CMSC 113 – COMPUTER SCIENCE 1
Recursive Functions – A Hands-on Tutorial

In this hands-on tutorial you will practice designing and writing simple recursive functions. Recall from class that recursive functions always have two essential ingredients: (1) One or more base cases (for termination), and (2) One or more recursive calls. [See Section 2.3 of your text).

Task#1: Write a recursive definition for the following function:

$$0! = 1$$
$$n! = n * (n-1)!$$

Call the function `factorial()`. Write its definition below (if you are having trouble, please refer to Section 2.3 of your text. Page 264 has the code):

Next, write a small Java program that includes `factorial()` as defined above, and test it using the following loop:

```
for (int i = 0; i <= 25; i++)
   StdOut.printf("%d! = %d\n", i, factorial(i));
```

Observe the output. Is it correct? If not, how would you fix it?

Task#2: In the same Java program as above, write and test the function to compute the nth Harmonic number (recursive version) (Page 265 of your text). Once again, compute the first 25 harmonic numbers.

Task#3: Sometimes it is helpful to write a recursive helper function to compute something. For example, consider the definition below:

To compute **sum(n)** = 1 + 2 + … + n
call the function **sumHelper(n, 0)**

`sumHelper()` takes two arguments: **n** and the accumulating **sum** so far (therefore in the call above, it is 0) and can be defined as follows:

sumHelper(0, x) = x
sumHelper(n, x) = sumHelper(n-1, x+n)

Code both functions: **`sum()`** and **`sumHelper()`** as defined above. Test them, in the same program as above to compute **`sum(25, 0)`**.

Task#4: Doing graphics with recursive functions. Let us grow a tree:

A tree is a trunk (a thick line) rooted at (x, y) and is of length L. The command:

```
tree(0.5, 0, Math.toRadians(90), 0.3);
```

will draw a tree trunk at coordinates `(0.5, 0)` that is tilted by `90` degrees (i.e. pointing straight up) and of length `0.3`. The scale of the canvas is `(0.0, 1.0)`. Here is the function that accomplishes this:

```
public static void tree(double x, double y, double theta, double len) {
   // Draw a tree rooted at <x, y> tilted theta radians and of length, len
   // Compute coordinates for the other end of the tree trunk (cx, cy)
   double cx = x + len * Math.cos(theta);
   double cy = y + len * Math.sin(theta);

   // Draw the tree trunk <x, y> to <cx, cy>
   StdDraw.line(x, y, cx, cy);
} // tree()
```

Next write a Java program that calls `tree()` to draw a tree:

```
public static void main(String[] args) {
   tree(0.5, 0, Math.toRadians(90), 0.3);
} // main()
```

Put together the complete program (including the **`tree()`** function above) in a file called **`Tree.java`**. Compile and run it. You will see a vertical line (representing a tree trunk stump) in the canvas. Change the tilt angle to 60 degrees, and also 100 degrees to observe how the tree can be tilted.

Task#5: Next, let us modify the program from Task#4 to draw a branch on the trunk. If each tree trunk branches into a branch, we can recursively call **`tree()`** to draw each branch. To draw one branch, coming out of the top of the trunk can be done as shown below (new commands are highlighted):

```
public static void tree(int n, double x, double y, double theta, double len)
{
   // Draw a tree rooted at <x, y> tilted theta radians and of length, len
   // The tree branches, each branch tilts a little
   // Compute coordinates for the other end of the tree trunk (cx, cy)
   double cx = x + len * Math.cos(theta);
   double cy = y + len * Math.sin(theta);

   // Draw the tree trunk <x, y> to <cx, cy>
   StdDraw.line(x, y, cx, cy);

   if (n == 0) return;  // Done when n=0

   // Branches come out of the trunk at <cx, cy>, tilted
   // branchAngle radians apart. We will begin with the angle = 30 degrees
   double branchAngle = Math.toRadians(30);

   tree(n - 1, cx, cy, theta + branchAngle, len);


} // tree()
```

Modify the **tree()** function as shown. Also, notice that we added another parameter (**n**) to **tree()**. This is the number of times the tree branches. Modify your main program to input n from the command line and then use it:

```
tree(n, 0.5, 0, Math.toRadians(90), 0.3);
```

Observe the results of the tree. See that it branches n times (try it for n=3) after the initial trunk. Also notice that the branches are of equal length. Let's fix that.

Let us say that each subsequent branch is a fraction smaller than the original (say, 0.7). We can accomplish this by creating a new variable:

```
double branchRatio = 0.7;
```

and using the following recursive call in **tree()**:

```
tree(n - 1, cx, cy, theta + branchAngle, len * branchRatio);
```

Modify your program to incorporate the two commands above. You should now see the tree branches, each subsequent one being 70% shorter than the previous one.

We can make one more aesthetic adjustments: like we did for the length, we can also adjust the thickness of each branch to get thinner and thinner. Add the command:

```
StdDraw.setPenRadius(0.001 * n * n);      // Pen radius is proportional to n
```

Just before the **StdDraw.line()** command in the **tree()** function.

Task#6: Adding more branches! This is easy. Just add another recursive call to **tree()**, just below the earlier recursive call. That is:

```
tree(n - 1, cx, cy, theta + branchAngle, len * branchRatio);
tree(n - 1, cx, cy, theta - branchAngle, len * branchRatio);
```

Test the program for n=3, then try n=4, 5, 6! Do you see a tree?

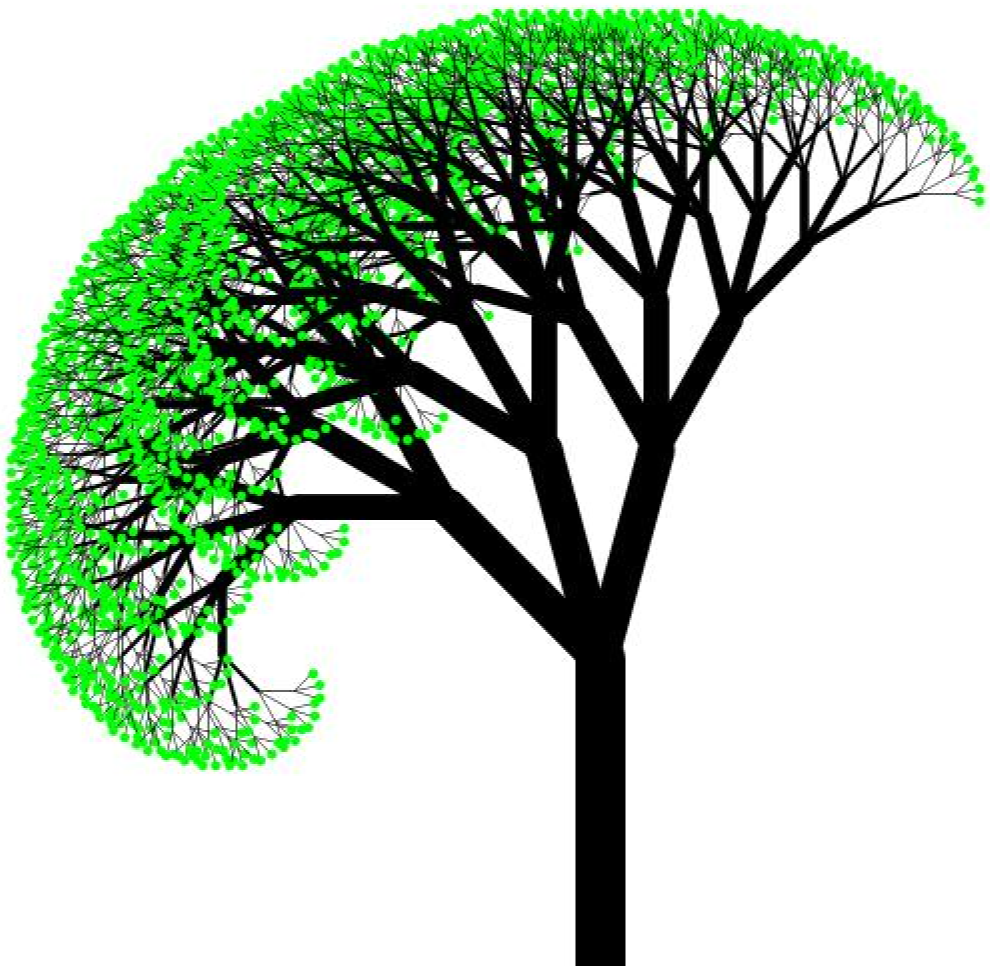
You can add another branch:

```
tree(n - 1, cx, cy, theta + branchAngle, len * branchRatio);
tree(n - 1, cx, cy, theta - branchAngle, len * branchRatio);
tree(n - 1, cx, cy, theta              , len * branchRatio);
```

Task#7: Add a bend angle to each branch:

```
double bendAngle = Math.toRadians(15);
tree(n - 1, cx, cy, theta + bendAngle + branchAngle, len * branchRatio);
tree(n - 1, cx, cy, theta + bendAngle - branchAngle, len * branchRatio);
tree(n - 1, cx, cy, theta + bendAngle              , len * branchRatio);
```

etc. Play with these parameters and some other ideas. For instance, draw some green dots (for leaves) at the tip of the final branches (See figure on next page).



I think that I shall never see
A poem lovely as a tree.
-: Joyce Kilmer