Collection Classes (Part 2: Maps, Sets, and Lexicons)

Optional Movie

Martin Luther King, Jr.
“I Have a Dream”
Gates B-12
Monday, January 19
2:15 P.M.

Methods in the Map Classes

- A map associates keys and values.
- The Stanford library offers two flavors of maps—Map and HashMap—both of which implement the following methods:

  - `map.size()`
    - Returns the number of key/value pairs in the map.
  - `map.isEmpty()`
    - Returns true if the map is empty.
  - `map.put(key, value)` or `map[key] = value;`
    - Makes an association between key and value, discarding any existing one.
  - `map.get(key)` or `map[key]`
    - Returns the most recent value associated with key.
  - `map.containsKey(key)`
    - Returns true if there is a value associated with key.
  - `map.remove(key)`
    - Removes key from the map along with its associated value, if any.
  - `map.clear()`
    - Removes all key/value pairs from the map.

Using Maps in an Application

- Before going on to create new applications of maps, it seems worth going through the example from the text, which uses a map to associate three-letter airport codes with their locations.
- The association list is stored in a text file that looks like this:

  - ATL=Atlanta, GA, USA
  - ORD=Chicago, IL, USA
  - LHR=London, England, United Kingdom
  - HND=Tokyo, Japan
  - LAX=Los Angeles, CA, USA
  - CDG=Paris, France
  - DFW=Dallas/Ft Worth, TX, USA
  - FRA=Frankfurt, Germany

- The Airports.cpp program shows how to read this file into a Map<string,string>, where it can be more easily used.

Sample Program: Symbol Tables

A map is often called a symbol table when it is used in the context of a programming language, because it is precisely the structure you need to store variables and their values. For example, if you are working in an application in which you need to assign floating-point values to variable names, you could do so using a map declared as follows:

```cpp
Map<string, double> symbolTable;
```

Write a C++ program that declares such a symbol table and then reads in command lines from the user, which must be in one of the following forms:

- A simple assignment statement of the form var = number.
- A variable alone on a line, which is a request to display its value.
- The command list, which lists all the variables.
- The command quit, which exits from the program.

The Range-Based for Statement

- One of the common operations that clients need to perform when using a collection is to iterate through the elements.
- While it is easy to implement iteration for vectors and grids using for loops, it is less clear how you would do the same for other collection types. The modern approach to solving this problem is to use a general tool called an iterator that delivers the elements of the collection, one at a time.
- C++11 uses a range-based for statement to simplify iterators:

  ```cpp
  for (string key : map) {
    // code to process that key...
  }
  ```
Methods in the \texttt{Set<type>} Class

- \texttt{set.size()}: Returns the number of elements in the set.
- \texttt{set.isEmpty()}: Returns true if the set is empty.
- \texttt{set.add(value)}: Adds \texttt{value} to the set.
- \texttt{set.remove(value)}: Removes \texttt{value} from the set.
- \texttt{set.contains(value)}: Returns true if the set contains the specified value.
- \texttt{set.clear()}: Removes all words from the set.
- \texttt{s1.isSubsetOf(s2)}: Returns true if \texttt{s1} is a subset of \texttt{s2}.
- \texttt{set.first()}: Returns the first element of the set in the ordering specified by the value type.

Methods in the \texttt{Lexicon} Class

- \texttt{lexicon.size()}: Returns the number of words in the lexicon.
- \texttt{lexicon.isEmpty()}: Returns true if the lexicon is empty.
- \texttt{lexicon.add(word)}: Adds \texttt{word} to the lexicon, always in lowercase.
- \texttt{lexicon.addWordsFromFile(filename)}: Adds all the words in the specified file to the lexicon.
- \texttt{lexicon.contains(word)}: Returns true if the lexicon contains the specified word.
- \texttt{lexicon.containsPrefix(prefix)}: Returns true if the lexicon contains any word beginning with \texttt{prefix}.
- \texttt{lexicon.clear()}: Removes all words from the lexicon.

Why Do Both \texttt{Lexicon} and \texttt{Set} Exist?

- The \texttt{Lexicon} representation is extremely space-efficient. The data structure used in the library implementation stores the full English dictionary in 350,000 bytes, which is shorter than a text file containing those words.
- The underlying representation makes it possible to implement a \texttt{containsPrefix} method that is useful in many applications.
- The representation makes it easy for iterators to process the words in alphabetical order.

Exercise: Finding “S” Hooks

- In Scrabble, one of the most important strategic principles is to conserve your \texttt{S} tiles so that you can hook longer words (ideally, the high-scoring seven-letter plays called \textit{bingos}) onto existing words.
- Some years ago, I was in a hotel where the shower taps were prominently labeled with \texttt{HOT} and \texttt{COLD}:

```
• Being a Scrabble player, it happened to occur to me that each of these words takes an \texttt{S} on either end, making them ideally flexible for Scrabble plays.
• Write a C++ program that finds all such words.
```

Iterator Order

- When you look at the documentation for an iterator, one of the important things to determine is whether the collection class specifies the order in which elements are generated. The Stanford C++ libraries make the following guarantees:
  - Iterators for arrays operate in index order.
  - Iterators for grids operate in \textit{row-major order}, which means that the iterator runs through every element in row 0, then every element in row 1, and so on.
  - Iterators for the \texttt{Map} class deliver the keys in the order imposed by the standard comparison function for the key type; iterators for the \texttt{HashMap} class return keys in a seemingly random order.
  - Iterators for the \texttt{Set} class deliver the elements in the order imposed by the standard comparison function for the value type; the \texttt{HashSet} class is unordered.
  - Iterators for lexicons always deliver words in alphabetical order.

Challenge for Next Time: Anagrams

- Write a program that reads in a set of letters and sees whether any anagrams of that set of letters are themselves words:

```
  Enter tiles:
  ehrsyz
  zephyrs
  Enter tiles:
  aeinstr
  anestri
  nastier
  ratines
  retains
  retinas
  retsina
  stainer
  stearin
```

- Generating all anagrams of a word is not a simple task. Most solutions require some tricky recursion, but can you think of another way to solve this problem? Hint: What if you had a function that sorts the letters in a word. Would that help?