Homework 1 — Simple ML programs — assigned Wednesday 31 January — due Sunday 11 February

In all exercises, pay attention to program style. Avoid the temptation to write code in the Scheme style; use ML's pattern matching and function definition by clauses.

1.1 Simple ML programs (5pts) [A.1]

Write a function `innerproduct: (real * real * real) * (real * real * real)- > real` to compute the inner product of two real vectors in $\mathbf{R}^3$ represented as ML tuples.

1.2 Simple ML programs (5pts) [A.1]

Write a function `innerproduct: real list * real list -> real` to compute the inner product of two real vectors in $\mathbf{R}^n$ represented as ML lists.

1.3 Pragmatics of integer arithmetic (15pts) [A.3]

The factorial function may be defined as:

```
fun fact 0 = 1
  | fact n = n * fact (n-1)
```

which allows the binomial coefficient $\binom{n}{m}$ to be defined as:

```
fun binom (n, m) = fact n div (fact m * fact (n-m))
```

A shortcoming of this definition is that the computation may trigger integer overflow even when the final result fits in an integer. (We assume here that we are using SML/NJ, in which the type `int` has a finite range.)

1. (5pts) Experimentally determine the range of `int` integers in SML/NJ.
2. (10pts) Write a different definition of `binom` that avoids this problem.

1.4 Using lists for sets: writing recursive functions over lists (25pts) [A.1; K.2.2]

Let us use the ML type `int list` 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 an ML function `union: int list * int list -> int list` that takes two sets and returns their union.
2. (5pts) Write an ML function `intersection: int list * int list -> int list` that takes two sets and returns their intersection.

3. (5pts) Write an ML function `difference: int list * int list -> int list` that takes two sets and returns their set difference.

4. (5pts) Write an ML function `equal: int list * int list -> bool` that takes two sets and returns `true` if and only if the two sets are equal.

5. (5pts) Write an ML function `powerset: int list -> int list list` 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 ML type `int list list` 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.

1.5 Using lists for text (20pts) [K.2.2]

Write a function `split_into_words` to split English text into words. Punctuation characters (commas, semicolons, etc.) and white space characters (spaces, tabs, and new lines) are word separators. The resulting words should be free of word separators.

The function you write will replace the comment in the code fragment below, to make the whole work.

```
local
   fun acceptfile (fileName: string) : string =
      let
         val f = TextIO.openIn fileName
         val s = TextIO.inputAll f
         val _ = TextIO.closeIn f
      in
         s
      end
   type word = char list
   type sentence = char list
   (* your code goes here *)
   fun split_into_words (s: sentence): word list =
   (* begin your code *)

   (* end your code *)
in
   fun main (filename: string) =
      let
         val s = acceptfile filename
         val cl = explode s
         val wl = split_into_words cl
      in
         wl
      end
end
```

The declaration of `split_into_words` should be expressed in terms of the language primitives (i.e., without the use of specialized library functions such as `String.tokens`).

1.6 Drawing: writing recursive functions over lists; manipulating strings (30pts) [A.4; E; K.2.2]

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 ML 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 an ML function `makeCommand: (real * real) list list -> string`.

The result returned by `makeCommand` is an ML 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 string:

```
%!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 will be printed by a PostScript printer as in Figure 1.

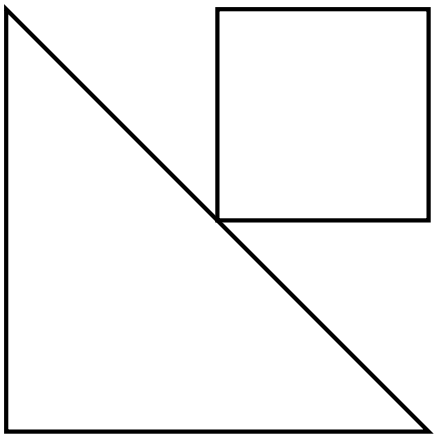

Figure 1: A triangle and a square.

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.

Implement a function `dumpStringToFile: string * string -> unit`, which takes two strings; the first is the name of the file to be written, and the second is the string that is to be dumped to the file. (Hint: Look at the preceding exercise, and consult the documentation of the Standard ML Basis Library.)

Now you can write your main function `makePictureFile: (real * real) list list * string -> unit` as follows:

```
fun makePictureFile (polygons: (real * real) list list, fileName: string): unit =
   dumpStringToFile (fileName, makeCommand polygons)
```

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.