CMSC330 Spring 2010 Quiz #3

Name ________________________________

Discussion Time (circle one):    9am    10am    11am    12pm    1pm    2pm

Instructions

• You have 20 minutes for this quiz.
• This is a closed book exam. No notes or other aids are allowed.
• For partial credit, show all of your work and clearly indicate your answers.
• Write neatly. Credit cannot be given for illegible answers.

1. (10 pts) Lambda calculus
   (2 pts each) Use beta-reduction to fully evaluate the following λ-expressions.
   a. \((λx.x) (λy.y) (λz.z)\)

   b. \((λx.λy.x) y\)

   c. (6 pts) Show that the Church numeral encoding of 1*1 evaluates, after 0 or more beta reductions, to the encoding for 1. The encodings are in the box below. In your reduction sequence you may write ~ whenever you replace a definition from the box with its right-hand-side, or vice-versa. Write => when you perform a beta reduction. Here is a bit to get you started:

   \[
   \begin{align*}
   1 * 1 & \sim \\
   \lambda x. (1 (1 x)) & \sim \\
   \lambda x. (\lambda f. \lambda y. f y) (\lambda x. x) & \Rightarrow \\
   \lambda x. (\lambda y. x) & \Rightarrow
   \end{align*}
   \]

   \[
   \begin{align*}
   M * N & = \lambda x. (M (N x)) \\
   0 & = \lambda f. \lambda y. y \\
   1 & = \lambda f. \lambda y. f y \\
   2 & = \lambda f. \lambda y. f (f y) \\
   3 & = \lambda f. \lambda y. f (f (f y))
   \end{align*}
   \]
2. (10 pts) Multithreading
   Consider the following attempt to implement producer/consumer pattern w/ Java 1.4.

   ```java
   class Buffer {
   Buffer ( ) {
      Object buf = null;
      boolean empty = true;
   }
   void produce(o) {
      synchronize (buf) {
         while (!empty) wait( );
      }
      synchronize (buf) {
         empty = false;
         notifyAll( );
         buf = o;
      }
   }
   Object consume( ) {
      synchronize (buf) {
         while (empty) wait( );
      }
      synchronize (buf) {
         empty = true;
         notifyAll( );
         return buf; // also releases lock
      }
   }
   }
   t1 = Thread.run { produce(1); }
   t2 = Thread.run { produce(2); }
   t3 = Thread.run { x = consume( ); }
   t4 = Thread.run { y = consume( ); }
   }
   (We are abusing the thread creation notation here, and the call to produce() is not type correct---
   assume produce(1) is like produce(new Integer(1)).) In the following, give schedules as a list
   of thread name/line number/range pairs, e.g., (t1, 1-4), (t2, 1), (t3, 5-8) …
   a. (2 pts) Give a schedule under which x = 1 and y = 2.
   b. (2 pts) Give a schedule under which x = 2 and y = 1.
   c. (3 pts) Give a schedule under which x = 1 and y = 1, or argue that no such
      schedule is possible.
   d. (3 pts) Give a schedule under which x = 2 and thread 4 never terminates.
   ```