Homework 3 — assigned Monday 31 March — due Sunday 6 April

(Questions 1 and 2 previewed on 16 March.)

3.1 Using lists for sets: writing recursive functions over lists (35pts)

Let us use the Haskell type `[Int]` to represent sets of integers. The representation invariants are that there are no duplicates in the list, and that the order of the list elements is increasing.

1. (5pts) Write a Haskell function `setUnion :: [Int] -> [Int] -> [Int]` that takes two sets and returns their union.
2. (5pts) Write a Haskell function `setIntersection :: [Int] -> [Int] -> [Int]` that takes two sets and returns their intersection.
3. (5pts) Write a Haskell function `setDifference :: [Int] -> [Int] -> [Int]` that takes two sets and returns their set difference.
4. (5pts) Write a Haskell function `setEqual :: [Int] -> [Int] -> Bool` that takes two sets and returns `True` if and only if the two sets are equal.
5. (15pts) Write a Haskell function `powerset :: [Int] -> [[Int]]` that takes a set S and returns its powerset 2^S. (The powerset 2^S of a set S (sometimes written $P(S)$) is the set of all subsets of S.) Note that the result uses the Haskell type `[[Int]]` to represent sets of sets of integers. Here the representation invariant is that there are no duplicates in the list; the order of the sublists is immaterial.

Your program should be prepared as a Haskell script `hw31.hs`; submit a detailed log of your interaction with `ghci`, showing a comprehensive test suite.

3.2 Drawing: writing recursive functions over lists; manipulating strings (35pts)

In this exercise, we develop a simple tool for drawing. A drawing is just a line drawing consisting of some number of polygons. A polygon is given as a list of vertices, and a vertex is simply a pair of real numbers for the x and y coordinates. For instance,

```
[[(100.0,100.0),(100.0,200.0),(200.0,100.0)],
 [(150.0,150.0),(150.0,200.0),(200.0,200.0),(200.0,150.0)]]
```

is an internal representation in Haskell of a drawing consisting of a triangle and a square.

Your task is to convert such a representation of a drawing into a simple page description in the PostScript language. Specifically, you are to write a Haskell function `makeCommand :: [[(Float,Float)]] -> String`.

The result returned by `makeCommand` is a Haskell value of type String, which must contain valid PostScript commands for drawing the given polygons.

For instance, the expression

```
makeCommand [[(100.0,100.0),(100.0,200.0),(200.0,100.0)],
             [(150.0,150.0),(150.0,200.0),(200.0,200.0),(200.0,150.0)]]
```

should evaluate to the text:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 100.0 100.0 200.0 200.0

100.0 100.0 moveto
100.0 200.0 lineto
200.0 100.0 lineto
closepath
stroke

150.0 150.0 moveto
150.0 200.0 lineto
200.0 200.0 lineto
200.0 150.0 lineto
closepath
stroke

showpage
%%EOF
```

which would be printed by a PostScript printer as in Figure 1.

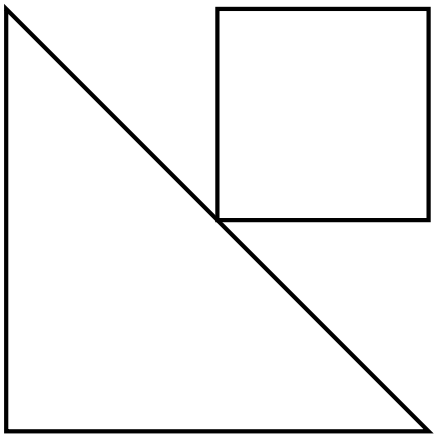

Figure 1: A triangle and a square.

Note that the bounding box is the smallest upright rectangle such that no points of the drawing lie outside it; it is specified by giving the coordinates of its lower left and upper right corners, in our example $(100.0, 100.0)$ and $(200.0, 200.0)$.

The example PostScript code shown here entirely suffices as a pattern to follow; however, if you would like to learn more about the PostScript language you can follow the links on the course web page.

Your program should be prepared as a Haskell script `hw32.hs`; submit a detailed log of your interaction with `ghci`, showing a comprehensive test suite.

3.3 Matrices (30pts)

In this exercise, we adopt the type `[Float]` as our representation of column vectors and the type `[[Float]]` as our representation of matrices, and we develop functions for matrix arithmetic.

Preface your code with the declaration:

```
type Realvector = [Float]
type Realmatrix = [[Float]]
```

1. (10pts) Write a function `rmvp :: Realmatrix -> Realvector -> Realvector` that computes a matrix-vector product.
2. (10pts) Write a function `rmmp :: Realmatrix -> Realmatrix -> Realmatrix` that computes a matrix-matrix product.
3. (10pts) Write a function `rmt :: Realmatrix -> Realmatrix` to compute the matrix transpose.

You may assume that the supplied vectors and matrices are compatible.

Use Haskell's predefined higher-order functions as much as possible.

Your program should be prepared as a literate Haskell script `hw33.lhs` containing, in addition to your program code, some descriptive text outlining your design decisions, and a detailed log of your interaction with `ghci`, showing a comprehensive test suite.

How to turn in

Submission instructions: see course mailing list.

Include the following statement with your submission, signed and dated:
I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual.