meant to make sure you're familiar with gcc/g++, gdb, and git. Second, it will get you in the frame of mind of doing the sort of C++ programming you'll be continuing throughout this course for the programming project. This lab asks you to use a fundamental data structure (a list) to implement a more complex data structure (a multimap).

A ***multimap*** holds key-value pairs, and can contain multiple key-value pairs that have the same key (and the same *or* different values). For example, the following is a valid data set contained in a multimap:

```
{'key' => 'value'}
{'key' => 'lock'}
{'chair' => 'classroom'}
{'year' => '2018'}
{'year' => '12 months'}
```

Your multimap should be written using a list, namely `std::list` or `std::vector`. Your multimap should support templated keys and values. It should support the following operations:

1. **`bool insert(const K& key, const V& value);`** Should return true if the operation succeeded, or false otherwise. It's likely you won't have a case where you need to return false in this simple implementation.
2. **`bool find(const K& key);`** Should return true if `key` is in the multimap, or false otherwise.
3. **`bool find(const K& key, std::list<V>& values);`** Same as (2), but if true is returned, also populates the passed-by-reference list `values` with each of the values present for the given `key`.
4. **`int remove(const K& key);`** Should remove any key-value pair with the given key, and return the number of key-value pairs removed. As with `insert()`, it's likely you won't have a case where you need to return -1 in this simple implementation.

I expect that your implementation be able to perform each of these operations in at best O(n) time. If you can do them faster (you must still use `std::list` or `std::vector` as the backing store), I'll be impressed. I very strongly suggest that your multimap be a class, and that your test executable (see the next section) create and use instances of that class. Remember, you will be graded not just on functionality, but on style and cleanliness as well.

Deliverables: You're expected to produce at least two things:

1. The source code for an executable that implements the list-based multimap, and performs tests on that set. If the source code needs to be multiple files for cleanliness, by all means do so.

   The executable should perform 10 tests, in which integers should be used for the keys and values. In each test, it selects 100 random integers between 0 and 200 (inclusive) as keys, and a corresponding 100 random integers between 0 and 200 (inclusive) as values, adds those to the multimap, then checks that each of those integers exists in the multimap. Hint: I recommend using `std::uniform_int_distribution` to generate your integers, a list to store the integers inserted as keys, and a loop over that list to check that each of the values in the list is in the multimap. At the end of the test, it should report whether the test succeeded or failed. I would prefer that you have a submission that correctly tests itself and finds that the implementation was incorrect than a submission that is correct but in which you didn't have time to add the test.

2. A `README` that explains how I can compile and run your code. I would prefer that you make a simple Makefile so I can make your code. Otherwise, provide a `build.sh` that invokes g++, or at worst, put the g++ command that should be used to compile your code in the `README`.

Grading: This lab will contribute to your lab score, but the magnitude of its contribution has not yet been determined. The lab will be graded on (1) functionality, (2) code style, and (3) adherence to the specification above.

Getting Help: If you're struggling, your first recourse should be the class mailing list. *Important:* you should solicit your fellow students for high-level help, such as "how can I iterate through a vector?". You should not ask for specific help with your code, such as "why doesn't the following code work?". If you're still stuck in a week, come to my office hours. See the first lecture for the policy on collaboration with other students (tl;dr: don't, outside of the mailing list). As is university policy, instances of cheating will be taken very seriously. If you believe there's an omission or error in this document, you're welcome to email me directly.

Due date: September 23, 2018, by 11:59:59pm EDT. See the first lecture for the late policy.

Submission: You should create a *private* Git repository called "MulticoreProgramming" in your GitHub account, push your code to your repository in a folder entitled "lab0", and add me (GitHub username: KermMartian) as a collaborator so I can review your code. Please also send me an email once you have committed and pushed your final submission with the tag (a 7-character hexadecimal string) of that commit; I will use the timestamp GitHub records for that commit to determine whether your submission is on-time or late.