Collections, Part Three
Friday Four Square!
Today at 4:15PM, Outside Gates
Lexicon
Lexicon

- A **Lexicon** is a container that stores a collection of words.
- No definitions are associated with the words; it is a “lexicon” rather than a “dictionary.”
- Contains operations for
  - Checking whether a word exists.
  - Checking whether a string is a prefix of a given word.
Tautonyms

• A **tautonym** is a word formed by repeating the same string twice.
  • For example: murmur, couscous, papa, etc.

• What English words are tautonyms?
foreach

- You can loop the elements of any collection class using the `foreach` macro:

```cpp
foreach (type var in collection) {
  /* ... do something with var ... */
}
```

- `foreach` is *not* a part of standard C++; it's a *macro* that we've built to keep things simple.
Some Aa
One Bulbul

http://travel.paintedstork.com/blog/image/yellow_browed_bulbul.jpg
More than One Caracara
Introducing the Dikdik
Anagrams

• Two phrases are **anagrams** of one another if they have the same letters, but in a different order.

• Examples:
  • Stanford University → A Trusty Finned Visor
  • Keith Schwarz → Zither Whacks
  • Dawson Zhou → Whoa! Zounds!

• **Question**: Given an English word, can we find all anagrams of that word?
Anagram Clusters

• An **anagram cluster** is a set of words that are all anagrams of one another.

  stop ↔ tops ↔ pots ↔ spot ↔ opts ↔ post

• If we want to find all anagrams of a word, we can find its anagram cluster, then list off all the words in that cluster.

• Two questions:
  • How do we store an anagram cluster?
  • How do we find the anagram cluster associated with a given word?
Set
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
Set

• The **Set** represents an unordered collection of distinct elements.

• Elements can be added and removed, and you can check whether or not an element exists.
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
Operations on Sets

- You can add a value to a set by writing
  \[ \text{set} += \text{value}; \]
- You can remove a value from a set by writing
  \[ \text{set} -= \text{value}; \]
- You can check if a value exists by writing
  \[ \text{set}.\text{contains} (\text{value}) \]
- Many more operations available (union, intersection, difference, subset, etc.), so be sure to check the documentation.
Map
Map

- The **Map** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.
Map

- The **Map** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.
Map

- The *Map* class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

<table>
<thead>
<tr>
<th>CS106B</th>
<th>Awesome!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dikdik</td>
<td>Cute!</td>
</tr>
</tbody>
</table>
Map

- The **Map** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.
Map

- The **Map** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CS106B</strong></td>
<td><strong>Awesome!</strong></td>
</tr>
<tr>
<td><strong>Dikdik</strong></td>
<td><strong>Very Cute!</strong></td>
</tr>
<tr>
<td><strong>This Slide</strong></td>
<td><strong>Self Referential</strong></td>
</tr>
</tbody>
</table>
Using the Map

• You can create a map by writing

```java
Map<KeyType, ValueType> map;
```

• You can add or change a key/value pair by writing

```java
map[key] = value;
```

If the key doesn't already exist, it is added.

• You can read the value associated with a key by writing

```java
map[key]
```

If the key doesn't exist, it is added and associated with a default value.

• You can check whether a key exists by calling

```java
map.containsKey(key)
```
Sorting Letters

• One way to check whether two words are anagrams of one another is to reorder the letters into ascending order:

  bleat → abelt
  table → abelt
Sorting Letters

- One way to check whether two words are anagrams of one another is to reorder the letters into ascending order:
  
  bleat → abelt
  
  table → abelt

- **Idea**: Build a `Map<string, Set<string>>` to represent anagram clusters.
  - Each key is the letters of a word in sorted order.
  - Each value is the set of all words with those letters.
Ordering in foreach

- When using `foreach` to iterate over a collection:
  - In a `Vector`, `string`, or array, the elements are retrieved in order.
  - In a `Map`, the `keys` are returned in sorted order.
  - In a `Set` or `Lexicon`, the values are returned in sorted order.
  - In a `Grid`, the elements of the first row are returned in order, then the second row, etc. (this is called `row-major order`).
Counting Sort
Counting Sort

banana
Counting Sort

banana

Map<char, int>
Counting Sort

Map<char, int>
Counting Sort

Map<char, int>
Counting Sort

\[
\begin{array}{cc}
\text{a} & 1 \\
\text{b} & 1 \\
\end{array}
\]

Map<char, int>
Counting Sort

Map<char, int>
Counting Sort

Map<char, int>

banana

Map<char, int>
Counting Sort

Map<char, int>
Counting Sort

Map<char, int>

bananana

Map<char, int>:

<table>
<thead>
<tr>
<th>a</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
</tr>
</tbody>
</table>
Counting Sort

Map<char, int>

banana

a 2
b 1
n 1
Counting Sort

Map<char, int>

banana

Map<char, int>

a  2
b  1
n  2
Counting Sort

Map<char, int>
Counting Sort

Map<char, int>

banana

a 3
b 1
n 2
Counting Sort

Map<char, int>

b a n a n a

a 3
b 1
n 2

Map<char, int>
Counting Sort

Map<char, int>
Counting Sort

Map<char, int>

banana

Map<char, int>

| a | 3 |
| b | 1 |
| n | 2 |

aaa
Counting Sort

Map<char, int>
Counting Sort

Map<char, int> banana

a 3
b 1
n 2

aaab
Counting Sort

Map<char, int>

banana

Map<char, int>
Counting Sort

Map<char, int>

banana

\[
\begin{array}{|c|c|}
\hline
\text{a} & 3 \\
\hline
\text{b} & 1 \\
\hline
\text{n} & 2 \\
\hline
\end{array}
\]

a a a a b n n n
Counting Sort

Map<char, int>

banana

\[
\begin{array}{ccc}
\text{a} & \text{3} \\
\text{b} & \text{1} \\
\text{n} & \text{2} \\
\end{array}
\]

aaaabbnnn
Next Time

• **Queue**
  • A data structure for waiting lines.

• **Password Security**
  • How do you properly store passwords?
  • And what on earth is a hash code?