Computer Science 1 — CSci 1100
Lab 12 — Recursion
Fall Semester 2014

Lab Overview

This is the last lab of the semester! It only has two checkpoints. You will explore the basic mechanisms of recursion, the design of simple recursive functions, and conversion between recursion and iteration.

As discussed during class, recursion is most often used as a formal way of modeling algorithms and data structures and less often used as a practical programming technique. There are some problems, however, that are almost impossible to solve using a non-recursive algorithm. Many of the search algorithms for tree-like data structures — which you will see if you take Data Structures — fit into this category. Unfortunately, we will not see any algorithms here that fit this category. Still, building an understanding of recursion is an important step in your development as a programmer and as a computer scientist.

Checkpoint 1

Attempts to define the formal foundations of mathematics at the start of the 20th century depended heavily on the mathematical notion of recursion. While ultimately completing this formalization was shown to be impossible, the theory that was developed contributed heavily to the formal basis of computer science.

The idea is to start with primitive, axiomatic definitions and build from there. We will build basic arithmetic starting with just one value — 0 — and three operations: (1) adding 1 to a value (the *successor* operation), (2) subtracing 1 from a value (the *predecessor* operation), and (3) testing a value to see if it is 0. In this case, the value 1 is the successor of 0, the value 2 is the successor of the successfor of 0, etc.

Using just these operations, we can implement addition of positive integers using a Python recursive function:

```
def add(m,n):
    if n == 0:
        return m
    else:
        return add(m,n-1) + 1
```

To make sure you understand this, show the sequence of calls made by

```
print add(5,3)
```

You may add print statements to show the results.

Building on this idea, please write a recursive function to multiply two non-negative integers using only the `add` function we've just defined, together with +1, -1 and the equality with 0 test. Call this function `mult`. Demonstrate the result by multiplying 8 and 3.

Now, define the integer power function, power$(x, n) = x^n$, in terms of the `mult` function you just wrote, together with +1, -1, and equality. Demonstrate the result by computing `power(6,4)`.

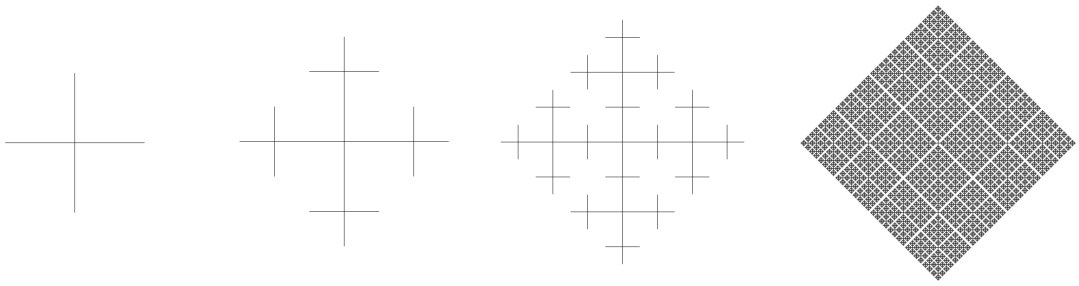
To complete Checkpoint 1: Show:

1. the calls from `add(5,3)`,
2. a working version of `mult`, and
3. a working version of `power`.

Checkpoint 2

In this section, you will write a recursive function to draw a self repeating plus sign. You are given the code to draw the first plus sign in the middle of the canvas. First, remember that `(0,0)` is the upper left corner, and `(800,800)` is the lower right corner.

At each iteration, your code will draw the same pattern at the four end points of the of the current plus sign and then reduce the length of the sign to half of its current value. The expected figure at iterations 0, 1, 2 and a much higher level is shown below.



When you start, your origin is at location `(400,400)` and original length is given as `150` on each side. You first draw a plus sign by drawing two lines, between:

`(250,400)-(550,400)` (horizontal line, total length `300`)

and

`(400,250)-(400,550)` (vertical line, total length `300`)

Now, you must start from the end points of this plus sign and draw four new plus signs (recursively) of length 75 at on each side (the center of these four new plus signs will be: `(250,400)`, `(400,550)`, `(400,250)`, and `(400,550)`.)

Every time you call the function recursively, you will increase a variable called `depth`. Your function will stop executing when depth reaches the max level that is given as input to the function.

To solve this, you are going to follow the same pattern we have used for the Sierpinski triangle in class. The code is provided to you under class modules. First, examine this program carefully and ask your TAs and mentors questions about how it works. You can then adopt the same solution to drawing the plus sign.

To complete Checkpoint 2: Show your working program.

Congratulations! You have completed all the labs in the course!

Now, you can try some different variations to see different effects for fun. Instead of decreasing the length by half, try setting the length to:

```
length = 2*length/3
length = 3*length/5
```

and other variations. You will see that as patterns overlap, you start seeing some very interesting Moiré patterns:

`http://en.wikipedia.org/wiki/Moire_pattern`

Now, try changing the length to `3/2` of its current value every third iteration and halving it at other iterations. Try shifting the starting values a little every time and see new patterns emerge.