CSE 143
Lecture 10

Recursive Programming

reading: 12.2 - 12.3

slides adapted from Marty Stepp and Hélène Martin
http://www.cs.washington.edu/143/
Exercise

• Write a recursive method $\text{pow}$ accepts an integer base and exponent and returns the base raised to that exponent.
  – Example: $\text{pow}(3, 4)$ returns 81

  – Solve the problem recursively and without using loops.
// 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 \texttt{pow} method?
– What is the benefit of this trick if the method already works?
// 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 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

- Write the method recursively and without using any loops.
• 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)?
Seeing the pattern

Suppose we are examining some arbitrary integer $N$.

- if $N$'s binary representation is $10010101011$
- $(N \div 2)$'s binary representation is $1001010101$
- $(N \mod 2)$'s binary representation is $1$

- What can we infer from this relationship?
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?
// 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`
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);
    }
}
// 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) {
    return s.length() < 2 ||
            (s.charAt(0) == s.charAt(s.length() - 1) &&
             isPalindrome(s.substring(1, s.length() - 1)));}
Exercise

• Write a recursive method `reverseLines` that accepts a file `Scanner` and prints the lines of the file in reverse order.

  – Example input file:

    ```
    Roses are red,
    Violets are blue.
    All my base
    Are belong to you.
    ```

  – Expected console output:

    ```
    Are belong to you.
    All my base
    Violets are blue.
    Roses are red,
    ```

  – What are the cases to consider?

    • How can we solve a small part of the problem at a time?
    • What is a file that is very easy to reverse?
Reversal pseudocode

• Reversing the lines of a file:
  – Read a line L from the file.
  – Print the rest of the lines in reverse order.
  – Print the line L.

• If only we had a way to reverse the rest of the lines of the file....
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        // recursive case
        String line = input.nextLine();
        reverseLines(input);
        System.out.println(line);
    }
}

- Where is the base case?
Tracing our algorithm

• **call stack:** The method invocations running at any one time.

```java
reverseLines(new Scanner("poem.txt"));
```
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.
A **File object** (from the `java.io` package) represents a file or directory on the disk.

<table>
<thead>
<tr>
<th>Constructor/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>File(String)</code></td>
<td>creates File object representing file with given name</td>
</tr>
<tr>
<td><code>canRead()</code></td>
<td>returns whether file is able to be read</td>
</tr>
<tr>
<td><code>delete()</code></td>
<td>removes file from disk</td>
</tr>
<tr>
<td><code>exists()</code></td>
<td>whether this file exists on disk</td>
</tr>
<tr>
<td><code>getName()</code></td>
<td>returns file's name</td>
</tr>
<tr>
<td><code>isDirectory()</code></td>
<td>returns whether this object represents a directory</td>
</tr>
<tr>
<td><code>length()</code></td>
<td>returns number of bytes in file</td>
</tr>
<tr>
<td><code>listFiles()</code></td>
<td>returns a File[] representing files in this directory</td>
</tr>
<tr>
<td><code>renameTo(File)</code></td>
<td>changes name of file</td>
</tr>
</tbody>
</table>
• We cannot vary the indentation without an extra parameter:

```java
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
        for (File subFile : f.listFiles()) {
            crawl(subFile, indent + "    ");
        }
    }
}