CS 603 Exam 2 Spring 2008 Solution

1. Write an ML function, diagonal M, where M is a square matrix stored as a list of row lists. It should return a list of the main diagonal elements of M. Example: diagonal [[1,2,3],[4,5,6],[7,8,9]] returns [1,5,9].

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
fun diagonal [ ] = [ ]
  | diagonal (row1::other) = hd row1 :: diagonal (map tl other);
```

2. Write two ML functions, curry and uncurry, that convert between curried and uncurried forms of a binary function. Example:

```
val plus = curry op+;
plus 3 4;       (* returns 7 *)
val add = uncurry plus;
add (3,4);      (* returns 7 *)
```

```
fun curry f x y = f (x,y);
fun uncurry f (x,y) = f x y;
```

3. Write an ML function, scan f id L, where f is a binary function, id is the identity value, and L is a list. It should return a list of the values obtained by folding f across each possible prefix of L. Example: scan op+ 0 [2,3,5,7,11] returns [0,2,2+3,2+3+5,2+3+5+7,2+3+5+7+11] = [0,2,5,10,17,28]. You may assume that f is associative.

```
fun scan f id [ ] = [id]
  | scan f id (h::t) = id :: map (curry f h) (scan f id t);
```

4. Consider these ML definitions:

```
datatype Primitive = I of int | R of real | B of bool | C of char | S of string;
datatype RecList = P of Primitive | L of RecList list;
val mylist = L [P (I 2), P (R 3.14), L [P (B true), L [P (C #"w"), L [ ]], P (S "xyz")]];
        (* same as [2, 3.14, [true, [#"w", [ ]], "xyz"]] but satisfies type checking *)
```

   Write an ML function, flatten Z, that returns a flat list with the same primitive values as Z. Example: flatten mylist returns [I 2, R 3.14, B true, C #"w", S "xyz"].

```
fun flatten (P x) = [x]
 | flatten (L [ ]) = [ ]
 | flatten (L (h::t)) = flatten h @ flatten (L t);
```

5. Specify the polymorphic types of the ML functions from the previous problems.

   val 'a diagonal = fn : **'a list list -> 'a list**

   val ('a, 'b, 'c) curry = fn : **('a * 'b -> 'c) -> 'a -> 'b -> 'c**

   val ('a, 'b, 'c) uncurry = fn : **('a -> 'b -> 'c) -> 'a * 'b -> 'c**

   val ('a, 'b) scan = fn : **('a * 'b -> 'b) -> 'b -> 'a list -> 'b list**

   val flatten = fn : **RecList -> Primitive list**

6. Show the output values that are printed by this μSmalltalk code.

```
(val block1 [(set x (+ 1 (* 3 x)))])
(val block2 [(set x (div: x 2))])
(val block3 [(> x 1)])
(val block4 [(println x) (if (= (mod: x 2) 1) block1 block2)])
(val x 18)
(while block3 block4)
```

   18 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2

7. Write two μSmalltalk classes, Bag and Set, so that the client below will operate as shown. Note that a Bag allows duplicate values, but a Set does not. Minimize code via inheritance, but do not use μSmalltalk's predefined Collection hierarchy.

```
((class Node Object (value next)
   (method init:: (v n) (set value v) (set next n) self)
   (method value ( ) value)
   (method next ( ) next)
)

(class Bag Object (head size)
    (method init ( ) (set head nil) (set size 0) self)

    (method add: (v)
        (set size (+ size 1))
        (set head (init:: (new Node) v head))
    )

    (method size ( ) size)

    (method count: (v) (count:: self v head))

    (method count:: (v n)
        (if (isNil n) [0]
            [(+ (if (= v (value n)) [1] [0]) (count:: self v (next n)) )])
    )
)

(class Set Bag ( )
    (method add: (v)
        (ifTrue: (= (count: self v) 0) [(add: super v)] )
    )
)
```

```
(val b (init (new Bag)))
(add: b 10)
(add: b 20)
(add: b 10)
(size b)              ; returns 3
(count: b 10)         ; returns 2
(count: b 20)         ; returns 1
(count: b 30)         ; returns 0
```

```
(val s (init (new Set)))
(add: s 10)
(add: s 20)
(add: s 10)
(size s)              ; returns 2
(count: s 10)         ; returns 1
(count: s 20)         ; returns 1
(count: s 30)         ; returns 0
```

8. Using either C or C++ or Java, write definitions for Bag and Set abstract data types, represented as linked lists of ints. Provide these operations with the same functionality as in the preceding problem: add, size, count. Reuse code wherever possible.

```
// Java
class Node {
    int value;
    Node next;
    Node(int v, Node n) { value=v; next=n; }
}

class Bag {
    Node head;
    int size;

    Bag( ) { head=null; size=0; }

    void add(int v) {
        size++;
        head = new Node(v, head);
    }

    int size( ) { return size; }

    int count(int v) { return count(v, head); }

    int count(int v, Node n) {
        if (n==null) return 0;
        return ((v==n.value) ? 1 : 0) + count(v, n.next);
    }
}

class Set extends Bag {
    void add(int v) {
        if (count(v) == 0) super.add(v);
    }
}
```