Problem Set 4

NOTE: Some students have had problems compiling problem sets on Mac OS. This is due to the Apple compiler being very conservative with its choice of C++ standards. Using the C++11 standard should make this work -- use `g++ -std=c++11` to enable this.

Makefile

For the next two parts, create a single makefile contains a target for compiling the source files, and a target for running the resulting binaries.

Sorting w/ Lookup

The file `sort_atoms.cpp` contains a vector of strings, with each string containing an element symbol.

1. Sort this list such that all elements are in order by their element number (not in alphabetical order). Print out this sorted vector.

2. Find the unique elements of the vector (ie, remove duplicates), and print those out.

Hint: For #1, you should create a lookup table with an std::map, which you can use to find the element number given a symbol. Then you need to write a custom comparison function.

Calculating the Molecular Formula

The file `molecule.cpp` contains some pieces of the now-familar molecule class. Your task is to write a function to calculate the molecular formula as a string from the atoms contained in the class.

The symbols should be in alphabetical order, with the appropriate element count (ie, H2O), not HHO).

Hint: It would be helpful to have another lookup table as in the previous problem, but going in the opposite direction.

Class Design

Below is a block of code for a class that contains information about molecular conformers. A conformer, in this case, has the same atoms/elements with the same bonds, but a different orientation in space. It is common for molecules to be able to adopt different conformations. For example, cyclohexane can adopt the chair, boat, and twist-boat conformations (see https://en.wikipedia.org/wiki/Cyclohexane_conformation).

```
class MoleculeConformers
{
    private:
        std::vector<int> atoms_; // Z number. 1 = H, 6 = C, etc
        std::vector<Bond> bonds_;
        std::vector<ConformerCoordinates> conformers_;

    public:
        // must construct with atoms and bonds, or things
        // get complicated very quickly
        MoleculeConformers(std::vector<int> atoms, std::vector<Bond> bonds)
           : atoms_(atoms), bonds_(bonds)
        { }

        void add_conformer(const ConformerCoordinates & ccoords)
        {
            conformers_.push_back(ccoords);
        }

        void remove_conformer(int index)
        {
            conformers_.erase(conformers_.begin()+index);
        }

        ConformerCoordinates & get_conformer(int index)
        {
          return conformers_.at(index);
        }

        const ConformerCoordinates & get_conformer(int index) const
        {
          return conformers_.at(index);
        }

        size_t n_conformers() const { return conformers_.size(); }
};
```

Questions

Answer these questions in a README file. No need to write/compile any code for this, although you are free to do so to test some of your answers.

1. Should I have written a destructor for this class? Why or why not?
2. Should I have written a copy constructor? Why or why not?
3. When adding conformations with `add_conformer`, what kinds of checks should be performed to maintain consistency?
4. I mention in the code that `MoleculeConformers` objects should always be constructed with atoms and bonds first, otherwise things get complicated. What did I mean by that? What would a better comment say?
5. I have two functions named `get_conformer`. Is this allowed? If so, when is each used?
6. Both `get_conformer` functions return a reference. Is this a good idea? What happens if I change the data returned by these functions (for example, `mc.get_conformer(2).clear()`).
7. Can you explain what the code in `remove_conformer` does? What does `.begin()+index` mean and why am I doing that?
8. Is `remove_conformer` safe (that is, can it cause a crash)? If so, what can you do?