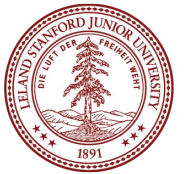
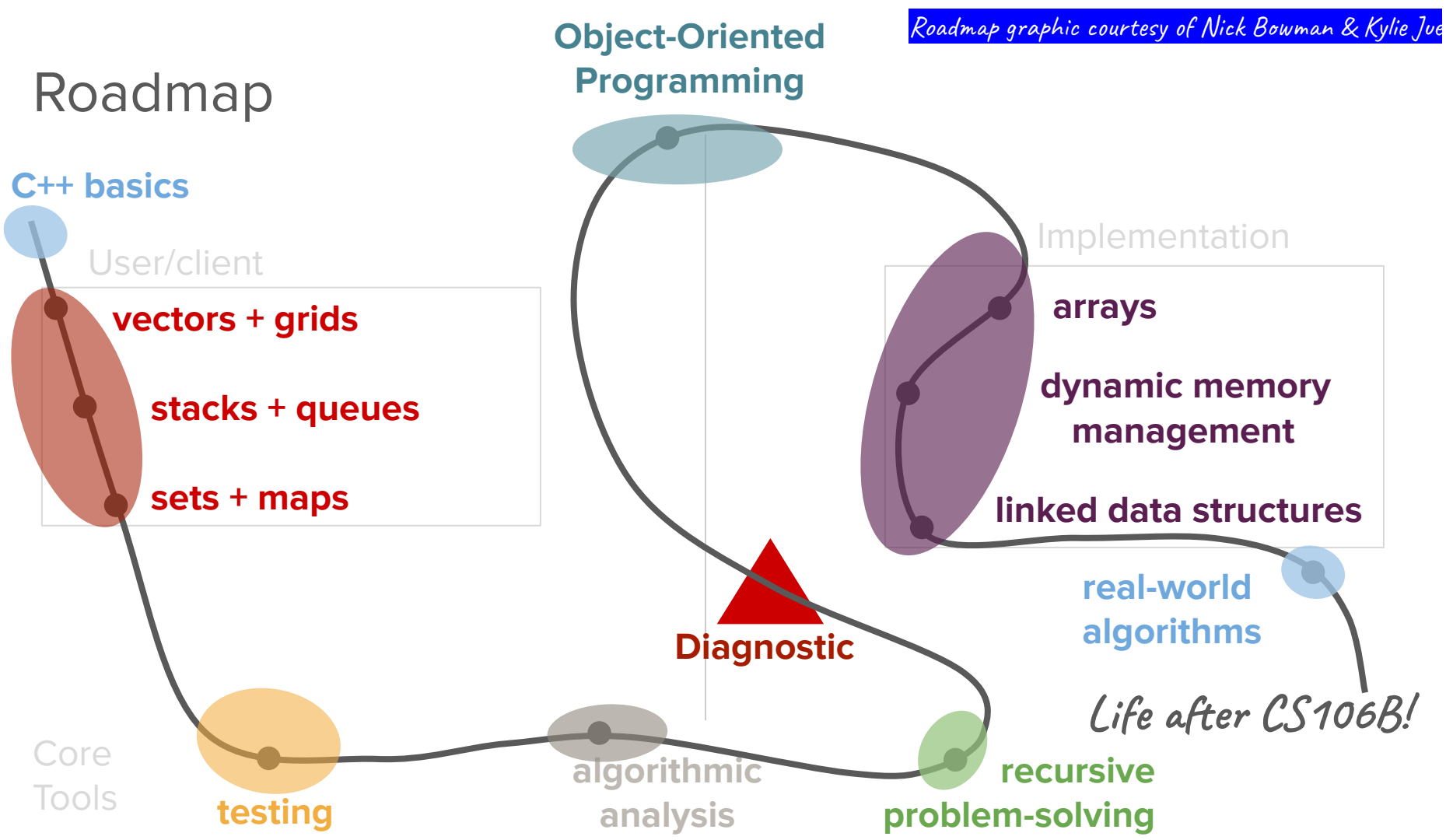
Object-Oriented Programming

Can you think of an app that would make your community or school better for all students or for a group of people?

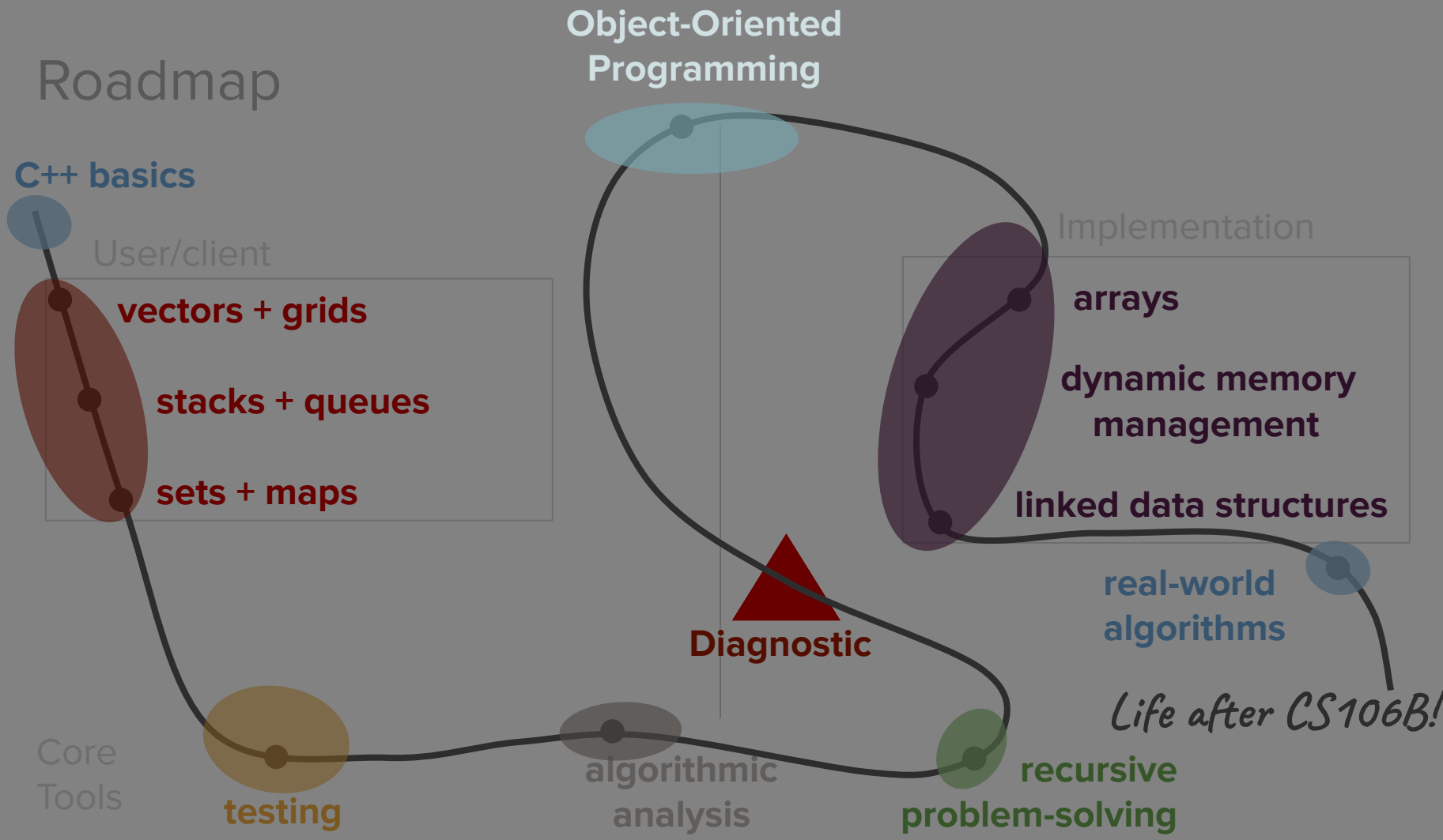
(put your answers the chat)



Roadmap


Roadmap graphic courtesy of Nick Bowman & Kylie Jue
Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Today's question

How do we design and define our own abstractions?

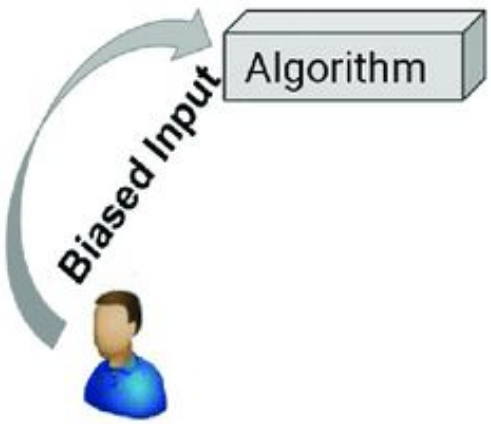
Today's topics

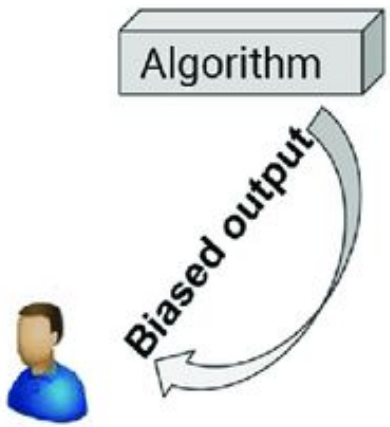
1. Review
2. What is a class?
3. Designing C++ classes
4. Writing classes in C++

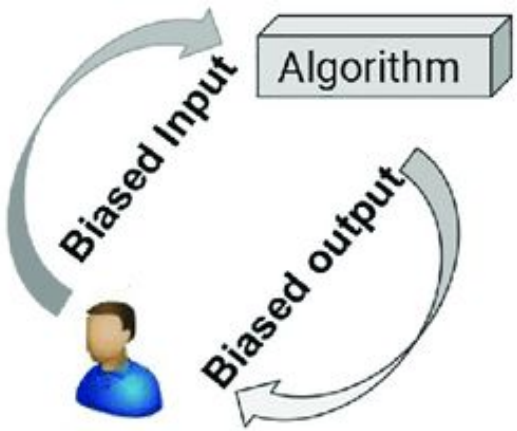
Review

The Knapsack Problem

You have a list full of supplies (each of which has a survival value and a weight associated with it) to choose from.


Algorithm
Biased Input


Algorithm
Biased output


Algorithm
Biased Input
Biased output

Key Questions

Who decides the target audience?

Who needs the app or software the most?

If there are unintended consequences, are these consequences fairly distributed among groups of people?

How do we define fairness?

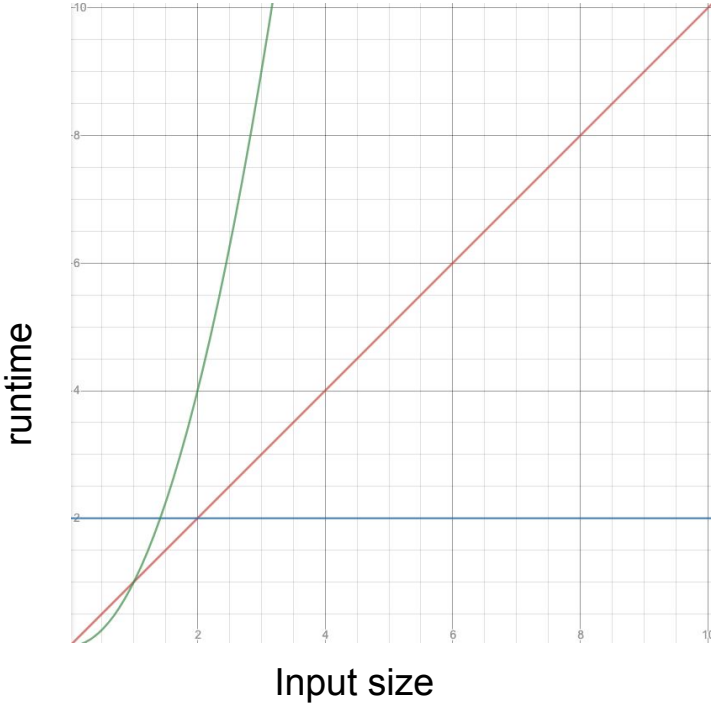
If we need funds to develop our software,
who is able to buy it and does the cost to develop it
inherently make it inequitable?

Should the government play a role in regulating this?

Gaining Perspective

Efficiency Categorizations So Far

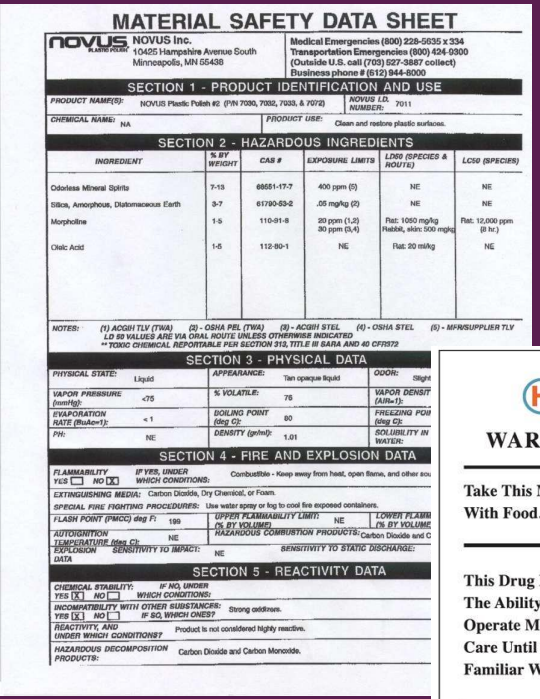
- Constant Time – O(1)
  - Super fast, this is the best we can hope for!
  - example: Euclid's Algorithm for Perfect Numbers
- Linear Time – O(n)
  - This is okay, we can live with this
- Quadratic Time – $O(n^2)$
  - This can start to slow down really quickly
  - example: Exhaustive Search for Perfect Numbers



CALCULATING THE EFFICIENCY GAP

$$\text{Efficiency Gap (EG)} = \frac{\text{abs(wasted green votes - wasted teal votes)}}{\text{total votes cast}}$$

Software Usage Labels?



MATERIAL SAFETY DATA SHEET

NOVUS Inc.
10425 Hampshire Avenue South
Minneapolis, MN 55438

Medical Emergencies (800) 228-5635 x 334
Transportation Emergencies (800) 424-9300
(Outside U.S. call (703) 527-3887 collect)
Business phone # (612) 944-8000

SECTION 1 - PRODUCT IDENTIFICATION AND USE

PRODUCT NAME(S): NOVUS Plastic Polish #2 (P/N 7030, 7032, 7033, & 7072) — NOVUS I.D. NUMBER: 7011

CHEMICAL NAME: NA — PRODUCT USE: Clean and restore plastic surfaces.

SECTION 2 - HAZARDOUS INGREDIENTS

INGREDIENT	% BY WEIGHT	CAS #	EXPOSURE LIMITS	LD50 (SPECIES & ROUTE)	LC50 (SPECIES)
Odorless Mineral Spirits	7-13	68551-17-7	400 ppm (5)	NE	NE
Silica, Amorphous, Diatomaceous Earth	3-7	61790-53-2	.05 mg/kg (2)	NE	NE
Morpholine	1-5	110-91-8	20 ppm (1,2) 30 ppm (3,4)	Rat: 1050 mg/kg Rabbit, skin: 500 mg/kg	Rat: 12,000 ppm (8 hr.)
Oleic Acid	1-5	112-80-1	NE	Rat: 20 ml/kg	NE

NOTES: (1) ACGIH TLV (TWA) (2) - OSHA PEL (TWA) (3) - ACGIH STEL (4) - OSHA STEL (5) - MFR/SUPPLIER TLV
LD 50 VALUES ARE VIA ORAL ROUTE UNLESS OTHERWISE INDICATED
** TOXIC CHEMICAL REPORTABLE PER SECTION 313, TITLE III SARA AND 40 CFR372

SECTION 3 - PHYSICAL DATA

PHYSICAL STATE:	Liquid	APPEARANCE:	Tan opaque liquid	ODOR:	Slight
VAPOR PRESSURE (mmHg):	<75	% VOLATILE:	78	VAPOR DENSITY (AIR=1):	
EVAPORATION RATE (BuAc=1):	< 1	BOILING POINT (deg C):	80	FREEZING POINT (deg C):	
PH:	NE	DENSITY (gr/ml):	1.01	SOLUBILITY IN WATER:	

SECTION 4 - FIRE AND EXPLOSION DATA

FLAMMABILITY YES ☐ NO ☒ — IF YES, UNDER WHICH CONDITIONS: Combustible - Keep away from heat, open flame, and other so

EXTINGUISHING MEDIA: Carbon Dioxide, Dry Chemical, or Foam.

SPECIAL FIRE FIGHTING PROCEDURES: Use water spray or fog to cool fire exposed containers.

FLASH POINT (PMCC) deg F:	199	UPPER FLAMMABILITY LIMIT: % BY VOLUME:	NE	LOWER FLAMM % BY VOLUME	
AUTOIGNITION TEMPERATURE deg C):	NE	HAZARDOUS COMBUSTION PRODUCTS:	Carbon Dioxide and C		
EXPLOSION DATA	SENSITIVITY TO IMPACT:	NE	SENSITIVITY TO STATIC DISCHARGE:		

SECTION 5 - REACTIVITY DATA

CHEMICAL STABILITY: YES ☒ NO ☐ — IF NO, UNDER WHICH CONDITIONS:

INCOMPATIBILITY WITH OTHER SUBSTANCES: YES ☒ NO ☐ — IF SO, WHICH ONES? Strong oxidizers.

REACTIVITY, AND UNDER WHICH CONDITIONS? Product is not considered highly reactive.

HAZARDOUS DECOMPOSITION PRODUCTS: Carbon Dioxide and Carbon Monoxide.

H

WARNINGS

Take This Medication With Food.

This Drug May Impair The Ability To Drive Or Operate Machinery. Use Care Until You Become Familiar With Its Effects.

Do Not Take Other Medicines Without Checking With Your Doctor Or Pharmacist.

A Miss Member
1111 Meadow Drive, Anywhere, RI 00000

B DATE: 01/01/2016

C Metformin 500mg
IC Glucophage 500mg

D TAKE 1 TABLET BY MOUTH UP TO 2 TIMES DAILY

E RX 1234567-09

F QTY: 60
2 Refills
03/01/2016

G Your Pharmacy
1111 State Road, Rhode Island 00000

Rx

Nutrition Facts

Serving Size 1/2 cup (about 82g)
Servings Per Container 8

Amount Per Serving

Calories 200 Calories from Fat 130

	% Daily Value*
Total Fat 14g	**22%**
Saturated Fat 9g	**45%**
Trans Fat 0g	
Cholesterol 55mg	**18%**
Sodium 40mg	**2%**
Total Carbohydrate 17g	**6%**
Dietary Fiber 1g	**4%**
Sugars 14g	
Protein 3g	

Vitamin A 10% • Vitamin C 0%

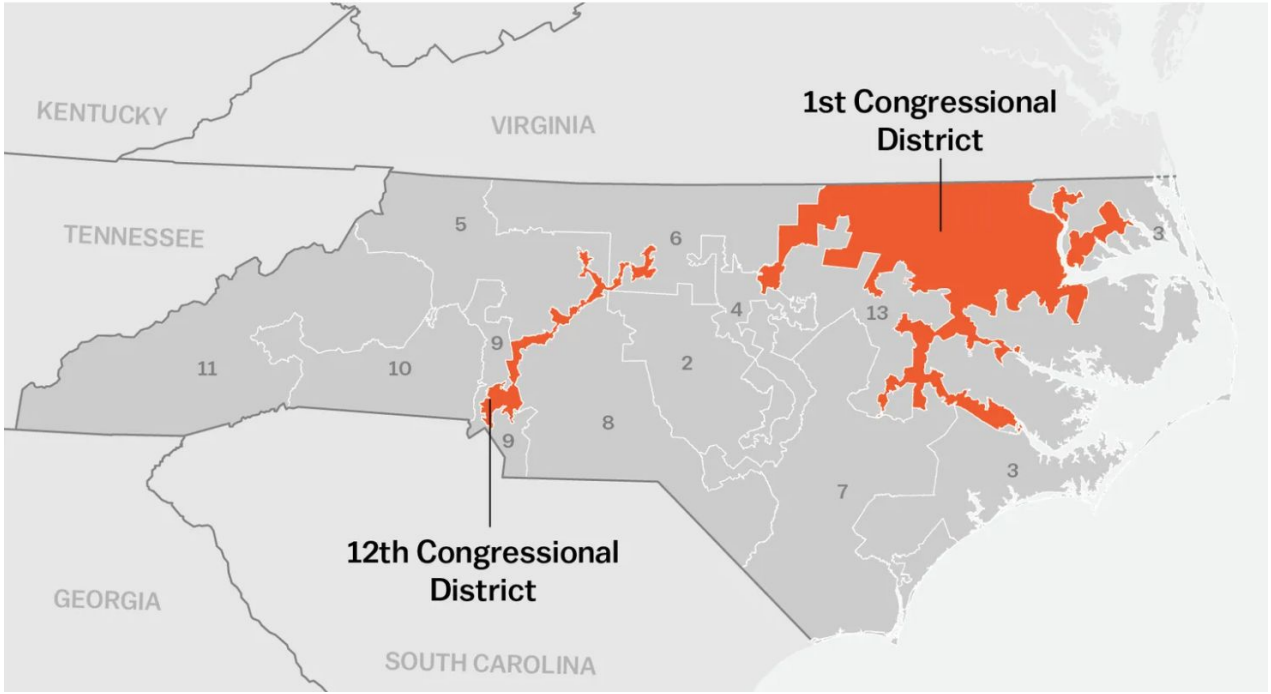
Calcium 10% • Iron 6%

*Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calorie needs:

	Calories:	2,000	2,500
Total Fat	Less than	65g	80g
Saturated Fat	Less than	20g	25g
Cholesterol	Less than	300mg	300 mg
Sodium	Less than	2,400mg	2,400mg
Total Carbohydrate		300g	375g
Dietary Fiber		25g	30g

Calories per gram:
Fat 9 • Carbohydrate 4 • Protein 4

Gerrymandering & Algorithmic Thinking



based on slides created by Katie Creel

What should we prioritize in redistricting? And what is a legitimate way to choose?

PRINCIPLE 1: ONE PERSON, ONE VOTE

PRINCIPLE 2: COMMUNITIES OF INTEREST

Why has Gerrymandering Gotten Worse?

Veteran redistricter: “Give the chairman of a state redistricting committee a powerful enough computer and neighborhood-block-level Census data, so that he suddenly discovers he can draw really weird and aggressive districts—and he will.”

Software + big data making a problem worse?

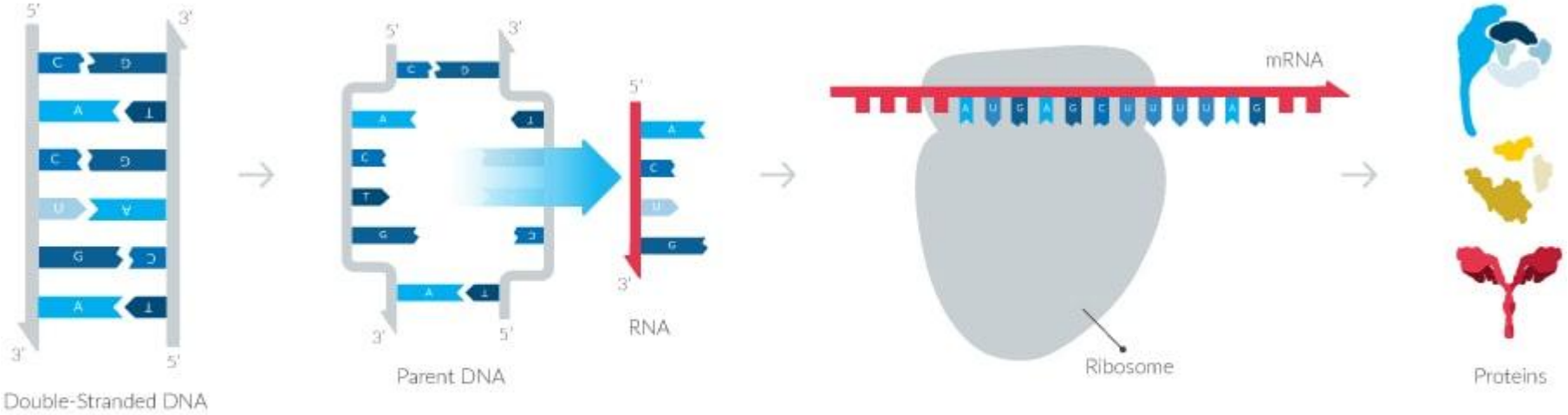
- New Software: Maptitude, RedAppl, and autoBound
- New Data: block-by-block census data

"POVERTY IS THE WORST FORM OF VIOLENCE"

-MAHATMA GANDHI

TRANSCRIPTION

TRANSLATION



Where are we now?

classes

object-oriented programming

abstract data structures
(vectors, maps, etc.)

arrays

dynamic memory
management

linked data structures

testing

algorithmic analysis

recursive problem-solving

classes
object-oriented programming

abstract data structures
(vectors, maps, etc.)

arrays

dynamic memory
management

linked data structures

testing

algorithmic analysis

recursive problem-solving

classes
object-oriented programming

This is our abstraction boundary!

abstract data structures
(vectors, maps, etc.)

arrays

dynamic memory management

linked data structures

testing *algorithmic analysis* *recursive problem-solving*

Revisiting abstraction

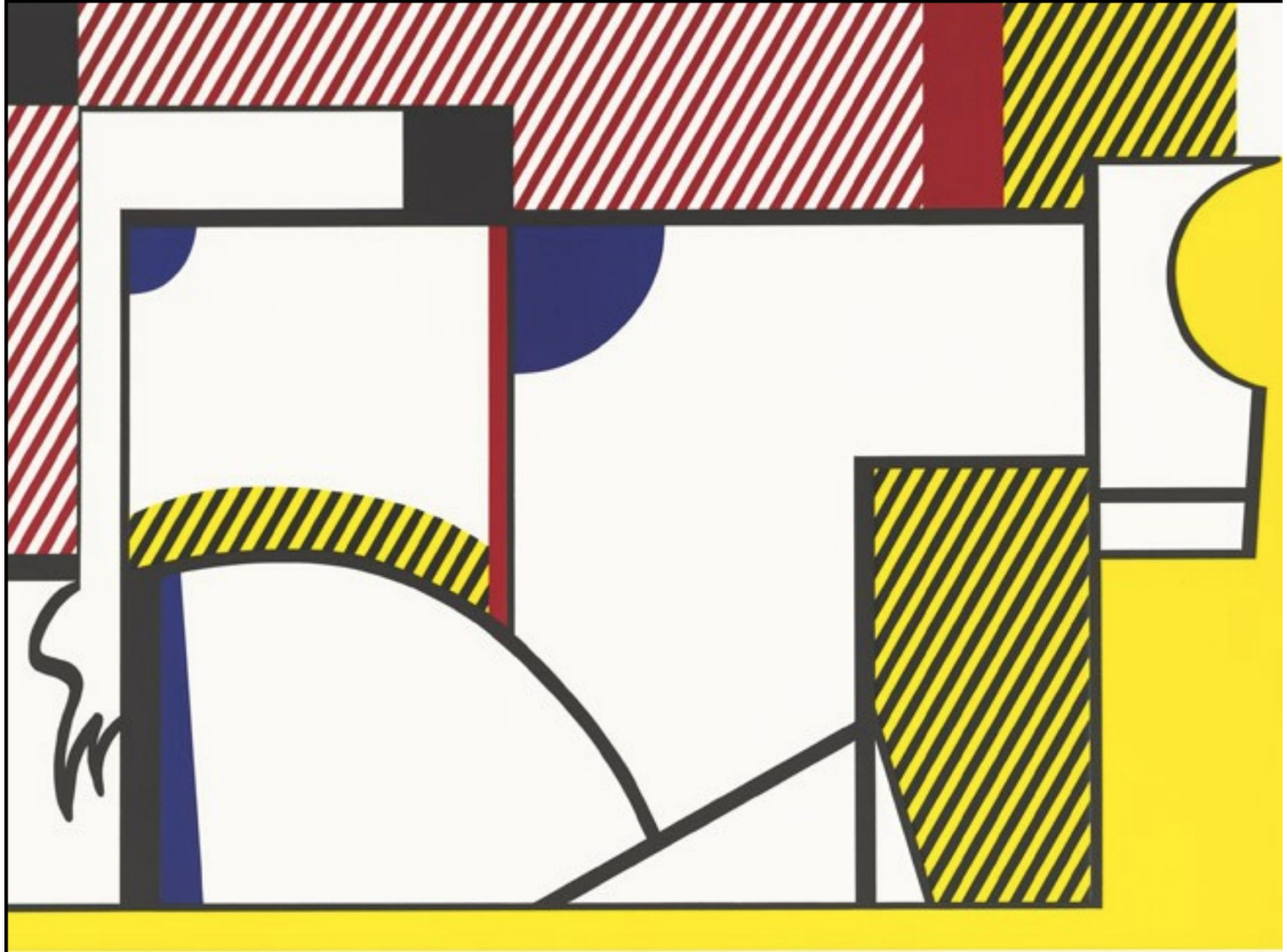
ab·strac·tion

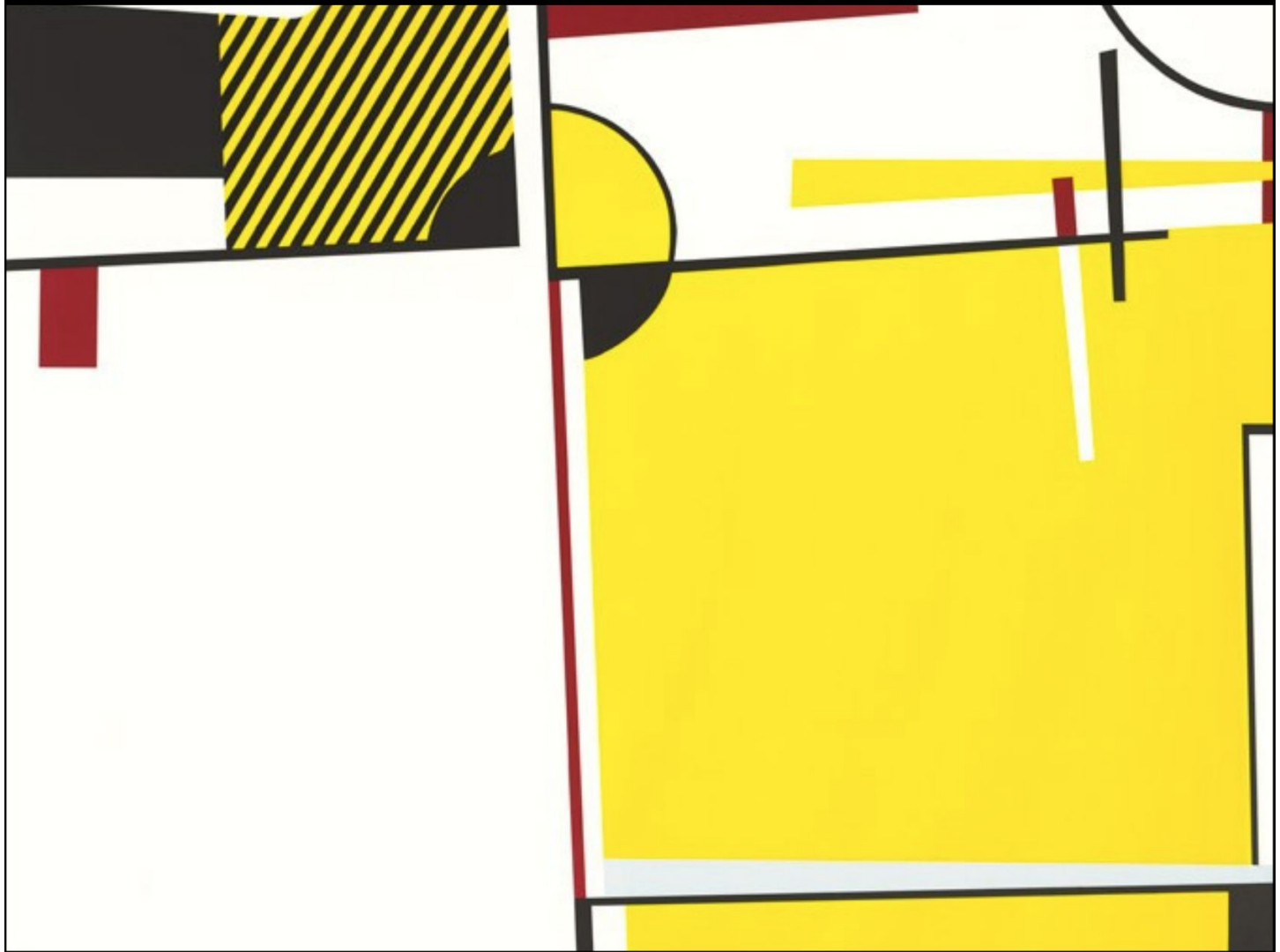
[...]

freedom from representational qualities in art

Source: Google

Example demonstration borrowed from Keith Schwarz

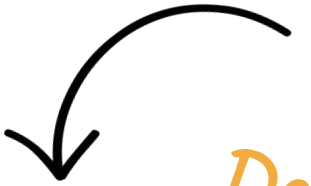




Definition

abstraction

Design that hides the details of how something works while still allowing the user to access complex functionality



How do we accomplish this in C++? With classes!

Definition

abstraction

Design that hides the details of how something works while still allowing the user to access complex functionality

What is a class?

Definition

class

A class defines a new data type for our programs to use.

Definition

class

A class defines a new data type for our programs to use.

This sounds familiar...

Remember structs?

```
struct BackpackItem {
    int survivalValue;
    int weight;
};

struct Juror {
    string name;
    int bias;
};
```

Remember structs?

```
struct BackpackItem {
    int survivalValue;
    int weight;
};

struct Juror {
    string name;
    int bias;
};
```

Definition

struct

A way to bundle different types of information in C++ – like creating a custom data structure.

Then what's the difference between a struct and a class?

```
GridLocation chosen;
int curRow = chosen.x;
int curCol = chosen.y;
```

```
GWindow canvas;
int displayWidth = canvas.getWidth();
int displayHeight = canvas.getHeight();
```

What's the difference in how you use a struct vs. a class?

Remember structs?

```
GridLocation chosen;
int curRow = chosen.x;
int curCol = chosen.y;

chosen.x = 3;
chosen.y = 4;
```

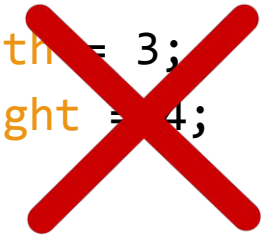
```
GWindow canvas;
int displayWidth = canvas.getWidth();
int displayHeight = canvas.getHeight();

canvas.width = 3;
canvas.height = 4;
```

What's the difference in how you use a struct vs. a class?

Remember structs?

```
GridLocation chosen;
int curRow = chosen.x;
int curCol = chosen.y;

chosen.x = 3;
chosen.y = 4;
```

```
GWindow canvas;
int displayWidth = canvas.getWidth();
int displayHeight = canvas.getHeight();

canvas.width = 3;
canvas.height = 4;
```



We don't have direct access to the variables in a class!

What is a class?

- Examples of classes we've already seen: **`Vector`**s, **`Map`**s, **`Stack`**s, **`Queue`**s

What is a class?

- Examples of classes we've already seen: **`Vector`**s, **`Map`**s, **`Stack`**s, **`Queue`**s

- Every class has two parts:
  - an **interface** specifying what operations can be performed on instances of the class (this defines the abstraction boundary)
  - an **implementation** specifying how those operations are to be performed

What is a class?

- Examples of classes we've already seen: **`Vector`**s, **`Map`**s, **`Stack`**s, **`Queue`**s

- Every class has two parts:
  - an **interface** specifying what operations can be performed on instances of the class (this defines the abstraction boundary)
  - an **implementation** specifying how those operations are to be performed

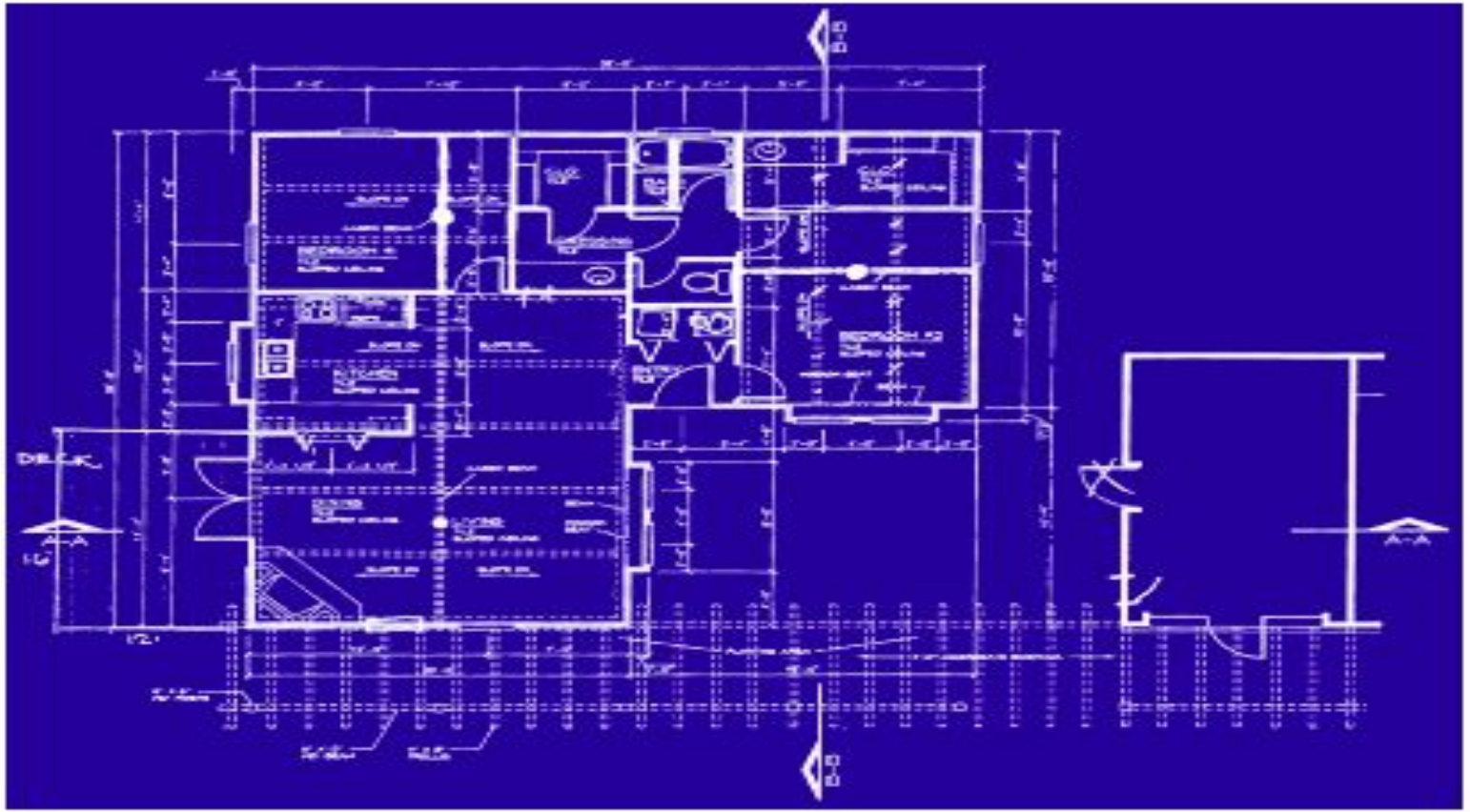
- The only difference between structs + classes are the **encapsulation** defaults.
  - A struct defaults to **public** members (accessible outside the class itself).
  - A class defaults to **private** members (accessible only inside the class implementation).

Definition

encapsulation

The process of grouping related information and relevant functions into one unit and defining where that information is accessible

Another way to think about classes...



Another way to think about classes...

- A blueprint for a new type of C++ **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

Another way to think about classes...

- A blueprint for a new type of C++ **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

Definition

instance

When we create an object that is our new type, we call this creating an instance of our class.

Another way to think about classes...

- A blueprint for a new type of C++ **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

```
Vector<int> vec;
```

*Creates an **instance** of the Vector **class***
(i.e. an object of the type Vector)

How do we design C++ classes?

Three main parts

- Member variables
- Member functions (methods)
- Constructor

Three main parts

- Member variables
  - These are the variables stored within the class
  - Usually not accessible outside the class implementation

- Member functions (methods)

- Constructor

Three main parts

- Member variables

- Member functions (methods)
  - Functions you can call on the object
  - E.g. **`vec.add()`**, **`vec.size()`**, **`vec.remove()`**, etc.

- Constructor

Three main parts

- Member variables
- Member functions (methods)
- Constructor
  - Gets called when you create the object
  - E.g. **`Vector<int> vec;`**

Three main parts

- Member variables
  - These are the variables stored within the class
  - Usually not accessible outside the class implementation

- Member functions (methods)
  - Functions you can call on the object
  - E.g. **`vec.add()`**, **`vec.size()`**, **`vec.remove()`**, etc.

- Constructor
  - Gets called when you create the object
  - E.g. **`Vector<int> vec;`**

How do we design a class?

We must specify the 3 parts:

1. Member variables: *What subvariables make up this new variable type?*

2. Member functions: *What functions can you call on a variable of this type?*

3. Constructor: *What happens when you make a new instance of this type?*

How do we design a class?

We must specify the 3 parts:

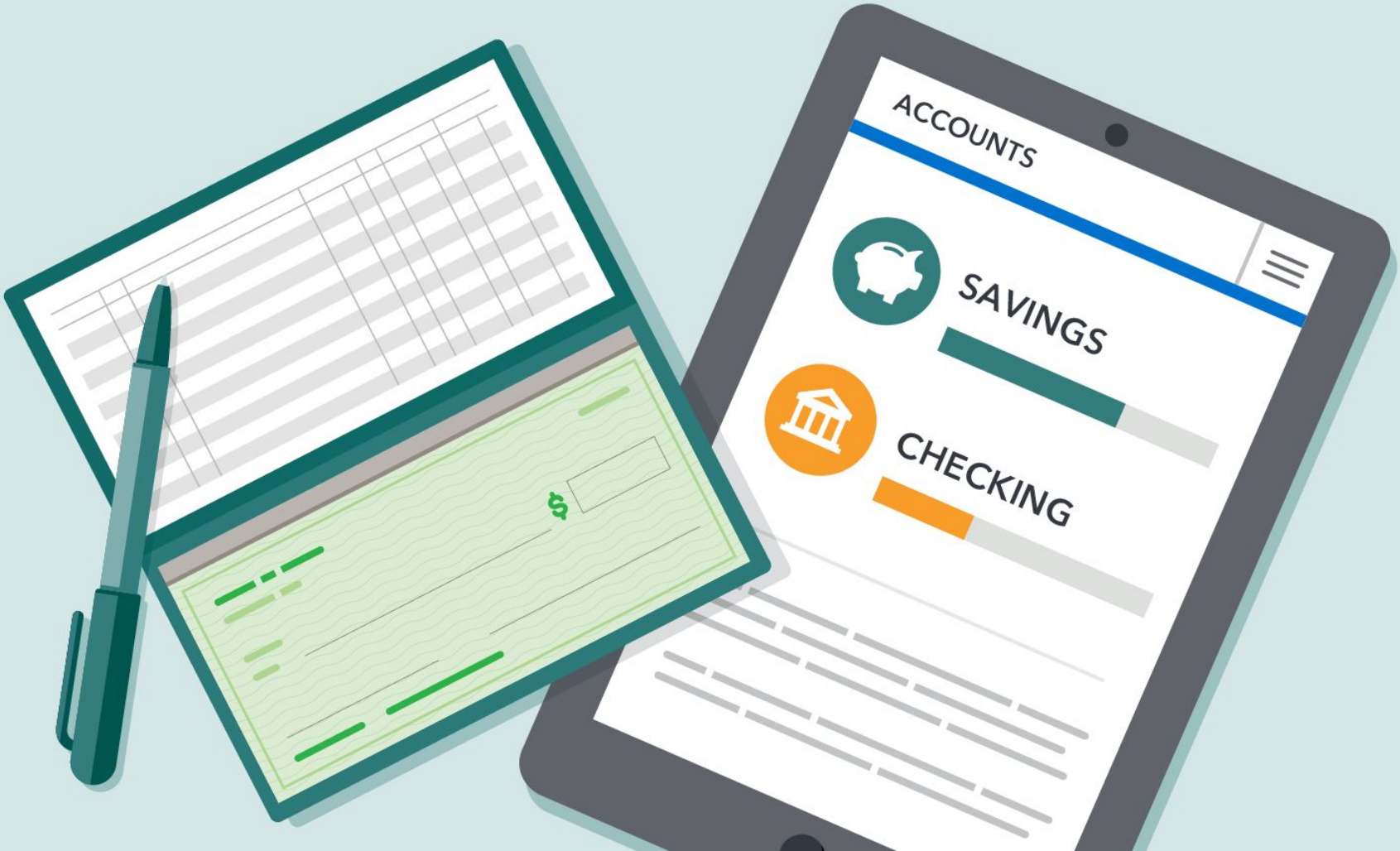
1. Member variables: *What subvariables make up this new variable type?*

2. Member functions: *What functions can you call on a variable of this type?*

3. Constructor: *What happens when you make a new instance of this type?*

In general, classes are useful in helping us with complex programs where information can be grouped into objects.

Breakout design activity


ACCOUNTS
SAVINGS
CHECKING

Home
Browse
Radio

Albums
Artists
Local Files
Podcasts

PLAYLISTS
2019 Great alb...
2018 Great Albums
70s Punk
70s Funk

New Playlist

Search

Sam Moore

PLAYLIST

2019 Great albums

Created by Sam Moore • 29 songs, 1 hr 45 min

PAUSE

FOLLOWERS
0

Filter

Download

	TITLE		ALBUM		
♡	Take It Away There		Thank You for Singing ...	2019-07-17	2:03
♡	Shrugging Match		Thank You for Singing ...	2019-07-17	3:58
♡	Yaskool		Thank You for Singing ...	2019-07-17	3:04
♡	Chicken Sized Nugget		Thank You for Singing ...	2019-07-17	3:04
♡	Salmon Rushdie		Thank You for Singing ...	2019-07-17	1:34
♡	Johnny Bravo		Thank You for Singing ...	2019-07-17	3:02
♥	Kamane Tarhanin	Mdou Moctar	Ilana (The Creator)	13 days ago	5:08
♡	Asshet Akal	Mdou Moctar	Ilana (The Creator)	13 days ago	4:51

Go to Playlist Radio
Collaborative Playlist
Make Secret
Edit Details
Report
Delete
Create Similar Playlist
Download
Share

Take It Away There
Bobbing

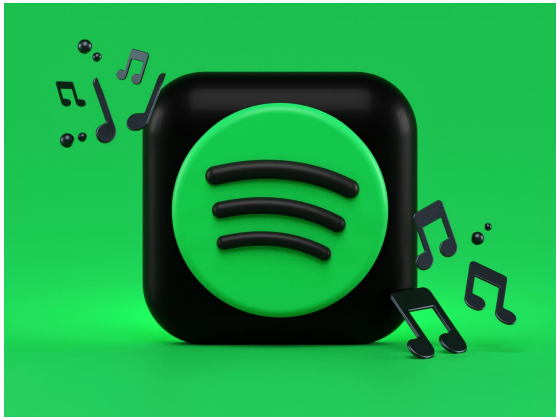
0:12
2:03

How would you design a class for...

- A bank account that enables transferring funds between accounts
- A Spotify (or other music platform) playlist

We must specify the 3 parts:

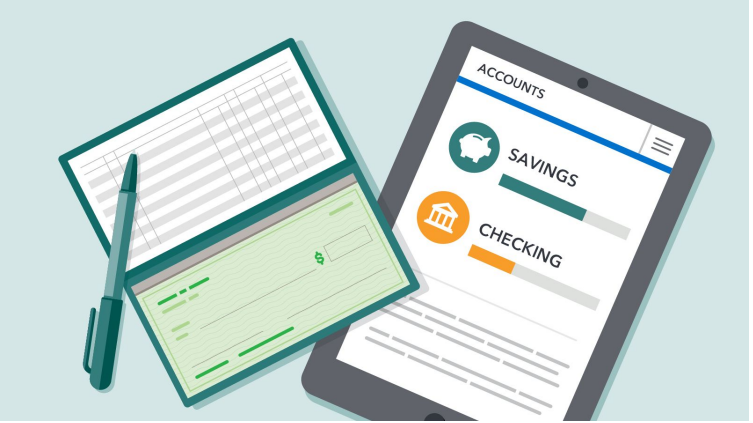
1. Member variables: *What subvariables make up this new variable type?*
2. Member functions: *What functions can you call on a variable of this type?*
3. Constructor: *What happens when you make a new instance of this type?*



Let’s design a music platform class!

We must specify the 3 parts:

1. Member variables: *What subvariables make up this new variable type?*
2. Member functions: *What functions can you call on a variable of this type?*
3. Constructor: *What happens when you make a new instance of this type?*



Let's design a bank account class!

We must specify the 3 parts:

1. Member variables: *What subvariables make up this new variable type?*
2. Member functions: *What functions can you call on a variable of this type?*
3. Constructor: *What happens when you make a new instance of this type?*

Announcements

Announcements

- The mid-quarter diagnostic will be released a minute after midnight!
  - **For some problems, you will need to upload .cpp files of your code. Shortly before the diagnostic, we will upload a starter .cpp file titled "midquarter.cpp" that you can download, fill in, and upload at the end. You do not need to use this resource, but we think it will be helpful.**
  - You should be able to see the diagnostic when you click on the CS106B8 class in gradescope.
  - Do not click on the diagnostic itself until you are ready to take it!
  - Once you start, you will have 3 hours to take the diagnostic.
- Assignment 3 is due tonight, **Thursday, July 15 at 11:59pm.**

Words of Advice

- Best of luck on the diagnostic! We hope that you all rock it!
- This is chance to demonstrate how much you've learned in just 3 weeks. The purpose of the diagnostic is truly "diagnostic" – to help you self-assess your own areas of strength and areas of potential growth. We expect everyone to have areas of improvement!
- Make sure to collect the resources that you plan to use in advance.
- Get a good night's sleep, eat a solid meal, get some exercise, and rock the diagnostic!

How do we write classes in C++?

Random Bags

Random Bags

- A **random bag** is a data structure similar to a stack or queue. It supports two operations:
  - **add**, which puts an element into the random bag, and
  - **remove random**, which returns and removes a random element from the bag.

Random Bags

- A **random bag** is a data structure similar to a stack or queue. It supports two operations:
  - **add**, which puts an element into the random bag, and
  - **remove random**, which returns and removes a random element from the bag.

- Random bags have a number of applications:
  - Simpler: Shuffling a deck of cards.
  - More advanced: Generating artwork, designing mazes, and training self-driving cars to park and change lanes!

Random Bags

- A **random bag** is a data structure similar to a stack or queue. It supports two operations:
  - **add**, which puts an element into the random bag, and
  - **remove random**, which returns and removes a random element from the bag.

- Random bags have a number of applications:
  - Simpler: Shuffling a deck of cards.
  - More advanced: Generating artwork, designing mazes, and training self-driving cars to park and change lanes.

- Let's go create our own custom **`RandomBag`** type!

Creating our own class

Classes in C++

- Defining a class in C++ (typically) requires two steps:

Classes in C++

- Defining a class in C++ (typically) requires two steps:
  - Create a **header file** (typically suffixed with `.h`) describing what operations the class can perform and what internal state it needs.

Classes in C++

- Defining a class in C++ (typically) requires two steps:
  - Create a **header file** (typically suffixed with `.h`) describing what operations the class can perform and what internal state it needs.
  - Create an **implementation file** (typically suffixed with `.cpp`) that contains the implementation of the class.

Classes in C++

- Defining a class in C++ (typically) requires two steps:
  - Create a **header file** (typically suffixed with **`.h`**) describing what operations the class can perform and what internal state it needs.
  - Create an **implementation file** (typically suffixed with **`.cpp`**) that contains the implementation of the class.

- Clients of the class can then include (using the **`#include`** directive) the header file to use the class.

Header files

What's in a header?

What's in a header?

```
#pragma once
```

This boilerplate code is called a *preprocessor directive.* It's used to make sure weird things don't happen if you include the same header twice.

What's in a header?

```
#pragma once

class RandomBag {



};
```

This is a **class definition**. We're creating a new class called **RandomBag**. Like a `struct`, this defines the name of a new type that we can use in our programs.

What's in a header?

```
#pragma once

class RandomBag {



};
```

Don't forget to add the semicolon!

You'll run into some scary compiler errors if you leave it out!

What's in a header?

```
#pragma once

class RandomBag {
public:




private:

};
```

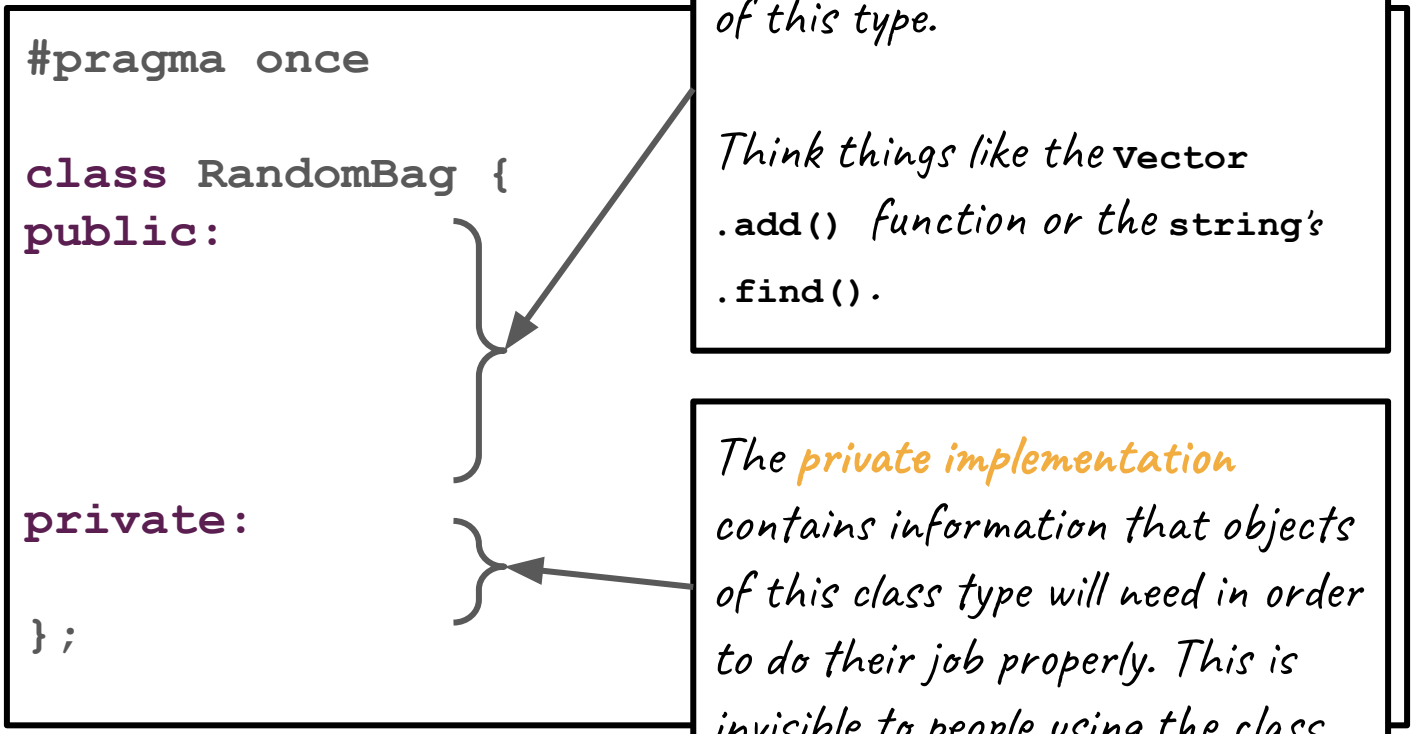
What's in a header?

```
#pragma once

class RandomBag {
public:




private:

};
```

The public interface specifies what functions you can call on objects of this type.

Think things like the `Vector` `.add()` function or the `string`'s `.find()`.

What's in a header?



What's in a header?

```
#pragma once

class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:

};
```

These are member functions of the `RandomBag` class. They're functions you can call on objects of type `RandomBag`.

All member functions must be defined in the class definition. We'll implement these functions in the C++ file.

What's in a header?

```
#pragma once
#include "vector.h"
class RandomBag {
public:
   void add(int value);
   int removeRandom();


private:
   Vector<int> elems;
};
```

This is a **data member** of the class. This tells us how the class is implemented. Internally, we're going to store a `Vector<int>` holding all the elements. The only code that can access or touch this `Vector` is the `RandomBag` implementation.

Header summary

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

Header summary

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

Implementation files

`RandomBag.cpp`

```
#include "RandomBag.h"
```

```
#include "RandomBag.h"
```

If we're going to implement the `RandomBag` type, the `.cpp` file needs to have the class definition available. All implementation files need to include the relevant headers.

```
#include "RandomBag.h"
```

If we're going to implement the `RandomBag` type, the `.cpp` file needs to have the class definition available. All implementation files need to include the relevant headers.

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
    elems.add(value);
}
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}
```

The syntax `RandomBag::add` means "the add function defined inside of `RandomBag`." The :: operator is called the **scope resolution operator** in C++ and is used to say where to look for things.

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}
```

If we had written something like this instead, then the compiler would think we were just making a free function named **add** *that has nothing to do with* **RandomBag's** *version of* **add**. *That's an easy mistake to make!*

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}
```

We don't need to specify where `elems` is. The compiler knows that we're inside `RandomBag`, and so it knows that this means "the current `RandomBag's` collection of elements."

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
    elems.add(value);
}

int RandomBag::removeRandom() {
    if (elems.isEmpty()) {
        error("Aaaaahhh!");
    }
    int index = randomInteger(0, elems.size() - 1);
    int result = elems[index];
    elems.remove(index);
    return result;
}
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
    elems.add(value);
}

int RandomBag::removeRandom() {
    if (elems.isEmpty()) {
        error("Aaaaahhh!");
    }
    int index = randomInteger(0, elems.size() - 1);
    int result = elems[index];
    elems.remove(index);
    return result;
}
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size();
  bool isEmpty();
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
    elems.add(value);
}

int RandomBag::removeRandom() {
    if (elems.isEmpty()) {
        error("Aaaaahhh!");
    }
    int index = randomInteger(0, elems.size() - 1);
    int result = elems[index];
    elems.remove(index);
    return result;
}

int RandomBag::size() {
    return elems.size();
}
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size();
  bool isEmpty();
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
    elems.add(value);
}

int RandomBag::removeRandom() {
    if (elems.isEmpty()) {
        error("Aaaaahhh!");
    }
    int index = randomInteger(0, elems.size() - 1);
    int result = elems[index];
    elems.remove(index);
    return result;
}

int RandomBag::size() {
    return elems.size();
}

bool RandomBag::isEmpty() {
    return size() == 0;
}
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size();
  bool isEmpty();
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
    elems.add(value);
}

int RandomBag::removeRandom() {
    if (elems.isEmpty()) {
        error("Aaaaahhh!");
    }
    int index = randomInteger(0, elems.size() - 1);
    int result = elems[index];
    elems.remove(index);
    return result;
}

int RandomBag::size() {
    return elems.size();
}

bool RandomBag::isEmpty() {
    return size() == 0;
}
```

This code calls our own **size()** *function. The class implementation can use the public interface.*

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size();
  bool isEmpty();
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
    elems.add(value);
}

int RandomBag::removeRandom() {
    if (elems.isEmpty()) {
        error("Aaaaahhh!");
    }
    int index = randomInteger(0, size() - 1);
    int result = elems[index];
    elems.remove(index);
    return result;
}

int RandomBag::size() {
    return elems.size();
}

bool RandomBag::isEmpty() {
    return size() == 0;
}
```

What a good idea! Let's use it here as well.

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size();
  bool isEmpty();
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
    elems.add(value);
}

int RandomBag::removeRandom() {
    if (elems.isEmpty()) {
        error("Aaaaahhh!");
    }
    int index = randomInteger(0, size() - 1);
    int result = elems[index];
    elems.remove(index);
    return result;
}

int RandomBag::size() {
    return elems.size();
}

bool RandomBag::isEmpty() {
    return size() == 0;
}
```

This use of the `const` *keyword means "I promise that this function doesn't change the state of the object."*

```
public:
  void add(int value);
  int removeRandom();
  int size() const;
  bool isEmpty() const;
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}

int RandomBag::removeRandom() {
     if (elems.isEmpty()) {
          error("Aaaaahhh!");
     }
     int ind                              size() - 1);
     int res
     elems.r
     return
}

int RandomBag::size() const {
     return elems.size();
}

bool RandomBag::isEmpty() const {
     return size() == 0;
}
```

We have to remember to add it into the implementation as well!

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size() const;
  bool isEmpty() const;
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
    elems.add(value);
}

int RandomBag::removeRandom() {
    if (elems.isEmpty()) {
        error("Aaaaahhh!");
    }
    int index = randomInteger(0, size() - 1);
    int result = elems[index];
    elems.remove(index);
    return result;
}

int RandomBag::size() const {
    return elems.size();
}

bool RandomBag::isEmpty() const {
    return size() == 0;
}
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size() const;
  bool isEmpty() const;
private:
  Vector<int> elems;
};
```

Using a custom class

[Qt Creator demo]

Takeaways

- Public member variables declared in the header file are automatically accessible in the **`.cpp`** file

Takeaways

- Public member variables declared in the header file are automatically accessible in the **`.cpp`** file

- As a best practice, member variables should be private, and you can create public member functions to allow users to edit them

Takeaways

- Public member variables declared in the header file are automatically accessible in the **`.cpp`** file

- As a best practice, member variables should be private, and you can create public member functions to allow users to edit them

- Member functions have an implicit parameter that allows them to know what object they're operating on

Takeaways

- Public member variables declared in the header file are automatically accessible in the **`.cpp`** file

- As a best practice, member variables should be private, and you can create public member functions to allow users to edit them

- Member functions have an implicit parameter that allows them to know what object they're operating on

- When you don't have a constructor, there's a default 0 argument constructor that instantiates all private member variables
    - (We'll see an explicit constructor tomorrow!)

Summary

Object-Oriented Programming

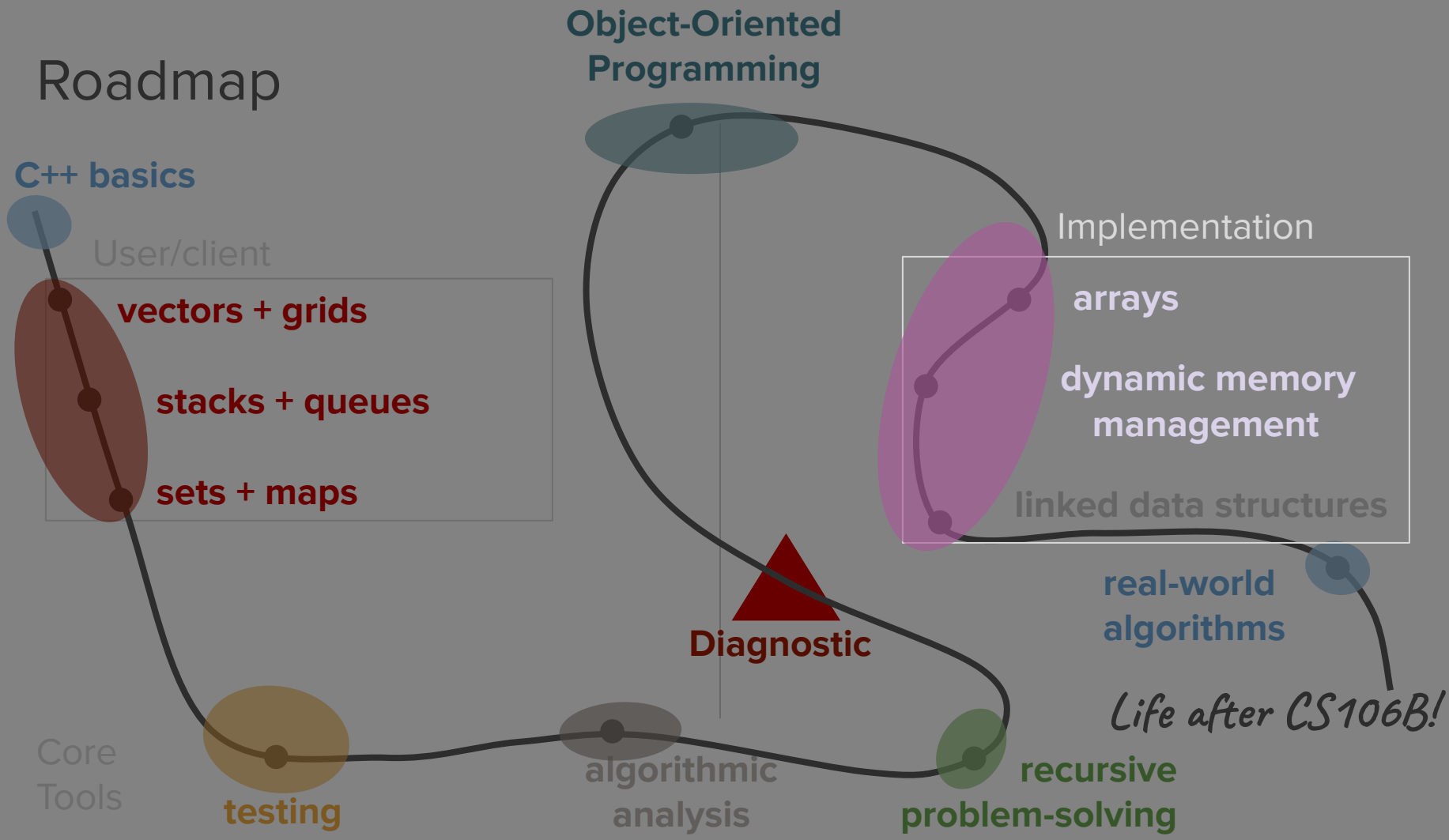
- We create our own abstractions for defining data types using classes. Classes allow us to encapsulate information in a structured way.

- Classes have three main parts to keep in mind when designing them:
  - Member variables ➜ these are always private
  - Member functions (methods)
  - Constructor ➜ this is created by default if you don’t define one

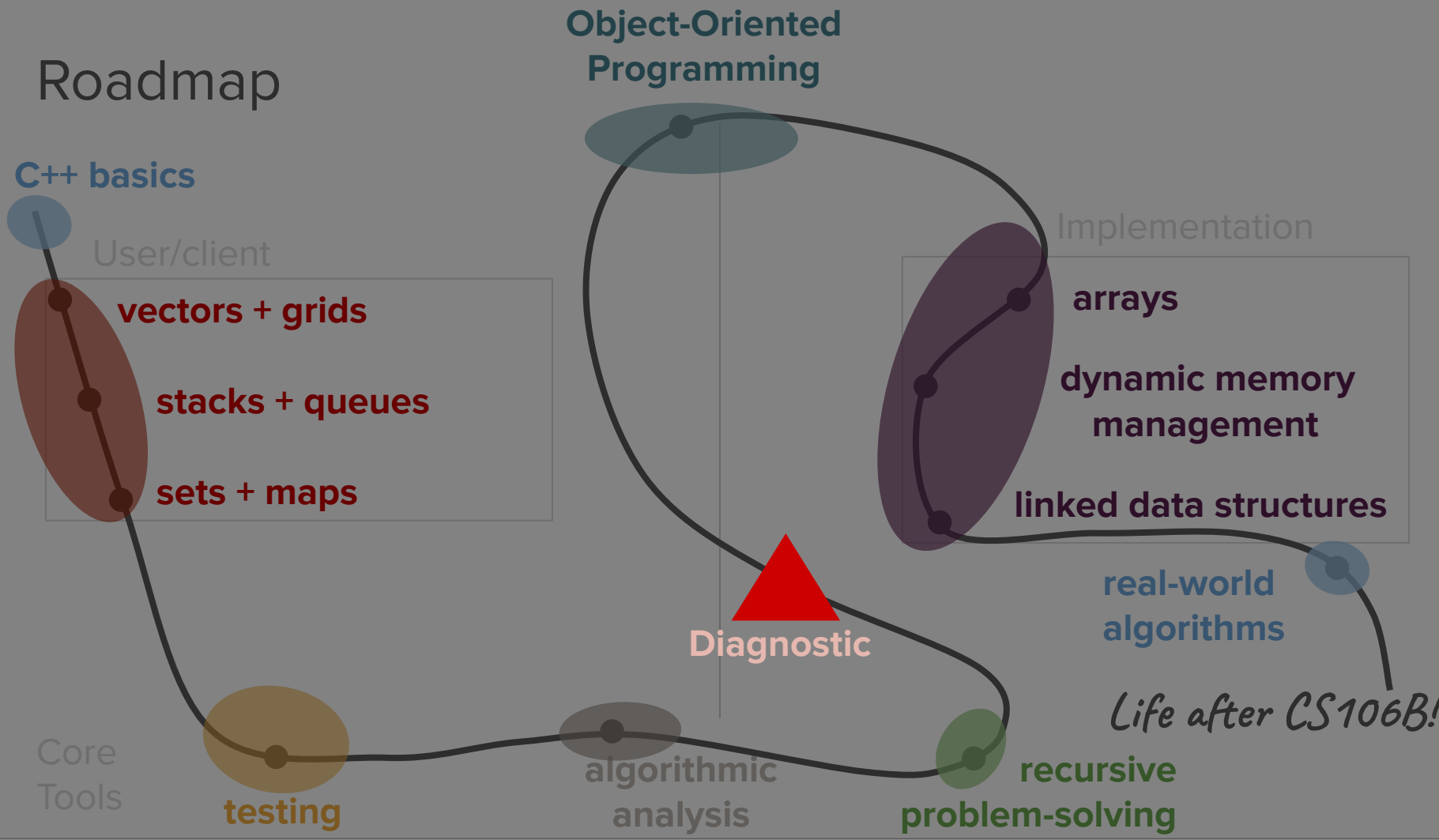
- Writing classes requires the creation of a header (**`.h`**) file for the interface and an implementation (**`.cpp`**) file.

What’s next?

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Dynamic memory and arrays