Postscript Assignment

Due Tuesday Feb. 10, 5:00 AM

This assignment is a gentle introduction to the agony and joy of programming in PostScript. It is to be your *individual work*. The problems are simple--you could probably do them in your sleep in C++. Your whole assignment should end up being less than 100 lines of code. However, remember that you have to do them in an entirely new language - get started early and ask for help if you need it!

Postscript Help

Postscript BlueBook: http://partners.adobe.com/public/developer/en/ps/sdk/sample/BlueBook.zip
A First Guide to Postscript: http://www.cs.indiana.edu/docproject/programming/postscript/postscript.html
Intro to Ghostscript: http://pages.cs.wisc.edu/~ghost/doc/intro.htm

Getting Started

There are two things to download for this assignment. The first thing to download is the starter code which contains problem stubs and some example test/driver code. You'll find the starter/test code in the psFiles.zip file. Next download the PostScript interpreter.

Windows/Linux

You can download a PostScript interpreter, GhostScript, at http://www.ghostscript.com/download/gsdnld.html. A PostScript viewer called GSview is also available at http://pages.cs.wisc.edu/~ghost/gsview/index.htm, which will allow you see higher quality renderings of your PostScript.

Macintosh

For Mac, try: https://www.macupdate.com/app/mac/9980/gpl-ghostscript

Turning in your work

All code should be placed inside of a folder named ps-firstLast (i.e. ps-billJohnson). Your folder will contain a total of eight files (**ternary.ps**, **ackermann.ps**, **hypotrochoid.ps**, and **fractal.ps**) which should contain only your solutions to the problems. Driver/test code should reside in the other provided files (**testTernary.ps**, **testAckermann.ps**, **testHypotrochoid.ps**, and **testFractal.ps**) which you are not required to modify – but feel free to do so to aid in your testing. Zip that folder and submit it via canvas.
Note: To allow the test code to find your solution code - you will need to either specify the full path to your solution file inside the test file (uncomment and modify the run statement) or load each file individually.

Grading

The assignment will be graded for good programming style (indentation and appropriate comments), as well as clean execution/output. The assignment will also be graded to ensure that each operator leaves the stacks and the drawing environment in the same state as when the functions were called (though

operands will be popped from the stack). In other words, the operator should not leave extra values on the stack and should restore the graphics environment to that when the operator was called if the environment was changed.

Problem #1: ternary.ps -- 10%

This program should go into the file **ternary.ps**. Define an operator, ?, that mimics ternary operator in C++. In C++, the ternary operator behaves as follows:

```
expr1 ? expr2 : expr3
```

This results in the evaluation of expr2 if expr1 is non-zero and the evaluation of expr3 otherwise.
In PostScript the ? operator will need three values on the stack where the first value is an integer, and the second and third values are code (you do not have to error check the types of the values on the stack).

```
op1 op2 op3 ?
```

It evaluates as follows: if op1 is non-zero evaluate op2 otherwise evaluate op3. So, for example

```
2 {(ok)} {(zero)} ?       % (ok) will be pushed onto the stack
0 {(ok)} {(zero)} ?       % (zero) will be pushed onto the stack
-1 {1} {0} ?              % 1 pushed onto the stack
1 {2 3 add} {5 4 add} ?   % 5 pushed onto the stack
0 {2 3 add} {5 4 add} ?   % 9 pushed onto the stack
```

HINT: You might find the ifelse and roll operators to be very useful!

Problem #2: ackermann.ps -- 20%

Put this solution into the file **ackermann.ps**. Wikipedia describes Ackermann's function. The Ackermann function is recursively defined as follows. You many assume that the inputs, m and n, are nonnegative integers.

```
ackermann(m, n) =
    n + 1                                  if m == 0
    ackermann(m-1, 1)                      if m > 0 and n == 0
    ackermann(m-1, ackermann(m, n-1))      otherwise
```

Define a recursive ackermann function in PostScript. **Your implementation of the Ackermann function must be done as a recursive function, not a loop!** This operator has one operand and one result so the stack must contain the same number of elements before and after the operation. Here are some examples.

```
0 0 ackermann     % pushes 1 on the stack
1 0 ackermann     % pushes 2 on the stack
3 1 ackermann     % pushes 13 on the stack
1 3 ackermann     % pushes 5 on the stack
3 4 ackermann     % pushes 125 on the stack
```

Problem #3: hypotrochoid.ps -- 30%

EXTRA CREDIT: Impress me - capture an image of an awesome hypotrochoid your program created (you can "Copy to Clipboard" in Ghostscript or convert to different files from GSView) and my favorite 5 will receive 5% extra credit on the assignment and be shown off in class after the assignment is due.

This program should go into the file **hypotrochoid.ps**. You can read more about hypotrochoids at the following sites if you are interested:

- http://en.wikipedia.org/wiki/Hypotrochoid
- http://mathworld.wolfram.com/Hypotrochoid.html
- http://www.archimy.com/examples/2d-hypotrochoid.html

Write a PostScript definition for a function, hypotrochoid, which takes four operands from the stack as follows

a b d r hypotrochoid

and produces a hypotrochoid using the following formulas:

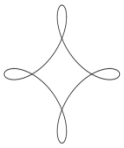
x(t) = ((a - b) * cos(t)) + (d * cos(((a-b) * t)/b))
y(t) = ((a - b) * sin(t)) - (d * sin(((a-b) * t)/b))

where **t = 0..360*r** (If your **r** is 5, then **t** should be 0 in your first iteration, 5 in your second, 10, etc. up to 1800)

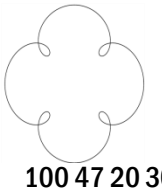
Each time your run these equation you get an x,y coordinate. Run it again, incrementing **t** appropriately, and then draw a line by using **lineto** to continue the line from the previous iteration's position.

Some example output:

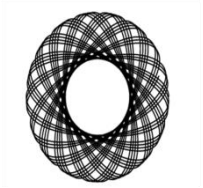
100 25 40 1 hypotrochoid



100 0 25 sub 40 1 hypotrochoid



100 47 20 30 hypotrochoid



Problem #4: fractal.ps -- 40%

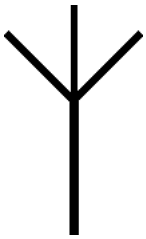
EXTRA CREDIT: Capture an image or PDF of a sweet fractal tree you created and my top 5 favorites will receive 5% extra credit on the assignment and also be shown off to the class.

This program should go into the file **fractal.ps**. Fractal is a term coined by Benoit Mandelbrot to describe geometric objects that are highly irregular and fail to fit into traditional geometry. They typically exhibit self-similarity, which means that a fractal has a structure, which at any scale, is an image of the overall structure. Recursive procedures are often used to draw fractals, with the same image drawn during every recursive call, but at an increasingly finer scale, often rotated and translated (HINT: use the PostScript coordinate transformation operations). Assume that the dimension of a fractal is the level of recursion used in drawing the fractal (i.e., one level of recursion is a one-dimensional fractal, two levels is a two-dimensional fractal, etc.).

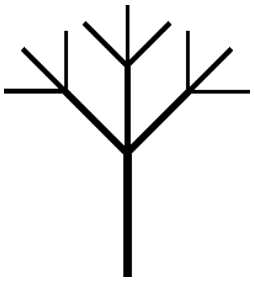
HINT: Implement this fractal by drawing a single line and then using the graphics transformations to scale, rotate, and translate the drawing coordinates and recursively call the fractal as many times as needed (i.e., three recursive calls for a tree with three main branches). Use **gsave** and **grestrore** to save and restore your graphics environment as needed.

For this assignment, you are to create a fractal that draws a slender or bushy tree-like object at higher dimensions. To get you started, here is what a basic tree-like fractal, based on drawing a vertical line at each recursive call, looks like at several dimensions. Note that each level adds three shorter and skinnier lines to the end of the line drawn at the previous level.
For dimension 1:

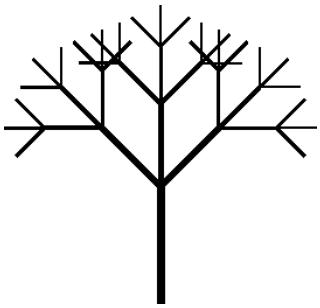
For dimension 2:



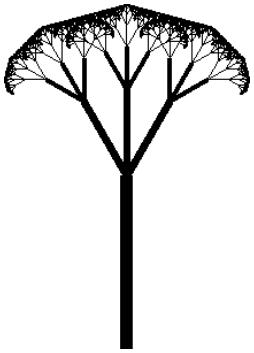
For dimension 3:



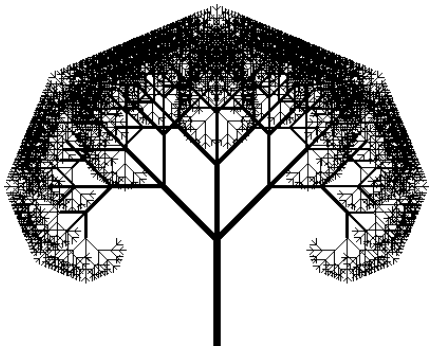
For dimension 4:



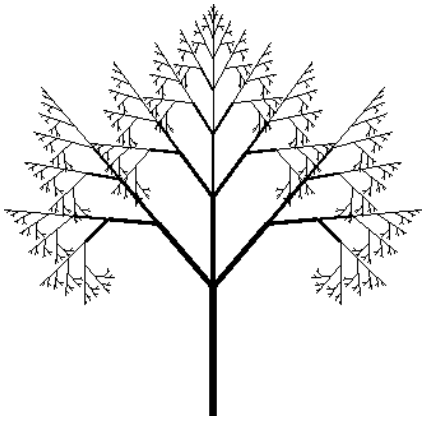
Your task is to define a fractal, fractal that draws tree. Assume that the dimension of the tree fractal is passed as an argument. For example, here is a slender looking tree at dimension 8, i.e., **8 fractal**:



Here is a bushy-looking tree at dimension 8:



Here would be a different bushy tree at dimension 8:



You don't have to copy these trees – make them look however you want, as long as they are “tree-like”.