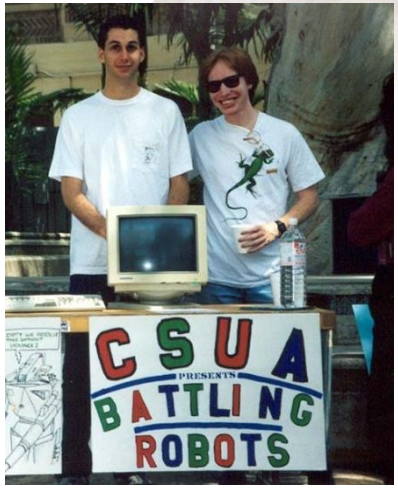
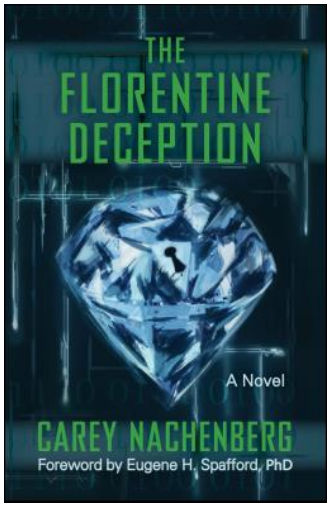
Lecture #1

Professor: Carey Nachenberg (*Please call me "Carey"*)
E-mail: climberkip@gmail.com

Class info: M/W 10am-12pm, Moore Hall 100
Office hours: Engineering VI 289
Mondays 12pm-1pm (after class)
Wednesdays 9am-10am (before class)
My Office: Engineering VI Room 297 (for 1-on-1 meetings)

Who Am I?





Carey Nachenberg

Age: 48

School: BS+MS in CS, UCLA '95

Work: Adjunct Prof at UCLA
Principal Engineer @ Google

Hobbies: Rock climbing, weight training, teaching CS!

My goal: To make an impact on your life (through getting you excited about programming)!

What You'll Learn in CS32

Advanced C++ Topics

Object Oriented Programming and C++ language features

Data Structures

The most useful data structures (e.g., lists, trees, hash tables)

Algorithms

The most useful algorithms (e.g., sorting, searching)

Building Big Programs

How to write large (> 1,000 lines of code) programs

Basically, once you complete CS32, you'll know 95% of what you need to succeed in industry!

Class Website

Official Class Website:
http://www.cs.ucla.edu/classes/winter20/cs32/

You should check the Official Class Website at least 2-3 times a week for homework info, projects, etc.

I will not always announce homework/projects so you have to track this on your own and be on top of the trash*!

* Being on top of the trash means being responsible**.

** If you're not responsible, you could get *rekt*.

CS32 Course Reader

If you'd like a course reader with all of my slides, you can either…

Download them from my website @
www.careynachenberg.com
and print them out yourself, or…

Buy a printed course reader from:

Copymat Westwood
10919 Weyburn Ave.
(310) 824 5276

Why buy a reader? Because you can take notes on it, use it during open-note exams and study from it!

Important Dates

Project #1: Due Tues, Jan 14th (next tues!)

Midterm #1: Thurs, Jan 30th
6pm-8pm (not during class hours!!)

Midterm #2: Tues, Feb 25th
6pm-8pm (not during class hours!!)

Final Exam: Sat, March 14th
One choice: 11:30-2:30pm
(This is the Saturday BEFORE Finals Week. Don't forget!)

Project #1: Due NEXT TUES!

In P1, we provide you with a simple C++ program that you have to add a few features to.

The goal of P1 is to allow you to self-evaluate to see if you're ready for CS32.

If you feel lost on P1…

You should seriously consider one of two things:

1. Drop the class, review your C++, and take CS32 in Spring, or
2. Suffer… Based on history, you'll get a C or lower in CS32… ☹

Random Administrative Stuff

We grade on a curve with the **average student getting a B-**, but if everyone gets above ~90%, **everyone gets an A/A-**!

But be careful! CS32 is a weeder class! If you take 16 units be prepared to **drop a class**! Also… **Start CS32 projects EARLY!**

Read & sign the **academic integrity agreement** and **don't cheat** – we catch people every year!

We have a **special grading policy** to **discourage cheating** on projects. Read the syllabus to understand it!

Compilers, Compilers, Compilers!

You must make sure that your homework/project code works properly with TWO different compilers to get credit!

Test your projects against both compilers at least a day or two before submitting them!

See the CS32 website

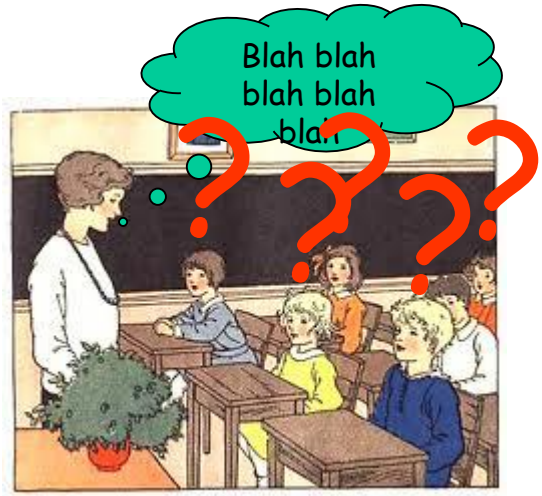
http://cs.ucla.edu/classes/winter20/cs32

For all of the gory details!

Carey's Thoughts on Teaching

It's more important that everyone understand a topic than I finish a lecture on time.

Don't be shy!!!
If something confuses you…
it probably confuses 5 other people too.
So always ask questions if you're confused!



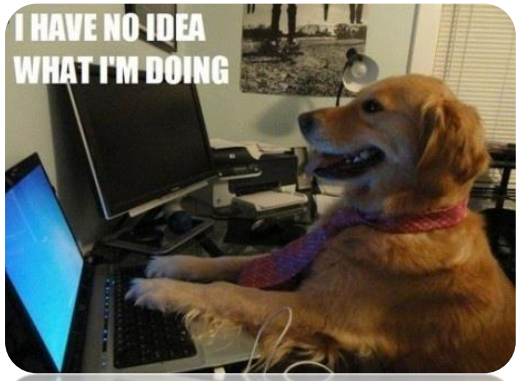
Always save more advanced questions for office hours or break.

I reserve the right to wait until office hours to answer advanced questions.

Carey's Thoughts on CS

Some people just "get" programming…

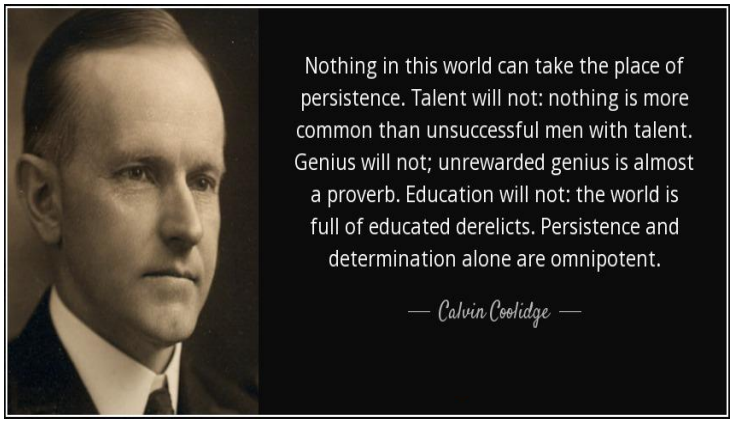
Others need to work hours and hours on it…



Don't freak out if you're in the latter camp.

If you enjoy coding… don't give up, don't worry about your grade, just practice and practice and practice.

There's nothing more important than grit when it comes to programming.

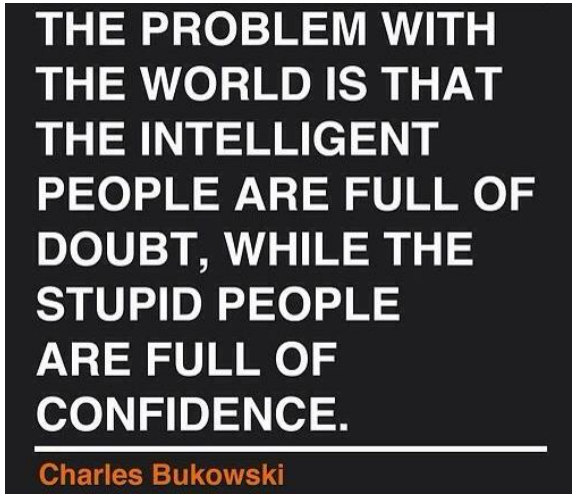


Use Me As a Resource!

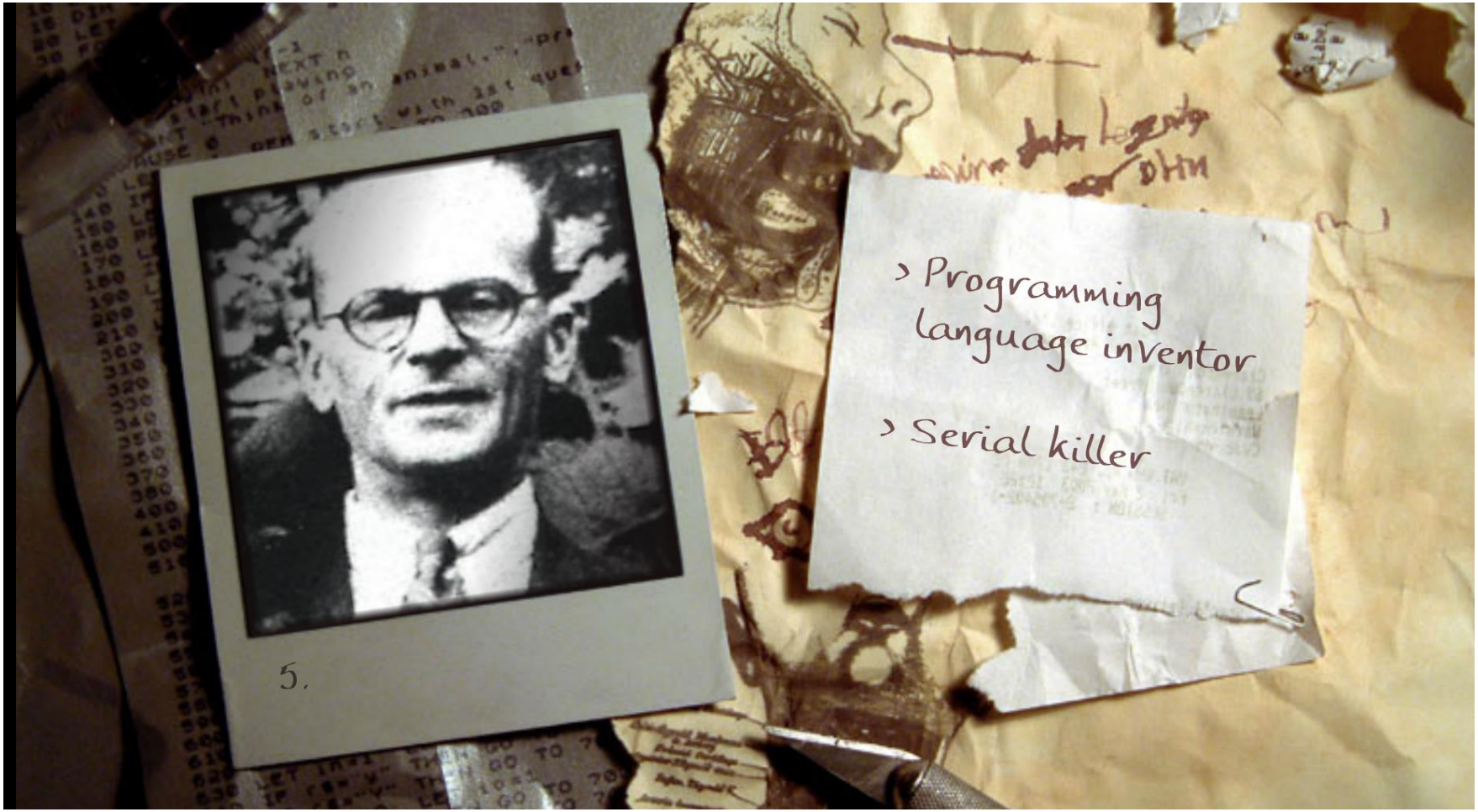
If you're struggling with projects, stressing out on exams, not getting the material, or just need some advice about life.

Come meet with me
Schedule a video chat
Let's talk on the phone
Or grab a coffee

Don't wait until 8th week when it's too late!



And now for a fun game!

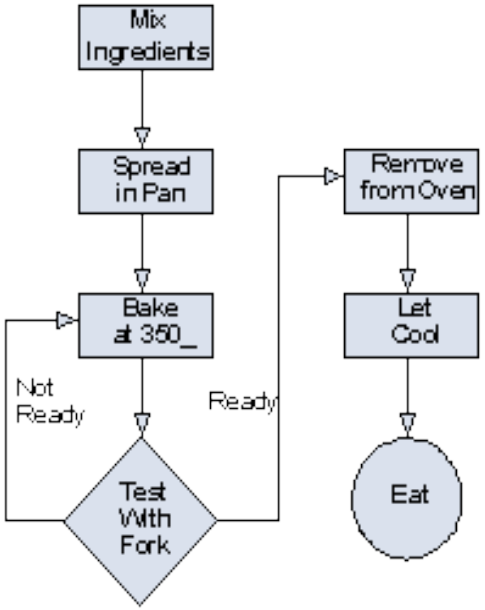


Is this guy a programming language inventor… or a SERIAL KILLER?!?!

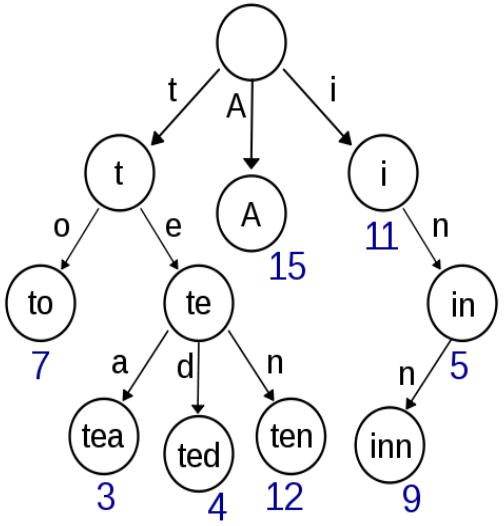
Alright… Enough administration!

Let's learn about…

Algorithms



Data Structures



What is an Algorithm

An algorithm is a set of instructions/steps that solves a particular problem.

Each algorithm operates on input data.

Each algorithm produces an output result.

Each algorithm can be classified by how long it takes to run on a particular input.

Each algorithm can be classified by the quality of its results.

Algorithm Comparison
"Guess My Word"

Let's say you're building a word guessing game.

The user secretly picks a random word from the dictionary.

Our program then must figure out what word the user picked as quickly as possible.

Let's consider two different algorithms…

Algorithm #1: Linear Search

Let's try a simple algorithm: We'll search linearly from top to bottom and ask the user about each word:

```
Start with the first word in the dictionary
While I haven't reached the end of the dictionary
{
    Read the current word out loud to the user
    Ask the user: "Is this the word you picked?"
    If they answer "Yes!", you're done!
    Otherwise advance to the next word
}
```

Question: If there are 100,000 total words in our dictionary, on average, how many guesses will our algorithm require?

Algorithm #2: Binary Search

Alright, for our second strategy let's try a more intelligent approach called binary search:

```
SearchArea = The entire dictionary
While I haven't guessed the user's word
{
    Pick the middle word w in the SearchArea
    Ask the user: "Is w your word?"
    If so, you're done! Woohoo!
    If not, ask: "Does your word come before or after w?"
    If their word comes before our middle word w
        SearchArea = first ½ of the current SearchArea
    If their word comes after our middle word w
        SearchArea = second ½ of the current SearchArea
}
```

Question: If there are 100,000 total words in our dictionary, on average, how many guesses will our Binary Search algorithm require?

Binary Search: How Many Guesses?

We keep on dividing our search area in half until we finally arrive at our word.

In the worst case, we must keep halving our search area until it contains just a single word – our word!

If our dictionary had 16 words, how many times would we need to halve it until we have just one word left?

16 8 4 2 1 It would take 4 steps

Ok, what if our dictionary had 131,072 words?

131072 65536 32768 16384 8192 4096 2048 1024 512 256 128 64 32 16 8 4 2 1

It would take just 17 steps!!! WOW!

Wow! That's Significant!

Our linear search algorithm requires an average of 50,000 steps to guess the user's secret word from a 100k word dictionary.

But our binary search algorithm requires just 17 steps, on average, to guess the user's secret word.

In CS32, you'll learn all of the major algorithms *and* how to analyze different algorithms so you can pick the best one for each task!

Data Structures

A data structure is the set of variable(s) that an algorithm uses to solve a problem.

A data structure could be as simple as an array…

Cold	0
Flu	1
Herpes	2
Measles	3
Rabies	4

Does David… have Herpes?

Cedric	0
David	1
Isaac	2
James	3
Nincy	4
Nora	5
Sally	6

	0	1	2	3	4
0	F	F	T	F	F
1	T	F	F	F	T
2	F	F	F	T	T
3	F	T	F	F	F
4	T	F	T	F	F
5	F	F	F	F	F
6	F	T	F	F	F

Algorithm
1. Find the person's row #.
2. Find the disease's column #.
3. If the cell has True in it, the person has had that disease.

Data Structures

Choosing the right data structure can make your algorithms far more efficient!

So in addition to learning all of the major algorithms this quarter…

You'll also learn all of the most powerful data structures in CS32!

Data Structures + Algorithms = Confusion!

Of course, your data structures and algorithms can get quite complex.

If you gave your code to another programmer, he/she would have no idea how to use it!

Therefore, every time you create a new set of data structures/algorithms, it helps to also create a few simple functions that hide the gory details…

Such a collection of simple functions is called an "interface."

```
bool hasDisease(string person, string disease)
void infectPerson(string person, string disease)
void curePerson(string person, string diasese)
```

An interface lets any programmer use your code without understanding the sordid details…

```
int main()
{
    infectPerson("Linda", "Cold");
    if (hasDisease("David","Herpes") == true)
        cout << "Time to go to the Ashe center!\n";
}
```

Algorithms & Data Structures Are a Perfect Couple!

Algorithms and data structures are like peas and carrots – they belong together!

To solve a problem, you have to design both the algorithms and the data structures together.

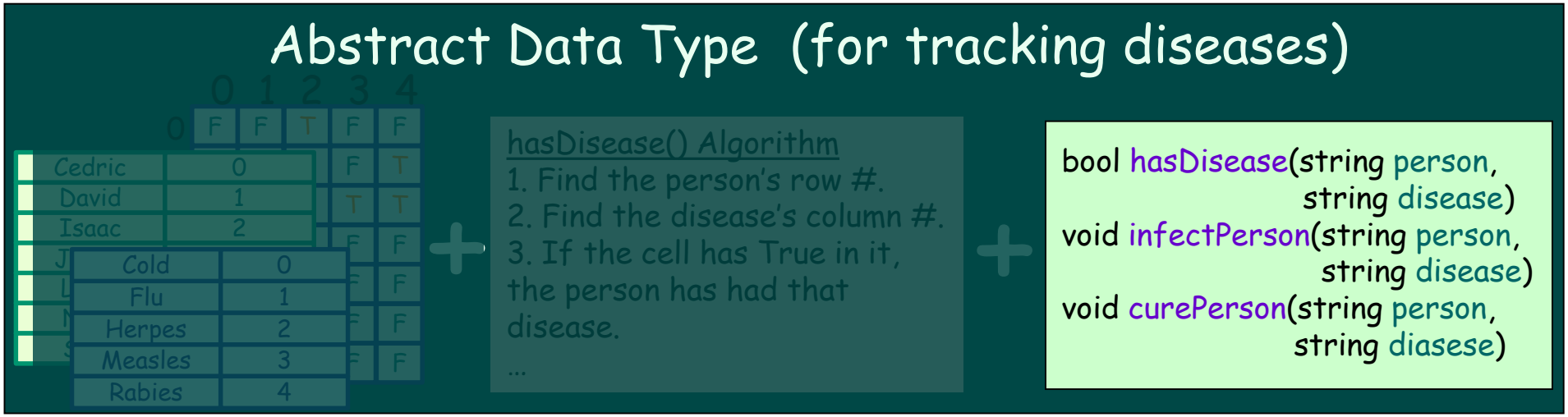
Then you provide a set of simple "interface" functions to let any programmer use them easily.

And in fact, we even have a name for such a (data structure + algorithm + interface) solution to a problem…

The Abstract Data Type (ADT)

An Abstract Data Type or ADT is:

A coordinated group of
(a) data structures, (b) algorithms and (c) interface functions
that is used to solve a particular problem.



In an ADT, the data structures and algorithms are secret.

The ADT provides an interface (a simple set of functions)
to enable the rest of the program to use the ADT.

Typically, we build programs from a collection of ADTs, each of which
solves a different sub-problem.

ADTs in C++

We can use C++ classes to define ADTs in our C++ programs! Each C++ class can hold algorithms, data and interface funcs.

```
int main()
{
  DiseaseTracker x;

  x.infectPerson("Carey",
                 "Cooties");
  x. infectPerson("David",
                 "Explosive Diarrhea");
  …

  string person, disease;

  cin >> person >> disease;
  if (x.hasDisease(person,
                   disease) == true)
    cout << person << " has got "
         << disease;
}
```

```
// A C++ disease tracking class…
// (this is really an ADT!)

class DiseaseTracker
{
public:
  // our interface functions go here
  void hasDisease(…);
  void infectPerson(…);
    …
private:
  // secret algorithms go here
  …
  // secret data structures go here
  …
};
```

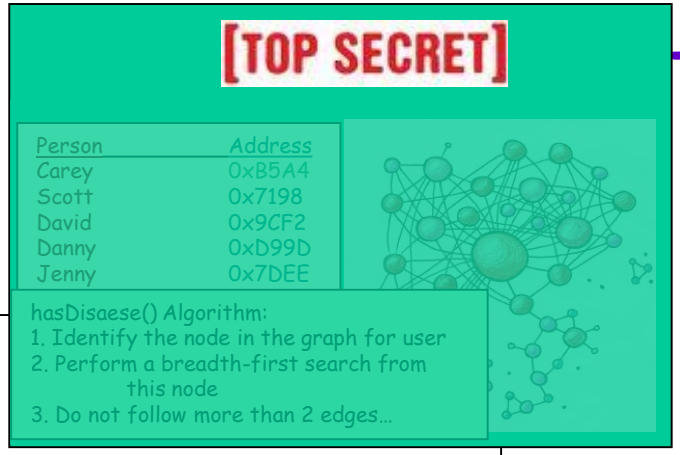
Once we've defined our class, the rest of our program can use it trivially.

All our program needs to do is call the functions in our class's public interface!

And yet all of its underlying data structures and algorithms are hidden!

The rest of the program can ignore the details of how our class works and just use its features!

This is the power of the ADT/class!



++

```
int main()
{
  DiseaseTracker x;

  x.infectPerson("Carey",
                 "Cooties");
  x. infectPerson("David",
                 "Explosive Diarrhea");
  ...

  string person, disease;

  cin >> person >> disease;
  if (x.hasDisease(person,
                   disease) == true)
    cout << person << " has got "
         << disease;
}
```

Now what if I wanted to improve my class's data structures and algorithms?

Let's say I made a radical change to my disease tracker data structures…

Would the user of my class need to change any part of her program?

No! Because the rest of the program doesn't rely on these details - it knows nothing about them!

This is the power of ADTs (and of classes)!

If you design your programs using this approach, you can reduce complexity by:

Breaking your programs up into small, self-contained chunks.

Combining these chunks together to solve bigger (more complex) problems.

What is Object Oriented Programming?

Object Oriented Programming (OOP) is simply a programming model based on the Abstract Data Type (ADT) concept we just learned!

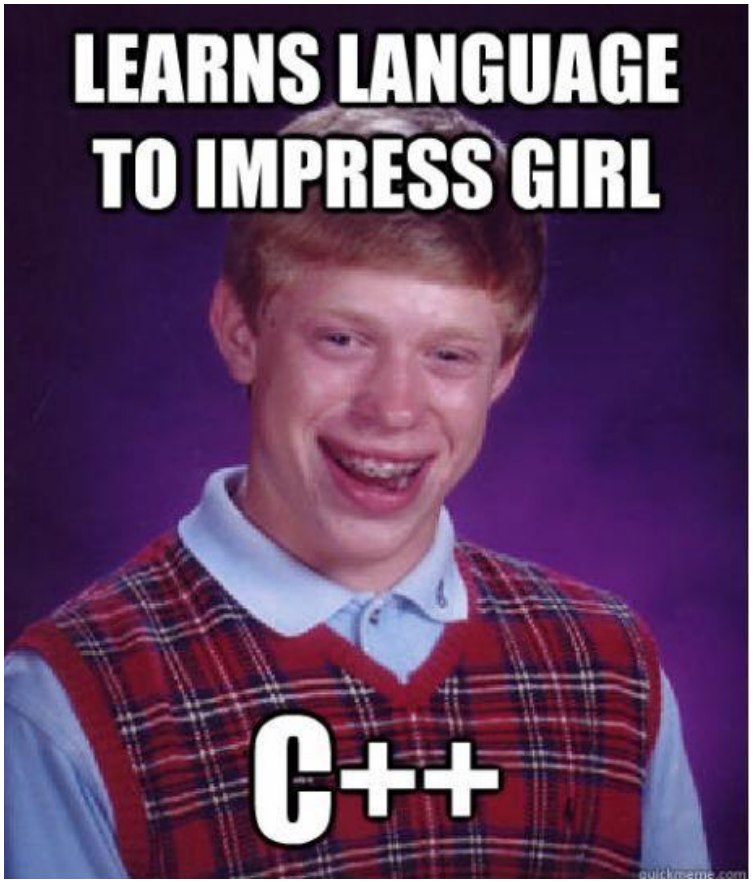
In OOP, programs are constructed from multiple self-contained classes.

Each class holds a set of data and algorithms - we then access the class using a simple set of interface functions!

Classes talk to each other only by using public interface functions – each class knows nothing about how the others work inside.

So to sum up:
OOP is using classes (aka ADTs) to build programs – plus some other goodies we'll learn soon!

Intermission Meme



C++ Class Review

As we've seen, a "class" is a self-contained problem solver that contains:

- Data structures
- Algorithms
- Interface functions

Since you've probably forgotten everything about classes…

Let's do a quick review of classes by defining our own Nerd class!

Defining a New Class

```
class Nerd
{
 public:
    void init() {
      myStinkiness = 0;
      myIQ = 100;
    }
    void study(int hours) {
      myStinkiness += 3*hours;
      myIQ *= 1.01;
    }
    int getStinkyLevel() {
       int total_stink = myIQ * 10 +
            myStinkiness;
       return total_stink;
    }
 private:
    int myStinkiness, myIQ;

};
```

First, we write the outer shell of our class and give it a name.

Then we define our class's public interface functions…

Then we define our class's private variables and functions…

Our class defines an entirely new data type, like string, that we can now use in our program.

Alert: Nerd is not a variable! It's a new C++ data type!

Using a New Class

Once we define a new class, like Nerd, we can use it to define variables like any traditional data type.

nerd.h

```
class Nerd
{
 public:
    void init() {
      myStinkiness = 0;
      myIQ = 100;
    }
    void study(int hours) {
      myStinkiness += 3*hours;
      myIQ *= 1.01;
    }
    …
 private:
    int myStinkiness, myIQ;
};
```

ucla.cpp

```
#include "nerd.h"

int main()
{
   int num_nerds = 1;
   Nerd david;

   david.init();
}
```

You typically define each new class in its own header file.

To use your new class, simply include its header file using quotation marks…

And then define variables with it throughout your program.

Just as num_nerds is an integer variable…

david is a Nerd variable.

Using a New Class

Alright, let's see our class in action!

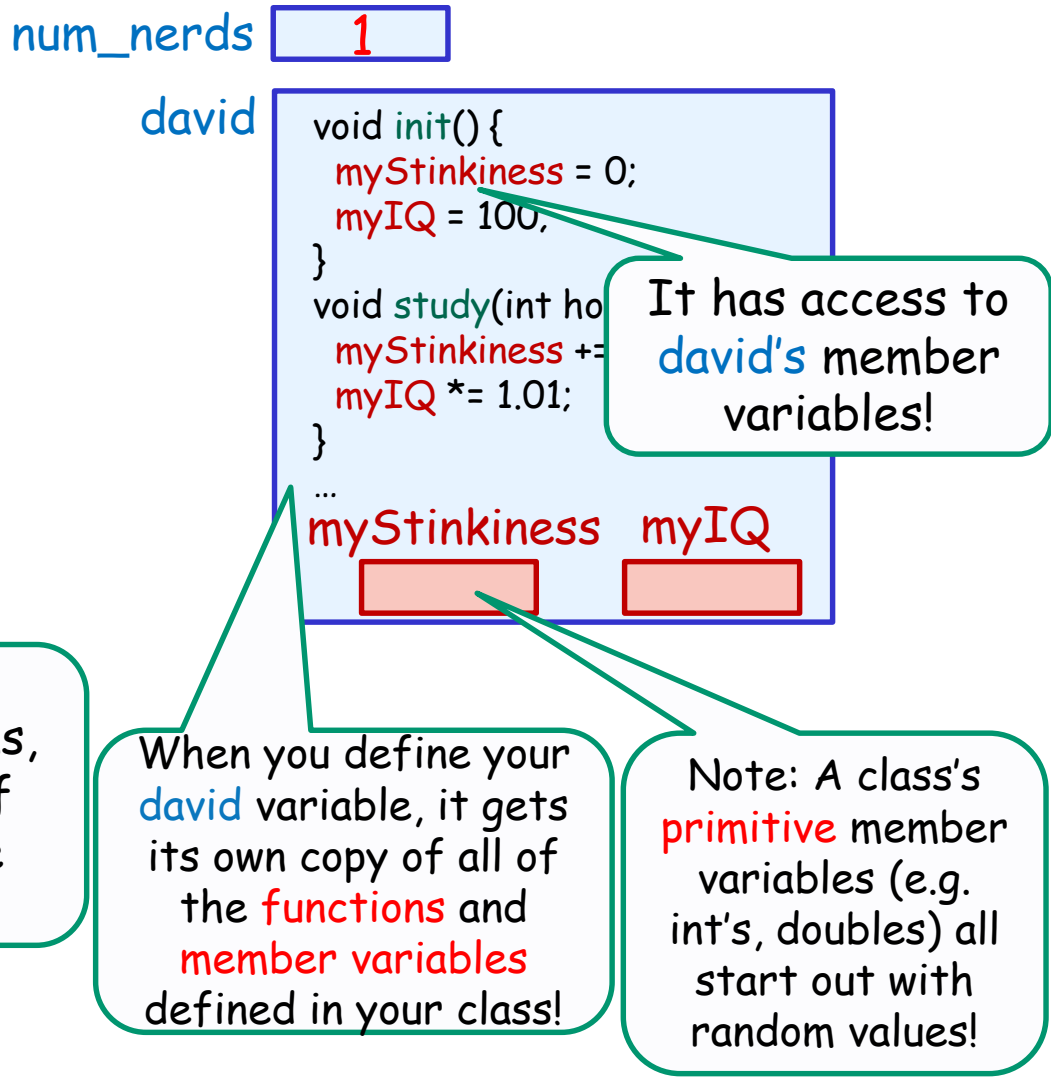
nerd.h

```
class Nerd
{
 public:
    void init() {
      myStinkiness = 0;
      myIQ = 100;
    }
    void study(int hours) {
      myStinkiness += 3*hours;
      myIQ *= 1.01;
    }
    ...
 private:
    int myStinkiness, myIQ;
};
```

ucla.cpp

```
#include
int main(
{
   int num
   Nerd david

   david.init();
}
```



Other Details

You typically only use member variables to store permanent attributes of your class.

nerd.h

```
class Nerd
{
 public:
    void init() {
      myStinkiness = 0;
      myIQ = 100;
    }
    int getStinkyLevel() {
       int total_stink = myIQ
            * 10 + myStinkiness;
       return total_stink;
    }
 private:
    int myStinkiness, myIQ;
};
```

ucla.cpp

```
#include "nerd.h"

int main()
{
   int num_nerds = 1;
   Nerd david;

   david.init();
}
```

Stinkiness and IQ are inherent attributes of every Nerd.

So we make these member variables.

On the other hand, we use local variables for temporary computations.

nerd.h

```
class Nerd
{
 public:
    void init() {
      myStinkiness = 0;
      myIQ = 100;
    }
    void study(int hours) {
      myStinkiness += 3*hours;
      myIQ *= 1.01;
    }
    ...
 private:
    int myStinkiness, myIQ;
};
```

So they can only be accessed by your class's member functions.

C++ strictly enforces class privacy!

Hiding the internal details of a class is called "encapsulation."

All functions in the public section can be used by all parts of your program.

However, these members are private.

ucla.cpp

```
#include "nerd.h"

int main()
{
   int num_nerds = 1;
   Nerd david;

   cout << david.myStinkiness;
}
```

Code outside your class can't directly access them, however.

ERROR!