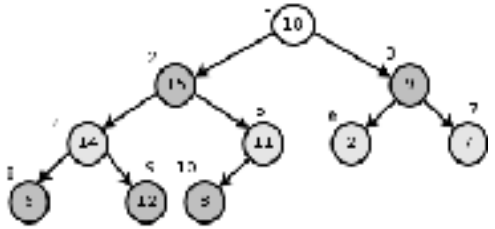
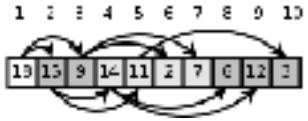
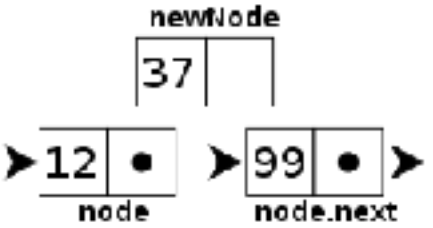
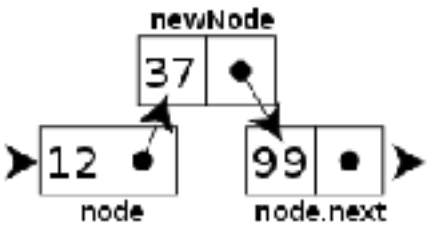




INSERTION-SORT(A)

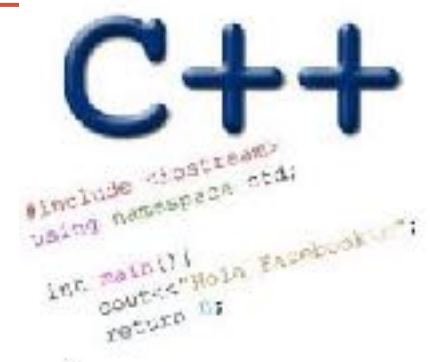
		cost	times
1	**for** $j = 2$ **to** $A.length$	c_1	n
2	$key = A[j]$	c_2	$n-1$
3	// Insert $A[j]$ into the sorted sequence $A[1..j-1]$	0	$n-1$
4	$i = j - 1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] = key$	c_8	$n-1$





WELCOME TO CS 24!

Problem Solving with Computers-II



https://ucsb-cs24-s18.github.io/

Read the syllabus. Know what's required. Know how to get help.

Enrollment status: 117/105

About me

- Diba Mirza (diba@ucsb.edu)
  - PhD (Computer Engineering, UCSD)
  - First year as faculty at UCSB!
  - Before this: Teaching faculty at UCSD for three years
- Office hours (starting next week 1/22):
  - M: 3:30p - 5p (right after lecture)
  - R: 11a – 1p
  - Or by appointment
  - Location: HFH 1155
  - Check the Google calendar on course website
  -
- You can reach me via
  - Piazza (highly recommended)
  - Email: Include [CS24] on the subject line

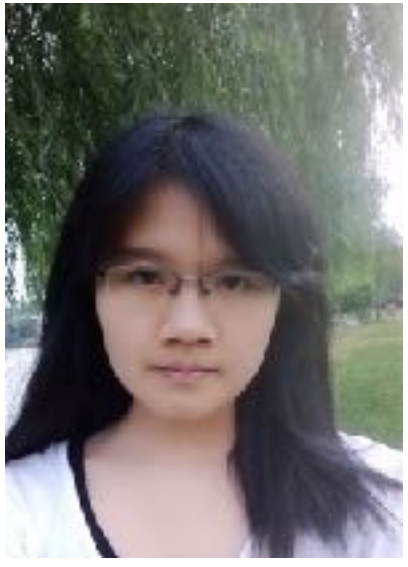
Ask me about:

- Course content!
- The how and why of what we are learning

Tell me about:

- Yourself!
- Experience in the class
- Interaction with the staff
- Climate of the labs

Course staff



TAs and peer mentors about:

- One-one help in labs
- Feedback on code
- Answer questions on course content
- Available during “schedule” and “open labs” in Phelps 3525

Peer Mentors

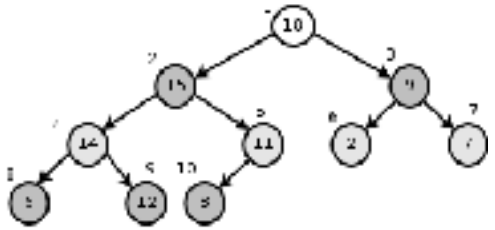
How to succeed in this course - first steps

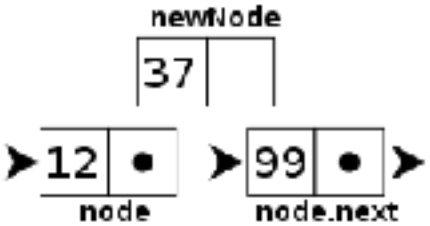
- Come to office hours and introduce yourself
- Setup a regular time to meet outside of section time with your
  - Mentor
  - Programming partner
- Communicate with the staff in person and remotely on:

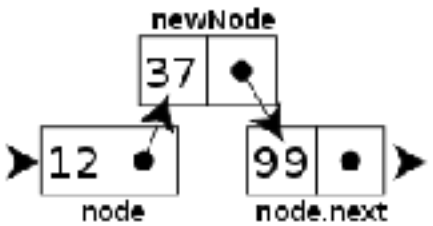
PIAZZA

About this course

Data Structures







Complexity Analysis

INSERTION-SORT(A)	*cost*	*times*
1 for $j = 2$ to $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

Course Logistics

- Grading
  - Class and section participation (iclickers): : 2%
  - Homeworks (due every week in sections) : 8%
  - Lab (programming) Assignments(due weekly) : 20%
  - Projects (programming assignments) : 20%
  - Midterm exams (2): : 20%
  - Final exam : 30%

- NO MAKEUPS ON EXAMS!

- You have 48 hours grace period to submit the labs. DO NOT contact the instructor or TAs for extensions unless you have a real emergency

- ATTENDENCE in sections and lectures is REQUIRED!
- To complete the labs you need a college of engineering account. If you don't have one yet, send an email to help@engineering.ucsb.edu

Clickers out – frequency AB

About you…

What is your familiarity/confidence with programming in C++?

A. Know nothing or almost nothing about it.
B. Used it a little, beginner level.
C. Some expertise, lots of gaps though.
D. Lots of expertise, a few gaps.
E. Know too much; I have no life.

About you…

What is your familiarity/confidence with C++ pointers?

A. Know nothing or almost nothing about it.
B. Used it a little, beginner level.
C. Some expertise, lots of gaps though.
D. Lots of expertise, a few gaps.
E. Know too much; I have no life.

About you…

What is your familiarity/confidence with C++ classes?

A. Know nothing or almost nothing about it.
B. Used it a little, beginner level.
C. Some expertise, lots of gaps though.
D. Lots of expertise, a few gaps.
E. Know too much; I have no life.

About you…

What is your familiarity/confidence with using version control – git or subversion?

A. Know nothing or almost nothing about it.
B. Used it a little, beginner level.
C. Some expertise, lots of gaps though.
D. Lots of expertise, a few gaps.
E. Know too much; I have no life.

iClickers: You must bring them

- Buy an iClicker at the Bookstore
- Register it on GauchoSpace (wait for my announcement on Piazza)
- Bring your iclicker to class

Required textbook

- Michael Main and Walter Savitch. *Data Structures and Other Objects Using C++ (4th edition)*, Addison-Wesley, 2011.

Recommended textbook

- Problem Solving with C++, Walter Savitch, Edition 9

You must attend class and lab sections
You must prepare for class
You must participate in class

Clickers, Peer Instruction, and PI Groups

- Find 1-2 students sitting near you. If you don't have any move.
- Introduce yourself.
- This is your initial PI group (at least for today)

Procedural Programming

- Break down a problem into sub tasks (functions)
- Algorithm to bake a cake

    Preheat the oven to 350F

    Get the ingredients: 2 eggs, 1cup flour, 1 cup milk

    Mix ingredients in a bowl

    Pour the mixture in a pan

    Place in the over for 30 minutes

Object Oriented Programming

- A different approach to solving problems
- Solution to a problem is a system of interaction **objects**
- An object has attributes and behavior
- What are the objects in this example?

Preheat the oven to 350F

Get the ingredients: 2 eggs, 1cup flour, 1 cup milk

Mix ingredients in a bowl

Pour the mixture in a pan

Place in the over for 30 minutes

Objects have attributes and behavior: A cake baking example

Object	Attributes	Behaviors
Oven	Size Temperature Number of racks	Turn on Turn off Set temperature
Bowl	Capacity Current amount	Pour into Pout out
Egg	Size	Crack Separate(white from yolk)

Example: A “Rabbit” object

- You could (in a game, for example) create an object representing a rabbit
- It would have data:
  - How hungry it is
  - How frightened it is
  - Its Location
- And methods:
  - eat, hide, run, dig

A class: pattern for describing similar objects

A generic pattern that is used to describe objects that have similar attributes and behaviors

e.g. a bowl and a pan may be described by the same class

```
class Dish{
      void pourIn( double amount);
      void pourOut(double amount);
      double capacity;
      double currentAmount;
};
```

Another example

A generic pattern that is used to describe objects that have similar attributes and behaviors

e.g. a bowl and a pan may be described by the same class

```
class Dish{
     void pourIn( double amount);
     void pourOut(double amount);
     double capacity;
     double currentAmount;
};
```

Concept: Classes describe objects

- Every object belongs to (is an instance of) a class
- An object may have fields, or variables
  - The class describes those fields
- An object may have methods
  - The class describes those methods
- A class is like a template, or cookie cutter
  - You use the class's constructor to make objects

Concept: Classes are like Abstract Data Types

- An Abstract Data Type (ADT) bundles together:
  - some data, representing an object or "thing"
  - the operations on that data
- The operations defined by the ADT are the *only* operations permitted on its data
- ADT = classes + information hiding

```
class Dish{
public:
      void pourIn( double amount);
      void pourOut(double amount);
private:
      double capacity;
      double currentAmount;
};
```

Another example of a class

```
class Employee{
public:




private:
      double salary;
      string name;
};
```

Notation: How to create objects of a class

```
Employee secretary;

secretary.setname( “Rabbit”);

cout<<secretary.getSalary();

Employee president;

Dish pan, bowl;
```

Approximate Terminology

- instance = object
- field = instance variable
- method = function
- sending a message to an object = calling a function
- These are all *approximately* true

Concept: *Classes* can have fields and methods

- Usually a class describes fields (variables) and methods for its objects (instances)
  - These are called instance variables and instance methods
- Every object has its own instance of the class variables
- A method of the class is always activated on a specific object and acts on the variables of that objects
- Use the special keyword static to say that a field or method belongs to the class instead of to objects

Advice: Restrict access

- Always, *always* strive for a narrow interface
- Follow the principle of information hiding:
  - the caller should know as little as possible about how the method does its job
  - the method should know little or nothing about where or why it is being called
- Make as much as possible private
- Your class is responsible for it's own data; don't allow other classes to screw it up!

What we have spoken about so far?

- Class = Data + Member Functions.
- Abstract Data Type = Class + information hiding
- How to define a new class type, and place the definition in a header file.
- How to use the header file in a program which declares instances of the class type.
- How to activate member functions.

? But you still need to learn how to write the bodies of a class's methods.

Next time

- Demo converting a procedural program to a OOP style program
- The big 3