CS 603 Exam 2 **Spring 2007** **Name _____**

1. Explain the differences between (i) imperative programming, (ii) object-oriented programming, and (iii) functional programming. Specify the relevant features that are characteristic of each paradigm, that is, the features that must be present in any programming language that fully supports the paradigm. **[9 points]**

2. Describe how class variables and class methods are different from instance variables and instance methods in an object-oriented programming language such as Smalltalk, C++, or Java. **[8 points]**

3. *Pass-by-need* is a kind of parameter passing such that evaluation of each actual parameter is delayed until the value of the formal parameter is needed. If and when this evaluation does occur, the result will be saved for any subsequent evaluations of the formal parameter, so that each actual parameter is evaluated at most once. Write an example in ML or μScheme such that the result would be different if using pass-by-need rather than pass-by-value, and state the result in each case. **[7 points]**

4. Write a suffixes function in ML that works as follows:
suffixes [10, 20, 30, 40] returns [[10, 20, 30, 40], [20, 30, 40], [30, 40], [40], [ ]].

   a. First write suffixes using recursion and pattern matching. Do not use map, foldr, or foldl. **[6 points]**

   b. Next write suffixes using map, foldr, and/or foldl. Do not use any explicit recursion. **[6 points]**

   c. Specify the polymorphic type of the suffixes function. **[3 points]**

5. Write a filter function in ML that works as follows:
filter (fn x => x>0, [3, ~2, 5, 0, 6, ~4, 7]) returns [3, 5, 6, 7].

   a. First write filter using recursion and pattern matching. Do not use map, foldr, or foldl. **[6 points]**

   b. Next write filter using map, foldr, and/or foldl. Do not use any explicit recursion. **[6 points]**

   c. Specify the polymorphic type of the filter function. **[3 points]**

6. Write a superExp function in ML, such that superExp(x, k) computes x^{2^k}.
Example: superExp(3.0, 4) returns $3.0^{2^4} = 3.0^{16} = 43046721.0$. Assume that $k \geq 0$. For full credit, make only one recursive call within your function, and do not use any helper functions. Also, your function should be efficient, that is, superExp(x, k) should perform a total of only k multiplication steps.
Hints: think of x^{2^k} as $(\ldots(x^2)^2\ldots)^2$, and use a let expression.

   a. Write the superExp function. **[9 points]**

   b. Specify the type of the superExp function. **[2 points]**

7. Write a compare function in ML that works as follows:
compare op<= [1,2,5] [1,3,4] returns [Equal, Less, Greater].
compare (fn ((a,b),(c,d)) => a<=c andalso b<=d)
[(1,2), (3,5), (4,6), (7,8)] [(1,2), (4,6), (3,5), (8,7)]
returns [Equal, Less, Greater, Incompatible].

   a. First write a datatype that is needed by the compare function. **[3 points]**

   b. Next write the compare function. **[12 points]**

   c. Specify the polymorphic type of the compare function. **[5 points]**

8. Write a µSmalltalk class called Interval such that each instance represents a continuous finite range on the real number line. Here is a client of class Interval which illustrates how the class Interval should operate. **[25 points]**

```
(val X (left:right: Interval 0 4))      ; X is the interval 0 … 4
(val Y (left:right: Interval 2 7))      ; Y is the interval 2 … 7
(val Z (left:right: Interval 5 11))     ; Z is the interval 5 … 11
(length X)                              ; returns 4
(length Y)                              ; 5
(length Z)                              ; 6
(disjoint: X Y)                         ; <False>
(disjoint: X Z)                         ; <True>
(disjoint: Y X)                         ; <False>
(disjoint: Y Z)                         ; <False>
(disjoint: Z X)                         ; <True>
(disjoint: Z Y)                         ; <False>
(intersects: X Y)                       ; <True>
(intersects: X Z)                       ; <False>
(intersects: Y X)                       ; <True>
(intersects: Y Z)                       ; <True>
(intersects: Z X)                       ; <False>
(intersects: Z Y)                       ; <True>
```