CSE 143
Lecture 11

Recursive Programming

slides created by Marty Stepp
http://www.cs.washington.edu/143/

Decimal Numbers

- A visual review of decimal numbers:
  - We get 348 by adding powers of 10

    348 =

    300 + 40 + 8 =

    $3 \times 10^2 + 4 \times 10^1 + 8 \times 10^0$

- That's why the decimal system is **base 10**

Binary Numbers

- Binary is exactly the same, but **base 2**

Decimal	Binary	Sum
0	0	0×2^0
1	1	1×2^0
2	10	$1 \times 2^1 + 0 \times 2^0$
3	11	$1 \times 2^1 + 1 \times 2^0$
4	100	$1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
5	101	$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
6	110	$1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
7	111	$1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
8	1000	$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$

3

Exercise

- Write a recursive method `printBinary` that accepts an integer and prints that number's representation in binary (base 2).
  - Example: `printBinary(7)` prints 111
  - Example: `printBinary(12)` prints 1100
  - Example: `printBinary(42)` prints 101010

place	10	1
value	**4**	**2**

32	16	8	4	2	1
1	**0**	**1**	**0**	**1**	**0**

  - Write the method recursively and without using any loops.

4

Case analysis

- Recursion is about solving a small piece of a large problem.
  - What is 69743 in binary?
    - Do we know *anything* about its representation in binary?
  - Case analysis:
    - What is/are easy numbers to print in binary?
    - Can we express a larger number in terms of a smaller number(s)?
  - Suppose we are examining some arbitrary integer `N`.
    - if `N`'s binary representation is **10010101011**
    - `(N / 2)`'s binary representation is **1001010101**
    - `(N % 2)`'s binary representation is **1**

5

`printBinary` solution

```
// Prints the given integer's binary representation.
// Precondition: n >= 0
public static void printBinary(int n) {
    if (n < 2) {
        // base case; same as base 10
        System.out.println(n);
    } else {
        // recursive case; break number apart
        printBinary(n / 2);
        printBinary(n % 2);
    }
}
```

- Can we eliminate the precondition and deal with negatives?

6

printBinary solution 2

```
// Prints the given integer's binary representation.
public static void printBinary(int n) {
    if (n < 0) {
        // recursive case for negative numbers
        System.out.print("-");
        printBinary(-n);
    } else if (n < 2) {
        // base case; same as base 10
        System.out.println(n);
    } else {
        // recursive case; break number apart
        printBinary(n / 2);
        printBinary(n % 2);
    }
}
```


Exercise

- Write a recursive method `isPalindrome` accepts a `String` and returns `true` if it reads the same forwards as backwards.
  - `isPalindrome("madam")` → `true`
  - `isPalindrome("racecar")` → `true`
  - `isPalindrome("step on no pets")` → `true`
  - `isPalindrome("able was I ere I saw elba")` → `true`
  - `isPalindrome("Java")` → `false`
  - `isPalindrome("rotater")` → `false`
  - `isPalindrome("byebye")` → `false`
  - `isPalindrome("notion")` → `false`

Exercise solution

```
// Returns true if the given string reads the same
// forwards as backwards.
// Trivially true for empty or 1-letter strings.
public static boolean isPalindrome(String s) {
    if (s.length() < 2) {
        return true;   // base case
    } else {
        char first = s.charAt(0);
        char last  = s.charAt(s.length() - 1);
        if (first != last) {
            return false;
        }              // recursive case
        String middle = s.substring(1, s.length() - 1);
        return isPalindrome(middle);
    }
}
```

9

Exercise solution 2

```
// Returns true if the given string reads the same
// forwards as backwards.
// Trivially true for empty or 1-letter strings.
public static boolean isPalindrome(String s) {
    if (s.length() < 2) {
        return true;   // base case
    } else {
        return s.charAt(0) == s.charAt(s.length() - 1)
            && isPalindrome(s.substring(1, s.length() - 1));
    }
}
```

10

Exercise

- Write a method `crawl` accepts a `File` parameter and prints information about that file.
  - If the `File` object represents a normal file, just print its name.
  - If the `File` object represents a directory, print its name and information about every file/directory inside it, indented.

```
cse143
    handouts
        syllabus.doc
        lecture_schedule.xls
    homework
        1-sortedintlist
            ArrayIntList.java
            SortedIntList.java
            index.html
            style.css
```

  - **recursive data**: A directory can contain other directories.

11

`File` objects

- A `File` object (from the `java.io` package) represents a file or directory on the disk.

Constructor/method	Description
`File(`**String**`)`	creates `File` object representing file with given name
`canRead()`	returns whether file is able to be read
`delete()`	removes file from disk
`exists()`	whether this file exists on disk
`getName()`	returns file's name
`isDirectory()`	returns whether this object represents a directory
`length()`	returns number of bytes in file
`listFiles()`	returns a `File[]` representing files in this directory
`renameTo(`**File**`)`	changes name of file

12

Public/private pairs

- We cannot vary the indentation without an extra parameter:

```
public static void crawl(File f, String indent) {
```

- Often the parameters we need for our recursion do not match those the client will want to pass.

  In these cases, we instead write a pair of methods:
  1) a public, non-recursive one with the parameters the client wants
  2) a private, recursive one with the parameters we really need

Exercise solution

```
// Prints information about this file,
// and (if it is a directory) any files inside it.
public static void crawl(File f) {
    crawl(f, "");   // call private recursive helper
}

// Recursive helper to implement crawl/indent behavior.
private static void crawl(File f, String indent) {
    System.out.println(indent + f.getName());
    if (f.isDirectory()) {
        // recursive case; print contained files/dirs
        for (File subFile : f.listFiles()) {
            crawl(subFile, indent + "    ");
        }
    }
}
```

Sum

- Write a method `sum` that takes an integer array as a parameter and computes the sum of the array elements recursively.

```
// returns the sum of the numbers in the given array
public static int sum(int[] list) {
   return sum(list, 0);
}

// computes the sum of the list starting at the given index
private static int sum(int[] list, int index) {
   if (index == list.length) {
      return 0;
   } else {
      return list[index] + sum(list, index + 1);
   }
}
```

15

Towers of Hanoi

16

Towers of Hanoi

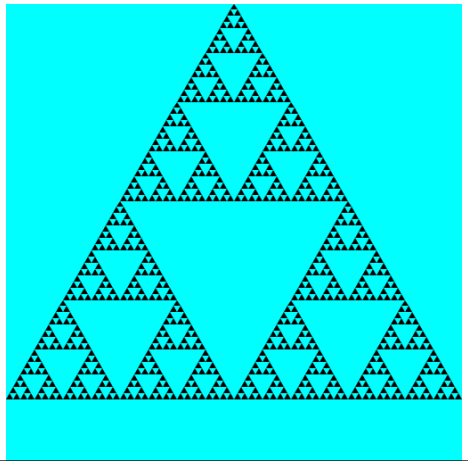
```
public class Towers {
   public static void main(String[] args) {
      towers(3, "A", "C", "B");
   }

   public static void towers(int num, String from, String to, String free) {
      if (num == 1) {
         System.out.println("Move " + num + " from " + from + " to " + to);
      } else {
         towers(num - 1, from, free, to);
         System.out.println("Move " + num + " from " + from + " to " + to);
         towers(num - 1, free, to, from);
      }
   }
}
```

17

Example: `Sierpinski`

- Chapter 12 in the book discusses the Sierpinski triangle, a fractal (recursive image):



18