Welcome to Computer Science 32

Lecture #1

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

Class info: M/W 10am-12pm, 3400 Boelter Hall
Office hours: 4549 Boelter Hall
Mondays 12pm-1pm (after class)
Wednesdays 9am-10am (before class)
My Office: 4531N Boelter Hall

Pick a # between 1 - 60

16	21	26	31	52	57
17	22	27	48	53	58
18	23	28	49	54	59
19	24	29	50	55	60
20	25	30	51	56	*

4	13	22	31	44	53
5	14	23	36	45	54
6	15	28	37	46	55
7	20	29	38	47	60
12	21	30	39	52	*

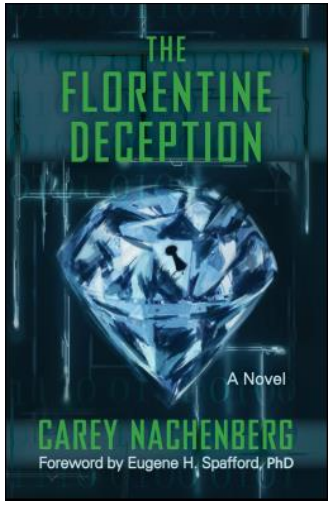
2	11	22	31	42	51
3	14	23	34	43	54
6	15	26	35	46	55
7	18	27	38	47	58
10	19	30	39	50	59

8	13	26	31	44	57
9	14	27	40	45	58
10	15	28	41	46	59
11	24	29	42	47	60
12	25	30	43	56	*

32	37	42	47	52	57
33	38	43	48	53	58
34	39	44	49	54	59
35	40	45	50	55	60
36	41	46	51	56	*

1	11	21	31	41	51
3	13	23	33	43	53
5	15	25	35	45	55
7	17	27	37	47	57
9	19	29	39	49	59

Who Am I?



Carey Nachenberg

Age: 44

School: BS, MS (No PhD!) in CS/E, UCLA '95

Work: UCLA: Adjunct Prof
Symantec: Vice President

Hobbies: Rock climbing, weight training, teaching CS, writing novels!

My goal: To get you *excited* about programming!

Class Website

Official Class Website:
http://www.cs.ucla.edu/classes/winter16/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!

I'll post my PowerPoint slides on my private website:
http://careynachenberg.com

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 (> 500 lines of code) programs

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

Important Dates

Project #1: Due Tues, Jan 12th (next tues!)

Midterm #1: Thurs, Jan 28th
Two choices: 5-6:05pm OR 5:45-6:50pm
(You must sign up 1 week in advance for either midterm)

Midterm #2: Thurs, Feb 25th
Two choices: 5-6:05pm OR 5:45-6:50pm
(You must sign up 1 week in advance for either midterm)

Final Exam: Sat, March 12th
One choice: 11:30-2:30pm
(This is the Saturday BEFORE Finals Week. Don't forget!)

Project #1: Due Tues, Jan 12th

Your first CS32 project is already posted on the class website and is due next Tues!

In P1, we provide you with a simple C++ program.

Your job is to get the program compiling and then make a few changes to it.

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++ this quarter and take CS32 in Spring, or
2. Suffer… Based on history, if you have problems on P1, 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**!

But be careful! CS32 is a weeder class and can easily take 30-40 hours per week of work! **Start 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 all programs you turn in compile and work with both Visual C++ and gcc (under linux or Mac)!

Note: Make sure your projects work with both compilers at least a day or two before submitting them!

You can get a free copy of Visual C++ from:
http://www.visualstudio.com/downloads/download-visual-studio-vs#d-express-windows-desktop

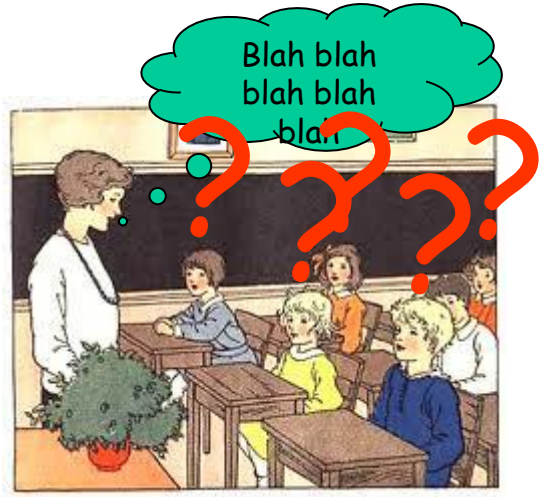
You can get a free copy of Apple Xcode/gcc from the Apple App store, or http://developer.apple.com/xcode/

You can also use gcc and Visual C++ in the lab.

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!

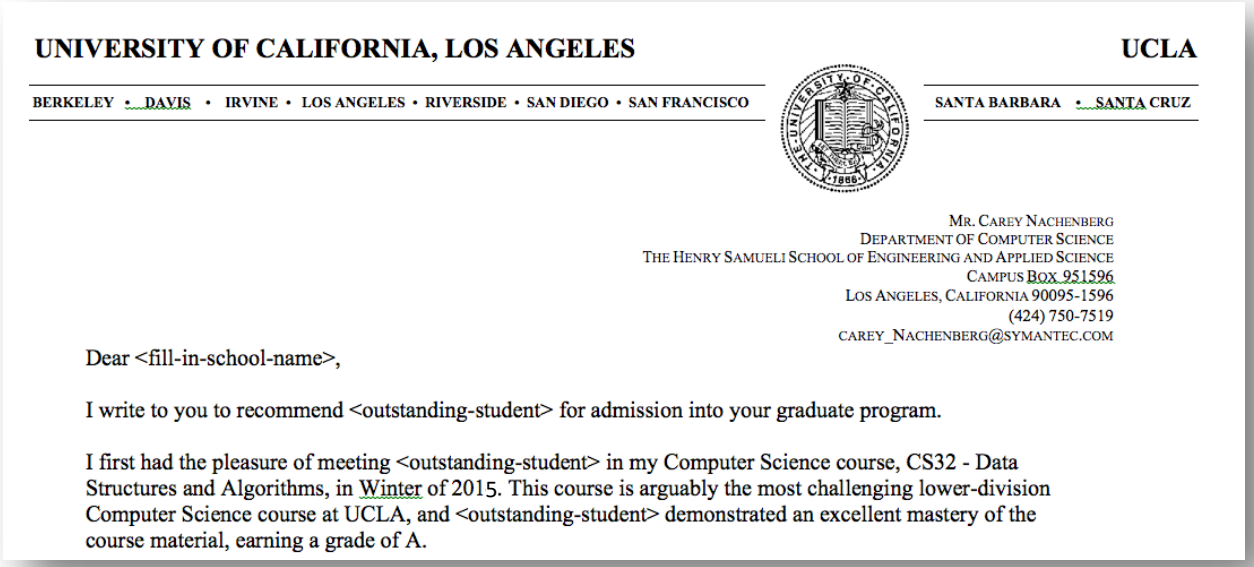


Always save more advanced questions for office hours or break.

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

Letters of Recommendation

If you want a letter of recommendation from me…



UNIVERSITY OF CALIFORNIA, LOS ANGELES **UCLA**

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO

SANTA BARBARA • SANTA CRUZ

MR. CAREY NACHENBERG
DEPARTMENT OF COMPUTER SCIENCE
THE HENRY SAMUELI SCHOOL OF ENGINEERING AND APPLIED SCIENCE
CAMPUS BOX 951596
LOS ANGELES, CALIFORNIA 90095-1596
(424) 750-7519
CAREY_NACHENBERG@SYMANTEC.COM

Dear <fill-in-school-name>,

I write to you to recommend <outstanding-student> for admission into your graduate program.

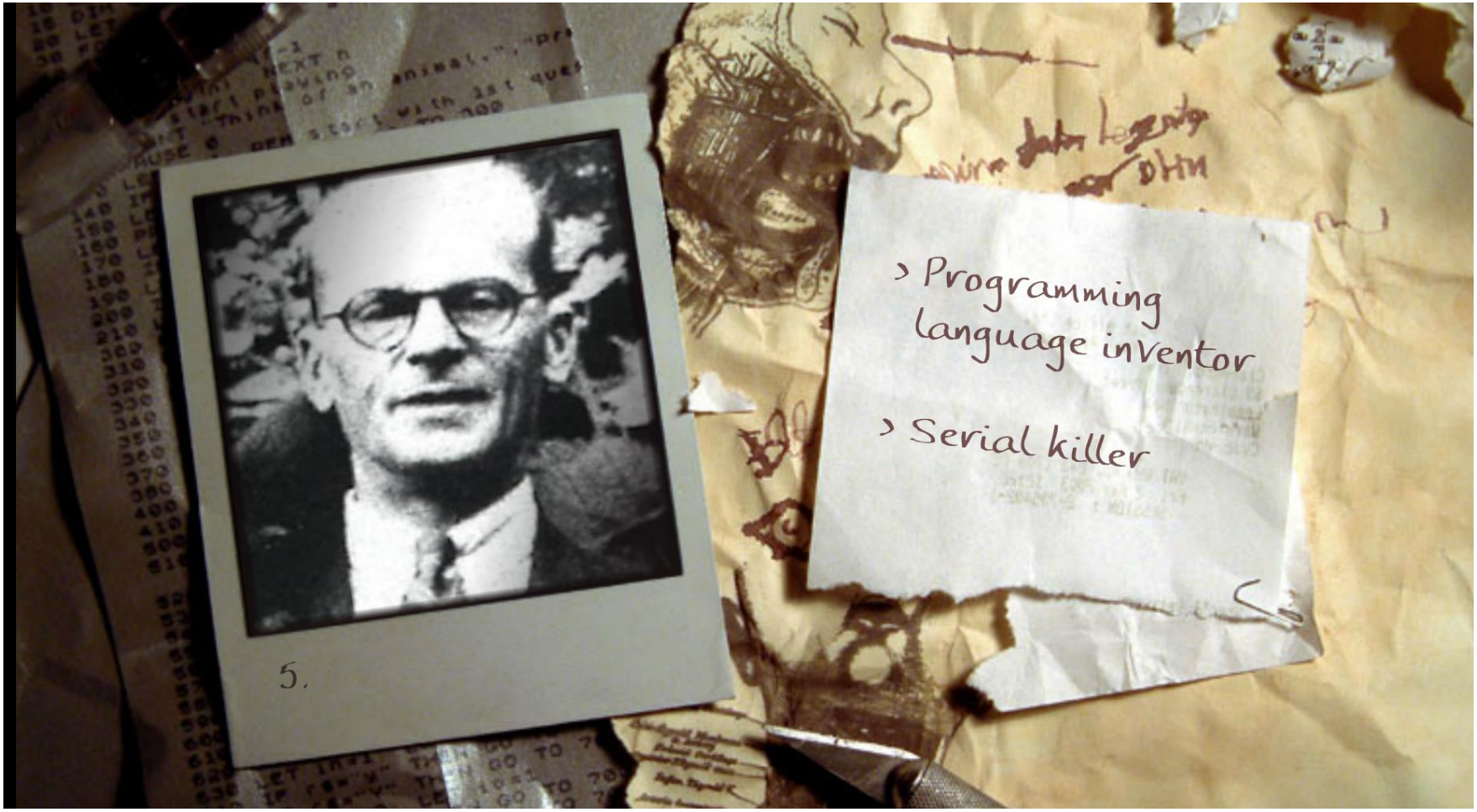
I first had the pleasure of meeting <outstanding-student> in my Computer Science course, CS32 - Data Structures and Algorithms, in Winter of 2015. This course is arguably the most challenging lower-division Computer Science course at UCLA, and <outstanding-student> demonstrated an excellent mastery of the course material, earning a grade of A.

Make sure to *come to office hours every week* so I get to know you: your hobbies, accomplishments, goals, etc!

If I don't know you extremely well, I won't write a letter!

(Oh, and also make sure to get an A in CS32 ☺)

And now for a fun game!

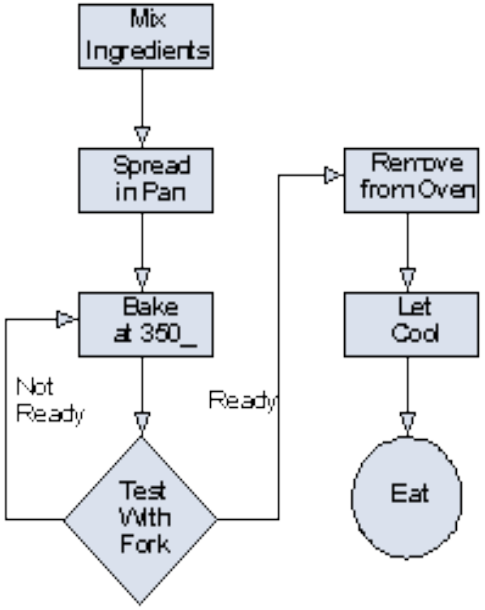


Is this guy a programming language inventor… or a SERIAL KILLER?!?!

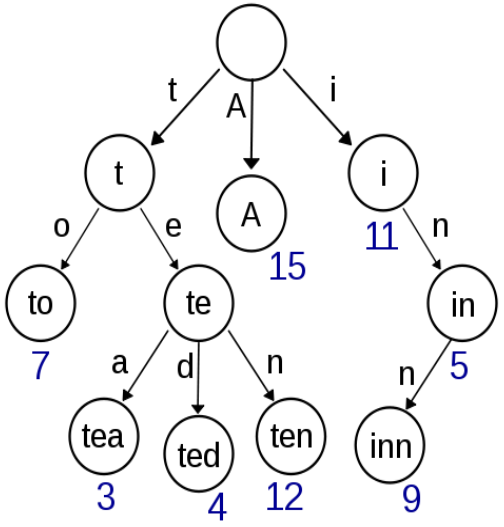
Alright… Enough administration!

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.

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:

```
j = 0
While I haven't guessed the user's word
{
    Select word j from the dictionary
   Ask the user if word j is the word they picked
   j = j + 1
}
```

Question: If there are N 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 N total words in our dictionary, on average, how many guesses will our 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 N=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 N=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!

Can we come up with a general formula to determine the # of steps?

Binary Search: Estimating the # of Steps

Well, if you remember your math…

You know that $\log_2(N)$ is equal to the number of times you can divide N by 2 before you reach a value of 1.

$\log_2(8) = 3$, since 8 → 4 → 2 → 1

$\log_2(64) = 6$, since 64 → 32 → 16 → 8 → 4 → 2 → 1

$\log_2(1 \text{ billion}) = ?$ 1B → 500M → 250M → 125M→ …

So no matter what word the user picks, we can find it in $\log_2(N)$ steps for a dictionary with N words!

Wow! That's Significant!

Our linear search algorithm requires N/2 steps, on average, to guess the user's secret word.
(~50,000 steps for a dictionary with 100,000 words)

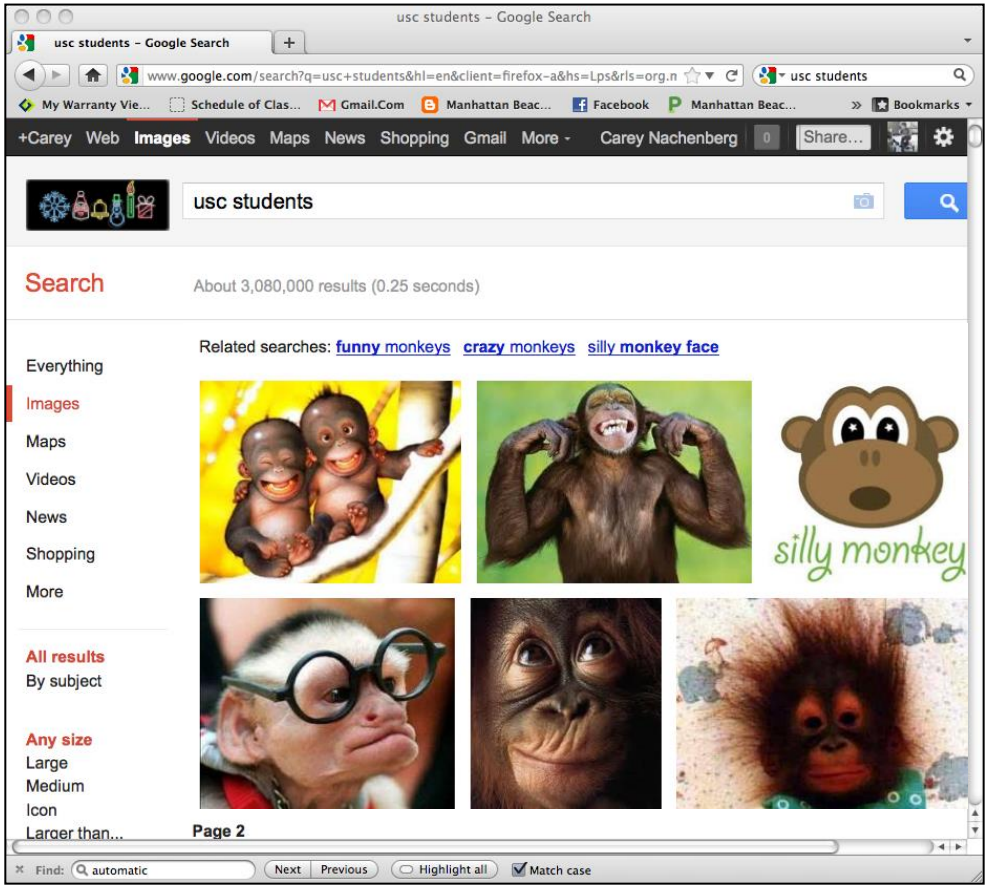
But our binary search algorithm only requires $\log_2(N)$ steps, on average, to guess the user's secret word.
(~17 steps for a dictionary with 100,000 words)

If, instead of a dictionary, we had to search a database of 300 million people, which algorithm would you choose?

Bad Algorithms Can Kill a Program!

Bad algorithms can be way too slow!

Bad algorithms can produce bad results:



Data Structures

A data structure is the data that's operated on by an algorithm to solve a problem.

Sometimes an algorithm can operate just on the data passed in to it:

```
void printNumbersBackward(int array[], int numItems)
{
    while (numItems > 0)
    {
        --numItems;
        cout << array[numItems];
    }
}
```

Solves the problem with no new variables or data!

Other times an algorithm will have to create its own secondary data structures to solve a problem.

A Data Structure for Facebook

Imagine that you're building a social network website like Facebook…

Friend A	Friend B
Danny Hsu	Jenny Oh
Danny Hsu	Rich Lim
Jenny Oh	David Sun
Danny Hsu	Carey Nash
Carey Nash	Scott Wilhelm
Melani Kan	Danny Hsu
Carey Nash	David Small
Jenny Oh	Len Kleinrock
Jenny Oh	Mario Giso
Mario Giso	Rich Nguyen

Hmmm. But if we had 100 million users with 10 billion friendships, that might be a bit too slow! ☺

And you want to keep track of who made friends with whom…

How would you do it?

Well, one *data structure we could use would be a long* list of every pair of friends…

Now how could I find out if Jenny Oh is a friend of Mario Giso?

Well, we could search through every pair until we find the friendship we are looking for…

A Better Friendship Data Structure

What if instead we assigned each of our N users a number…

And kept an alphabetized list of each user → number pair.

User	Number
Carey Nash	0
Danny Hsu	1
David Small	2
David Sun	3
Jenny Oh	4
Len Kleinrock	5

FriendA	FriendB
Danny Hsu	Jenny Oh
David Sun	Carey Nash
Jenny Oh	David Sun
Danny Hsu	Carey Nash
Carey Nash	Scott Wilhelm
Melani Kan	Danny Hsu
Carey Nash	David Small

Now given a user's name, how can we quickly find their number?

Right – just use a Binary Search!

1. Look up their numbers A, B
2. Check (A,B) *or* (B,A) in our array
3. If we find a *, they're buddies!

to represent relationships!

	0	1	2	3	4	5	6	7	8	9	10
0				*							
1	*										
2											
3	*				*						*
4		*		*			*	*	*		
5											
6					*						
7					*						
8					*					*	
9									*		
10				*							

A Better Friendship Data Structure

So what we've done is created a new data structure that makes it faster to determine if two people are friends!

Coupled with our new, more efficient algorithm, we can quickly find out if two people are friends!

User	Number
Carey Nash	0
Danny Hsu	1
David Small	2
David Sun	3
Jenny Oh	4
Len Kleinrock	5
Mario Giso	6
Melani Kan	7

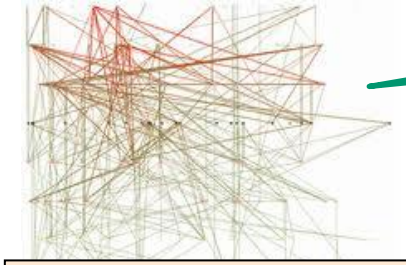
	0	1	2	3	4	5	6	7	8	9	10
0		*	*	*							
1	*										
2	*										
3	*				*	*					*
4				*			*	*	*		
				*							
					*						
					*						
					*					*	
									*		
				*							

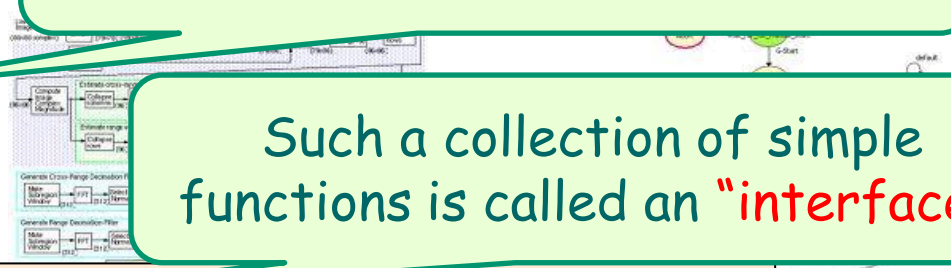
Friend-finding Algorithm:

1. Look up their numbers A, B
2. Check (A,B) or (B,A) in our array
3. If we find a *, they're buddies!

Data Structures + Algorithms = Confusion!

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





Without having to understand any of the sordid details!

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

```
bool AreTheyFriends(string UserA, string UserB)
void BecomeFriends(string UserA, string UserB)
void BreakUp(string UserA, string UserB)
```

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…

An interface lets any programmer use your code…

```
int main()
{
    BecomeFriends("Rob Crouch", "Eric Chien");
    if (AreTheyFriends("Carey N","David S") == true)
        cout << "Carey & David are BFFs!\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.

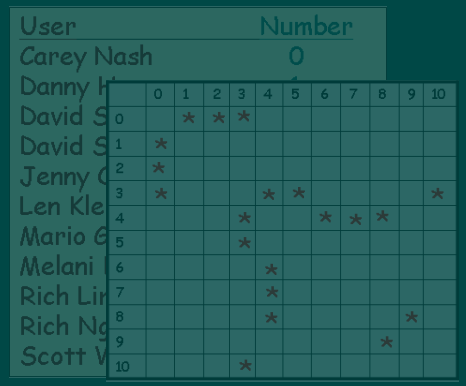
And in fact, we even have a name for such a (data structure + algorithm + interface) solution to a problem…

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."

Abstract Data Type (for finding friends)



+

Add Friend Algorithm:

1. Look up each person's #s
2. Stick a * in grid[n1][n2]
3. Stick a * in grid[n2][n1]

Friend-finding Algorithm

1. Look up each person's #s
2. Check grid[n1][n2] and grid[n2][n1]
3. If we find a *, they're buddies!

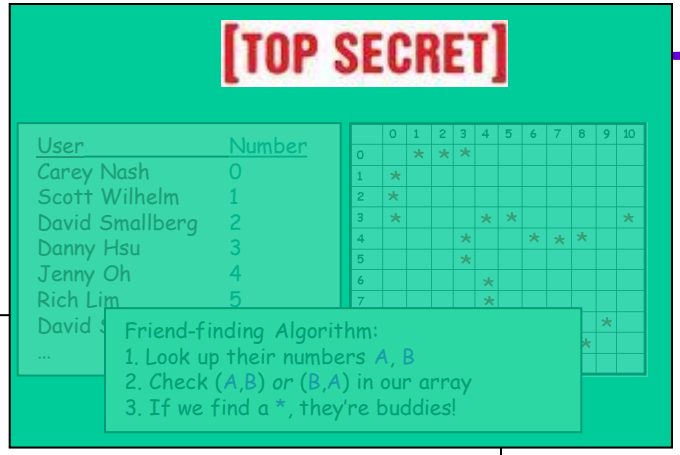
+

bool AreTheyFriends(UserA,UserB)
void BecomeFriends(UserA,UserB)
void BreakUp(UserA,UserB)
...

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.



++

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

```
int main()
{
  FriendFinder x;

  x.BecomeFriends("Carey Nash",
                  "Scott Wilhelm");
  x.BecomeFriends("Jenny Oh",
                  "David Sun");
  ...

  string userA, userB;

  cin >> userA >> userB;
  if (x.AreTheyFriends(userA,
                       userB) == true)
    cout << userA << " is friends with "
         << userB;
}
```

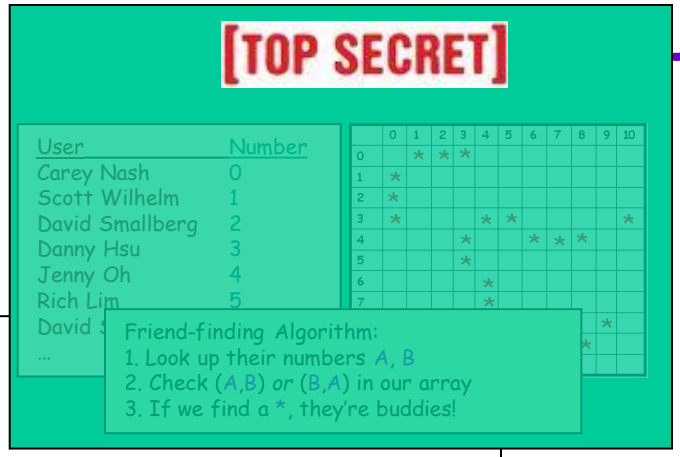
Once we've defined our class, the rest of our program can use it trivially.

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!



++

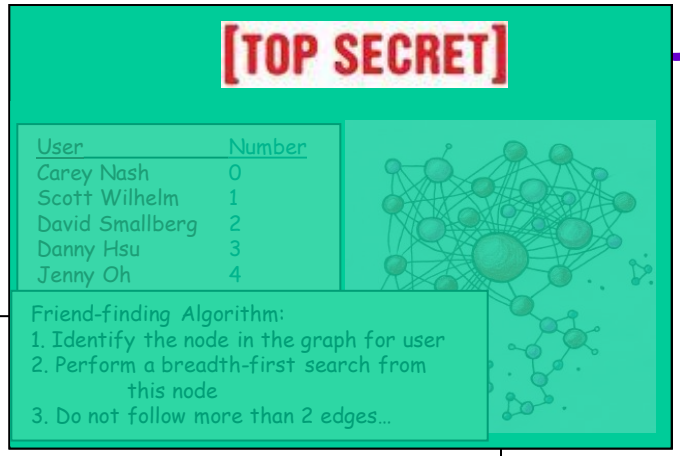
```
int main()
{
  FriendFinder x;

  x.BecomeFriends("Carey Nash",
                  "Scott Wilhelm");
  x.BecomeFriends("Jenny Oh",
                  "David Sun");
  ...

  string userA, userB;

  cin >> userA >> userB;
  if (x.AreTheyFriends(userA,
                       userB) == tr
    cout << userA << " is friends with
         << userB;
}
```

We use a similar approach to construct real-life objects!

Consider your car…

It has an easy-to-use interface like the steering wheel, brake and accelerator pedals.

The complicated machinery is hidden from you – you don't need to know how it works to use it!



++

```
int main()
{
  FriendFinder x;

  x.BecomeFriends("Carey Nash",
                  "Scott Wilhelm");
  x.BecomeFriends("Jenny Oh",
                  "David Sun");
  ...

  string userA, userB;

  cin >> userA >> userB;
  if (x.AreTheyFriends(userA,
                       userB) == true)
    cout << userA << " is friends with "
         << userB;
}
```

Now what if I wanted to improve my FriendFinder data structures?

Let's say I made a radical change to our friend network 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!

The same is true for your car! If you replace your engine, you don't care – you just use the steering wheel, accelerator and brakes like you did before!

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!

C++ Class Review

Ok, since I know you're rusty, let's do a quick review of C++ structs and classes.

Topic #1: C++ structs

Sometimes we want to group together multiple related pieces of data and store them in a single variable.

For instance, to represent a person in my program, I need to store their name, age, phone number, address, etc.

The problem is that I have to define four separate variables to represent a single entity (like a person).

It'd be nice if we had some way to define a single variable that can hold all of our related values.

```
int main()
{
   string name;
   int age;
   int phoneNum;
   string address;
   name = "Carey";
   age = 44;
   …


   PERSON   p;
   p.name = "Carey";
   p.age = 44;
   …
}
```

Topic #1: C++ structs

The C++ struct allows us to define a whole new type of variable that holds multiple related pieces of data rather than just one value.

Let's see how to use this feature to define a nerd type!

Each nerd has:

A name

An IQ

A GPA

C++ struct types

```
struct NERD
{
   string name;
   int    IQ;
   float  GPA;
};
```

First, write the word struct.

Then we give our new struct a name…

Then we specify all of the variables that our new structure contains in between { and }

Finally, we add a semicolon ; at the end.

This 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 data type!

C++ struct types

Once we define our new data type, like NERD, we can use it to define variables like any traditional data type.

nerd.h

```
struct NERD
{
     string name;
     int IQ;
     float GPA;
};
```

You typically define a new struct type in a header file.

nerd.cpp

```
#include "nerd.h"

int main()
{
    float f1;
    NERD n1, n2;

    n1.name = "David";
    …
```

After you define a new struct type, you can create variables with it throughout your program.

Just as f1 is a floating-point variable…

n1 and n2 are NERD variables…

C++ struct ty

Let's see how we can use our new struct type in action!

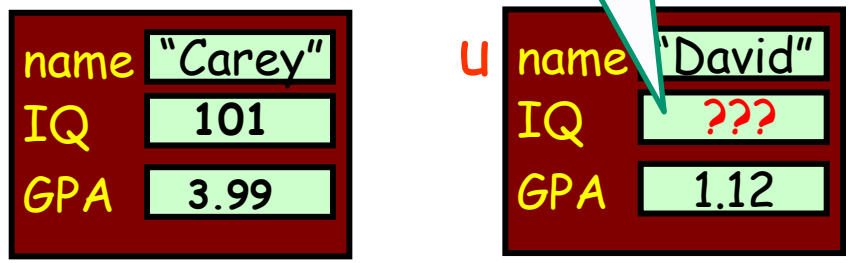
Notice that if you don't initialize a member in a struct variable, then it has a random value! This holds for all built-in types: chars, ints, doubles, etc…

nerd.h

```
struct NERD
{
    string name;
    int IQ;
    float GPA;
};
```

nerd.cpp

```
#include "nerd.h"
int main()
{
   NERD t, u;

   t.name = "Carey";
   t.IQ = 101;
   t.GPA = 3.99;

   u.name = "David";
   u.GPA  = 1.12;
```



Neato! C++ structs let us group related data together into one common unit.

But wait! Wouldn't it be cool if our struct could also hold algorithms to operate on that data, as well as interface functions?

Then we could represent a complete, self-contained Abstract Data Type!

Topic #2: C++ classes

Consider the following structure and associated function(s).

```
struct Rect
{
   int width;
   int height;
};

int GetArea(Rect &a)
{
  int area = a.width *
             a.height;
  return(area);
}
```

a

area 200

10 * 20

200

r

width 10

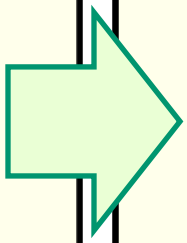
height 20

```
int main(void)
{
   Rect r;

   r.width = 10;
   r.height = 20;

   cout << GetArea(r);
```

Topic #2: C++ classes

A class is basically a struct with built-in functions in addition to member variables.

```
struct Rect
{
   int width;
   int height;
};

int GetArea(Rect &a)
{
  int area = a.width *
             a.height;
  return(area);
}
```



```
class Rect
{

   int GetArea()
   {
     int area =   width *
                  height;
     return(area);
   }

   int width;
   int height;
};
```

Using a Class

```
class Rect
{
    int GetArea()
    {
      int area = width * height;
      return(area);
    }

    int width;
    int height;
};


int main(void)
{
   Rect  r;

   r.width = 10;
   r.height = 20;

   cout << r.GetArea();
```

GetArea() is a "member function" of our Rect class.

width and height are called "member variables"

When you define your r variable, it gets its own copy of all of the functions and variables defined in your class!

r

```
int GetArea()
{
  int area = width * height;
  return(area);
}

width
height
```

area

In addition to calling r a variable (which it is), we also sometimes call it an "instance" or an "object".
"r is an instance of the Rect class"
"r is a Rect object"

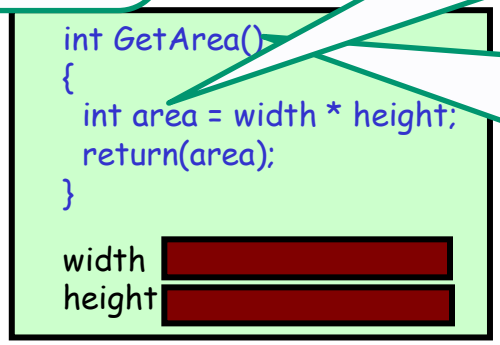
Using a Class

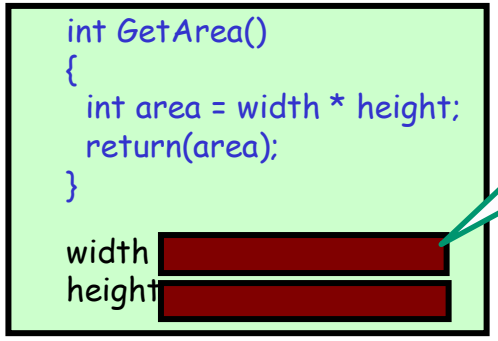
```
class Rect
{
    int GetArea()
    {
      int area = width * height;
      return(area);
    }

    int width;
    int height;
};


int main(void)
{
   Rect  r, s;

   r.width = 10;
   r.height = 20;

   cout << r.GetArea();
```

You should ONLY add a member variable to your class if you want it to remember the value permanently.

And of course, I can define as many Rect objects as I like!

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





Each "instance" gets its own copy of the GetArea() function…

and its own width and height member variables.

Classes

```
class Rect
{

    int GetArea()
    {
      int area = width * height;
      return(area);
    }
   void Initialize(int startW, int startH)
   {
      width = startW;
      height = startH;
   }


   int width;
   int height;
};
```

We can add as many member functions as we like to our class…

```
int main(void)
{
   Rect r;
   r.Initialize(10,20);
   cout << r.GetArea();
```

Public

```
class Rect
{
public:
    int GetArea()
    {
      int area = width * height;
      return(area);
    }
   void Initialize(int startW, int startH)
   {
      width = startW;
      height = startH;
   }
private:
    int width;
    int height;
};
```

The public keyword tells C++ what part of your class you want to expose publicly…This is how you "interface" with it!

Since GetArea() is in the public section, it can be used by all parts of your program.

Similarly, since Initialize() is in the public section, it can be used by all parts of your program.

And here we see our main() function using the GetArea() function!

Oh, and there're two more things we have to add to make our syntax correct…

```
int main(
{
   Rect r;

   r.Initialize(10,7);
   cout << r.GetArea();
```

And here we see our main() function using the Initialize() function!

Public vs. Private

```
class Rect
{
public:
   int GetArea()
   {
     int area = width * height;
     return(area);
   }
   void Initialize(int startW, int startH)
   {
     width = startW;
     height = startH;
   }
private:
   int width;
   int height;
};
```

Only your member functions may access this private stuff!

The private keyword tells C++ what part of your class you want to hide from the outside world.

This is where you put your class's private data & functions.

Any attempt to access private members outside your class will result in an error!

```
int main(void)
{
   Rect r;

   r.Initialize(10,7);
   cout << r.GetArea();
   r.width = 10; // ERROR!
```

Using Public+Private Properly

```
class Rect
{
public:
   int GetArea()
   {
     int area = width * height;
     return(area);
   }

   void Initialize(int startW, int startH)
   {
      width = startW;
      height = startH;
   }
private:
   void myComplexAlgorithm(float angle)
   {
     width = height * exp(arctan(angle),2);
   }

   int width;
   int height
};
```

Put all member variables and any internal algorithm functions in the private section to hide them…

(Only your class should know what variables and algorithms it has and how they're used!)

And place all of the functions used to interface with your class in the public section… So they're usable by the rest of your program.

This is called "encapsulation."

```
int main(void)
{
   Rect r;

   r.Initialize(10,7);
   cout << r.GetArea();
```

Class Challenge!

Part 1: Convert the following code into a Cylinder class.

The class must have two methods:

init(r,h) to initialize the cylinder's radius/height

getVolume() to get its volume

Part 2: Show how to convert the main() function so it uses your new class.

```
double getVolume(int r, int h)
{
    const double pi = 3.1415926535;
    double vol = pi * r*r * h;

     return vol;
}
```

```
int main()
{
   int radius;      // define the attributes
   int height;      // of a cylinder

   radius = 10;  // init radius and height
   height = 20;

   double v = getVolume(radius,height);
   cout << "The volume is: " << v;
}
```

Self-contained Problem Solvers

Each class is a self-contained, problem-solving unit… *an ADT!*

It contains all of the necessary data and algorithms to solve a certain problem – and simple interface functions to use it.

The rest of our program doesn't need to know how it works, only how to talk to it using simple interface functions!

Since we hide our class's algorithms and variables …

We can improve its internal logic and variables later and the rest of our program will still work as-is!