Programming Abstractions

CS106X

Cynthia Lee

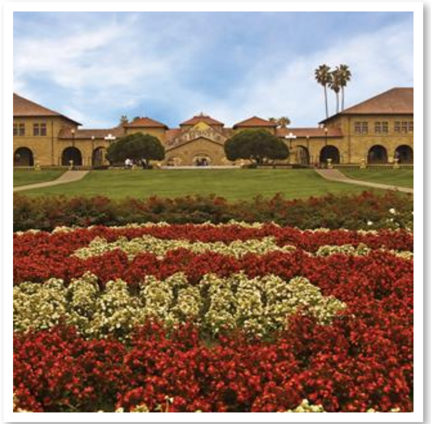
Today’s Topics

ADTs

- Map
  - Example: counting words in text
- Containers within containers
  - Example: reference tests
  - Example: anagram finder

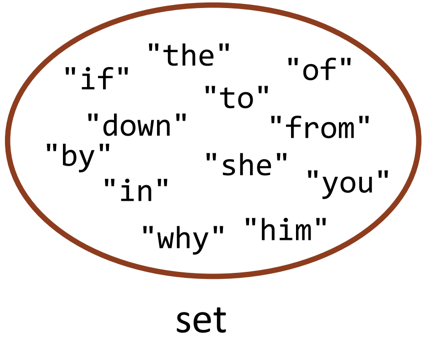
Map

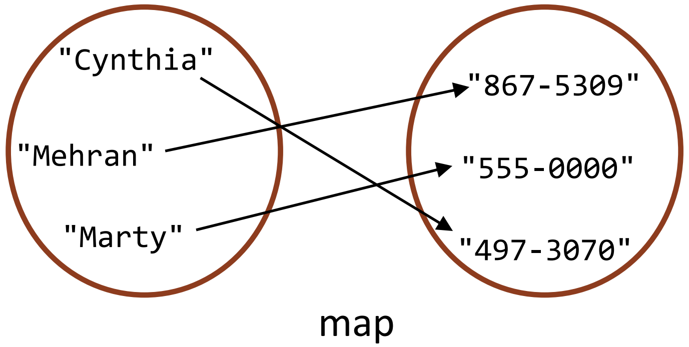
WHAT ARE THEY?
EXAMPLE APPLICATION



Associative containers

- Map
- Set
- Lexicon





Not as concerned with order but with matching

- Set: associates **keys** with **membership** (yes or no)
- Map: associates **keys** with **values** (could be any type)

Stanford library Map *(selected member functions)*

```
template <typename KeyType, typename ValueType> class Map {
public:
    void add(const KeyType& key, const ValueType& value);

    bool containsKey(const KeyType& key) const;

    ValueType get(const KeyType& key) const;

    ValueType operator [](const KeyType& key) const;
...
}
```

Map programming exercise

Write a program to count the number of occurrences of each unique word in a text file (e.g. *Poker* by Zora Neale Hurston).

- Report all words that appeared in the book at least 10 times, in alphabetical order
- Allow the user to type a word and report *how many times* that word appeared in the book

What would be a good design for this problem?

A. Map<int, string> wordCounts;
B. Map<string, Vector<string>> wordCounts;
[✓ C. Map<string, int> wordCounts;]
D. Map<string, Vector<int>> wordCounts;
E. Other/none/more

Stanford University

Write a program to count the number of occurrences of each unique word in a text file (e.g. *Poker* by Zora Neale Hurston).

```
Map<string,int> wordCounts;
string word;
infile >> word;
while (!infile.fail()){
    //record count here
    infile >> word;
}
```

How can we record the count?

A. `wordCounts[word]+=word;`
B. `wordCounts[word]+=1;` ✓
[✓ C.] `wordCounts[word]++;`
D. B and C are good, but you need to first detect new (never seen before) words so you can start at zero before you start adding +1
E. Other/none/more

wordcounts ["hello"] = 10;

Stanford University

Write a program to count the number of occurrences of each unique word in a text file (e.g. *Poker* by Zora Neale Hurston).

- Report all words that appeared in the book at least 10 times, in alphabetical order

```
cout << “Most common words:" << endl;
for (string word : wordCounts){
   if (wordCounts[word] >= 10){
      cout << word << "\t“;
      cout << wordCounts[word] << endl;
   }
}
```

New (C++11) useful tool!
for loop that iterates over all elements of a container class

Does this work for our alphabetical use case?

- Yes!
- Stanford library Map returns its keys in sorted order

Compound Containers

IT'S TURTLES ALL THE WAY DOWN...

Compound containers

```
Map<string,Vector<int>> mymap;

Vector<int> numbers;
numbers.add(1);
numbers.add(2);
numbers.add(3);

mymap["123"] = numbers;
Vector<int> test = mymap["123"];
test.add(4);
cout << “New size: " << mymap["123"].size() << endl;
```

Predict the outcome:

(A) 3 (B) 4 (C) other # (D) Error

Compound containers

```
Map<string,Vector<int>> mymap;

Vector<int> numbers;
numbers.add(1);
numbers.add(2);
numbers.add(3);

mymap["123"] = numbers;
mymap["123"].add(4);
cout << “New size: " << mymap["123"].size() << endl;
```

Predict the outcome:

(A) 3 (B) 4 (C) other # (D) Error

C++ bonus details:

This works by <u>returning</u> a reference (!)

C++ also allows you to define a return type to be a reference

Gives you a *reference* to the item being returned

In the case of map, this returns a *reference* to the value at map[key]:

```
ValueType & operator[](const KeyType & key);
```

Stanford library Map *(selected member functions)*

```
template <typename KeyType, typename ValueType> class Map {
public:
    void add(const KeyType& key, const ValueType& value);

    bool containsKey(const KeyType& key) const;

    ValueType get(const KeyType& key) const;

    ValueType operator [](const KeyType& key) const;

    ValueType& operator [](const KeyType& key);
...
private:
```

Redacted…until the second half of the quarter!

```
}
```

Returning a reference

```
Map<string,Vector<int>> mymap;

Vector<int> numbers;
numbers.add(1);
numbers.add(2);
numbers.add(3);

mymap["123"] = numbers;

Vector<int>& referenceTest = mymap["123"];
referenceTest.add(4);

cout << “New size: " << mymap["123"].size() <<
endl;
```

Predict the outcome:

(A) 3 (B) 4 (C) other # (D) Error

Anagram Finder

AN APPLICATION OF COMPOUND MAP

“Abstractions”

Bacon artists
Cab stain rots
Crab in toasts
Bonsai tracts
…

http://www.wordsmith.org/anagram/

What would be a good design for this problem?

Concept:

- Unlike the website, we will only show anagrams that are 1 word ↔ 1 word (“moored” ↔ “roomed”, not “abstractions” ↔ “bacon artists”)
- Have a string that is a “representative” of a group of words that are anagrams of each other
- Have that string map to a list of those words
- `Map<string, Vector<string>> anagrams;`

- Key trick idea: the representative is the string with the letters sorted (use a function “`string sortWord(string word);`”)
  - *moored* becomes *demoor*
  - *roomed* becomes *demoor*

What would be a good design for this problem?

Concept:

- `Map<string, Vector<string>> anagrams;`

How would we add a word stored in the string variable `word` to our collection?

A. `anagrams[word]+=word;`

B. `anagrams[word]+=sortWord(word);`

C. `anagrams[sortWord(word)]+=word;`

D. `anagrams[sortWord(word)]+=sortWord(word);`

E. Other/none/more

What would be a good design for this problem?

Concept:

- `Map<string, Vector<string>> anagrams;`

To add a word to our collection:

```
anagrams[sortWord(word)]+=word;
```

To look up a word in our collection to find its anagrams:

```
Vector<string> matches = anagrams[sortWord(query)];
```