COS126/128: General Computer Science	Spring 1996
Problem Set # 6	
Handed out:Mar. 26, 1996	*Due: Apr. 3, 1996*

Read the following before attempting this assignment: Course Reader pp 121–123, and 154–160

In this assignment you will continue to use the Postscript language to draw pictures (but you will use a somewhat richer set of instructions than before). You will write a program that outputs Postscript instructions to print the "H-tree" pattern shown at the end of this handout. Then you will modify the program to output some other recursive pattern of your own design. (Use your creativity to the fullest for this part; your designs will be entered in a contest!).

The H-tree is an example of a self-similar (fractal) pattern that is defined using recursion. An H-tree with parameter r is defined by drawing an "H" and then recursively drawing an H-tree with parameter $r/2$ at each of the four ends of the "H" that was just drawn. The thickness of each "H" drawn depends on the value of r when it is drawn.

(Recall that the Koch snowflake in the lecture notes was similarly defined using recursion.) H-trees are of practical interest in designing microprocessors chips; they a good pattern to use for running the wires that connect circuit components when a tree of interconnections is needed.

Part I (75%):

Write a program that outputs Postscript instructions for generating the "H-tree" pattern shown. Though the pattern looks complex, it can be generated with a recursive program very similar to the `star()` subroutine given on page 123 of your course reader. You should only need to write a function `htree()` quite similar to `star()`, a function `drawH()` to output the PostScript for an H of given parameters, and a `main` procedure to call the others and produce the desired effect.

Your postscript programs should do all their line-drawing inside a 512×512 integer coordinate system. To guide you, we provide the postscript instructions that should be output by your `htree` function when it is called for the first time with `r = 2`. This postscript program draws the five biggest "H"'s in the diagram at the end of this handout (though with a thinner line width than the one shown there), first the four in the corners, centered at $(128, 384)$, $(384, 384)$, $(128, 128)$, and $(384, 128)$, then the one in the middle, centered at $(256, 256)$. Note that the first "H" is big, even though $r = 2$. This means the size of the "H" has to be independent of the value of r – another difference from `star()`.

```
%!
50 50 translate .5 setgray
1 setlinewidth
  64  384 moveto  128    0 rlineto stroke
```

```
  64  320 moveto    0  128 rlineto stroke
 192  320 moveto    0  128 rlineto stroke
1 setlinewidth
 320  384 moveto  128    0 rlineto stroke
 320  320 moveto    0  128 rlineto stroke
 448  320 moveto    0  128 rlineto stroke
1 setlinewidth
  64  128 moveto  128    0 rlineto stroke
  64   64 moveto    0  128 rlineto stroke
 192   64 moveto    0  128 rlineto stroke
1 setlinewidth
 320  128 moveto  128    0 rlineto stroke
 320   64 moveto    0  128 rlineto stroke
 448   64 moveto    0  128 rlineto stroke
2 setlinewidth
 128  256 moveto  256    0 rlineto stroke
 128  128 moveto    0  256 rlineto stroke
 384  128 moveto    0  256 rlineto stroke
showpage
```

In general, you should use `setlinewidth` to set the thickness of the lines to `r`. The diagram on the back corresponds to starting with `r = 32`. For larger `r`, the thick wires overlap some of the small ones—in applications, this overlap might create problems, but don't worry about that for this assignment.

Part II (25%):

Now do another, different, recursive design. Experiment with implementing a different function to replace `drawH()`, or with a different function to replace `htree()`. For example, the `box()` function used for the figure in the book uses a `fill`, not a `stroke`. Placing the call to the drawing function (like `drawH()` or `box()`) in differing places with respect to the recursive calls will produce different and interesting effects. Or, work with "X"'s or "S"'s instead of "H"'s or use three recursive calls and triangles, or whatever. You will be graded on the originality and creativity in your design.

Extra Credit:

Try introducing some randomness in your recursive design. You can choose the way in which you use randomness. For example, change the pattern between recursive calls (a little? a lot ?). If your solution for part II seems aesthetically best, you can still submit your best design in this category as well.

More than Grades!

The best design – either a solution to Part II or Extra Credit – will be used on the class T-shirts that we'll be selling. We will select 5 "finalist" designs, and then hold an election in class to determine the winning design.

What to submit:

You should submit your program for part I as "htree.c" and your program with your best design from part II as "graphic.c":

```
/u/cs126/bin/submit 6 htree.c graphic.c
```

or

```
/u/cs128/bin/submit 6 htree.c graphic.c
```

as appropriate.

For your hardcopy submission, include the printed output for the H-tree design with $r = 32$ and the printed output for your best design from part II with r (and any other parameters) set as you think best. If you wish to submit a solution for extra credit, submit it as a separate program and separate picture.

Due: Wednesday, April 3

Submit electronically (`/u/cs126/submit`) before 10:00PM Wednesday, April 3. Annotated hardcopy must be handed in by lecture on Thursday April 4.