Lecture 16

Anastasio, Cannon, Lexi & Sarah Like Cookies (1/2)

- They would each like to have one of these cookies:
  - How many ways can they distribute the cookies amongst themselves?
    - Anastasio has 4 choices
    - Cannon has 3 choices
    - Lexi only 2 choices
    - Sarah must take what remains (poor Sarah!)

Thus, there are 24 different ways the characters can choose cookies ($4! = 4 \times 3 \times 2 \times 1 = 24$)

What if we wanted to solve this problem for all the (H)TAs?
Factorial Function

- Model this problem mathematically: factorial(n) calculates the total number of unique permutations, or the number of different ways to arrange/order n items.

- Small examples:
  - 1! = 1
  - 2! = 2 * 1 = 2
  - 3! = 3 * 2 * 1 = 6
  - 4! = 4 * 3 * 2 * 1 = 24
  - 5! = 5 * 4 * 3 * 2 * 1 = 120

- **Iterative definition**: \( n! = n \times (n - 1) \times (n - 2) \times \ldots \times 1 \)

- **Recursive definition**: \( n! = n \times (n-1)! \) for \( n \geq 1 \) and \( 0! = 1 \)

Recursion (1/2)

- Models problems that are self-similar:
  - breaks down a whole task into smaller, similar subtasks
  - each subtask can be solved by applying the same technique

- Whole task solved by combining solutions to sub-tasks:
  - special form of divide and conquer at every level

Recursion (2/2)

- Task is defined in terms of itself:
  - in Java, recursion is modeled by method that calls itself, but each time with simpler case of the problem, hence the recursion will "bottom out" with a base case eventually

  - base case is a case simple enough to be solved directly, without recursion; without base case, the method would recurse indefinitely, causing a StackOverflowError

  - what is the base case of the factorial problem?
  - Java will bookkeep each execution of the same method just as it does for nested methods that differ, so there is no confusion

  - usually, you combine the results from the separate executions
Factorial Function Recursively (1/2)

- Recursive factorial algorithm
  - the factorial function is not defined for negative numbers
  - the first conditional checks for this precondition
  - it is good practice to document and test preconditions (see code example)
  - number of times method is called is the depth of recursion (1 for 0!)
  - what is depth of 4!?

```java
public class RecursiveMath {
    public int factorial(int num) {
        if (num < 0) {
            System.out.println("Input must be >= 0");
            return -1; // return -1 for invalid input
        }
        int result = 0;
        if (num == 0) {
            // base case: 0! = 1
            result = 1;
        } else {
            // general case
            result = num * this.factorial(num - 1);
        }
        return result;
    }
}
```

Factorial Function Recursively (2/2)

Factorial(4) = factorial(4)
= 4 * 3!
= 4 * 3 * 2!
= 4 * 3 * 2 * 1!
= 4 * 3 * 2 * 1 * 0!
= 24

TopHat Question

Given the following non-practical code:
```
public class RecursiveMath {
    public int recursiveAddition(int n) {
        if (n <= 1) {
            return 1;
        } else {
            return this.recursiveAddition(n - 1);
        }
    }
}
```
What is the output of `this.recursiveAddition(4)`?

A. 1  
B. 9  
C. 10  
D. StackOverflowError
TopHat Question

Given the following code:

```java
public class RecursiveMath {
    public int funkyFactorial(int n) {
        if (n == 0) {
            return 1;
        } else {
            return n * this.funkyFactorial(n - 2);
        }
    }
}
```

What is the output of `this.funkyFactorial(5)`?

A. 1  
B. 5  
C. 15  
D. StackOverflowError

If you want to know more about recursion...

Turtles in Recursion – from Wikipedia

The following anecdote is told of William James. After a lecture on cosmology and the structure of the solar system, James was accosted by a little old lady.

"Your theory that the sun is the center of the solar system, and the earth is a ball which rotates around it has a very convincing ring to it, Mr. James, but it's wrong. I've got a better theory, " said the little old lady.

"And what is that madam?" inquired James politely.

"That we live on a crust of earth which is on the back of a giant turtle."

Not wishing to demolish her absurd theory with his scientific arsenal, James decided to gently dissuade his opponent.

"If your theory is correct, madam," he asked, "what does this turtle stand on?"

"You're a very clever man, Mr. James, and that's a very good question," replied the little old lady, "but I have an answer to it. And it's this: The first turtle stands on the back of a second, far larger, turtle, who stands directly under him."

"But what does this second turtle stand on?" persisted James patiently.

To this, the little old lady crowed triumphantly, "It's no use, Mr. James – it's turtles all the way down."

— J.R. Ross, Constraints on Variables in Syntax 1967
Mandelbrot Fractals as Recursive Functions

- Benoit Mandelbrot developed Fractals, a mathematical branch whose principal characteristic is self-similarity at any scale, one of the characteristics of recursion.
- Fractals are common in nature (botany, lungs, blood vessels, kidneys…).

\[ Z' = Z^2 + c \] in the complex plane (x, iy) where \( i = \sqrt{-1} \).

- For any choice, if the sequence converges, the point is in the set and colored black.
- If it diverges, the color is based on how rapidly the sequence changes, and to look cool.

Simpler Recursive Functions

- Some simpler, non-fractal, but still self-similar shapes composed of smaller, simpler copies of some pattern are simple spirals, trees, and snowflakes.

Designing Spiral Class (1/2)

- **Spiral** class defines single draw method
  - turtle functions as pen to draw spiral, so class needs reference to turtle instance
- Constructor’s parameters to control its properties:
  - position at which spiral starts in turtle’s position
  - length of spiral’s starting side
  - angle between successive line segments
  - amount to change length of spiral’s side at each step
- Note: this info is passed to each execution of recursive method, so next method call depends on previous one.
Designing **Spiral** Class (2/2)

```java
public class Spiral {
    private Turtle turtle;
    private double angle;
    private int lengthDecrement;

    // passing in parameters to set the properties of the spiral
    public Spiral(Turtle myTurtle, double myAngle, int myLengthDecrement) {
        this.turtle = myTurtle;
        this.angle = myAngle;
        this.lengthDecrement = 1; // default, handles bad parameters
        if (myLengthDecrement > 0) {
            this.lengthDecrement = myLengthDecrement;
        }
        // draw method defined soon...
    }
}
```

Drawing **Spiral**

- First Step: Move turtle forward to draw line and turn some degrees.
  What's next?
- Draw smaller line and turn! Then another, and another...

Sending Recursive Messages (1/2)

- **draw** method uses turtle to trace spiral

- How does **draw** method divide up work?
  - draw first side of spiral
  - then draw smaller spiral (this is where we implement recursion)

```java
public void draw(int sideLen) {
    // general case: move sideLen, turn
    // angle and draw smaller spiral
    this.turtle.forward(sideLen);
    this.turtle.left(this.angle);
    this.draw(sideLen - this.lengthDecrement);
}
```
Sending Recursive Messages (2/2)

• What is the base case?
  - when spiral is too small to see, conditional statement stops method so no more recursive calls are made
  - since side length must approach zero to reach the base case of the recursion, the draw method gets a smaller side length each time

  ```java
  public void draw(int sideLen) {
    // base case: spiral too small to see
    if (sideLen <= 3) {
      return; // stops method
    }
    // general case: move sideLen, turn and draw smaller spiral
    this.turtle.forward(sideLen);
    this.turtle.left(this.angle);
    this.draw(sideLen - this.lengthDecrement);
  }
  ```

Recursive Methods

• We are used to seeing a method call other methods, but now we see a method calling itself
• Method must handle successively smaller versions of original task

Method’s Variable(s)

• As with separate methods, each execution of the method has its own copy of parameters and local variables, and shares access to instance variables
• Parameters let method execution (i.e., successive recursive calls) “communicate” with, or pass info between, each other
• Java’s record of current place in code and current values of parameters and local variables is called the activation record
  - with recursion, multiple activations of a method may exist at once
  - at base case, as many activation records exist as depth of recursion
  - each activation of a method is stored on the activation stack (you’ll learn about stacks soon)
Spiral Activation

draw(int sideLen)

draw Activation
int sideLen = 25

draw Activation
int sideLen = 14

draw Activation
int sideLen = 3

Initial value of sideLen: 25
Length decrement: 11

TopHat Question

What is the value of count after calling collatzCounter(5)?

A. 4
B. 5
C. 6
D. StackOverflowError

Towers of Hanoi (1/4)

- Game invented by French mathematician Edouard Lucas in 1883
- Goal: move tower of n disks, each of a different size (in order, with smallest at top), from left-most peg to right-most peg
- Rule 1: no disk can be placed on top of a smaller disk
- Rule 2: only one disk can be moved at a time

Play it here: https://www.mathsisfun.com/games/towerofhanoi.html
Watch a solution here: https://www.youtube.com/watch?v=l4w1b9mmeFE
One Disk Solution

Two Disk Solution

Three Disk Solution
Pseudocode for Towers of Hanoi (1/2)

- Try solving for 5 non-recursively... too hard!
  - try solving it recursively
- One disk:
  - move disk to final pole
- Two disks:
  - use one disk solution to move top disk to intermediate pole
  - use one disk solution to move bottom disk to final pole
- Three disks:
  - use two disk solution to move top disks to intermediate pole
  - use one disk solution to move bottom disk to final pole
  - use two disk solution to move top disks to final pole

Pseudocode for Towers of Hanoi (2/2)

- In general (for \( n \) disks)
  - use \( n-1 \) disk solution to move top disks to intermediate pole
  - use one disk solution to move bottom disk to final pole
  - use \( n-1 \) disk solution to move top disks to final pole

Note: a method can have multiple recursive calls as seen in the code in the next slide

Lower-level pseudocode

```java
public void hanoi(int n, Pole src, Pole dst, Pole other) {
    if (n == 1) {
        this.move(src, dst);
    } else {
        this.hanoi(n - 1, src, other, dst);
        this.move(src, dst);
        this.hanoi(n - 1, other, dst, src);
    }
}
public void move(Pole src, Pole dst) {
    // take the top disk on the pole src and make it the top disk on the pole dst
}
```

That's it!

otherPole and move are fairly simple methods, so the is not much code.

But try hand simulating this when \( n \) is greater than 4. It is tedious (but not hard)!

Iterative solution far more complex, and much harder to understand.
Fibonacci Sequence (1/2)

- 1, 1, 2, 3, 5, 8, 13, 21,...
- Each number is calculated by adding the two previous numbers
  - \( F_n = F_{n-1} + F_{n-2} \)

Fibonacci Sequence (2/2)

- What is the base case?
  - there are two: \( n=0 \) and \( n=1 \)
- Otherwise, add two previous values of sequence together
  - this is also two recursive calls!

Given the following code:

```java
public int fib(int n)
{
    // error check
    if (n < 0) {
        return -1;
    }
    // base case
    if (n == 0 || n == 1) {
        return 1;
    }
    // general case: add previous two values
    return fib(n - 1) + fib(n - 2);
}
```

What number would be returned if you excluded \( n == 1 \) from the base case and called `fib(2)`?

A. 5  
B. 3  
C. 2  
D. 1  

TopHat Question
**Loops vs. Recursion (1/2)**

- **Spiral** uses simple form of recursion
  - each sub-task only calls on one other sub-task
  - this form can be used for the same computational tasks as iteration
  - loops (iteration) and simple recursion are computationally equivalent in the sense of producing the same result, if suitably coded (not necessarily the same performance, though -- looping is more efficient)

**Loops vs. Recursion (2/2)**

- Iteration is often more efficient in Java because recursion takes more method calls (each activation record takes up some of the computer's memory)
- Recursion is more concise and more elegant for tasks that are "naturally" self-similar (Towers of Hanoi is very difficult to solve iteratively)
- we will begin doing recursion on data structures soon (stay tuned!)
- this type of recursion is emphasized in courses like CS200

```java
public void drawIteratively(int sideLen) {
    while (sideLen > 3) {
        // while loop
        this.turtle.forward(sideLen);
        this.turtle.left(this.angle);
        sideLen -= this.lengthDecrement;
    }
}
```

**Recursive Binary Tree (1/2)**

- The tree is composed of a trunk that splits into two smaller branches that sprout in opposite directions at the same angle
- Each branch then splits as the trunk did until sub-branch is deemed too small to be seen. Thus it is drawn as a leaf
- The user can specify the length of a tree's main trunk, the angle at which branches spread, and the amount by which to decrement each branch
Recursive Binary Tree (2/2)

- Compare each left branch to its corresponding right branch
- Right branch is simply rotated copy
- Branches are themselves smaller trees!
  - Branches are themselves smaller trees!
  - Branches are themselves smaller trees!
- Our tree is self-similar and can be programmed recursively!
  - Base case is a leaf

Designing the Tree Class

- Tree has properties that user can set:
  - Start position (myTurtle's built-in position)
  - Angle between branches (myBranchAngle)
  - Amount to change branch length (myTrunkDecrement)
- Tree class will define a single draw method
  - Like Spiral, also uses a Turtle to draw

```java
public class Tree {
    private Turtle turtle;
    private double branchAngle;
    private int trunkDecrement;

    public Tree(Turtle myTurtle, double myBranchAngle, int myTrunkDecrement) {
        this.turtle = myTurtle;
        if (myTrunkDecrement > 0) {
            this.trunkDecrement = myTrunkDecrement;
        } else {
            this.trunkDecrement = 1;
        }
        if (myBranchAngle > 0) {
            this.branchAngle = myBranchAngle;
        } else {
            this.branchAngle = 45;
        }
    }

    // draw method coming up...
}
```

Tree's draw Method

- Base case: Branch size too small, add a leaf
- General case:
  - move Turtle forward
  - move Turtle left
  - recursively draw left branch
  - move Turtle right
  - move Turtle starting orientation
  - back up to prepare for next branch

```java
private void draw(int trunkLen) {
    if (trunkLen <= 0) {
        this.addLeaf();
    } else {
        this.turtle.forward(trunkLen);
        this.turtle.left(this.branchAngle);
        this.draw(trunkLen - this.trunkDecrement);
        this.turtle.right(2 * this.branchAngle);
        this.draw(trunkLen - this.trunkDecrement);
        this.turtle.left(this.branchAngle);
        this.turtle.back(trunkLen);
    }
}
```
Overall Program View

A class that creates a Tree and utilizes its recursive methods in order to draw it.

```java
public class BuildTree {
    private Tree myTree;

    public BuildTree() {
        Turtle turtle = new Turtle();
        double branchAngle = 30;
        int trunkDecrement = 1;
        int trunkLen = 6;
        // Remember that draw() in Tree class took in a trunkLen
        this.myTree = new Tree(turtle, branchAngle, trunkDecrement);
        this.createTree(trunkLen);
    }

    public void createTree(int currTrunkLen) {
        this.myTree.draw(currTrunkLen);
    }
}
```

Recursive Snowflake

- Invented by Swedish mathematician, Helge von Koch, in 1904; also known as Koch Island
- Snowflake is created by starting an equilateral triangle and partitioning each side into three equal parts. Each side's middle part is then replaced by another equilateral triangle (with no base) whose sides are one third as long as the original.
- This process is repeated for every remaining line segment.
- The user can specify the length of the initial equilateral triangle's side.
- "Mathematical monster": infinite length with a bounded area

Snowflake's draw Method

- Can draw equilateral triangle iteratively
- `drawSnowFlake` draws the snowflake by drawing smaller, rotated triangles on each side of the triangle (compare to iterative `drawTriangle`)
- For loop iterates 3 times
- Each time, calls the `drawSide` helper method (defined in the next slide) and reorients turtle to be ready for the next side

```java
public void drawTriangle(int sideLen) {
    for (int i = 0; i < 3; i++) {
        this.turtle.forward(sideLen);
        this.turtle.right(120.0);
    }
}

public void drawSnowFlake(int sideLen) {
    for (int i = 0; i < 3; i++) {
        this.drawSide(sideLen);
        this.turtle.right(120.0);
    }
}
```
Snowflake's `drawSide` method

- `drawSide` draws single side of a recursive snowflake by drawing four recursive sides.
- **Base case:** simply draw a straight side.
- `MIN_SIDE` is a constant we set indicating the smallest desired side length.
- **General case:** draw complete recursive side.

```java
private void drawSide(int sideLen) {
    if (sideLen < MIN_SIDE) {
        this.turtle.forward(sideLen);
    } else {
        this.drawSide(Math.round(sideLen / 3));
        this.turtle.left(60.0);
        this.drawSide(Math.round(sideLen / 3));
        this.turtle.right(120.0);
        this.drawSide(Math.round(sideLen / 3));
        this.turtle.left(60.0);
    }
}
```

Hand Simulation

MIN_SIDE: 20
sideLen: 90

1) Call `draw(90)`, which calls `drawSide(90)`, which calls `drawSide(30)`, which calls `drawSide(10)`. Base case reached because 10 < MIN_SIDE.

2) `drawSide(10)` returns to `drawSide(30)`, which tells this.turtle to turn left 60 degrees and then calls `drawSide(10)` again.

3) `drawSide(10)` returns to `drawSide(30)`, which tells this.turtle to turn right 120 degrees and then calls `drawSide(10)` for a third time.

4) `drawSide(10)` returns to `drawSide(30)`, which tells this.turtle to turn left 60 degrees and then calls `drawSide(10)` for a fourth time. After this call, `drawSide(30)` returns to `drawSide(90)`, which reorients this.turtle and calls `drawSide(30)` again.

Again: Koch Snowflake Progression

```
private void drawKoch(int sideLen) {
    if (sideLen < MIN_SIDE) {
        this.turtle.forward(sideLen);
    } else {
        drawKoch(sideLen / 3);
        this.turtle.left(60.0);
        drawKoch(sideLen / 3);
        this.turtle.right(120.0);
        drawKoch(sideLen / 3);
        this.turtle.left(60.0);
    }
}
```
Indirect Recursion

- Two or more methods act recursively instead of just one
- For example, `methodA` calls `methodB` which calls `methodA` again
- Methods may be implemented in same or different classes
- Can be implemented with more than two methods too
- Example:
  - `Brownopoly` smart auto-sell algorithm
    - Brown-inspired Monopoly Game
    - Created as a CS15 Final Project

Brownopoly Example

- Monopoly is a game of money. When in debt, prioritize valuable assets:
  - Most expensive properties (high value)
  - Monopolies (higher value)
- Let’s conceptually recurse until we can pay off the debt:
  - First mortgage cheapest properties not belonging to a monopoly
  - Then, sell properties that are part of a monopoly but have no houses
  - Next, sell houses of cheapest monopolies
    - BUT after selling all houses do not want to continue selling houses, instead we want to mortgage those properties

Brownopoly Code

```java
private void autosell()
{
    if (this.player.getDebt() < this.player.getAssetsValue())
    {
        this.declareBankruptcy();
    } else {
        this.mortgageCheapestNonMonopoly();
    }
}

private void mortgageCheapestNonMonopoly()
{
    // sets toBeMortgaged = cheapest non-monopoly property
    if (toBeMortgaged != null)
    {
        toBeMortgaged.mortgage();
        if (this.player.getDebt() > 0)
        {
            this.mortgageCheapestNonMonopoly();
        } else {
            this.mortageNonHouseCheapest();
        }
    }
}

private void mortgageNonHouseCheapest()
{
    // mortgage cheapest property without a house
    if (this.player.getDebt() > 0)
    {
        this.mortgageNonHouseCheapest();
    } else {
        this.sellHouses();
    }
}

private void sellHouses()
{
    // sells cheapest property in a monopoly with most houses
    if (this.player.getDebt() > 0)
    {
        if (property.getMonopoly().hasHouses())
        {
            this.sellHouses();
        } else {
            this.mortageNonHouseCheapest();
        }
    }
}
Summary
- Recursion models problems that are self-similar, breaking down a task into smaller, similar sub-tasks.
- Whole task solved by combining solutions to sub-tasks (divide and conquer)
- Since every task related to recursion is defined in terms of itself, method will continue calling itself until it reaches its base case, which is simple enough to be solved directly.

Announcements
- DoodleJump Deadline!!
  - on-time hand-in: 11/01
  - late hand-in: 11/03
- Lab 7 2D-Arrays this week:
  - pre-lab video and pre-lab quiz
- Mentorship Program: Mentors will reach out this week to schedule next meeting; look out for emails from them.
Breaking News: Executive Order on AI Just Passed

- Executive Order includes provisions that
  - Order requires developers to share safety test results with the government.
  - Demand that AI-generated content be watermarked.
  - Touches matters of privacy, civil rights, consumer protections, scientific research, and worker rights.
  - Directed at both government agencies and companies.

- Gina Raimondo, the U.S. Secretary of Commerce and former Governor of Rhode Island is leading this effort.
- Brown University Professor Suresh Venkatasubramanian has been involved as an advisor and was there for the signing of the order.

EU AI Laws

- The EU may agree on a final text for the AI act by Wednesday.
- Incumbent points include the use of AI for surveillance:
  - Should states be able to use AI-powered facial recognition?
  - Will hold AI companies responsible if their AI is used to create something illegal?
- Will likely hold protections for artists, musicians, and researchers to give them legal protection from AI plagiarism.

Case Study: Elon Musk Twitter Acquisition

- On April 14, 2022, Elon Musk made an unsolicited offer to acquire Twitter:
  - Cited combating spam, promoting free speech, and making algorithms open source.
- The deal was closed on October 27, 2022, for a total price tag of 44 billion dollars.
- Spent ~30 billion of his own money taking on debt to pay the rest.
Historical Context

- 1895: William Randolph Hearst buys The Morning Journal
- Hearst's publication inflamed public opinion against Spain leading the U.S. to enter the Spanish-American War
- 1976, 1985, 2007: Australian Rupert Murdoch buy the New York Post, Twentieth Century Fox (Fox News), and Dow Jones & Company (parent company of WSJ)
- 2013: Jeff Bezos acquires Washington Post
- 2023: Elon Musk acquires Twitter

Many other examples of billionaires buying media!!

Twitter Timeline

- October 27, 2022: Acquisition concludes, Musk becomes CEO
- November 2022: Mass layoffs begin (~80% of employees)
  * These layoffs include members of Twitter's content moderation teams and the dissolving of Twitter's Trust and Safety Council
- Late November 2022: Twitter begins reinstating formerly banned accounts (high profile accounts include: Trump, Jordan Peterson, Andrew Tate)
- February 2023: Musk took control of the Twitter API
- March 2023: Musk makes Twitter’s algorithm open source
- April 2023: Leaked Pentagon documents spread widely on Twitter

Source: Associated Press, NBC News, Announcement on Twitter website on Covid disinformation

Section 230 of Communications Decency Act (1/2)

- Section 230 shields big tech companies from lawsuits regarding content posted on their platform

  "No provider or user of an interactive computer service shall be held liable on account of — any action voluntarily taken in good faith to restrict access to or availability of material that the provider or user considers to be obscene, lewd, lascivious, filthy, aversive, violent, harassing, or otherwise objectionable, whether or not such material is constitutionally protected;"
Section 230 of Communications Decency Act (2/2)

- "No provider or user of an interactive computer service shall be treated as the publisher or speaker of any information provided by another information content provider."

- Differing views on the extent to which Section 230 promotes free speech or censorship

- Recent Supreme Court ruling sided with big tech and didn’t expand scope of 230

Open-Source Algorithm

Source: Github

Lingering Questions

- Musk through his various companies now wields immense power over global discourse and geopolitics.
  - To what extent should billionaires be able to control the public discourse and geopolitical policy?

- How can we maintain the balance between safety and free speech?
  - With the context of Section 230, who should make these decisions and to what extent should tech companies be held liable for the spread of disinformation on their technology?