CS 170 **Lab 12**
Fall 2012 **Recursion**

Due: Midnight tonight

Objectives:

This lab will reinforce recursion which is traditionally a pretty complicated technique for students to master in Computer Science.

Preparation:

Copy-and-paste these commands to your terminal. They copy the files and cd to the lab directory:

```
mkdir ~/cs170/lab12
cd ~cs170003/share/lab12
cp  *.java  ~/cs170/lab12
cd ~/cs170/lab12
```

Background:

We will write 2 recursive algorithms in this lab. But first, some theory on how to think about recursive algorithms.

- How to write recursive methods:

  As discussed in class, a recursive method will invoke itself, but the invocation always uses a parameter value that is smaller that the original value.

- Example: factorial

```
public static int factorial( int n ) {
   int sol1, mySol;

   if ( n == 0 ) {
      return 1;     // A solution is available for a base case
   }
   else {
      sol1 = factorial ( n - 1 );

      mySol = n * sol1;

      return mySol;
   }
}
```

  The original parameter is `n`, and `factorial(n)` invokes itself with a smaller parameter value `(n–1)`.

- The key in understanding recursive methods is recognizing that you are:
  - Not: "solving a smaller problem" to solve the original problem
    But: "using the solution of a smaller problem" to solve the original problem.
  - Think of the invocation `factorial(n-1)` as something that has been completed.   And it has returned the solution of the smaller problem that you can use.
- Example: compute 20!
  - We know that 20! = 20 x 19!
  - Suppose we know that 19! = 121645100408832000. Then we can use the solution of this

smaller problem (19!) to solve the original problem of 20!:
20! = 20 x 121645100408832000
= 2432902008176640000

- The Base case(s): In every recursive method, there is at least one or more base cases that must have been solved (i.e., you know the solution to without breaking the problem down further.) These readily solved problems are called base cases.
  - General rule: If you use the solution of the smaller problem Problem(n-k) to solve the problem Problem(n), you will need k base cases.
- Structure of a recursive method
  - The easiest recursive method uses one smaller problem to solve the original problem.
  - The structure of a recursive method to solve a problem of size n using the solution of a problem of size n−1 is:

```
public static ReturnType solveProblem( int n ) {
    ReturnType sol1, mySol;

    if ( n is a base case )    {
       return solution;     // A solution is available for a base case
    }
    else {
       sol1 = solveProblem ( n - 1 );
       mySol = solve problem of size n using solution sol1;
       return mySol;
    }
}
```

- Example: factorial

```
public static int factorial( int n )   {
   int sol1, mySol;

    if ( n == 0 ) {
       return 1;     // A solution is available for a base case
    }
    else {
       sol1 = factorial ( n - 1 );
       mySol = n * sol1;
       return mySol;
    }
 }
```

Task 1: A simple recursive method with integer input

- Problem description: Write a recursive method `power(int a, int n)` that computes a^n
- Open the `Power.java` file in gedit.
- How to approach the problem:
  - Q: What is the problem solved by the method?
    - A: `power(a,n)` computes a^n (example: `power(2,10)` computes 2^{10})
  - Q: What is a smaller problem of the same nature?
    - A: `power(a, n-1)` computes a^{n-1}. (example: `power(2,9)` computes 2^9)
  - Q: How can you use use the solution of `power(a, n-1)` to solve the original problem?
    - A: If we are solving 2^{10} and we know the solution of 2^9 is 512, we can solve 2^{10} as: 2 x 512.
    - In general: If we are solving an and we know the solution a^{n-1} `= sol1`, we can solve

        a^n as `a x sol1`.
    - Q: What is a base case that we have a readily available solution for?
        - A: `power(a, 0)` because anything raised to the 0 power is equal to 1 (so, `power(2,0)` is 1).
- Now use the structured approach above to write the recursive method `power(int a, int n)`. Do not use any loops in the method. Your solution must be recursive to receive credit.

Other kinds of smaller problems:

- The most common smaller problem that a recursive method solves within itself uses the parameter `n−1`
- But, that is not always the case.
- Problems involving strings that are solved by recursive method usually uses a shorter length string.
- The most common smaller problem that a "string typed" recursive method solves within itself uses a string of shorter length `n−1`.

Task 2: Another recursion exercise

- Problem description: Given a non-negative number n, write a recursive method `sumDigits(int n)` that computes sum of the digits in the number `n`.
- Open the `SumDigits.java` file for editing/compiling/running.
- How to approach the problem:
    - It is easier to describe the method by listing out the digits of a number, so I will use the notation:

        `n = xyz`

        to represent the fact that the number `n` consists of the digits `xyz`.
    - Q: What is the problem solved by the method?
        - A: `sumDigits(n=xyz)` computes `x + y + z` (example: `sumDigits(345)` returns `3 + 4 + 5` or `12`
    - Q: What is a smaller problem of the same nature?
        - A: `sumDigits (n/10 = xy)` computes `x+y` (example: `sumDigits(34)` computes 7). Note that `34 == 345/10.`
    - Q: How can you use the solution of `sumDigits(n/10)` to solve the original problem ?
        - A: If we are solving `sumDigits(345)` and we know the solution for `sumDigit(34) = 7`, we can solve `sumDigits(345)` by adding 7 plus the last digit of 345 (5). In other words: `7 + 345%5.`
          More generally: If we are solving `sumDigits(n = xyz)` and we know the solution of `sumDigit(n/10 = xy = sol1`, we can solve `sumDigit(n=xyz)` by:
          `sol1 + z` or
          `sol1 + (n % 10)`

    - Q: What is a base case that we have a readily available solution for?
        - A: `sumDigits(0)` is equal to 0
- Use the above information to write the recursive method `sumDigits(int n).` You must use recursion (not loops!) in order to receive credit.

Turnin

Use the following commands to turn in your lab 12:

```
 cd ~/cs170/lab11
/home/cs170003/turnin-lab   Power.java      lab12a
/home/cs170003/turnin-lab   SumDigits.java  lab12b
```