Plan for the Week

● Review for Midterm
   ➢ Source code provided
   ➢ Handouts/What you bring

● Exam Format
   ➢ What you bring, how you complete exam

● Toward linked structures and recursion
Exam Reference Sheet and Code

● What questions do you have?

public int calc(int n) {
    return n * calc(n - 1);
}

● What is the Internet?
  ➢ A network of networks.
  Or …

● What is recursive DNS?
  ➢ What IP is fxyztl.com?

● What is PageRank?
  ➢ Where is it used?
PFTD, PFTW, PFYL

● **Structure motivates ‘self-referential’ code**
  - motivation

● **Basic understanding of recursion**
  - Principles
  - Examples

● **APTs with recursion**
Why Linked Lists and Recursion are …

- **Node has**
  - Information
  - Pointer to node

- **Self-referential**
  - You talking to me?

- **What is 'this' in code?**
  - Self-referential

- **Recursive method**
  - Does NOT call self
  - Does call "clone"

- **Methods call other methods (often?)**
  - How to remember where to return?
  - Abstraction!
Quota Exceeded: coping with storage

● You're running out of disk space
  ➢ Buy more
  ➢ Compress files
  ➢ Delete files

● How do you find your “big” files?
  ➢ What's big?
  ➢ How do you do this?
BlobCount or edge detection or …

● How do we find images? Components? Paths?
  ➢ https://git.cs.duke.edu/201fall16/blobstuff/tree/master/src
Tools: Solving Computational Problems

- **Algorithmic techniques and paradigms**
  - Brute-force/exhaustive, greedy algorithms, dynamic programming, divide-and-conquer, ...
  - Transcend a particular language
  - Designing algorithms, may change when turned into code

- **Programming techniques and paradigms**
  - Recursion, memo-izing, compute-once/lookup, tables, ...
  - Transcend a particular language
  - Help in making code work
    - Cope with correctness and maintenance
    - Cope with performance problems
Tools: Solving Computational Problems

- **Java techniques**
  - `java.util.*`, Comparator, LinkedList, Map, Set, ...
  - These aren’t really Java-specific, but realized in Java
  - Map, Comparator, Set: C++, Python, ....
  - We learn idioms in a language and talk about abstractions

- **Analysis of algorithms and code**
  - Mathematical analysis, empirical analysis
  - We need a language and techniques for discussion
  - Theory and practice, real problems and in-the-limit issues

- “In theory there is no difference between theory and practice, but in practice there is.”
  (attributed to many)
Recursive structure matches code

```java
public static final long THRESHOLD = 1000000L; // one million bytes

public static void findBig(File dir, String tab) {
  File[] dirContents = dir.listFiles();
  System.out.println(tab+"**:"+dir.getPath());
  for(File f : dirContents){
    if (f.isDirectory()) {
      findBig(f,tab+"\t");
    }
    else {
      if (f.length() > THRESHOLD){
        System.out.printf("%s%s%8d
",tab,f.getName(), f.length());
      }
    }
  }
}
```

Does findBig call itself?
Solving Problems Recursively

- **Recursion: indispensable in programmer’s toolkit**
  - Elegance can lead to better programs: easier to modify, extend, verify, more efficient, cure ...
  - Sometimes recursion isn't appropriate, when it's bad it can be very bad---every tool requires knowledge and experience in how to use it

- **The basic idea is to get help solving a problem from coworkers (clones) who work and act like you do**
  - Ask clone to solve a simpler/smaller, but similar problem
  - Use clone's result to put together your answer

- **Both: call on the clone and use the result**
Exponentiation

● **Computing** \(x^n\) **means multiplying** \(n\) **numbers**
  - Does it require \(n\) multiplies?
  - What’s the simplest value of \(n\) when computing \(x^n\)?
  - To only multiply once, what can you ask a clone?

```java
public static double power(double x, int n)
{
    if (n == 0)
    {
        return 1.0;
    }
    return x * power(x, n-1);
}
```

● **Number of multiplications? Structure?**
  - Note base case: no recursion, no clones
  - Note recursive call: moves toward base case (unless ... )
Faster exponentiation

- **Recursive calls made to compute $2^{1024}$?**
  - How many multiplies on each call? Is this better?

```
public static double power(double x, int n){
    if (n == 0) return 1.0;
    double semi = power(x, n/2);
    if (n % 2 == 0) return semi*semi;
    return x * semi * semi;
}
```

- **What about an iterative version of this function?**
  - Why might we want such a version?
Back to Recursion

● Recursive functions have two key attributes
  ➢ There is a base case, aka exit case: no recursion!
    • See print directories, exponentiation
  ➢ All other cases make a recursive call, with some measure (e.g., parameter value) that decreases towards the base case
    • Ensure that sequence of calls eventually reaches the base case
    • “Measure” can be tricky, but usually it’s straightforward

● Example: structural recursion: data meets code
  ➢ Why is directory code inherently recursive?
  ➢ How is this different from exponentiation?
More recursion recognition

```java
public static int sumit(int[] a, int index){
    if (index < a.length) {
        return a[index] + sumit(a,index+1);
    }
    return 0;
}
// original call: int v = sumit(a,0);
```

- What is base case, what value is returned?
- How is progress towards base case realized?
- How is recursive call used to return a value?
- What if we sum values in a linked list?

Recursive methods sometimes use extra parameters; helper methods set this up
Blob Counting, Flood Fill

- **Flood a region with color**
  - Erase region, make transparent
  - How do find the region?

- **Finding regions, blobs, edges, ..**
  - See blob counting code
  - What is a blob?

- **Recursion helps, but necessary?**
  - Performance, clarity, ...
  - Ease of development
Ideas behind blob fill code

● Ask your neighbors
  ➢ Return blob size
  ➢ Ensure no re-counts
  ➢ Sum and return

● What do neighbors do?
  ➢ Same thing!
  ➢ Colors indicate calls
Details and Idioms in blob code

- **Method `blobFill` has four parameters**
  - (row,column) of where search starts
  - Character being searched for (initially * or blob)
  - Character to fill with on success (e.g., count ‘2’ or ‘4’)
    - Mark for visualization
    - Mark to ensure we don't search again!

- **If (row,column) is part of blob, count it and ask neighbors for their counts**
  - They're part of blob (if never visited before)

- **Return total of yourself and neighbors**
  - Key to recursion: do one thing and ask for help
Blob questions

- **What changes if diagonal cells are adjacent?**
  - Conceptually and in code

- **How do we find blob sizes in a range?**
  - Not bigger than X, but between X and Y

- **How would we number blobs by size rather than by when they're found?**
  - Do we have the tools to do this in existing code?

- **Can we avoid recursion and do this iteratively?**