Programming Languages (CS 329) **Homework 1**
Computer Science and Engineering **Out: 2000-08-17**
Indian Institute of Technology Bombay **Due: 2000-09-07**

1. How many α-distinct reduction sequences are there for the expression

$$\texttt{((}\lambda\texttt{ a ((}\lambda\texttt{ b c) a)) (}\lambda\texttt{ d (e d)))}? \tag{1}$$

   Enumerate as many as you can.

2. Draw the parse tree of the following expressions. Reduce to normal form or state that a normal form does not exist.

   (a)
$$\texttt{((}\lambda\texttt{ y (y y)) (}\lambda\texttt{ x x))} \tag{2}$$

   (b)
$$\texttt{(y (}\lambda\texttt{ y (y (}\lambda\texttt{ x (x y)))))} \tag{3}$$

3. With Church numerals and associated operations as described in class, verify that

   (a) (INC (INC $\bar{2}$)) $\equiv \bar{4}$.

   (b) For any Church numeral $\bar{n}$, (DEC (INC $\bar{n}$)) $\equiv \bar{n}$.

4. Give an informal recipe to use an applicative order evaluator to implement a normal order evaluator. I.e., you have code which terminates under normal order but may hang when run under applicative order. How can you syntactically modify your code to run safely on an applicative order interpreter?

5. In class we defined a normal order Y combinator as

$$Y' \equiv \texttt{(}\lambda\texttt{ y (}\lambda\texttt{ t (t ((y y) t))))} \tag{4}$$
$$Y \equiv (Y'\ Y'), \tag{5}$$

   which works for normal order interpreters.

   (a) Why does Y not perform as intended for applicative order interpreters?

   (b) Using the answer to the previous question, turn Y into an applicative order Y combinator.

   (c) Compare your applicative order Y with the one given in the Friedman et al book (page 116):

```
(lambda f
  ( (lambda x (f (lambda y ((x x) y))) )
    (lambda x (f (lambda y ((x x) y))) ) )   )
```

   Are they equivalent?

6. Does the Scheme evaluator follow applicative or normal order? Write the simplest expression you can to figure out. Turn in the code.

7. Use the applicative order Y designed above to implement two functions without using recursion in Scheme: factorial of a positive integer and length of a list.

8. Write a simple Scheme function which, when given a Church numeral, will output a regular integer suitable for printing. Write in Scheme the exponentiation routine `EXP` which takes two Church numerals $\bar{m}$ and $\bar{n}$ and outputs $\overline{m^n}$. Verify the output to test cases using the transormation from Church numerals to integers.