The objective of this exam is to test your undertanding of weeks 5 and 6 of the CIS 194 Spring 2013 course (parametricity, type classes, and lazy evaluation).

Name: _____

1. Consider the following function:

```
sd :: Eq b => (a -> b) -> a -> b -> Bool
sd = ?
```

How would you define `sd` taking into account its type signature?

```
sd f x y =
```

2. Consider the following function:

```
g :: ?
g x y = x + 1 > y
```

What is the type of `g`?

A. `Ord a => a -> a -> Bool`

B. `(Eq a, Num a) => a -> a -> Bool`

C. `(Eq a, Ord a) => a -> a -> Bool`

D. `(Num a, Ord a) => a -> a -> Bool`

3. Given the following functions:

```
iterate :: (a -> a) -> a -> [a]
iterate f x = x : iterate f (f x)

take :: Int -> [a] -> [a]
take n _      | n <= 0 = []
take _ []              = []
take n (x:xs)          = x : take (n - 1) xs
```

Can you complete the step-by-step evaluation of `take 3 (iterate (+ 1) 0)`?

```
  take 3 (iterate (+ 1) 0)
= take 3 (0 : iterate (+ 1) (0 + 1))
= 0 : take (3 - 1) (iterate (+ 1) (0 + 1))
= 0 : take 2 (iterate (+ 1) (0 + 1))
=
```

4. A stream represents a list that must be infinite:

```
data Stream a = Cons a (Stream a) deriving Show
```

We can define the ruler function

$$0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0, 4, 0, 1, 0, 2, ...$$

as follows:

```
ruler :: Stream Integer
ruler = ruler' 0
  where
    ruler' x = interleave (repeat x) (ruler' (x + 1))
```

Here, the `repeat` function is defined as:

```
repeat :: a -> Stream a
repeat x = Cons x (repeat x)
```

And `interleave` could be defined as either:

```
interleaveA :: Stream a -> Stream a -> Stream a
interleaveA (Cons x xs) (Cons y ys) = Cons x (Cons y (interleave ys xs))
```

Or:

```
interleaveB :: Stream a -> Stream a -> Stream a
interleaveB (Cons x xs) ys = Cons x (interleave ys xs)
```

What is the difference between using `interleaveA` or `interleaveB` in `ruler`?