CS1110 29 September. Recursion

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

ProgramLive CD, page 15-3, has interesting recursive methods.

Download presented algorithms from the website

Prelim I
7:30–9:00PM
Thursday, 6 Oct

Gries, Marschner in Hollister 202 between lectures

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 later. Recursion is an important programming tool. Some languages ("functional languages") have no loops —only recursion.

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

Two simple examples

Imagine String.length() did not exist.

```
/** = the number of characters in s. */
public static int len(String s) {
    if (s.isEmpty())
        return 0;
    // { s has at least one character }
    return 1 + len(s.substring(1));
}

/** = the number of 'e's in s. */
public static int numEs(String s) {
    if (s.isEmpty())
        return 0;
    // { s has at least one character }
    return (s.charAt(0) == 'e' ? 1 : 0) + noe(s.substring(1));
}
```


How to use the specification of a function

```
/** = the length of (number of chars in) s */
public static int len(String s) {
   …
}
```

parameter   argument

What is the value of this call: len("abc") ?

To determine the value of a call:
Make a copy of the specification and replace the parameters by the arguments.

The specification is important!!

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 discussed the first issue earlier. 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. We do this in the next lecture.

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

How to think about recursive methods

1. Have a precise method specification.

2. Base case(s): when the parameter values are as small as possible and the answer is determined with little calculation.

3. Recursive case(s): recursive calls are used. When verifying that the recursive cases are programmed properly, understand recursive calls in terms of the method specification.

4. Termination: The arguments of the recursive calls have somehow to be "smaller" than the parameters so that each recursive call gets closer to a based case.


```
/** = the number of 'e's in s */
public String noe(String s) {
    if (s.length() == 0) {        Called the base case
        return 0;
    }
    // { s has at least one char }   Called the recursive case
    return (s[0] = 'e' ? 1 : 0) + noe(s.substring(1));
}
```

0 1 s.length()

s

Notation:
s[i] shorthand for s.charAt[i].

s[i..] shorthand for s.substring(i).

Express the answer with the same terminology as the specification, but on a smaller scale:

number of 'e's in s = (if s[0] = 'e' then 1 else 0) + number of 'e's in s[1..]

Understanding a recursive method

Step 1: HAVE A PRECISE SPECIFICATION

```
// = number of 'e's in s
public static int noe(String  s) {
   if (s.length() == 0) {
         return 0;                base case
   }
   // {s has at least one character} recursive case (has a recursive call)
   // return (s[0] = 'e' ? 1 : 0) + number of 'e's in s[1..];
   return (s[0] = 'e' ? 1 : 0) + noe(s.substring(1));
}
```

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

Step 2: Check the base case.
When s is the empty string, 0 is returned.
So the base case is handled correctly.

Understanding a recursive function

s = "" **base case**
s has at least one character **recursive case**

Step 3: Recursive calls make progress toward termination.

argument s[1..] is smaller than parameter s, so there is progress toward reaching base case 0

```
// = number of 'e's in s
public static int noe(String  s) {
   if (s.length() == 0) {
         return 0;      base case
   }
   // {s has at least one character}   recursive case (has a recursive call)
   return (s[0] = 'e' ? 1 : 0) + noe(s.substring(1));
}
```

parameter s
argument s[1..]

Step 4: Recursive case is correct.

Creating a recursive method

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

0. Specification:

/** = s but with its blanks removed */ ← **precise spec!**
public static String deblank(String s)

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

1. Base case: the smallest String s is "".

```
if (s.length() == 0)
      return s;
```

2. Other cases: String s has at least 1 character.
return (s[0] == ' ' ? "" : "") + s[1..] with its blanks removed

Creating a recursive method

```
// = 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] shorthand for s.charAt[i].
s[i..] shorthand for s.substring(i).


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

Check the four points:
0. Precise specification?
1. Base case: correct?
2. Recursive case: progress toward termination?
3. Recursive case: correct?

Checking palindrome-hood

A String with at least two characters is a palindrome if:
- its first and last characters are equal, and
- the rest of the characters form a palindrome:

have to be the same

e.g. AMANAPLANACANALPANAMA

has to be a palindrome

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

0. Precise specification?
1. Base case: correct?
2. Recursive case: progress toward termination?
3. Recursive case: correct?