Flolac 2012
Intro. to Type Systems
Partial solutions to Assignment 2

1. **(20%)** Perform full beta rduction on the following lambda term:
   - **a)** (λ x . x (x y)) (λz. z)

     →* y

   - **b)** (λx. (λz. z) x x) (λx. (λz. z) x x)

     →

     → (λx. (λz. z) x x) (λx. (λz. z) x x) --back to itself

     →* non-terminating

2. **(40%)** Please give the *type derivations* (proof trees) for the following Mini-Haskell expressions. You should try to derive the most general type for them.

   (a) **let id = \x -> x in id id**

   (b) **\f -> f (\x -> x)**

   Ans:

```
        f: (a->a)->b   |-- f : (a-> a)->b
          f: (a->a)->b   |-- \x->x : a->a    (omit some steps)
     -------------------------------------------------------
          f: (a->a)->b   |-- f (\x->x) : b
      ----------------------------------------------------------
              |--   \f -> f (\x -> x) : (a->a)->b->b
```

   (c) **\x-> let f = \y -> x in (f 1, f True)**

   Ans:

```
             x:a. y:b |-- x:a
      ------------------------------
           x:a |-- \y->x : b->a      Gen([x:a], b->a) = \forall b. b->a
      x:a. f: \forall b. b->a |-- f: Int->a      x:a. f: \forall b. b->a |-- f: Bool->a
      x:a. f: \forall b. b->a |-- 1: Int         x:a. f: \forall b. b->a |-- True:Bool
      ----------------------------------------   ----------------------------------------
      x:a. f: \forall b. b->a |-- f 1: a           x:a. f: \forall b. b->a |-- f True: a
      ------------------------------------------------------------------------------------
              x:a. f: \forall b. b->a |-- (f 1, f True) : (a, a)
      ------------------------------------------------------------------
           x:a |-- let f = \y -> x in (f 1, f True) : (a, a)
      ----------------------------------------------------------------
           |-- \x-> let f = \y -> x in (f 1, f True) : a->(a,a)
```

3. (20%) Mini-Haskell does not support recursive function definitions! One way to extend Haskell with recursive functions is to add a new form of function declaration as follows:

E ::= …

| letrec f = E1 in E2 --**E1** may contain a reference(s) to **f**

Ans:

$$\frac{TE + [f : \tau_1] \vdash E1 : \tau_1 \qquad TE + [f : Gen(TE, \tau_1)] \vdash E2 : \tau_2}{TE \vdash (\text{letrec } f = E1 \text{ in } E2) : \tau_2}$$

4. **(20%)** The canonical non-terminating computation, *(λx.xx) (λx.xx)*, was not expressible in the simply-typed λ-calculus. Neither was the self-application fragment, *self* ═ *λx.xx.* But *self* is indeed typable in the polymorphic lambda calculus. Please re-write *self* as a PLC expression.

Ans: One possible solution—let $\Theta = \forall \alpha .\ \alpha \rightarrow \alpha$

self = (λx: Θ **. x**Θ **x) and its type is** $\Theta \rightarrow \Theta$