CSE 143
Lecture 13

Recursive Programming

reading: 12.2 - 12.3

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

Exercise

- Write a 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.

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**

Exercise 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?

Exercise 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 method `pow` accepts an integer base and exponent as parameters and returns the base raised to that exponent.
  - Example: `pow(3, 4)` returns 81
  - Solve the problem recursively and without using loops.

Exercise solution

```
// Returns base ^ exponent.
// Precondition: exponent >= 0
public static int pow(int base, int exponent) {
    if (exponent == 0) {
        // base case; any number to 0th power is 1
        return 1;
    } else {
        // recursive case:  x^y = x * x^(y-1)
        return base * pow(base, exponent - 1);
    }
}
```

An optimization

- Notice the following mathematical property:

$3^{12} = 531441 = 9^6$

$= (3^2)^6$

$531441 = (9^2)^3$

$= ((3^2)^2)^3$

  - When does this "trick" work?
  - How can we incorporate this optimization into our `pow` method?
  - What is the benefit of this trick if the method already works?

Exercise solution 2

```
// Returns base ^ exponent.
// Precondition: exponent >= 0
public static int pow(int base, int exponent) {
    if (exponent == 0) {
        // base case; any number to 0th power is 1
        return 1;
    } else if (exponent % 2 == 0) {
        // recursive case 1:  x^y = (x^2)^(y/2)
        return pow(base * base, exponent / 2);
    } else {
        // recursive case 2:  x^y = x * x^(y-1)
        return base * pow(base, exponent - 1);
    }
}
```

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 that directory.

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

  - A recursive structure: A directory can contain other directories.

`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

Exercise solution

```
// Prints information about this file,
// and (if it is a directory) any files inside it.
public static void crawl(File f) {
    System.out.println(f.getName());
    if (f.isDirectory()) {
        // recursive case; print contained files/dirs
        for (File subFile : f.listFiles()) {
            crawl(subFile);
        }
    }
}
```

Exercise part 2

- Now write a version that indents: When printing a directory, indent it and its contents 4 spaces further than the parent.

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

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 2

```
// 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
```