CSE 8B Today

MORE RECURSION!

NEXT TUESDAY:
Get exam 3 back
We’ll discuss the solutions
public class RecursionDemo
{
    public int g( int x ) {
        return -1 * x;
    }
}

//.. In main:
RecursionDemo r = new RecursionDemo();
int result = r.g(-4);

Fill in the memory model for the code at left
public class RecursionDemo {
    public int g(int x) {
        return -1 * x;
    }
    public int f(int x) {
        return g(x/2);
    }
}

//.. In main:
RecursionDemo r = new RecursionDemo();
r.f(-4);

What does the memory model for this code look like? Draw the model at the point when g is executing its body.
public int demo(int x) {
    return x + f(x);
}
public int f(int x) {
    return 11*g(x) + g(x/2);
}
public int g(int x) {
    return -1 * x;
}

How many stack frames (total) are ever created when we call \texttt{r.demo(-4)}?

A. 0
B. 1
C. 2
D. 3
E. 4
"The Stack" – recursion?

REMEMBER is exactly the same situation... Only that now we call all the time to the same method.
Thinking sequentially

factorial

\[ 5! = 120 \]

\[ 5! = 5 \times 4 \times 3 \times 2 \times 1 \]

\[ N! = N \times (N-1) \times (N-2) \times \ldots \times 3 \times 2 \times 1 \]
Thinking *recursively*

**factorial**

\[ 5! = 120 \]
\[ 5! = 5 \times 4 \times 3 \times 2 \times 1 \]

\[ N! = N \times (N-1) \times (N-2) \times \ldots \times 3 \times 2 \times 1 \]
\[ N! = N \times (N-1)! \]
What is wrong with the following?

```java
public int fac( int n ) {
    return n * fac( n-1 );
}
```

A. A function cannot call itself in Java  
B. It will run forever because it is missing a base case  
C. It will not compile  
D. It will not compute the correct value of factorial because it is missing the recursive step
public int fac(int n) {
    return fac(n-1);
}
Thinking *recursively* 

factorial

\[ N! = N \times (N-1) \times (N-2) \times \ldots \times 3 \times 2 \times 1 \]

\[ N! = N \times (N-1)! \]

(and \(1! = 1\))

Recursive step, do a tiny bit of work, and then assume you have code that solves a smaller version of the same problem.

Base case (what to do when the problem can’t be simplified any further)
public int fac( int n ) {
    if ( (1) )
        (2)
    else
        return n*fac( n-1 );
}

What should go in blank (1)?
A. n > 1
B. n <= 1
C. fac(n) == 1
D. fac(n) > 1
How to think *recursively*?

• When writing a recursive method/algorithm you need to do two things:
  
  – **Identify a base case**
    • NO recursive call in the base case
    • A version of the problem so simple that you already know the answer, so just do it!
  
  – **The recursive step**
    • Assume that your function will WORK (magic!) for smaller versions of the input
    • Remember to do a small piece of the work
    • Remember that your new input must be heading toward your base case (UPDATE somehow the parameters that you input in the recursive call)

• You can think about these two steps in any order. Whatever works for you!
NOTE: All recursive code can be implemented with loops

• Simple: if, else recursion is usually “easy” to turn into a loop

• But some problems are naturally expressed in terms of a recursive structure, which makes writing the method easier and more elegant
Code tracing and recursion

// In the class RecursionDemo
public static void recFun( int num ) {
    System.out.println( num );
    num = num - 1
    if ( num > 0 ) {
        recFun( num );
        System.out.println( num );
    }
    // return;
}

// return;

RecursionDemo.recFun( 3 )

What does the above code print? (Can you explain why?)

A. 3
    3
    3
    3
    3
    ...

B. 3
    2
    2
    1
    1
    2

C. 3
    2
    2
    1
    1
    ...

D. 3
    2
    1
    1
    1
    1

E. 3
    3
    2
    2
    1
    1
Code tracing and recursion

// In the class RecursionDemo
public static void recFun2( int num ) {
    if ( num != 0 ) {
        recFun2( num - 2 );
        System.out.println( num );
    }
}

>> RecursionDemo.recFun2( 3 )

What does the above code print?

A. Nothing, it never gets to the print statement
B. 3
C. 3
D. 1
E. 3
Review, continued

// In the class RecursionDemo
public static void recFun2( int num ) {
    if ( num != 0 ) {
        recFun2( num - 2 );
        System.out.println( num );
    }
}

>> RecursionDemo.recFun2( 3 )

What does the above code print?

A. Nothing, it never gets to the print statement
B. 3 1
C. 3 1
D. 1 3
E. 3 3 3 3 3 ...

What small change would make the answer C?

What is the base case here?

Can you rewrite the code in recFun2 to make the answer D (with the same method call)?
Recursion, more practice: FIND!

Write a method to find an element in an array between `startIndex` and the end of the array and return its index as shown in the examples.

```java
public static int find(String[] myList, String toFind, int startIndex)
{
    ???
}
```

```java
>> int[] a = {"Hello", "Welcome", "Turtle", "fun"};
>> RecursionDemo.find(a, "Hello", 0)
0
>> RecursionDemo.find(a, "Happy", 0)
-1
>> RecursionDemo.find(a, "fun", 2)
3
>> RecursionDemo.find(a, "Hello", 2)
-1
```

What is the base case?
A. The element at startIndex is equal to toFind
B. The array is empty
C. startIndex is (greater than or) equal to myList.length
D. toFind is not in myList
E. More than one of these