CHAPTER 18
RECURSION

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO JAVA PROGRAMMING, LIANG (PEARSON 2014)
OVERVIEW

- **Recursion** is an algorithmic technique where a function calls itself directly or indirectly.

- Why learn recursion?
  - New mode of thinking.
  - Powerful programming paradigm.

- Many computations are naturally self-referential.
  - A folder containing files and other folders.
  - Mathematical sequences - \( f_{n+1} = f_n + 1 \) (whole numbers)
  - Exploring mazes – make a step in the maze, then keep exploring the maze
EXAMPLE COUNTING

• Lets take an example of counting from 0
• The next number is the previous number plus 1, or mathematically: 
  \[ f_n = f_{n-1} + 1, \text{ where } f_0 = 0 \]
• So lets compute \( f_5 \)
  • \( f_5 = f_4 + 1 \), this would be great if we knew \( f_4 \), so lets expand it
  • \( f_5 = (f_3 + 1) + 1 \), then
  • \( f_5 = ((f_2 + 1) + 1) + 1 \), then
  • \( f_5 = (((f_1 + 1) + 1) + 1) + 1 \), then
  • \( f_5 = ((((f_0 + 1) + 1) + 1) + 1) + 1 \), then finally
  • \( f_5 = (((((0) + 1) + 1) + 1) + 1) + 1 = 5 \)
PRACTICE RECURSIVE FORMULAS
FIBONACCI SEQUENCE

• The Fibonacci Sequence is used in various places in mathematics and computer science

\[ f_n = f_{n-1} + f_{n-2}, \text{ where } f_0 = 0, f_1 = 1 \]

• Expand and evaluate \( f_6 \), work with a partner and show your work
HOW DO WE DO RECURSION IN CODE?

• Simply call the function within its own body

```java
public static void foo() {
    // possibly do some stuff
    foo();  // This example of recursion
    // possibly do some more stuff
}
```
EXAMPLE COUNTING

// This function counts using recursion
public static int recursiveCount(int n) {
  // f_0 = 0
  if (n == 0)
    return 0;
  // f_n = f_{n-1} + 1
  return recursiveCount(n - 1) + 1;
}

• Together lets trace
  System.out.println(recursiveCount(3));
  • recursiveCount(3)
    • return recursiveCount(2) + 1
      • return recursiveCount(1) + 1;
        • return recursiveCount(0) + 1;
          • return 0;
          • return 0 + 1;
        • return 1 + 1;
      • return 2 + 1;
    • return 3
PRACTICE RECURSIVE CODE
FIBONACCI SEQUENCE

• Write a Java function Fibonacci for

\[ f_n = f_{n-1} + f_{n-2} \]

```java
public static int Fibonacci(int n) {
    if(n == 0) return 0;
    if(n == 1) return 1;
    return Fibonacci(n-1) + Fibonacci(n-2);
}
```

• Practice tracing Fibonacci(3)

Fibonacci(3)
  • return Fibonacci(2) + Fibonacci(1);
    • return Fibonacci(1) + Fibonacci(0);
      • return 1;
    • return 1 + 0;
    • return 1 + 1;
  • return 1;
  • return 1 + 1;

• 2
CHARACTERISTICS OF RECURSION

• All recursive methods have the following characteristics:
  • One or more base cases (the simplest case) are used to stop recursion.
  • Every recursive call reduces the original problem, bringing it increasingly closer to a base case until it becomes that case.

• In general, to solve a problem using recursion, you break it into subproblems. If a subproblem resembles the original problem, you can apply the same approach to solve the subproblem recursively. This subproblem is almost the same as the original problem in nature with a smaller size.
DESIGNING RECURSIVE FUNCTIONS

• Identify the base case
  • The base case is the part of the recursion not defined in terms of itself, e.g., \( f_0 = 0, f_1 = 1 \)
  • This is when the recursion stops! If you forget your base case, then the world will end
    • Really an infinite series of function calls until your computer crashes (if it ever does)

• Identify the recursive process
  • This is the algorithmic process or algorithmic formula

• Write the code
PRACTICE DESIGNING RECURSIVE FUNCTIONS
GREATEST COMMON DENOMINATOR (GCD)

• GCD
  • For two integers \( p \) and \( q \), if \( q \) divides \( p \) then the GCD of \( p \) and \( q \) is \( q \)
  • Otherwise the GCD of \( p \) and \( q \) is the same as \( q \) and \( p \% q \)

• Step 1: Identify the base case
  • \( q = 0 \) implies that the GCD is \( p \)

• Step 2: Identify the recursive step
  • \( GCD(q, p \% q) \)

• Step 3: Code

1. public static int gcd(int p, int q) {
2.     if (q == 0) return p;
3.     return gcd(q, p \% q);
4. }

RECURSIVE GCD DEMO

1. public class Euclid {
2.     public static int gcd(int p, int q) {
3.         if (q == 0) return p;
4.         else return gcd(q, p % q);
5.     }
6. }
7. public static void main(String[] args) {
8.     System.out.println(gcd(1272, 216));
9. }
10.}
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}

gcd(1272, 216)

Memory gcd call - 1

\[ p = 1272, \quad q = 216 \]
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}

gcd(1272, 216)

Memory gcd call - 1

\[ p = 1272, \ q = 216 \]
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}

p = 1272, q = 216

Memory gcd call - 1

gcd(1272, 216)
```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}

p = 1272, q = 216
Memory gcd call - 1
1272 = 216 \times 5 + 192

p = 216, q = 192
Memory gcd call - 2
```
```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

```
p = 1272, q = 216
Memory gcd call - 1
```

```
p = 216, q = 192
Memory gcd call - 2
```

```
gcd(1272, 216)
gcd(216, 192)
```

```
gcd(1272, 216)
```

```
gcd(216, 192)
```

```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}

gcd(1272, 216)
p = 1272, q = 216
Memory gcd call - 1

static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}

gcd(216, 192)
p = 216, q = 192
Memory gcd call - 2
```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

Memory gcd call - 1

```c
p = 1272, q = 216
```

gcd(1272, 216)

```c
p = 216, q = 192
```

gcd(216, 192)

```c
p = 192, q = 24
```

gcd(192, 24)

```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

Memory gcd call - 2

Memory gcd call - 3

```c
gcd(216, 192)
```

```c
gcd(192, 24)
```

```c
gcd(1272, 216)
```
```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

Memory gcd call - 1

- **p = 1272, q = 216**
- **p = 216, q = 192**
- **p = 192, q = 24**

```
gcd(1272, 216)
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

Memory gcd call - 2

```
gcd(216, 192)
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

Memory gcd call - 3

```
gcd(192, 24)
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```
```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

Memory gcd call - 1

```
test: gcd(1272, 216)
```

```
p = 1272, q = 216
```

Memory gcd call - 2

```
test: gcd(216, 192)
```

```
p = 216, q = 192
```

Memory gcd call - 3

```
test: gcd(192, 24)
```

```
p = 192, q = 24
```
```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

**Memory gcd call - 1**
```
p = 1272, q = 216
```

**Memory gcd call - 2**
```
p = 216, q = 192
```

**Memory gcd call - 3**
```
p = 192, q = 24
```

**Memory gcd call - 4**
```
p = 24, q = 0
```

**gcd(1272, 216)**
```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

**gcd(216, 192)**
```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

**gcd(192, 24)**
```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

**gcd(24, 0)**
```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```
```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

**gcd(1272, 216)**
```
p = 1272, q = 216
```

**Memory gcd call - 1**
```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

**gcd(216, 192)**
```
p = 216, q = 192
```

**Memory gcd call - 2**
```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

**gcd(192, 24)**
```
p = 192, q = 24
```

**Memory gcd call - 3**
```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

**gcd(24, 0)**
```
p = 24, q = 0
```

**Memory gcd call - 4**
```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```
static int gcd (int p, int q) {
    if (q == 0) return p;
    else return gcd (q, p % q);
}

Memory gcd call - 1

p = 1272, q = 216

gcd(1272, 216)

Memory gcd call - 2

p = 216, q = 192

gcd(216, 192)

Memory gcd call - 3

p = 192, q = 24

gcd(192, 24)

Memory gcd call - 4

p = 24, q = 0

gcd(24, 0)

p = 192, q = 24

Memory gcd call - 3

p = 216, q = 192

Memory gcd call - 2

p = 1272, q = 216

Memory gcd call - 1

p = 24, q = 0

Memory gcd call - 4

p = 192, q = 24
```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

1. **gcd(1272, 216)**
   - Memory gcd call - 1
   - p = 1272, q = 216

2. **gcd(216, 192)**
   - Memory gcd call - 2
   - p = 216, q = 192

3. **gcd(192, 24)**
   - Memory gcd call - 3
   - p = 192, q = 24

- **gcd(192, 24)**
  - p = 192, q = 24
  - Memory gcd call - 3

- **gcd(216, 192)**
  - p = 216, q = 192

- **gcd(1272, 216)**
  - p = 1272, q = 216

```
gcd(1272, 216)
gcd(216, 192)
gcd(192, 24)
```
```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

Example usage:

```c
gcd(1272, 216)
gcd(216, 192)
gcd(192, 24)
```
```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

Example:

```
p = 1272, q = 216
```

```
Memory gcd call - 1
```

```
gcd(1272, 216)
```

```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

```
p = 216, q = 192
```

```
Memory gcd call - 2
```

```
gcd(216, 192)
```

```
p = 216, q = 192
```

```
24
```
```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

`p = 1272, q = 216`

Memory gcd call - 1

`p = 216, q = 192`

Memory gcd call - 2

`gcd(216, 192)`

```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

`gcd(1272, 216)`

```c
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```
static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}

Memory gcd call - 1

p = 1272, q = 216

gcd(1272, 216)

24
public class Euclid {
    public static int gcd(int p, int q) {
        if (q == 0) return p;
        else return gcd(q, p % q);
    }

    public static void main(String[] args) {
        System.out.println(gcd(p, q));
    }
}
RECURSIVE HELPER METHODS

• Many of the problems presented in the early chapters can be solved using recursion if you think recursively. For example, the palindrome problem can be solved recursively as follows:

```java
1. public static boolean isPalindrome(String s) {
2.   if (s.length() <= 1) // Base case
3.     return true;
4.   else if (s.charAt(0) != s.charAt(s.length() - 1)) // Base case
5.     return false;
6.   else
7.     return isPalindrome(s.substring(1, s.length() - 1));
8. }
```
The preceding recursive `isPalindrome` method is not efficient, because it creates a new string for every recursive call. To avoid creating new strings, use a helper method:

```java
public static boolean isPalindrome(String s) {
    return isPalindrome(s, 0, s.length() - 1);
}

public static boolean isPalindrome(String s, int low, int high) {
    if (high <= low) // Base case
        return true;
    else if (s.charAt(low) != s.charAt(high)) // Base case
        return false;
    else
        return isPalindrome(s, low + 1, high - 1);
}
```
EXERCISE – PROGRAM TOGETHER

DOWNLOAD STDDRAW – LETS DRAW A TREE!
```java
import java.util.Scanner;

public class Tree {
    public static void drawTree(int o) {
        drawTree(o, 0., 0., 60, Math.PI/2, 35*Math.PI/180);
    }

    public static void drawTree(int o, double x, double y, double l, double t1, double t2) {
        if(o < 0) return;
        double x2 = x + l * Math.cos(t1);
        double y2 = y + l * Math.sin(t1);
        StdDraw.line(x, y, x2, y2);
        drawTree(o - 1, x2, y2, l*0.6, t1 + t2, t2);
        drawTree(o - 1, x2, y2, l*0.6, t1 - t2, t2);
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter an order: ");
        int order = in.nextInt();
        StdDraw.setXscale(-200, 200);
        StdDraw.setYscale(0, 400);
        StdDraw.enableDoubleBuffering();
        drawTree(order);
        StdDraw.show();
    }
}
```
DOWNSIDE OF RECURSION

• Recursion is not always efficient!
• Take for instance, the Fibonacci sequence

```java
public static long F(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return F(n-1) + F(n-2);
}
```

• F(50) is called once.
• F(49) is called once.
• F(48) is called 2 times.
• F(47) is called 3 times.
...  
• F(1) is called 12,586,269,025 times.
BEST PRACTICE
CONVERT RECURSIVE ALGORITHMS TO ITERATIVE ONES

• Try to do this whenever possible
• Example Fibonacci Sequence

```java
1. // This is an example conversion. You can be even more efficient!
2. public static long F(int n) {
3.     if (n == 0) return 0;
4.     if (n == 0) return 1;
5.     int fn2 = 0;
6.     int fn1 = 1;
7.     int fn = 1;
8.     // Iterative means repetition until failure condition,
9.     // typically done with loops and not recursion
10.    for (int i = 2; i <= n; i++) {
11.        fn = fn1 + fn2;
12.        fn2 = fn1;
13.        fn1 = fn;
14.    }
15.    return fn;
16. }
```
OR DO TAIL RECURSION

• Tail recursion is when the last operation of a function is the recursive call

• Example with factorial:
  \[ f_n = n \times f_{n-1} \]

1. public static long factorial(int n) {
2.   if (n == 0) return 1;
3.   else return n*factorial(n-1);
4. }

1. public static long factorial(int n) {
2.   return factorial(n, 1);
3. }
4. public static long factorial(int n, int result) {
5.   if(n == 0) return result;
6.   else
7.     return factorial(n - 1, n * result);
8. }
SUMMARY

• How to write simple recursive programs?
  • Base case, reduction step.

• Trace the execution of a recursive program.

• Why learn recursion?
  • New mode of thinking.
  • Powerful programming tool
TOWERS OF HANOI

PRACTICE
TOWERS OF HANOI

• Design recursive algorithm to move all the discs from the leftmost peg to the rightmost one.
  • Only one disc may be moved at a time.
  • A disc can be placed either on empty peg or on top of a larger disc.
SOLUTION
TOWERS OF HANOI

Move n-1 smallest discs right.

Move largest disc left.

Move n-1 smallest discs right.
public class TowersOfHanoi {
    public static void moves(int n, char from, char to, char aux) {
        if (n == 0) return;
        moves(n-1, from, aux, to);
        System.out.printf("Move disk %d from peg %c to peg %c", n, from, to);
        moves(n-1, aux, to, from);
    }
    public static void main(String[] args) {
        moves(3, 'A', 'C', 'B');
    }
}