Computer Science II — CSci 1200
Lab 7 — Maps

Overview

This lab explores programming with maps. Before getting started, download the `word_count.cpp` program example

```
http://www.cs.rpi.edu/academics/courses/fall04/cs2/lab07/word_count.cpp
```

and examine it. We discussed this program in class and will modify it in checkpoint 1.

Review your notes on maps (Lectures 10 and 11; Chapter 7 of the Koenig and Moo text). Recall that a map is an association between a key and a value. The key and the value are stored in the map in pairs. Using the `[]` operator, we provide a key, and get access to the associated value. This is great, except that the key is placed in the map if it wasn't already there. This is often ok, but not always. Maps also have find and insert functions (member functions) whose behavior is pretty much as expected. The challenge to using them is mastering the associated syntax.

Also, recall that maps have iterators, just like every other container. In particular, maps have begin and end iterators, just like lists and vectors. Map iterators also have `++` and `--` operators. Unlike vector iterators, but like list iterators, map iterators can NOT be combined with integers offsets. The trick about map iterators is that they indicate pairs. Thus, in the `word_count.cpp` code, the line

```
map<string, int>::const_iterator it = counters.begin();
```

initializes the map iterator to the first entry in the vector. Then,

```
it -> first
```

is the first string in alphabetical order and

```
it -> second
```

is the string's associated count.

Review your notes (Lecture 12) on sets as well. Sets may be viewed as a simplified version of maps, where only the key is stored, not the key/value pair. In particular, sets are automatically ordered by the key, which means an `operator<` and `operator==` must be defined for the type stored in the set. Because they lack the key/value association, sets do not have an `operator[]`. The contents of sets are manipulated through insert, find and erase operations. Set iterators are also defined, and they are simpler than map iterators.

You will use both maps and sets in this lab.

Checkpoints

1. Write a program that uses a map to find the mode of a sequence of input integers. If there are multiple modes then just find the smallest one (just to keep things simple). You should use a declaration of the form

   ```
   map< int, int > occurrences;
   ```

   This should be a fairly straightforward modification of the word count program from Lecture 10 (the first lecture on maps). You are welcome to start with this program.

2. For this checkpoint and the third you are to write a program that reads a sequence of words and determines, for each word, what other words are near it in the sequence (separated by less than 3 intervening words). This is similar to the problem you solved for Homework 5. For simplicity, the input file contains one word per line, with no capital letters and no punctuation marks. Don't ignore any words. Start by reading all the words into a vector.

   This is an example where a map of sets is a really good idea (and required for this lab!). In particular, your definition of the map should look like:

   ```
   map< string, set<string> > nearby_words;
   ```

   (Don't forget to put the space between the two > symbols.) Each entry in this map will be an association (a pair) between a word and the set of words that appear near it somewhere in the input.

   To complete this checkpoint write and compile code to read the words into the vector and then create and fill in the map. The code for this is relatively short and can make good use of `operator[]` for maps.

   Stop and show your results to a TA when you have gotten the code compiled.

3. Complete the program and test it. For each word (each key in the map), output it, followed by the words that appear near it.