CS 603 Exam 1 **Spring 2007** **Name ______________**

1. List all the strings with length exactly 8 that can be generated using this grammar: S → SaSbS | ε. Write these strings in the indicated boxes. **[14 points]**

Is the above grammar ambiguous, or is it unambiguous? Justify your answer. **[6 points]**

2. A *perfect number* is an integer that equals the sum of its proper positive divisors. Thus 6 is perfect because 1 + 2 + 3 = 6, and 28 is perfect because 1 + 2 + 4 + 7 + 14 = 28. Write a recursive Impcore function (perfect m) that returns 1 when m is perfect, but that returns 0 otherwise. You may write helper functions, and you may call functions provided in the text such as mod. But do not use while or set expressions. **[10 points]**

3. Suppose Impcore provides a new built-in expression (switch e_0 v_1 e_1 v_2 e_2 … v_n e_n) with the following meaning. First e_0 is evaluated. If the result is v_1, then evaluate e_1; else if it is v_2, then evaluate e_2; …; else if it is v_n, then evaluate e_n; else return the default value 0. Complete these Impcore-style natural operational semantics rules corresponding to the AST node SWITCH(e_0,v_1,e_1,v_2,e_2,…,v_n,e_n). **[15 points]**

   The first rule should handle the k^{th} case, for arbitrary k in the range from 1 to n:

   $\langle e_0, \xi, \phi, \rho\rangle \Downarrow$

   $$\frac{\qquad\qquad\qquad\qquad\qquad\qquad}{\langle \text{SWITCH}(e_0,v_1,e_1,\ldots,v_n,e_n), \xi, \phi, \rho\rangle \Downarrow} \quad \text{(SWITCH\_kTH\_CASE)}$$

   The second rule should handle the default case:

   $\langle e_0, \xi, \phi, \rho\rangle \Downarrow$

   $$\frac{\qquad\qquad\qquad\qquad\qquad\qquad}{\langle \text{SWITCH}(e_0,v_1,e_1,\ldots,v_n,e_n), \xi, \phi, \rho\rangle \Downarrow} \quad \text{(SWITCH\_DEFAULT)}$$

4. Draw a diagram that illustrates the internal representation of this S-expression:
   (1 (2 (3 4)) (( )) ((5 6) 7 . 8)) **[10 points]**

5. Implement a sum_of_products function in Scheme, as described below. You may write helper functions. But do not use while or set expressions.

   Example: (sum_of_products '((1 2 3 4) (5 6 7) (8 9))) returns 306, because 1*2*3*4 + 5*6*7 + 8*9 = 24 + 210 + 72 = 306.

   Example: (sum_of_products '((5 10) (15))) returns 65, because 5*10 + 15 = 65.

   a. First write the sum_of_products function using recursion. Do not use map, foldr, or foldl. **[10 points]**

   b. Next write the sum_of_products function using map, foldr, and/or foldl. Do not use any explicit recursion. **[10 points]**

6. Explain what unusual thing happens when each of these Scheme expressions is evaluated.

   a. ((lambda (x) (x x)) (lambda (x) (x x))) **[7 points]**

b. ((lambda (x) (list2 x (list2 (quote quote) x)))
(quote (lambda (x) (list2 x (list2 (quote quote) x))))) **[7 points]**

7. Scheme provides both let and let* for defining local variables. Consider this example:

```
-> (val a 10)
10
-> (let ((a 20) (b (* a a)))  (+ b 1))
101
-> (let* ((a 20) (b (* a a)))  (+ b 1))
401
```

With let, variable b gets value 10*10 = 100. With let*, variable b gets value 20*20 = 400.

a. First show that each use of let* can be replaced by uses of let. That is, write an expression using let that is equivalent to (let* ((x_1 e_1) (x_2 e_2) … (x_n e_n)) e). Hint: consider the above example. **[8 points]**

b. Next show that each use of let can be replaced by uses of let*. That is, write an expression using let* that is equivalent to (let ((x_1 e_1) (x_2 e_2) … (x_n e_n)) e). Hint: consider the above example. **[8 points]**