McMaster University
Department of Computing and Software
Dr. W. Kahl

CAS 781
Winter 2003
Assignment 2

CAS 781 — Functional Programming

2003-02-13

The following is the file `Assignment2.lhs` on the course page:

```
> type Vertex = (Float,Float)

> data Shape = Rectangle Vertex Vertex
>            | Polygon [Vertex]
>            | Polyline [Vertex]
>            | Ellipse Vertex Vertex
>            | ShearEllipse Vertex Vertex Vertex

> data Pic1 = EmptyPic1
>           | Above1 Pic1 Pic1
>           | Prim1 Shape
>           | WithColor1 Color Pic1

Produce an instance of Eq for Pic1 that respects the
equalities involving emptyPic and withColor, in particular:

< WithColor1 c1 (WithColor1 c2 p) == WithColor1 c2 p  =  True

> data Pic2 = EmptyPic2
>           | Above2 Pic2 Pic2
>           | Prim2 {shape :: Shape
>                   ,color :: Color
>                   }
>           | Move2 Pic2

Implement the following conversions:

> pic1FromPic2 :: Pic2 -> Pic1
> pic2FromPic1 :: Pic1 -> Pic2

Implement the following functions without using those conversions:

> graphicFromPic1 :: Pic1 -> Graphic
> graphicFromPic2 :: Pic2 -> Graphic

> move1 :: Vertex -> Pic1 -> Pic1
> move2 :: Vertex -> Pic2 -> Pic2

> withColor1 :: Color -> Pic1 -> Pic1
> withColor2 :: Color -> Pic2 -> Pic2

> prim1 :: Shape -> Pic1
> prim2 :: Shape -> Pic2

> above1 :: Pic1 -> Pic1 -> Pic1
> above2 :: Pic2 -> Pic2 -> Pic2
```

Implement instances of the following class for Pic1 and Pic2:

```
> class Picture p where
>   prim           :: Shape       -> p
>   move           :: Vertex -> p -> p
>   withColor      :: Color  -> p -> p
>   above          :: p      -> p -> p
>   graphicFromPic :: p           -> Graphic
```

Add a scaling function to this class and its instances.

Use the extended class to implement a function that, when invoked with

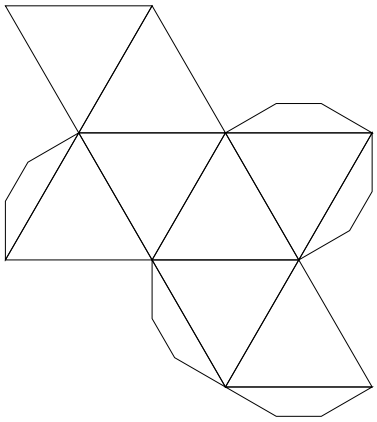
```
< sierpinski size d
```

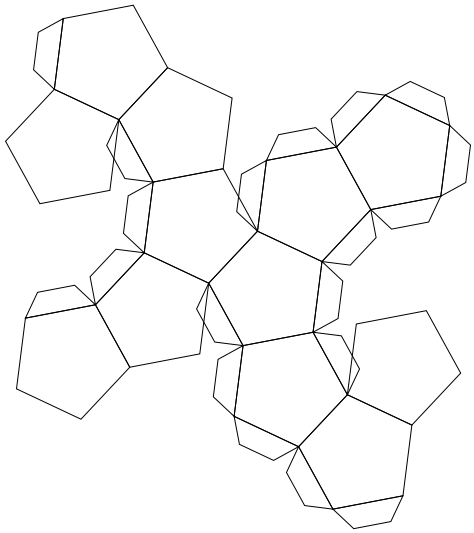
delivers a ‘‘Picture’’ Sierpinski triangle with outer side length size and recursion depth d:

```
> sierpinski :: Picture p => Double -> Int -> p
```

Perform experiments using graphicFromPic.

Use the other files on the course page to produce functions for producing paper folding models like the following (or better!), but ideally for arbitrary flat-sided three-diemnsional shapes:





The last two files need not be understood: `PSPic.lhs` provides instances:

```
> instance HasPS Shape
> instance HasPS s => HasPS (Pic s)
```

and the function

```
> writeEPS :: (HasVert a, HasPS a) => String -> a -> IO ()
```

to write pictures into PostScript files; it uses the module `PostScript`.

Tackle this problem as a proper **project:**

- Produce a requirements document
- Explore design alternatives
- Document the utilities you provide