Collections, Part Three
Announcements

• Two handouts online
  • Assignment 2: Fun with Collections
  • Section Handout
Announcements

- Section assignments finalized ~5pm tonight.
- No class on Thursday
- No sections on Thursday. You have two options:
  - Go to the SCPD section in person
  - Watch SCPD section online
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?
Some Aa
One Bulbul

http://travel.paintedstork.com/blog/image/yellow_browed_bulbul.jpg
More than One Caracara
Introducing the Dikdik
tautonyms (Pseudocode)
foreach

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

```c++
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.
tautonyms.cpp
(On Computer)
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.
  
  \[
  \text{stop} \leftrightarrow \text{tops} \leftrightarrow \text{pots} \leftrightarrow \text{spot} \leftrightarrow \text{opts} \leftrightarrow \text{post}
  \]

• 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.
Set

Set<int> numbers;

numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;
Set

Set<int> numbers;

numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;
Set

Set<int> numbers;

numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;
Set

Set<int> numbers;

numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;
```cpp
Set<int> numbers;
numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;
```

Set

Set<int> numbers;

numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;
Set

Set<int> numbers;
numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;

137
2718
Set

Set<int> numbers;

numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;
Set<int> numbers;
numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;
Set

Set<int> numbers;
numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;
numbers -= 42;
Set

Set<int> numbers;

numbers += 137;

numbers += 2718;

numbers += 42;

numbers += 42;

numbers -= 42;

42 already in numbers, no changes.
Set

Set<int> numbers;

numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;
Set

Set<int> numbers;
numbers += 137;
numbers += 2718;
numbers += 42;
numbers += 42;
numbers -= 42;
Anagram Clusters

• We can store each anagram cluster as a `Set<string>`.

• We still need a way of associating words to anagram clusters.
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.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CS106B</td>
<td>Awesome!</td>
</tr>
<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.

<table>
<thead>
<tr>
<th>CS106B</th>
<th>Awesome!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dikdik</td>
<td>Cute!</td>
</tr>
<tr>
<td>Djikstra</td>
<td>Pathfinding</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)
  ```
Anagram Clusters

- We can use `Map<string, Set<string> >` to match strings to anagram clusters
  - Key: Some sort of unique identifier for each anagram cluster
  - Value: Set of words in the anagram cluster
- What should we use for the key? How can we uniquely identify an anagram cluster?
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.
Counting Sort
Counting Sort

banana
Counting Sort

banana

Map<char, int>
Counting Sort

Map<char, int>
Counting Sort

Map<char, int>
Counting Sort

Map<char, int>

banana

b 1
Counting Sort

Map<char, int>

banana

Map<char, int>
Counting Sort

Map<char, int>
Counting Sort

Map<char, int>

banana

Map<char, int>

a 1
b 1
n 1
Counting Sort

Map<char, int>

banana

Map<char, int>

a 1
b 1
n 1
Counting Sort

Map<char, int>

banana

Map<char, int>
Counting Sort

Map<char, int>

banana

Map<char, int>
Counting Sort

Map<char, int>

b a n a n a
Counting Sort

Map<char, int>
Counting Sort

Map<char, int>

banana

Map

a 3
b 1
n 2
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

banana

Map<char, int>

<table>
<thead>
<tr>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
</tr>
</tbody>
</table>
Counting Sort

Map<char, int>

banana

Map<char, int>
Counting Sort

Map<char, int>

banana

<table>
<thead>
<tr>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
</tr>
</tbody>
</table>

aaa
Counting Sort

Map<char, int>
Counting Sort

Map<char, int>

banana

Map:

<table>
<thead>
<tr>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
</tr>
</tbody>
</table>

Result:

aab
Counting Sort

Map<char, int>
Counting Sort

Map<char, int>

banana

Map<char, int>

a 3
b 1
n 2

aaabnnn
Counting Sort

Map<char, int>

banana

Map<char, int>

a 3
b 1
n 2

aaabbnn
sort()
anagram-clusters.cpp
(On Computer)
anagram-clusters
(Pseudocode)
anagram-clusters.cpp

(Computer)
**foreach**

- Friends don't let friends modify a collection when using `foreach` to iterate over its elements
  - Will cause your program to crash.

```csharp
Set<int> s;
    s += 1; s += 2;
foreach (int i in s) {
    s.remove(i); //ERROR!!!
}
```
Lexicon or Set\(<\text{string}>\)?

- Both the Lexicon and Set\(<\text{string}>\) can be used to represent a collection of strings. So which should you use?
- It turns out that the Lexicon is better for storing very large collections of strings that don't change over time
  - Like words in a language
- Set\(<\text{string}>\) are much more general purpose.
  - We'll find out why in a couple weeks!
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?