CS 603 Exam 2 **Spring 2006** **Solution**

1. For each ML function definition below, write the type (usually a polymorphic type) that the ML interpreter would infer for the function. **[20 points]**

   a. fun f _ = [ ];

      fn : **'a -> 'b list**

   b. fun f (x,y) = (y,x);

      fn : **'a * 'b -> 'b * 'a**

   c. fun f (x,y,z) = [x,y,z];

      fn : **'a * 'a * 'a -> 'a list**

   d. fun f (w,x,y,z) = (w div x, y/z);

      fn : **int * int * real * real -> int * real**

   e. fun f x y = let val g = op^ in g(x,y) end;

      fn : **string -> string -> string**

   f. fun f (x,y,z) = z (y x);

      fn : **'a * ('a -> 'b) * ('b -> 'c) -> 'c**

   g. fun f x y z = x y z;

      fn : **('a -> 'b -> 'c) -> 'a -> 'b -> 'c**

   h. fun f w x y z = if w then x else y=z;

      fn : **bool -> bool -> ''a -> ''a -> bool**

   i. fun f (w,x) = let fun g (y,z) = y (y z) in g (w,x) end;

      fn : **('a -> 'a) * 'a -> 'a**

   j. fun f (u,[ ]) = [u] | f([ ],v) = [v] | f(w::x,y::z) = [w,y]::f(x,z);

      fn : **'a list * 'a list -> 'a list list**

2. Write an ML function inner(X,Y,f,g) that returns the inner product of the lists X and Y with respect to binary operations f and g. Assume that lists X and Y are nonempty lists of the same length, and that operation f is associative. You may use helper functions if you wish. For maximum credit, do not call the predefined hd or tl functions. Example: inner([1,2,3], [4,5,6], op+, op*) should return 32, because 1*4+2*5+3*6 = 32. This kind of inner product is often called a dot product. Another example: inner([1,2,3], [4,5,6], op*, op+) should return 315, because (1+4)*(2+5)*(3+6) = 315. **[10 points]**

```
fun inner ([h1],[h2],f,g) = g(h1,h2)
  | inner (h1::t1,h2::t2,f,g) = f(g(h1,h2),inner(t1,t2,f,g));
```

3. Write an ML function outer(X,Y,f) that returns the outer product of the lists X and Y with respect to binary operation f. You may use helper functions if you wish. For maximum credit, do not call the predefined hd or tl functions. The result is a list of length |X|, each of whose members is a sublist of length |Y|. The k^{th} element of the j^{th} sublist is obtained by applying operation f to the j^{th} element of X and the k^{th} element of Y. Example: outer([1,2,3], [4,5,6,7], op*) should return the list [[4,5,6,7],[8,10,12,14],[12,15,18,21]]. This kind of outer product is often called a multiplication table. Hints: Write a helper function that constructs one of the sublists. Also, consider using the map function. **[10 points]**

```
fun helper(a,Y,f) = map (fn b => f(a,b)) Y;

fun outer(X,Y,f) = map (fn a => helper(a,Y,f)) X;
```

OR

```
fun helper(a,[ ],f) = [ ]
  | helper(a,h::t,f) = f(a,h):: helper(a,t,f);

fun outer([ ],L,f) = [ ]
  | outer(h::t,L,f) = helper(h,L,f)::outer(t,L,f);
```

4. List five (or more) most significant ways that ML differs from Scheme. **[10 points]**

   Static typing using type inference.
   The val command always creates a new binding without destroying any old binding.
   Pattern matching.
   Syntax based on precedence/associativity rather than nested parentheses.
   Exception handling.
   Distinct type constructors for lists, tuples, and records (not all just S-expressions).
   Programmer-defined abstract datatypes.

5. List five (or more) most significant ways that Smalltalk differs from C++ and/or Java. **[10 points]**

   Pure OOP: everything is an object (no primitive type), uses dynamic method binding.
   Dynamic typing: no static type declarations.
   Subtypes based on protocols rather than inheritance.
   Implicit visibility rather than explicit public/private modifiers.
   Interpreted rather than compiled.
   No templates, method overloading, etc, because polymorphism is automatic.
   Blocks provide higher-order functions.
   No importing of libraries because all classes are always available.
   Syntax for message sends: unary, binary, or keyword-based.
   No multiple inheritance (as in C++) and no interfaces (as in Java).

6. In μSmalltalk, a message sent to a receiver named super is implemented as a message sent to self, except that method search does not begin in the receiver's class as it normally would. Two proposed implementations of super are described below. If these two implementations are equivalent, then explain why both implementations always produce the same result. Otherwise provide an example that shows how the two implementations can produce different results. Also state which, if either, of these two implementations is correct. **[10 points]**

   a. Method search begins in the superclass of the receiver's class.
   b. Method search begins in the superclass of the class that defines the method that sends the message to super.

(class A Object ( ) ** (method p ( ) true)** **)** **(class B A ( )** ** (method p ( ) false)** ** (method q ( ) (p super))** **)** **(class C B ( )** **)** **(val x (new C))** **(q x)**	**Using implementation b., message (q x) returns true, which is the correct answer.** **Using implementation a., message (q x) returns false, which is not correct.** **Implementation b. is correct, but implementation a. is not correct.**

7. First read this μSmalltalk code. Next determine the values that would be obtained upon sending each of the messages {p, q, r, s, t} to each of the receiver objects {b, c, d, e, f, g}. Write these values in the table below, where each row corresponds to a message and each column corresponds to a receiver. **[15 points]**

```
(class A Object ( )
        (method p ( ) (q self))
        (method q ( ) (r self))
        (method r ( ) (s self))
        (method s ( ) (t self))
        (method t ( ) 1)
)
(class B A ( )
        (method p ( ) (s self))
        (method r ( ) 2)
)
(class C B ( )
        (method q ( ) (p self))
        (method s ( ) 3)
)
(class D C ( )
        (method p ( ) 4)
        (method t ( ) (r self))
)
(class E D ( )
        (method q ( ) 5)
        (method s ( ) (t self))
)
(class F E ( )
        (method r ( ) (q self))
        (method t ( ) 6)
)
(class G F ( )
        (method p ( ) (q self))
        (method r ( ) (t self))
        (method s ( ) 7))

(val b (new B))
(val c (new C))
(val d (new D))
(val e (new E))
(val f (new F))
(val g (new G))
```

	b	c	d	e	f	g
p	**1**	**3**	**4**	**4**	**4**	**5**
q	**2**	**3**	**4**	**5**	**5**	**5**
r	**2**	**2**	**2**	**2**	**5**	**6**
s	**1**	**3**	**3**	**2**	**6**	**7**
t	**1**	**1**	**2**	**2**	**6**	**6**

8. Write a μSmalltalk class Queue such that each message send in the client would produce the output or return value as indicated below. **[20 points]**

```
(class Node Object (data next)
        (method data ( ) data)
        (method next ( ) next)
        (method data: (x) (set data x))
        (method next: (x) (set next x))
)


(class Queue Object (front rear)
        (method isEmpty ( ) (isNil front))
        (method enqueue: (x) (locals z)
                (set z (new Node))
                (data: z x)
                (if (isEmpty self)
                        [(set front z)]
                        [(next: rear z)])
                (set rear z)
                x)
        (method dequeue ( ) (locals x)
                (if (isEmpty self) [nil]
                        [(set x (data front))
                         (set front (next front))
                         x]))
        (method display ( ) (locals z)
                (set z front)
                (while [(notNil z)]
                        [(println (data z))
                         (set z (next z))]))
)


(val Q (new Queue))
(isEmpty Q)             ; <True>
(enqueue: Q #a)         ; a
(enqueue: Q #b)         ; b
(enqueue: Q #c)         ; c
(isEmpty Q)             ; <False>
(display Q)             ; a   b   c
(dequeue Q)             ; a
(dequeue Q)             ; b
(dequeue Q)             ; c
(isEmpty Q)             ; <True>
(dequeue Q)             ; nil
```