Recursion

Recursion: If you get the point, stop; otherwise, see Recursion.

Infinite recursion: See Infinite recursion.

Read: pp. 403-408 but SKIP sect. 15.1.2

Look in ProgramLive CD, page 15-3, for some interesting recursive methods.

Download presented algorithms from the website

Recursive definition: A definition that is defined in terms of itself.

Recursive method: a method that calls itself (directly or indirectly).

Recursion is often a good alternative to iteration (loops), which we cover next. Recursion is an important programming tool. Functional languages have no loops — only recursion.

Two issues in coming to grips with recursion

1. How are recursive calls executed?
2. How do we understand a recursive method and how do we create one?

We discuss the first issue later. Suffice it to say that if you execute a call on a recursive method, carefully using our model of execution, you will see that it works. Briefly, a new frame is created for each recursive call.

DON’T try to understand a recursive method by executing its recursive calls! Use execution only to understand how it works.

Understanding a recursive method

Factorial:

\[ n! = \begin{cases} 1 & \text{for } n = 0 \\ n \cdot (n-1)! & \text{for } n > 0 \end{cases} \]

Step 1: HAVE A PRECISE SPECIFICATION

\[ n! = \begin{cases} 1 & \text{for } n = 0 \\ n \cdot (n-1)! & \text{for } n > 0 \end{cases} \]

public static int fact(int n) {
    if (n == 0) {
        return 1; // base case
    }
    // {n > 0}, an assertion
    return n * fact(n-1); // recursive case
}

Step 2: Check the base case.

When n = 0, 1 is returned, which is 0!. So the base case is handled correctly.

Step 3: Recursive calls make progress toward termination.

\[ n! = \begin{cases} 1 & \text{for } n = 0 \\ n \cdot (n-1)! & \text{for } n > 0 \end{cases} \]

public static int fact(int n) {
    if (n == 0) {
        return 1; // base case
    }
    // {n > 0}, an assertion
    return n * fact(n-1); // recursive case
}

Argument n-1 is smaller than parameter n, so there is progress toward reaching base case 0.

Step 4: Recursive case is correct.

Creating a recursive method

Task: Write a method that removes blanks from a String.

0. Specification:

\[ \text{s but with its blanks removed} \]

public static String deblank(String s) {
    // {s.length() = 0}
    if (s.length() == 0) {
        return s; // base case
    }
    // {s.length() > 0}
    return s[0] + deblank(s[1..]); // recursive case
}

Argument s[1..] is shorthand for s.substring(1).

Notation:
- i means i is shorthand for s.charAt(i).
- i[..] is shorthand for s.substring(i).

Understanding a recursive function

Factorial:

\[ n! = \begin{cases} 1 & \text{for } n = 0 \\ n \cdot (n-1)! & \text{for } n > 0 \end{cases} \]

Step 3: Recursive calls make progress toward termination.

argument n-1 is smaller than parameter n, so there is progress toward reaching base case 0.

Step 4: Recursive case is correct.
Creating a recursive method

```java
// = s but with its blanks removed
public static String deblank(String s) {
    if (s.length() == 0)
        return s;
    // {s is not empty}
    if (s[0] is a blank)
        return s[1..] with its blanks removed
    // {s is not empty and s[0] is not a blank}
    return s[0] + (s[1..] with its blanks removed);
}
```

The tasks given by the two English, blue expressions are similar to the task fulfilled by this function, but on a smaller String! >>>Rewrite each as deblank(s[1..]) .

Notation: s[i] is shorthand for s.charAt(i). s[i..] is shorthand for s.substring(i).

Check palindrome-hood

A String with at least two characters is a palindrome if
(0) its first and last characters are equal, and
(1) chars between first & last form a palindrome:

e.g. AMANAPLANACANALPANAMA

```java
/** = "s is a palindrome"
 */
public static boolean isPal(String s) {
    if (s.length() <= 1)
        return true;
    // { s has at least two characters }
    return s.charAt(0) == s.charAt(s.length()-1)
        && isPal(s.substring(1, s.length()-1));
}
```

Hilbert's space-filling curve

```
As the size of each line gets smaller and smaller, in the limit, this algorithm fills every point in space. Lines never overlap.
```