Advanced Programming Languages	**Problem Set 2**
CS 6110 Spring 2015	**Due: Weds. Feb 25, 2015**

Problem Set 2

Exercises

1. Write a primitive recursive function $is\_sqr(x)$ which returns the value 0 if x is a perfect square and 1 otherwise, e.g. $is\_sqr(9) = 0$, $is\_sqr(5) = 1$

   Be as elegant as you can.

2. Higher order primitive recursion. We can define a sensible notion of a "function primitive recursive in f" for f an arbitrary general function. For example,

   $$h(0, f) = f(0)$$
   $$h(S(x), f) = f(h(x, f))$$

   Is there a "primitive recursive function in f" that computes $\mu y.(f(y) = 0)$? Explain your answer.

3. Give an example of a program than runs faster if ran lazily, and one that runs faster if ran eagerly.

4. Give an "interesting" example of a program that uses dynamic scoping to modify its behavior at runtime (using the Y combinator for example, or using a fix that computes as follows $fix(F) \rightarrow F(fix(F))$).

5. Background. The primitive recursive functions are a simple well known class of computable functions defined by a programming language. We introduced them using Kleene's definition of an inductively defined class of functions. Later we will see the Loop language as a very natural imperative PL whose class of definable functions is exactly the primitive recursive functions.

We call programming languages that define a subclass of the total computable functions *subrecursive*. The CoqPL is subrecursive. Another natural subrecursive class are the *elementary functions*. An interesting issue is whether there is a subrecursive PL for the *polynomial time computable* functions. There are several, and they are more complex to define. We obtain a nice definition by introducing step counting in subrecursive languages. We will investigate this topic later.

The goal of this problem is to describe how is to restrict the lambda calculus so that it defines only primitive recursive functions. We will assume an intuitively clear notations for natural numbers, but these are not practical notations for feasible computations. Nevertheless, Coq uses them for their logical simplicity.

(a) Write the *design requirements* for a λ-calculus based definition of primitive recursive functions. A language computing all and only primitive recursive functions. Have it use fix as a primitive operator. Assume numerical terms, 0, $S(0)$,... Add a primitive conditional testing for 0 term $cond(n; \_\_; \_\_)$.

(b) Show potential definitions of addition and multiplication in a language of this kind.