Collection Classes

(Part 2: Maps, Sets, and Lexicons)

Eric Roberts
CS 106B
January 14, 2015

Optional Movie

And so even though we face the difficulties of today and tomorrow, I still have a dream. It is a dream deeply rooted in the American dream.

I have a dream that one day this nation will rise up and live out the true meaning of its creed: "We hold these truths to be self-evident, that all men are created equal."

Martin Luther King, Jr.
"I Have a Dream"

Location TBA
Monday, January 19
2:15 P.M.

Outline

1. The **`Map`** class
2. Using maps in an application
3. Symbol tables
4. The range-based **`for`** loop
5. The **`Set`** class
6. The **`Lexicon`** class
7. Exercises

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

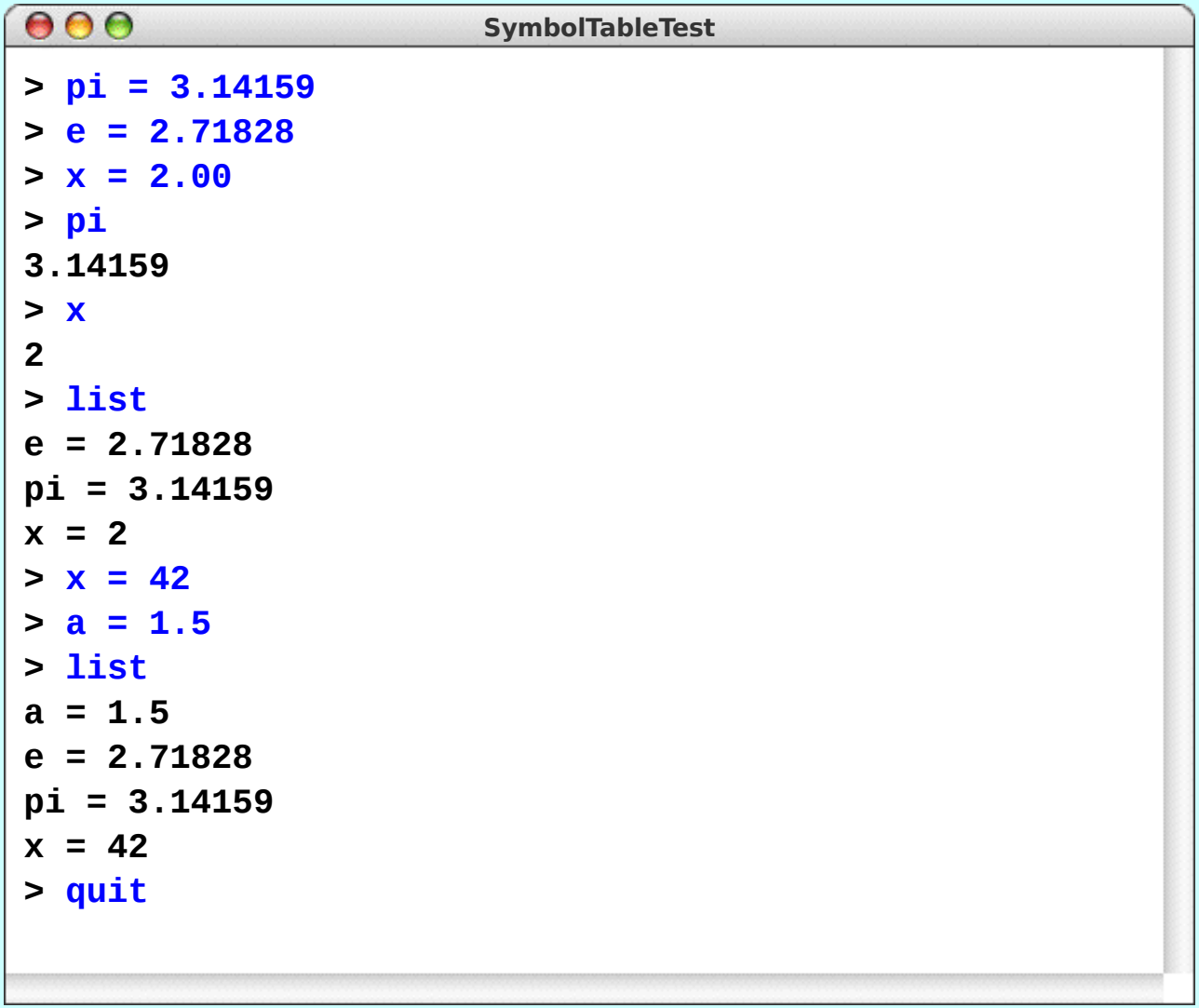
A map is often called a ***symbol table*** when it is used in the context of a programming language, because it is precisely the structure you need to store variables and their values. For example, if you are working in an application in which you need to assign floating-point values to variable names,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

```
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.

Symbol Table Sample Run



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:

```
for (string key : map) {
    . . . code to process that key . . .
}
```

Methods in the **`Set<`***type*`>`** Class

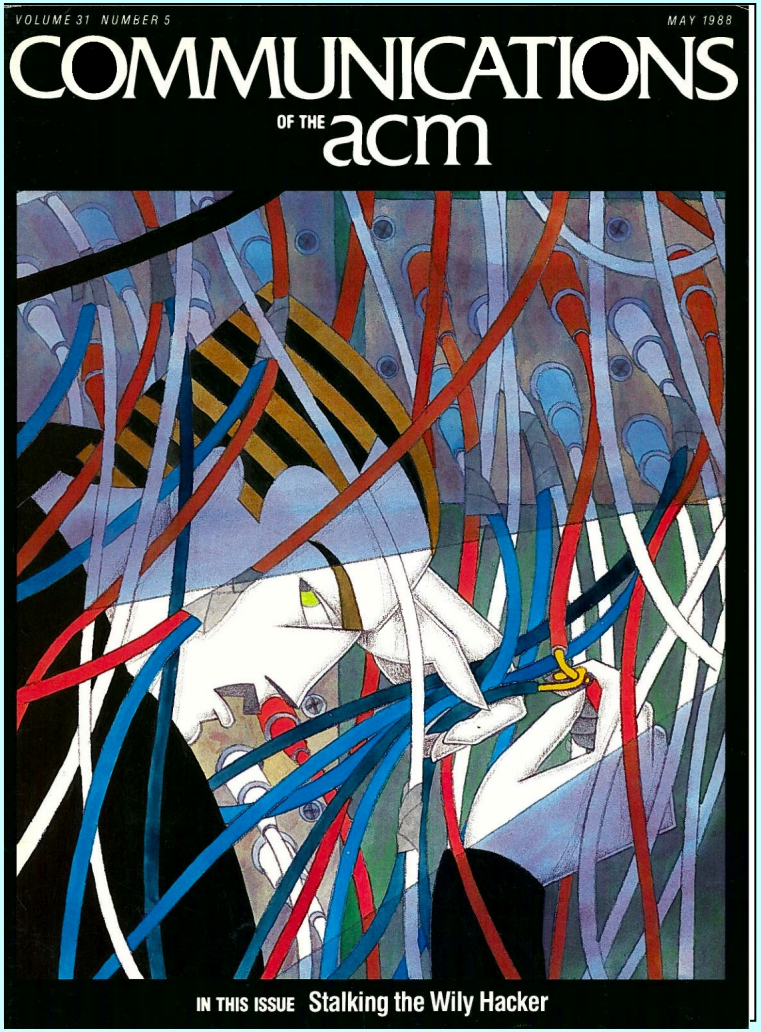
`set.size()` Returns the number of elements in the set.
`set.isEmpty()` Returns **`true`** if the set is empty.
`set.add(value)` Adds **`value`** to the set.
`set.remove(value)` Removes **`value`** from the set.
`set.contains(value)` Returns **`true`** if the set contains the specified value.
`set.clear()`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

Methods in the **`Lexicon`** Class

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

Why Do Both **`Lexicon`** and **`Set`** Exist?

- The **`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 **`containsPrefix`** method that is useful in many applications.
- The representation makes it easy for iterators to process the words in alphabetical order.

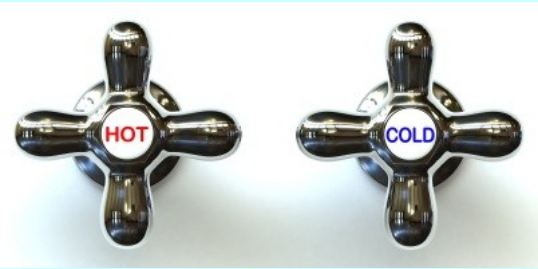


Iterator Order

- 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 ***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 **`Map`** class deliver the keys in the order imposed by the standard comparison function for the key type; iterators for the **`HashMap`** class return keys in a seemingly random order.
  - Iterators for the **`Set`** class deliver the elements in the order imposed by the standard comparison function for the value type; the **`HashSet`** class is unordered.
  - Iterators for lexicons always deliver words in alphabetical order.

Exercise: Finding “S” Hooks

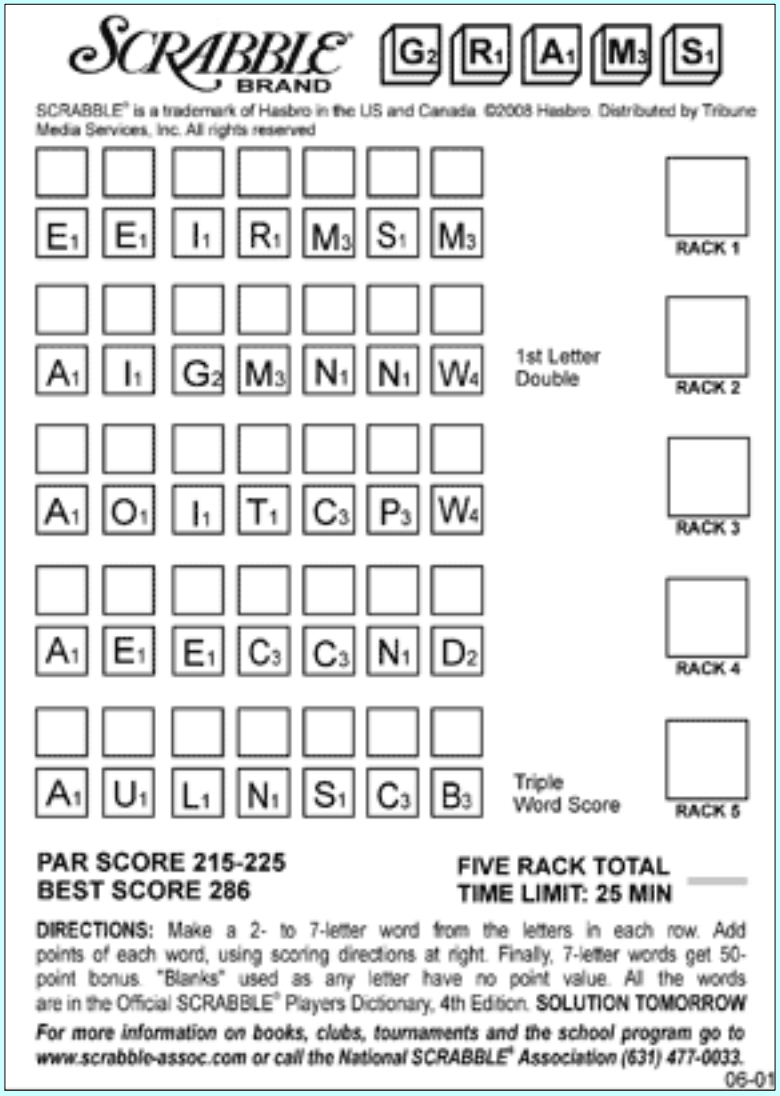
- In Scrabble, one of the most important strategic principles is to conserve your S tiles so that you can hook longer words (ideally, the high-scoring seven-letter plays called ***bingos***) onto existing words.
- Some years ago, I was in a hotel where the shower taps were prominently labeled with HOT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



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

Download: **FindSHooks.cpp**

Challenge for Next Time: Anagrams



SCRABBLE BRAND G2 R1 A1 M3 S1

E1 E1 I1 R1 M3 S1 M3		RACK 1
A1 I1 G2 M3 N1 N1 W4	1st Letter Double	RACK 2
A1 O1 I1 T1 C3 P3 W4		RACK 3
A1 E1 E1 C3 C3 N1 D2		RACK 4
A1 U1 L1 N1 S1 C3 B3	Triple Word Score	RACK 5

PAR SCORE 215-225
BEST SCORE 286

FIVE RACK TOTAL _____
TIME LIMIT: 25 MIN

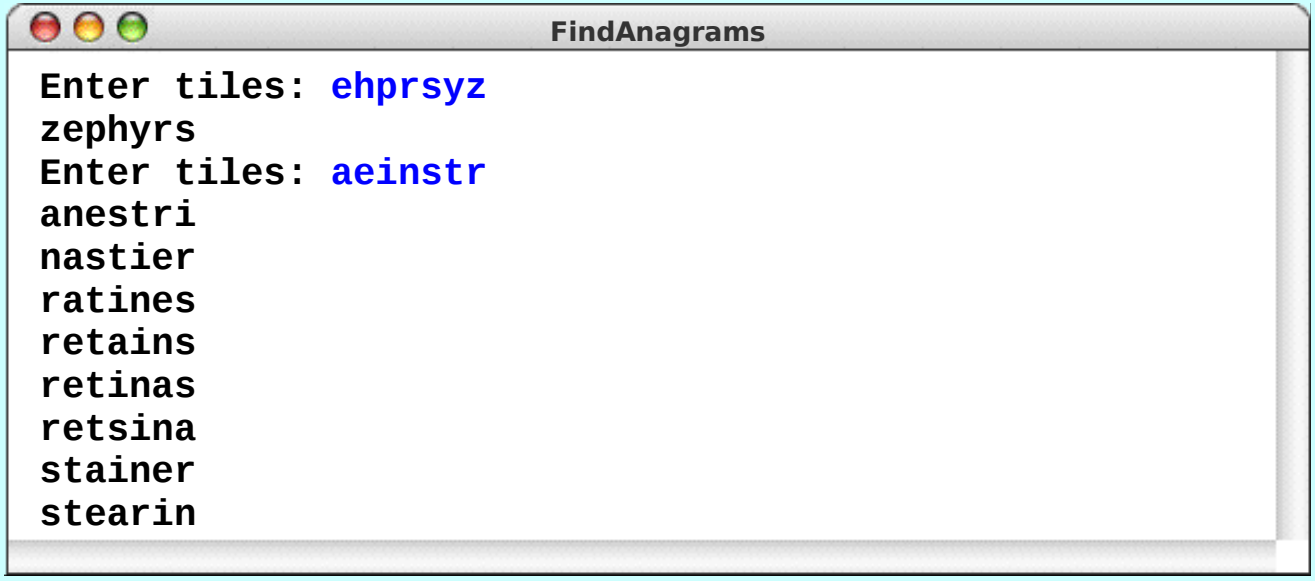
DIRECTIONS: Make a 2- to 7-letter word from the letters in each row. Add points of each word, using scoring directions at right. Finally, 7-letter words get 50-point bonus. "Blanks" used as any letter have no point value. All the words are in the Official SCRABBLE® Players Dictionary, 4th Edition. **SOLUTION TOMORROW**

For more information on books, clubs, tournaments and the school program go to www.scrabble-assoc.com or call the National SCRABBLE® Association (631) 477-0033.

06-01

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:



- 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?

The End