CS100J February 20 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.

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

Infinite recursion: See Infinite recursion.

Turn recursive definition into recursive function

Factorial:

!0 = 1 **base case**

!n = n * !(n-1) for n > 0 **recursive case**

Thus, !3 = 3 * !2
= 3 * 2 * !1
= 3 * 2 * 1 * !0
= 3 * 2 * 1 * 1 (= 6)

note the precise specification

```
// = !n  (for n ≥ 0)
public static int fact(int n) {
   if (n == 0) {
        return 1;                  base case
   }
   // {n > 0}                      an assertion
   return n * fact(n-1);           recursive case
}                                  (a recursive call)
```

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:

!0 = 1 **base case**

!n = n * !(n-1) for n > 0 **recursive case**

Step 1: HAVE A PRECISE SPECIFICATION

```
// = !n  (for n ≥ 0)
public static int fact(int n) {
   if (n == 0) {
        return 1;                  base case
   }
   // {n > 0}
   return n * fact(n-1);           recursive case
}                                  (a recursive call)
```

Step 2: Check the base case.

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

Understanding a recursive function

Factorial:

!0 = 1 **base case**

!n = n * !(n-1) for n > 0 **recursive case**

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

```
/** = !n  (for n>=0) */
public static int fact(int n) {
   if (n == 0) {
        return 1;
   }
   // {n > 0}
   return n * fact(n-1);           recursive case
}
```

parameter n

argument n-1

Step 4: Recursive case is correct.

Creating a recursive method

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

0. Specification: **precise specification!**

```
/** = s but with its blanks removed */
public static String deblank(String s)
```

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

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

2. Other cases: String s has at least 1 character. If it's blank, return s[1..] but with its blanks removed. If it's not blank, return

s[0] + (s[1..] but with its blanks removed)

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

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

Creating a recursive method

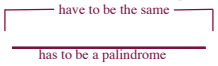
```
// = s but with its blanks removed
public static String deblank(String s) {
   if (s.length == 0)
         return s;
   // {s is not empty}
   if  (s.charAt(0) is a blank)
         return deblank(s.substring(1));
   // {s is not empty and s[0] is not a blank}
   return s.charAt(0) +
            deblank(s.substring(1));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

Check the four points:

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

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**

```
/** = "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));
   }
```

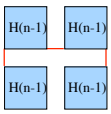

A man, a plan, a caret, a ban, a myriad, a sum, a lac, a liar, a hoop, a pint, a catalpa, a gas, an oil, a bird, a yell, a vat, a caw, a pax, a wag, a tax, a nay, a ram, a cap, a yam, a gay, a tsar, a wall, a car, a luger, a ward, a bin, a woman, a vassal, a wolf, a tuna, a nit, a pall, a fret, a watt, a bay, a daub, a tan, a cab, a datum, a gall, a hat, a fag, a zap, a say, a jaw, a lay, a wet, a gallop, a tug, a trot, a trap, a tram, a torr, a caper, a top, a tonk, a toll, a ball, a fair, a sax, a minim, a tenor, a bass, a passer, a capital, a rut, an amen, a ted, a cabal, a tang, a sun, an ass, a maw, a sag, a jam, a dam, a sub, a salt, an axon, a sail, an ad, a wadi, a radian, a room, a rood, a rip, a tad, a pariah, a revel, a reel, a reed, a pool, a plug, a pin, a peek, a parabola, a dog, a pat, a cud, a nu, a fan, a pal, a rum, a nod, an eta, a lag, an eel, a batik, a mug, a mot, a nap, a maxim, a mood, a leek, a grub, a gob, a gel, a drab, a citadel, a total, a cedar, a tap, a gag, a rat, a manor, a bar, a gal, a cola, a pap, a yaw, a tab, a raj, a gab, a nag, a pagan, a bag, a jar, a bat, a way, a papa, a local, a gar, a baron, a mat, a rag, a gap, a tar, a decal, a tot, a led, a tic, a bard, a leg, a bog, a burg, a keel, a doom, a mix, a map, an atom, a gum, a kit, a baleen, a gala, a ten, a don, a mural, a pan, a faun, a ducat, a pagoda, a lob, a rap, a keep, a nip, a gulp, a loop, a deer, a leer, a lever, a hair, a pad, a tapir, a door, a moor, an aid, a raid, a wad, an alias, an ox, an atlas, a bus, a madam, a jag, a saw, a mass, an anus, a gnat, a lab, a cadet, an em, a natural, a tip, a caress, a pass, a baronet, a minimax, a sari, a fall, a ballot, a knot, a pot, a rep, a carrot, a mart, a part, a tort, a gut, a poll, a gateway, a law, a jay, a sap, a zag, a fat, a hall, a gamut, a dab, a can, a tabu, a day, a batt, a waterfall, a patina, a nut, a flow, a lass, a van, a mow, a nib, a draw, a regular, a call, a war, a stay, a gam, a yap, a cam, a ray, an ax, a tag, a wax, a paw, a cat, a valley, a drib, a lion, a saga, a plat, a catnip, a pooh, a rail, a calamus, a dairyman, a bater, a canal ---Panama!

Hilbert's space-filling curve

Hilbert(1):

Hilbert(2):

Hilbert(n):



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

Hilbert's space-filling curve