# CS 106B

## Lecture 6: Sets and Maps

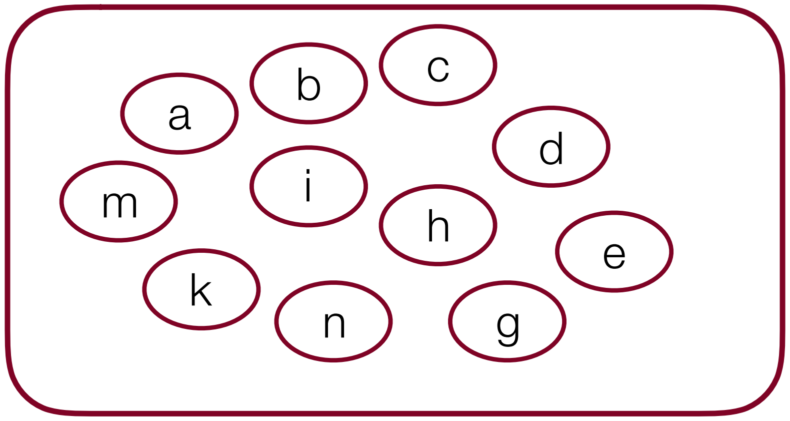
Friday, April 14, 2017

Programming Abstractions
Spring 2017
Stanford University
Computer Science Department

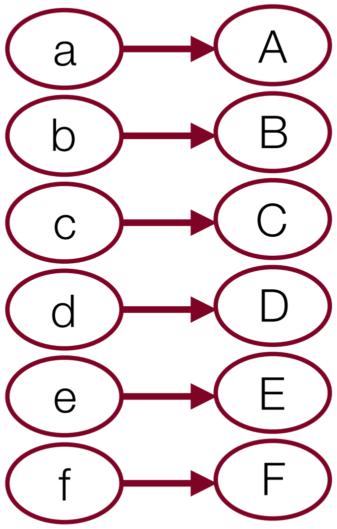
Lecturer: Chris Gregg

reading:
Programming Abstractions in C++, Chapter 5.4-5.6

Set:



Map:



## Today's Topics

- Logistics:
  - Tiny Feedback: some responses!
    - Not enough motivation for why we care about ADTs: good point!
    - More interactive classes: I'll see what I can do!
    - KEY pages ( 🔑 )

    - Late credits change: *up to two calendar days equals one late credit.* You get three automatic late credits per quarter.

- the "const" qualifier
- Postfix refresher
- Structs (details will come later!)
- Sets
- Maps

## const

- When we pass variables by reference into a function, we do so for a couple of reasons:
  - We don't want to make copies of big objects
    - As it turns out (thanks to the person who put a note on sayat.me/chrisgregg who reminded me to mention this), C++ has new functionality that allows us to return big objects in some cases without lots of copying (but the Stanford libraries don't have that functionality yet)

and / or

  - We need to modify an object in place (we will do this a great deal with recursion)

## const

- What if we want to pass a variable by reference, but we *know* we won't modify it?

  - We could just have self-control and not modify it.

  - Or, we could make the compiler keep us honest. To do this, we use the keyword **`const`**

## const

- **`const`** allows a programmer to tell the compiler that the object passed cannot be changed in the function. E.g.,

```
void printLifeGrid(Grid<char> const &lifeGrid);
```

- There is no need for the `printLifeGrid()` function to change the lifeGrid, but we would rather pass the grid by reference to avoid big copies.

## const

- We use `const` to tell the compiler to give us an error if we do try to modify a const-declared variable in a function.
- This *also* tells someone reading our code that we are guaranteeing that the object will be the same when the function ends as when it began.

## Postfix (RPN) Refresher

What does the following postfix (RPN) computation equal?

`10 3 5 * 9 4 - / +`

Feel free to use our stack algorithm:

**Read the input and push numbers onto a stack until you reach an operator.**
**When you see an operator, apply the operator to the two numbers that are popped from the stack.**
**Push the resulting value back onto the stack.**
**When the input is complete, the value left on the stack is the result.**

Answer: **`13`**

How would our stack-based RPN know that we had made an error, e.g.,
`10 3 5 * - + 9 4 -`

Answer: the stack is empty when we try to pop two operands

## Brief Introduction to Structs

Recall that in C++, we can only return one value from a function. We have overcome this in the past by using references:

```
/*
 * Solves a quadratic equation ax^2 + bx + c = 0,
 * storing the results in output parameters root1 and root2.
 * Assumes that the given equation has two real roots.
 */
void quadratic(double a, double b, double c,
               double& root1, double& root2) {
    double d = sqrt(b * b - 4 * a * c);
    root1 = (-b + d) / (2 * a);
    root2 = (-b - d) / (2 * a);
}
```

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

## Brief Introduction to Structs

- There is another way we can return variables by packaging them up in a type called a "struct"
- Structs are a way to *define a new type* for us to use.
- Once we define a struct, we can use that type anywhere we would normally use another type (e.g., an **`int`**, **`double`**, **`string`**, etc.)

new type name

```
struct Roots {
    double root1;
    double root2;
};
```

struct variables, referred to with dot notation

don't forget the semicolon

## Brief Introduction to Structs

- Let's re-write our quadratic equation solver to use the Roots struct.

## Brief Introduction to Structs

- Let's re-write our quadratic equation solver to use the Roots struct.

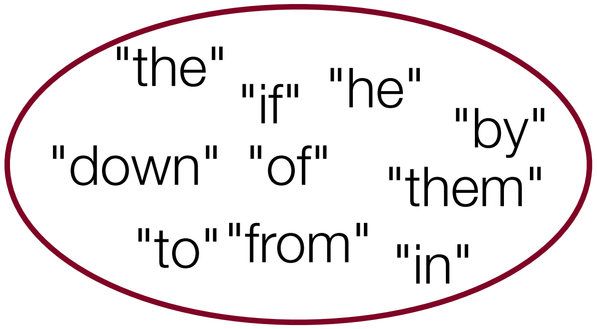
```
struct Roots {
    double root1;
    double root2;
};

/*
 * Solves a quadratic equation ax^2 + bx + c = 0,
 * storing the results in output parameters root1 and root2.
 * Assumes that the given equation has two real roots.
 */
Roots quadratic(double a, double b, double c) {
    Roots roots;
    double d = sqrt(b * b - 4 * a * c);
    roots.root1 = (-b + d) / (2 * a);
    roots.root2 = (-b - d) / (2 * a);
    return roots;
}
```

# Sets and Maps

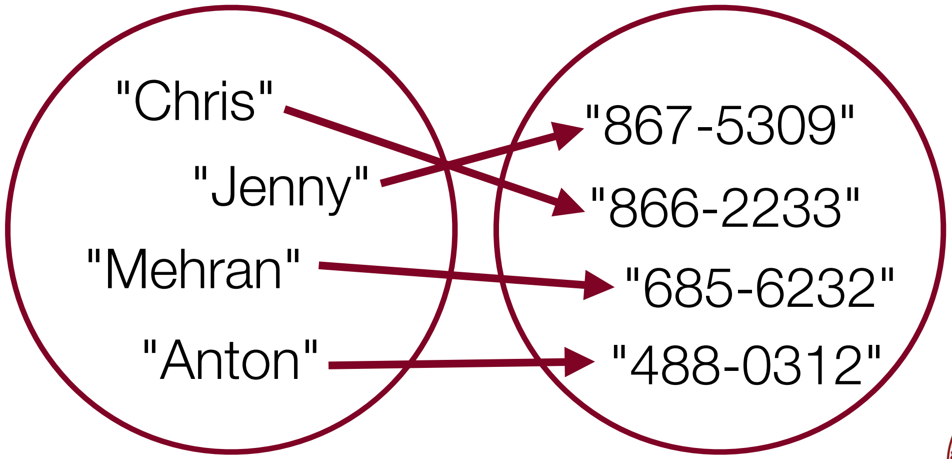
## Sets

- Collection of elements with *no duplicates*.



## Maps

- Collection of key/value pairs
- The key is used to find its associated value.

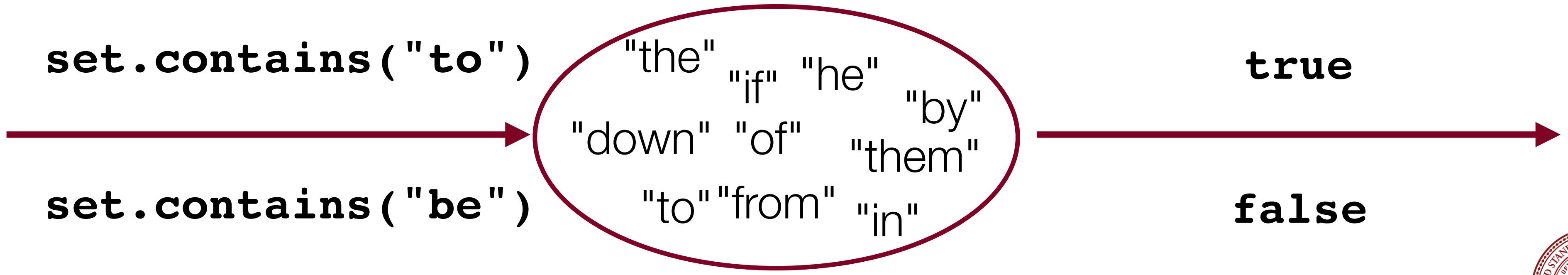


# Sets

🔑 • **set**: a collection of elements with no duplicates.

Operations include `add`, `contains`, and `remove`, and they are all fast

Sets *do not* have indexes



## Sets: Simple Example

```
Set<string> friends;
friends.add("chris");
friends.add("anton");
cout << friends.contains("voldemort") << endl;
for(string person : friends) {
    cout << person << endl;
}
```

## Set Essentials

| |
|---|
| **`int set.size()`**<br>Returns the number of elements in the set. |
| **`void set.add(value)`**<br>Adds the new value to the set (ignores it if the value is already in the set) |
| **`bool set.contains(value)`**<br>Returns `true` if the value is in the set, `false` otherwise. |
| **`void set.remove(value)`**<br>Removes the value if present in the set. Does not return the value. |
| **`bool set.isEmpty()`**<br>Returns `true` if the set is empty, `false` otherwise. |

Sets also have other helpful methods. See the online docs for more.

## Looping Over a Set

```
for(type currElem : set) {
    // process elements one at a time
}
```

can't use a normal `for` loop and get each `element [i]`

```
for(int i=0; i < set.size(); i++) {
    // does not work, no index!
    cout << set[i];
}
```

## Types of Sets

### Set

Iterate over elements in *sorted* order

**REALLY FAST!**

`O(log n) per retrieval`

Implemented using a "binary search tree"

### HashSet

Iterate over elements in *unsorted (jumbled)* order

**REALLY, RIDICULOUSLY FAST!**

`O(1) per retrieval`

Implemented using a "hash table"

# Set Operands

Sets can be compared, combined, etc.

| | |
|---|---|
| `s1 == s2` | true if the sets contain exactly the same elements |
| `s1 != s2` | true if the sets don't contain the same elements |
| `s1 + s2` | returns the union of s1 and s2 (all elements in both) |
| `s1 * s2` | returns intersection of s1 and s2 (elements must be in both) |
| `s1 - s2` | returns difference of s1, s2 (elements in s1 but not s2) |

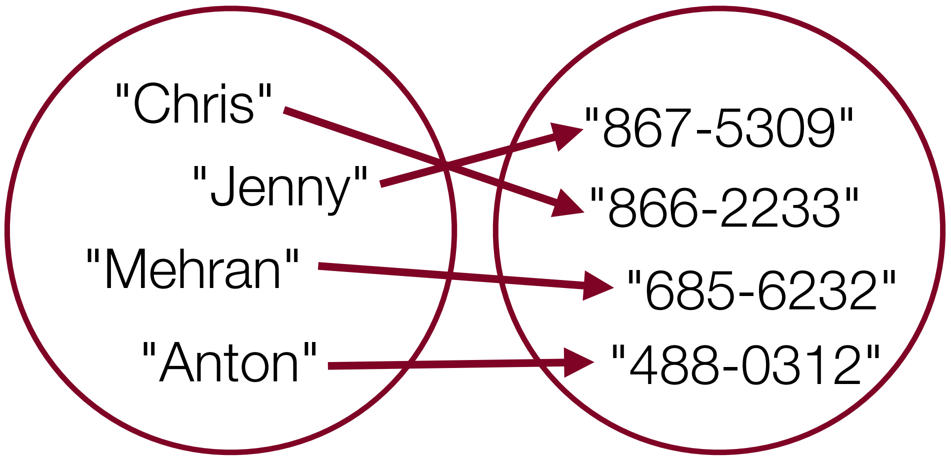
## Count Unique Words

# Maps

🔑 **map**: A collection of pairs (*k*, *v*), sometimes called **key/value** pairs, where *v* can be found quickly if you know *k*.

a.k.a. dictionary, associative array, hash

a generalization of an array, where the "indexes" need not be ints.

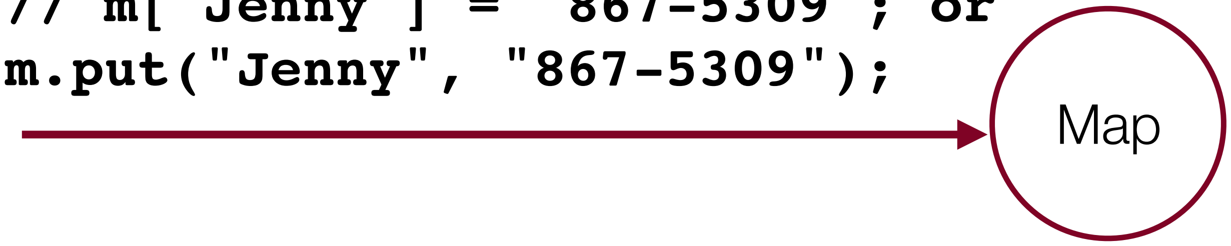


# Using Maps

A map allows you to get from one half of a pair to the other.

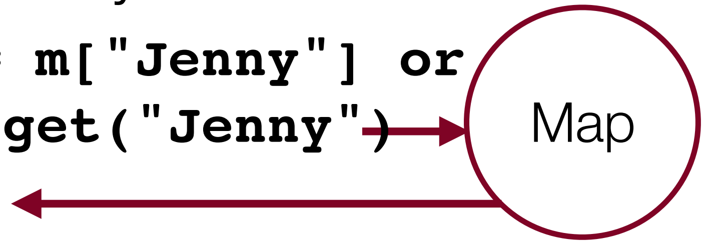
store an association from "Jenny" to "867-5309"

```
//      key            value
// m["Jenny"] = "867-5309"; or
m.put("Jenny", "867-5309");
```



What is Jenny's number?

```
// string ph = m["Jenny"] or
string ph = m.get("Jenny")
"206-685-2181"
```

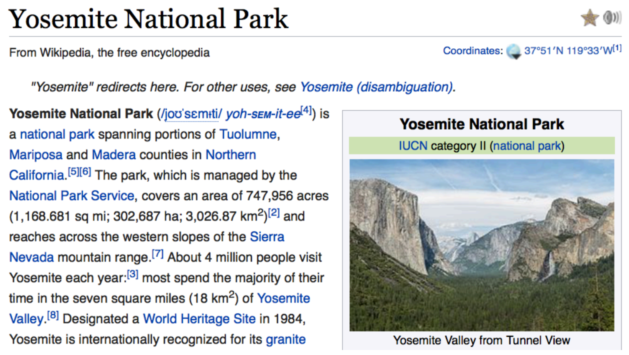


# Maps are Everywhere

key = title, value = article

key:
"Yosemite National Park"

value:



Yosemite National Park

From Wikipedia, the free encyclopedia

Coordinates: 37°51′N 119°33′W[1]

*"Yosemite" redirects here. For other uses, see Yosemite (disambiguation).*

**Yosemite National Park** (/joʊˈsɛmɪti/ *yoh-SEM-it-ee*[4]) is a national park spanning portions of Tuolumne, Mariposa and Madera counties in Northern California.[5][6] The park, which is managed by the National Park Service, covers an area of 747,956 acres (1,168.681 sq mi; 302,687 ha; 3,026.87 km$^2$)[2] and reaches across the western slopes of the Sierra Nevada mountain range.[7] About 4 million people visit Yosemite each year:[3] most spend the majority of their time in the seven square miles (18 km$^2$) of Yosemite Valley.[8] Designated a World Heritage Site in 1984, Yosemite is internationally recognized for its granite

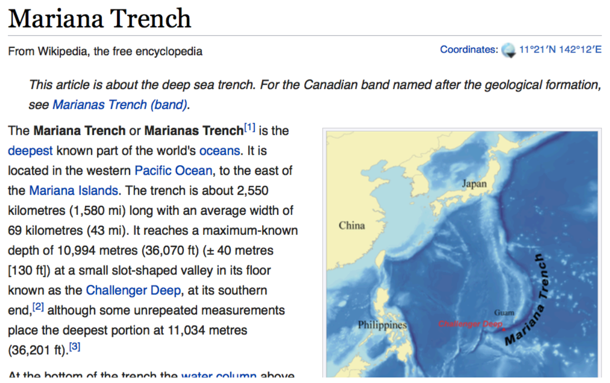
**Yosemite National Park**

IUCN category II (national park)

Yosemite Valley from Tunnel View

key:
"Mariana Trench"

value:



Mariana Trench

From Wikipedia, the free encyclopedia

Coordinates: 11°21′N 142°12′E

*This article is about the deep sea trench. For the Canadian band named after the geological formation, see Marianas Trench (band).*

The **Mariana Trench** or **Marianas Trench**[1] is the deepest known part of the world's oceans. It is located in the western Pacific Ocean, to the east of the Mariana Islands. The trench is about 2,550 kilometres (1,580 mi) long with an average width of 69 kilometres (43 mi). It reaches a maximum-known depth of 10,994 metres (36,070 ft) (± 40 metres [130 ft]) at a small slot-shaped valley in its floor known as the Challenger Deep, at its southern end,[2] although some unrepeated measurements place the deepest portion at 11,034 metres (36,201 ft).[3]

At the bottom of the trench the water column above

## Creating Maps

Requires 2 type parameters: one for keys, one for values.

```
// maps from string keys to integer values
Map<string, int> votes;

// maps from double keys to Vector<int> values
Map<string, Vector<string>> friendMap;
```

# Map Methods

| | |
|---|---|
| `m.clear();` | removes all key/value pairs from the map |
| `m.containsKey(key)` | returns `true` if the map contains a mapping for the given key |
| `m[key]` or<br>`m.get(key)` | returns the value mapped to the given key;<br>if key not found, **adds** it with a default value (e.g. 0, "") |
| `m.isEmpty()` | returns `true` if the map contains no k/v pairs (size 0) |
| `m.keys()` | returns a `Vector` copy of all keys in the map |
| `m[key] = value;` or<br>`m.put(key, value);` | adds a mapping from the given key to the given value;<br>if the key already exists, **replaces** its value with the given one |
| `m.remove(key);` | removes any existing mapping for the given key |
| `m.size()` | returns the number of key/value pairs in the map |
| `m.toString()` | returns a string such as `"{a:90, d:60, c:70}"` |
| `m.values()` | returns a `Vector` copy of all values in the map |

# Map Example

```
Map<string, string> wiki;

// adds name / text pair to dataset
wiki.put("Neopalpa donaldtrumpi", articleHTML);
```

SCIENCE JAN 17 2017, 9:43 PM ET

**Tiny Moth Named for President-Elect Donald Trump**

by MINDY WEISBERGER, LIVE SCIENCE

SHARE

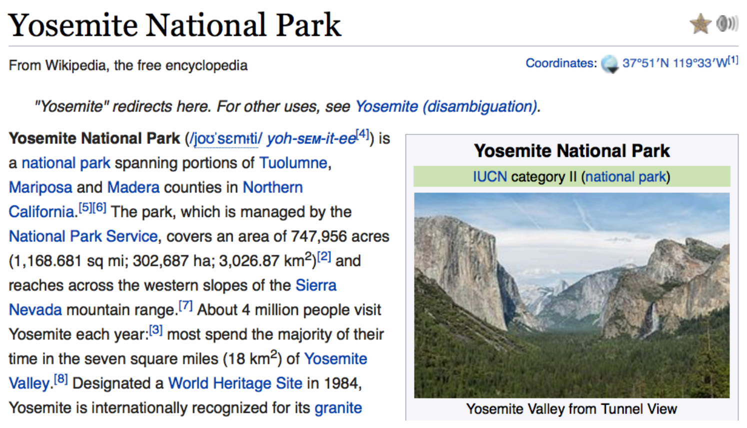
Moth species Neopalpa donaldtrumpi. Vazrick Nazari

## Map Example

```
Map<string, string> wiki;

// adds name / text pair to dataset
wiki.put(“Neopalpa donaldtrumpi”, articleHTML);

// returns corresponding articleHTML
cout << wiki.get(“Yosemite National Park”);
```



Yosemite National Park

From Wikipedia, the free encyclopedia

Coordinates: 37°51′N 119°33′W[1]

*"Yosemite" redirects here. For other uses, see Yosemite (disambiguation).*

**Yosemite National Park** (/joʊˈsɛmɪti/ *yoh-SEM-it-ee*[4]) is a national park spanning portions of Tuolumne, Mariposa and Madera counties in Northern California.[5][6] The park, which is managed by the National Park Service, covers an area of 747,956 acres (1,168.681 sq mi; 302,687 ha; 3,026.87 km$^2$)[2] and reaches across the western slopes of the Sierra Nevada mountain range.[7] About 4 million people visit Yosemite each year:[3] most spend the majority of their time in the seven square miles (18 km$^2$) of Yosemite Valley.[8] Designated a World Heritage Site in 1984, Yosemite is internationally recognized for its granite

**Yosemite National Park**

IUCN category II (national park)

Yosemite Valley from Tunnel View

# Map Example

```
Map<string, string> wiki;

// adds name / text pair to dataset
wiki.put(“Neopalpa donaldtrumpi”, articleHTML);

// returns corresponding articleHTML
cout << wiki.get(“Yosemite National Park”);

// removes the article
wiki.remove(“Britain in the E.U.”);
```

Brexit

From Wikipedia, the free encyclopedia

*For the 2016 Brexit referendum, see United Kingdom European Union membership referendum, 2016.*
*For other uses, see Brexit (disambiguation).*

The United Kingdom's withdrawal from the European Union is widely known as **Brexit**, a portmanteau of "British" and "exit".[1] Following a referendum held on 23 June 2016, in which 52% of votes cast were in favour of leaving the EU, the UK government intends to invoke Article 50 of the Treaty on European Union (the formal procedure for withdrawing) by the end of March 2017. This, within the treaty terms, would put the UK on a course to leave the EU by March 2019. Prime Minister Theresa May, elected by the ruling Conservative Party in the wake of the referendum, has promised a bill to repeal the European Communities Act 1972 and to incorporate existing EU laws into UK domestic law.[2] In January 2017, May announced a 12-point plan of negotiating objectives and confirmed that the UK government would not seek continued membership of the single market.[3] The terms of withdrawal have not yet been negotiated and in the meantime, the UK remains a full member of the European Union.

The UK joined the European Economic Community (EEC), a predecessor of the EU, in 1973, and confirmed its membership in a 1975 referendum by 67% of the votes. Historical opinion polls 1973–2015 tended to reveal majorities in favour of remaining in the EEC, EC or EU. In the 1970s and 1980s, withdrawal from the EEC was advocated mainly by some Labour Party and trade union figures. From the 1990s, withdrawal from the EU was advocated mainly by some Conservatives and by the newly founded UK Independence Party (UKIP).

**Contents** [hide]

1 The term "Brexit"
2 Historical background

Part of a series of articles on the
**United Kingdom**
**in the**
**European Union**

**Accession** [show]
**Membership** [show]
**Withdrawal** [show]

## Types of Maps

### Map

Iterate over elements in *sorted* order

**REALLY FAST!**

`O(log n) per retrieval`

Implemented using a "binary search tree"

### HashMap

Iterate over elements in *unsorted (jumbled)* order

**REALLY, RIDICULOUSLY FAST!**

`O(1) per retrieval`

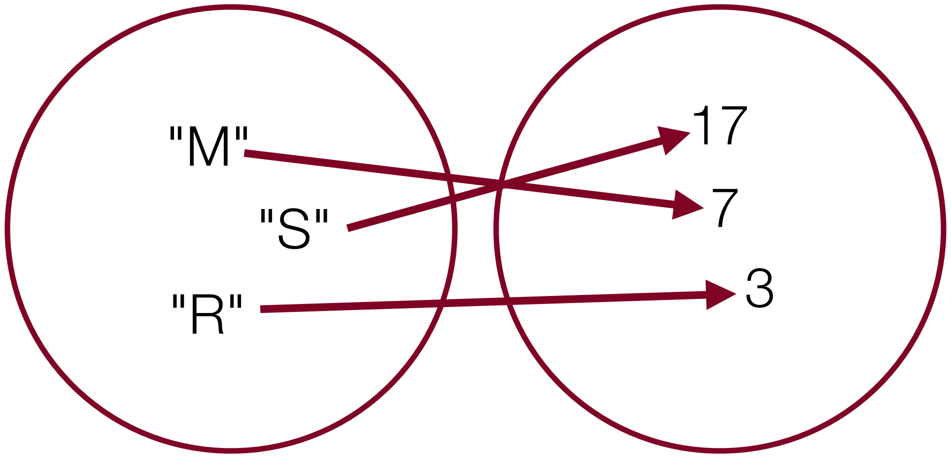
Implemented using a "hash table"

## Map Example: Tallying Votes

count votes:

```
// (M)ilk, (S)tokes, (R)ogers
    "MMMRMSSMSSMMMMMRRMMMMRRRMMM"
```

|  |  |  |  |
|---|---|---|---|
| key: | "M" | "S" | "R" |
| value: | 17 | 7 | 3 |



*In 1976 Harvey Milk became the first openly gay elected official in the US

## Tallying Words

## Looping Over a Map

```
Map<string, double> gpa = load();
for (string name : gpa) {
    cout << name << "'s GPA is ";
    cout << gpa[name] << endl;
}
```

*The order is unpredictable in a HashMap

## Recap

- **Structs**
  - Used to define a type that holds multiple other types.
  - Useful for returning more than one value, or keeping things together (e.g., a coordinate could be an x,y and it is nice to keep them together:

```
struct coordinate {
     double x,y;
}
```

- Uses dot notation to access elements.

- **Sets:**
  - Container that holds non-duplicate elements
  - O(log n) behavior per element access (**HashSet**: O(1), but unordered)

- **Map:**
  - Container that relates keys to values.
  - Needs two types when defining: `Map<keyType, valueType>`
  - O(log n) behavior per element access (**HashMap**: O(1), but unordered)

## References and Advanced Reading

- **References:**
  - Stanford Set reference: http://stanford.edu/~stepp/cppdoc/Set-class.html
  - Stanford Map reference: stanford.edu/~stepp/cppdoc/Map-class.html
  - const: http://www.cprogramming.com/tutorial/const_correctness.html

- **Advanced Reading:**
  - Hashing: https://en.wikipedia.org/wiki/Hash_table
  - Relational Databases: https://en.wikipedia.org/wiki/Relational_database (especially idecies)
  - const: http://duramecho.com/ComputerInformation/WhyHowCppConst.html

## Extra Slides