Written Assignment 4
Due October 23, 2017 at 12:30pm

You may discuss this assignment with other students and work on the problems together. However, your write-up should be your own individual work and you must acknowledge the students you work with. Written assignments must be turned in at the start of lecture on the indicated due date.

1. (20 points) After sifting through the hundredths of mails you get everyday from programmers after releasing your L interpreter on the web, you decide to add a three-way branching construct to L. Specifically, you want a construct of the following form:

   ```
   branch x { e1 e2 e3 }
   ```

   This construct evaluates to `e1` if x is less than 0, to `e2` if x is equal to zero and to `e3` otherwise.

   (a) (10 points) Give large-step operational semantics for this construct
   (b) (10 points) Give small-step operational semantics for this construct

2. (20 points) Consider adding a lazy llet statement to the L language. Specifically, you want such a construct to only get evaluated at the point where the defined name it is used. (Read up in the lecture notes on lazy lets before attempting this question.) Here are some examples together with the expected output:

   ```
   llet x = 3 + "Duck" in 0
   ```

   evaluates to 0 with no error.

   ```
   llet x = print "cs345H" in x + x
   ```

   evaluates to 0 with no error and prints "cs345H" *twice*.

   (a) (10 points) Give large-step operational semantics for this construct
   (b) (10 points) Give small-step operational semantics for this construct

3. (20 points) Consider the following language:

   $$S \quad \rightarrow \quad \textit{integer constant} \mid S_1 * S_2 \mid S_1 / S_2 \mid \textit{let } x = S_1 \textit{ in } S_2$$

   Assume that the only run-time error occurs from division by zero (i.e., S_2 is zero in a division)

   (a) (6 points) List three programs in this language
   (b) (14 points) Give the natural large-step operational semantics for this language that "get stuck" if a program divides zero.
   (c) (15 points) Using the types `zero` and `non-zero`, give sound typing rules for this language that prevent all run-time errors without being overly restrictive.
   (d) (10 points) Proof preservation of your type system with respect to your operational semantics
   (e) (10 points) Proof progress of your type system with respect to your operational semantics