CSSE 230 Day 3

Basic Data Structure Review

Check out from SVN:
ComparatorExample project

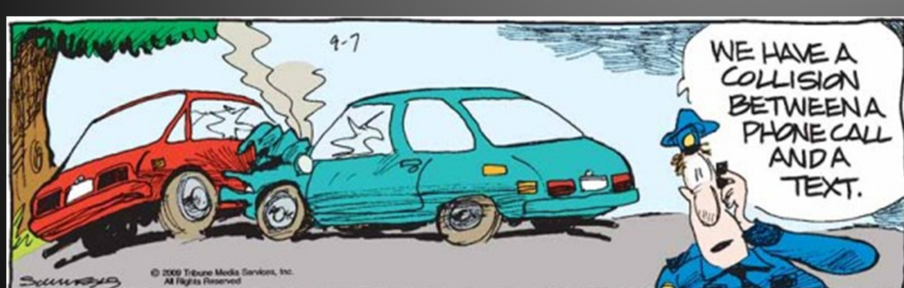
Reminders

- Written assignment 1 (to ANGEL dropbox) was due at 8 AM
  - You can use late day if you aren't done.
- See schedule page for things due soon
  - Warm Up and Stretching programs
  - Written Assignment 2
  - Pascal's Christmas Tree programming problem
- Randomly choose partners for Pascal's Christmas Tree project

Agenda

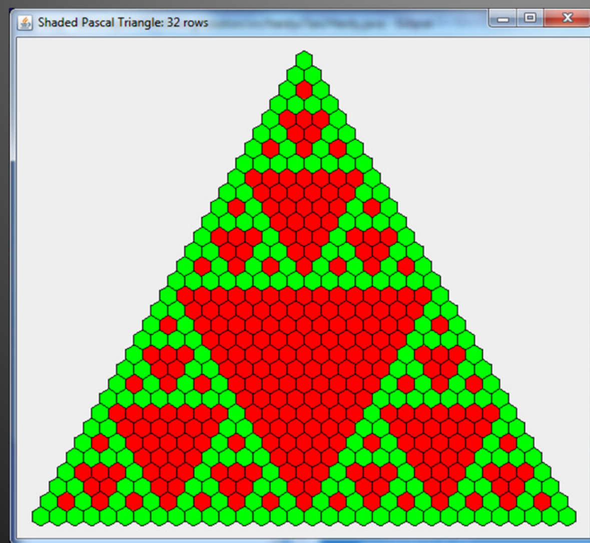
- Preview of PascalChristmasTree assignment
- Data Structures Overview
  - Mostly the same as 220, but with a few more details and a few more structures
- Review of Function Objects

Questions?



PascalChristmasTree

- Demo
- Meet your partner



Pascal partners, repos : Section 01

csse230-201230-pascal-11,amesen,piliseal
csse230-201230-pascal-12,dingx,elswicwj
csse230-201230-pascal-13,eubankct,goldthea
csse230-201230-pascal-14,harbisjs,rujirasl
csse230-201230-pascal-15,huangz,namdw
csse230-201230-pascal-16,jarvisnw,murphysw
csse230-201230-pascal-17,jonesel,postcn
csse230-201230-pascal-18,koestedj,millerns
csse230-201230-pascal-19,maglioms,rudichza
csse230-201230-pascal-20,mccullwc,semmeln
csse230-201230-pascal-21,mcdonabj,newmansr
csse230-201230-pascal-22,mehrinla,morrista
csse230-201230-pascal-23,nuanests,shahdk
csse230-201230-pascal-24,sanderej,timaeudg
csse230-201230-pascal-25,paulbi,yuhasmj
csse230-201230-pascal-26,weirjm,woolleld

Pascal partners, repos: Section 02

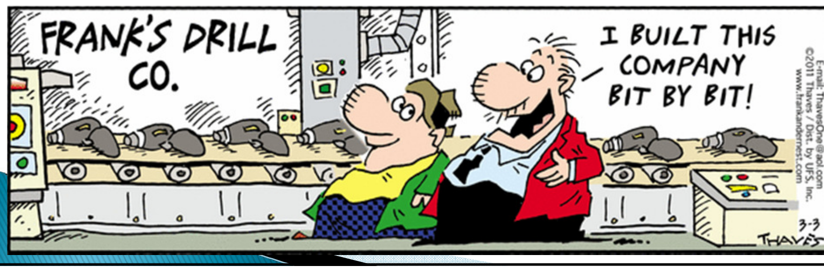
csse230-201230-pascal-27,bollivbd,ruthat
csse230-201230-pascal-28,davelldf,iwemamj
csse230-201230-pascal-29,ewertbe,hopwoocp
csse230-201230-pascal-30,faulknks,sprycet
csse230-201230-pascal-31,fendrirj,pohltm
csse230-201230-pascal-32,gartzkds,minardar
csse230-201230-pascal-33,haydr,lawrener
csse230-201230-pascal-34,lius,modivr
csse230-201230-pascal-35,memeriaj,scroggd
csse230-201230-pascal-36,mengx,stewarzt
csse230-201230-pascal-37,meyermc,uphusar
csse230-201230-pascal-38,qinz,zhangz
csse230-201230-pascal-39,roetkefj,taylorem
csse230-201230-pascal-40,tilleraj,toorha
csse230-201230-pascal-41,watterlm,weil
csse230-201230-pascal-42,zhenw,yuhasem

Abstract Data Types

Data Structures

- What is data?
- What do we mean by “structure”?

- A data type is an interpretation of the bits
  - Basically a set of operations
  - May be provided by the hardware (**int** and **double**)
  - By software (**java.math.BigInteger**)
  - By software + hardware (**int[]**)



What is an Abstract Data Type (ADT)?

- A mathematical model of a data type
- Specifies:
  - The type of data stored
  - The operations supported
  - The types and return values of these operations
  - What each operation does, but not how

An Example ADT: Non-negative integers

- One special value: **zero**
- Three basic operations:
  - **succ**
  - **pred**
  - **isZero**
- Derived operations include **plus**
- Sample rules:
  - **isZero(succ(n)) ➔ false**
  - **pred(succ(n)) ➔ n**
  - **plus(n, zero) ➔ n**
  - **plus(n, succ(m)) ➔ succ(plus(n, m))**

Data Structures: ADTs for storing several items

- Typically we're concerned with three things:
  - Specification (interface for the operations)
  - Implementation(s):
    - Representation (fields)
    - Operation implementations (method definitions)
  - Application (uses for the ADT)
- CSSE 220 emphasizes specification and uses
- CSSE 230 emphasizes specification and implementations

Data Structures Grand Tour

- Some review
  Some new
  All will appear again

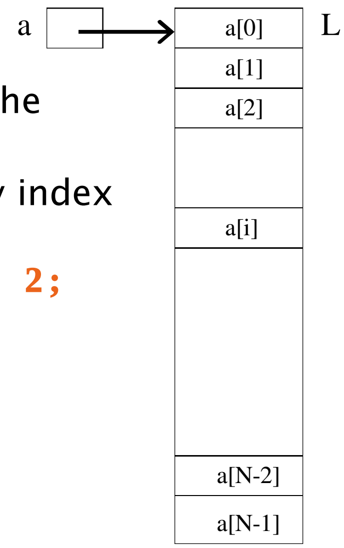
Common ADTs

- Array
- List
  - Array List
  - Linked List
- Stack
- Queue
- Set
  - Tree Set
  - Hash Set
- Map
  - Tree Map
  - Hash Map
- Priority Queue
- Tree
- Graph
- Network

Implementations for almost all of these are provided by the Java Collections Framework in the `java.util` package.

Q3

Array



- Size must be declared when the array is constructed
- Can look up or store items by index
  Example:

```
nums[i+1] = nums[i] + 2;
```

List

- A list is an ordered collection where elements may be added anywhere, and any elements may be deleted or replaced.
- **Array List**: Like an array, but growable and shrinkable.
- **Linked List**:

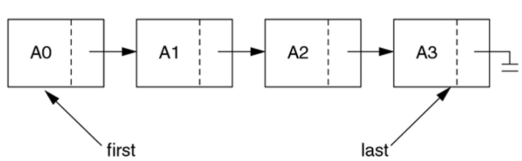


figure 6.19
A simple linked list

Q4

Array Lists and Linked Lists

Operations Provided	Array List Efficiency	Linked List Efficiency
Random access	O(1)	O(n)
Add/remove item	O(n)	O(1)

Stack

- A last-in, first-out (LIFO) data structure
- Real-world stacks
  - Plate dispensers in the cafeteria
  - Pancakes!
- Some uses:
  - Tracking paths through a maze
  - Providing "unlimited undo" in an application

```
public static void printInReverse(List<String> words) {
    // TODO: implement
    Stack<String> stack = new Stack<String>();
    for (String w : words) {
        stack.push(w);
    }
    while (!stack.isEmpty()) {
        System.out.println(stack.pop());
    }
}
```

Operations Provided	Efficiency
Push item	O(1)
Pop item	O(1)

Implemented by **Stack**, **LinkedList**, and **ArrayDeque** in Java

Queue

```java
/**
 * Uses a queue to print pairs of words consisting of
 * a word in the input and the word that appeared five
 * words before it.
 *
 * @param words
 */
public static void printCurrentAndPreceding(List<String> words) {
    // TODO: implement
    ArrayDeque<String> queue = new ArrayDeque<String>();
    // Preloads the queue:
    for (int i = 0; i < 5; i++) {
        queue.add("NotAWord");
    }
    for (String w : words) {
        queue.add(w);
        String fiveAgo = queue.remove();
        System.out.println(w + ", " + fiveAgo);
    }
}
```

- first-in, first-out (FIFO) data structure
- Real-world queues
  - Waiting line at the BMV
  - Character on Star Trek TNG
- Some uses:
  - Scheduling access to shared resource (e.g., printer)

Operations Provided	Efficiency
Enqueue item	O(1)
Dequeue item	O(1)

Implemented by **LinkedList** and **ArrayDeque** in Java

Set

Q5-8

- A collection of items **without duplicates** (in general, order does not matter)
  - If **a** and **b** are both in set, then **!a.equals(b)**
- Real-world sets:
  - Students
  - Collectibles
- One possible use:
  - Quickly checking if an item is in a collection

```java
public static void printSortedWords(List<String> words) {
    TreeSet<String> ts = new TreeSet<String>();
    for (String w : words) {
        ts.add(w);
    }
    for (String s : ts) {
        System.out.println(s);
    }
}
```

Example from 220

Operations	HashSet	TreeSet
Add/remove item	O(1)	O(lg n)
Contains?	O(1)	O(lg n)

Can hog space

Sorts items!

Map

How is a TreeMap like a TreeSet? How is it different?

- Associate **keys** with **values**
- Real-world "maps"
  - Dictionary
  - Phone book
- Some uses:
  - Associating student ID with transcript
  - Associating name with high scores

Operations	HashMap	TreeMap
Insert key-value pair	O(1)	O(lg n)
Look up the value associated with a given key	O(1)	O(lg n)

Can hog space

Sorts items by key!

HashMap/HashSet Example (220)

```
public static void printWordCountsByLength(List<String> words) {
    HashMap<Integer, HashSet<String>> map =
        new HashMap<Integer, HashSet<String>>();

    for (String w : words) {
        int len = w.length();
        HashSet<String> set;
        if (map.containsKey(len)) {
            set = map.get(len);
        } else {
            set = new HashSet<String>();
            map.put(len, set);
        }
        set.add(w);
    }
    System.out.printf("%d unique words of length 3.%n", getCount(map, 3));
    System.out.printf("%d unique words of length 7.%n", getCount(map, 7));
    System.out.printf("%d unique words of length 9.%n", getCount(map, 9));
    System.out.printf("%d unique words of length 15.%n", getCount(map, 15));
}
```

```
public static int getCount(HashMap<Integer, HashSet<String>> map, int key) {
    if (map.containsKey(key)) {
        return map.get(key).size();
    } else {
        return 0;
    }
}
```

Priority Queue

Not like regular queues!

Q9

- Each item stored has an associated priority
  - Only item with "minimum" priority is accessible
  - Operations: **insert**, **findMin**, **deleteMin**
- Real-world "priority queue":
  - Airport ticketing counter
- Some uses
  - Simulations
  - Scheduling in an OS
  - Huffman coding

```
PriorityQueue<String> stringQueue =
    new PriorityQueue<String>();

stringQueue.add("ab");
stringQueue.add("abcd");
stringQueue.add("abc");
stringQueue.add("a");

while(stringQueue.size() > 0)
    System.out.println(stringQueue.remove());
```

Operations Provided	Efficiency
Insert	O(log n)
Find Min	O(log n)
Delete Min	O(log n)

The version in Warm Up and Stretching isn't this efficient.

Trees, Not Just For Sorting

- Collection of nodes
  - One specialized node is the root.
  - A node has one parent (unless it is the root)
  - A node has zero or more children.
- Real-world "trees":
  - Organizational hierarchies
  - Some family trees
- Some uses:
  - Directory structure on a hard drive
  - Sorted collections

Only if tree is "balanced"

Operations Provided	Efficiency
Find	O(log n)
Add/remove	O(log n)

Graphs

- A collection of nodes and edges
  - Each edge joins two nodes
  - Edges can be directed or undirected
- Real-world "graph":
  - Road map
- Some uses:
  - Tracking links between web pages
  - Facebook

Operations Provided	Efficiency
Find	O(n)
Add/remove	O(1) or O(n) or O(n^2)

Depends on implementation (time/space trade off)

Networks

- Graph whose edges have numeric labels
- Examples (labels):
  - Road map (mileage)
  - Airline's flight map (flying time)
  - Plumbing system (gallons per minute)
  - Computer network (bits/second)
- Famous problems:
  - Shortest path
  - Maximum flow
  - Minimal spanning tree
  - Traveling salesman
  - Four-coloring problem for planar graphs

Common ADTs

- Array
- List
  - Array List
  - Linked List
- Stack
- Queue
- Set
  - Tree Set
  - Hash Set
- Map
  - Tree Map
  - Hash Map
- Priority Queue
- Tree
- Graph
- Network

We'll implement and use nearly all of these, some multiple ways. And a few other data structures.

Data Structure Summary

Structure	find	insert/remove	Comments
Array	O(n)	can't do it	Constant-time access by position
Stack	top only O(1)	top only O(1)	Easy to implement as an array.
Queue	front only O(1)	O(1)	insert rear, remove front.
ArrayList	O(log N)	O(N)	Constant-time access by position
Linked List	O(n)	O(1)	O(N) to find insertion position.
HashSet/Map	O(1)	O(1)	If table not very full
TreeSet/Map	O(log N)	O(log N)	Kept in sorted order
PriorityQueue	O(log N)	O(log N)	Can only find/remove smallest
Tree	O(log N)	O(log N)	If tree is balanced
Graph	O(N*M) ?	O(M)?	N nodes, M edges
Network			shortest path, maxFlow

Function Objects and Generics

Comparable and Comparator

Comparable review:

- **interface java.lang.Comparable<T>**
- **Type Parameter**: **T** – the type of objects that this object may be compared to
- **int compareTo(T o)**
  - Compares **this** with **o** for order.
  - Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object
  - Primitive type comparison: **x < y**
  - Comparable comparison: **obj1.compareTo(obj) < 0**

Limitations of Comparable!

- There is more than one natural way to compare Rectangles!
- What if we want to compare using
  - Height?
  - Width?
  - Closeness of aspect ratio to the golden ratio, ϕ
- It would be nice to be able to create and pass comparison methods to other methods …

$$\varphi = \frac{a+b}{a} = \frac{a}{b} = \frac{1+\sqrt{5}}{2}$$

Function Objects (a.k.a. Functors)

- Why do methods have arguments in the first place?
- We'd like to be able to pass a method as an argument to another method
- This is not a new or unusual idea.
  - You pass other functions as arguments to Maple's *plot* and *solve* functions (on a later slide).
  - C and C++ provide *qsort*, whose first argument is a comparison function.
  - Scheme and Python also have *sort* functions that can take a comparison function as an argument.

In Scheme

- Scheme has a sort function that takes a function as an argument:

```
Chez Scheme Version 7.4
Copyright (c) 1985-2007 Cadence Research Systems
> (sort > '(7 3 9 -2 5 -6 0 4 1 -8))
(9 7 5 4 3 1 0 -2 -6 -8)
> (sort (lambda (x y) (< (abs x) (abs y)))
        '(7 3 9 -2 5 -6 0 4 1 -8))
(0 1 -2 3 4 5 -6 7 -8 9)
```

Similar example in Python

```
>>> list = [4, -2, 6, -1, 3, 5, -7]
>>> list.sort()
>>> list
[-7, -2, -1, 3, 4, 5, 6]
>>> def comp (a, b):
       return abs(a) - abs (b)

>>> list.sort(comp)
>>> list
[-1, -2, 3, 4, 5, 6, -7]
```

The **comp** function is passed as an argument to the **sort** method

Similar example in Maple

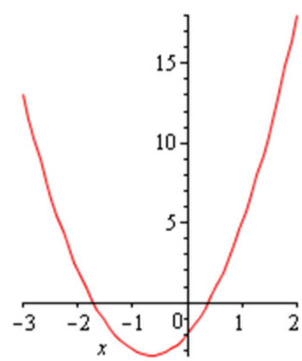
```
> sort([3, 7, −3, 4, −6, 1, 8], `<`);
                    [−6, −3, 1, 3, 4, 7, 8]
> sort([3, 7, −3, 4, −6, 1, 8], `>`);
                    [8, 7, 4, 3, 1, −3, −6]
> absless := (x, y) → abs(x) < abs(y);
                absless := (x, y) → |x| < |y|
> sort([3, 7, −3, 4, −6, 1, 8], `absless`)
                    [1, −3, 3, 4, −6, 7, 8]
```

More Maple

```
> f := x->3*x^2 + 4*x - 2;
```

$$f := x \to 3x^2 + 4x - 2$$

```
> plot(f(x), x=-3..2);
```



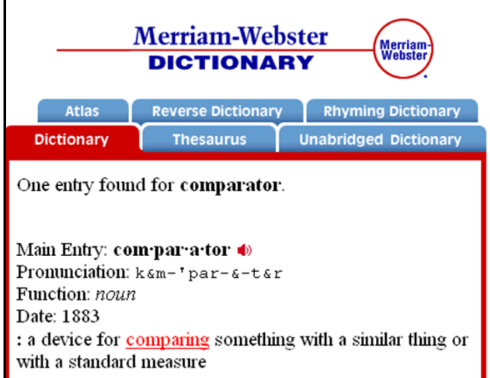

```
> solve(f(x), x);
```

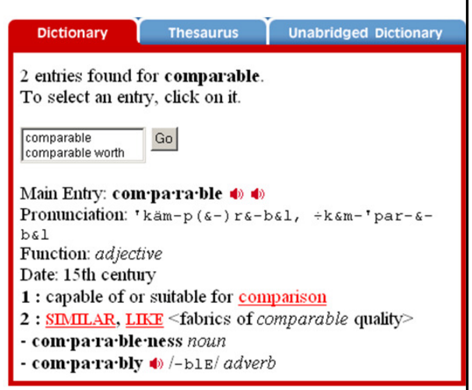
$$-\frac{2}{3} + \frac{\sqrt{10}}{3}, -\frac{2}{3} - \frac{\sqrt{10}}{3}$$

Java Function Objects

- What's it all about?
  - Java doesn't (yet) allow passing functions as arguments.
  - So, we create objects whose sole purpose is to pass a function into a method
  - Called **function objects**
    - a.k.a. functors, functionoids, more fun than a barrel of monkeys
- Weiss DS book's example: **`Comparator`**

You say "tomato", I say "toe-mah-toe"





Java: "imposed" ordering "natural" ordering

Sorting Arrays and Collections

- **java.util.Arrays** and **java.util.Collections** are your friends!
- On Written Assignment 2
  - The **CountMatches** implementation problem asks you to write code that creates and uses function objects.

Q10-11

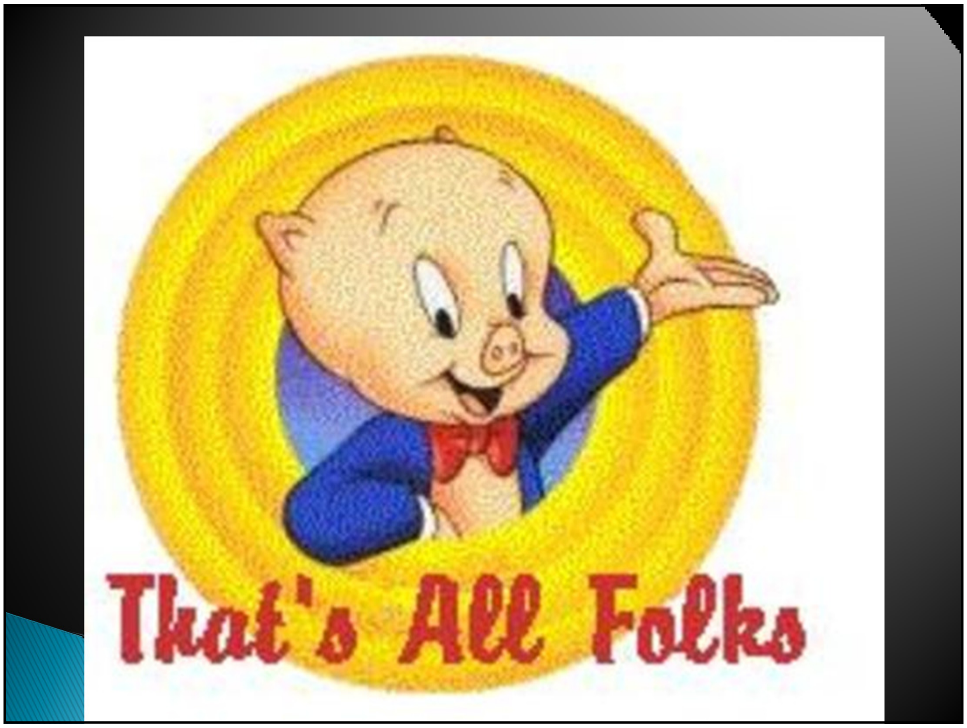
Work Time

- If a miracle occurs and we have itme left

  Make progress on **Warm Up and Stretching** problems

  Get help as needed, especially with Eclipse and SVN issues

  Work on WA2 if you have finished WarmUpAndStretching


That's All Folks