Fall, 1997 Name: ______________________________

Com S 541 — Programming Languages 1

Test on Functional (and a bit of Logic) Programming Languages

Special Directions for this Test

This test has 10 questions and pages numbered 1 through 8.
This test is open book and notes.
Before you begin, please take a moment to look over the entire test so that you can budget your time.
When you write Haskell functions on this test, you may use anything in the Haskell Prelude without writing it in your test. You are encouraged to define functions not specifically asked for if they are useful to your programming; if they are not in the Prelude, write them into your test.

1. (10 points) Define a Haskell function.

```
> subseteq :: (Eq a) => [a] -> [a] -> Bool
```

such that `subseteq s1 s2` is true just when all elements of `s1` are elements of `s2`. The following are examples.

```
subseteq [] [3,4] = True
subseteq [3,4] [] = False
subseteq [3,4,4,3] [4,3] = True
subseteq [3,4,4,3] [3,4,1] = True
```

2. (10 points) Define a Haskell function

```
> setProduct :: (Eq a, Eq b) => [a] -> [b] -> [(a,b)]
```

that takes two lists, `as` and `bs`, and returns the list of all pairs of elements, with the first element of each pair from `as` and the second from `bs`. For example

```
setProduct ([]::[Char]) [1,3]
    = []
setProduct [2, 7, 10] [1, 3]
    = [(2,1), (2,3), (7,1), (7,3), (10,1), (10,3)]
setProduct ['a', 'b', 'c'] [17, 21, 3, 4]
    = [('a',17), ('a',21), ('a',3), ('a',4),
       ('b',17), ('b',21), ('b',3), ('b',4),
       ('c',17), ('c',21), ('c',3), ('c',4)]
```

3. (15 points) An instrument in a factory emits an infinite list of temperatures. These temperatures are in degrees Kelvin, and have the type `Int`. The temperature readings may vary in an unsettled fashion; however, they are reliable when 3 consecutive readings are the same number and when they are between 273 and 373 (inclusive). For the convenience of human operators, these readings should be translated to degrees Celsius, by subtracting 273 from each reading. Write a Haskell function

```
> reliableCelsius :: [Int] -> [Int]
```

that takes the infinite list of readings in degrees Kelvin, and produces from it an infinite list of readings in degrees Celsius. For example:

```
reliableCelsius ([273, 273, 273, -3, 274, 274, 274] ++ [275, 275 ..])
   = [0, 1] ++ [2, 2 ..]
reliableCelsius ([273, 273, 273, 273, -3, 274] ++ [275, 275 ..])
   = [0, 0] ++ [2, 2 ..]
reliableCelsius ([5, 5, 5, 7, 300, 300, 300, -4, -10] ++ [373, 373 ..])
   = [27] ++ [100, 100 ..]
```

4. Consider the Haskell type and data definitions, which can be thought of as the abstract syntax for Prolog terms.

```
> type Id       = (Int,String)
> type Atom     = String
> data Term     = Var Id | Struct Atom [Term] deriving Eq
```

(a) (10 points) Write a function

```
> varsIn :: Term -> [Id]
```

that returns a list containing each `Id` that appears in the clause (duplicates are allowed). For example:

```
varsIn (Var (0,"x"))
    = [(0,"x")]
varsIn (Struct "foo" [Struct "bar" [], Var (0,"z"), Struct "baz" [Var (0,"y")]])
    = [(0,"z"), (0,"y")]
```

(b) (10 points) Write a function

```
> renameVars :: Int -> Term -> Term
```

that takes an integer `lev` and a term `t`, and returns a term that is just like `t`, except that every `Id` of the form `(i,s)` in `t`, is changed to the form `(lev,s)`. For example:

```
renameVars 3 (Var (0,"x"))
    = Var (3,"x")
renameVars 9 (Struct "foo" [Struct "bar" [],
                            Var (0,"z"),
                            Struct "baz" [Var (0,"y")]])
    = Struct "foo" [Struct "bar" [], Var (9,"z"), Struct "baz" [Var (9,"y")]]
```

5. (20 points) A *fuzzy set* is a generalization of the notion of a set in which membership is given by a degree, which is a real number between 0.0 to 1.0 (inclusive). Fuzzy sets can be used to describe imprecise knowledge or measurement. For example, we might say that Gary's membership degree in the fuzzy set of professional cooks is 0.0, in the fuzzy set of fat people is 0.3, in the fuzzy set of tall people is 0.6, and in the fuzzy set of computer science professors is 1.0.

   One way to describe a fuzzy set is with a characteristic function that gives the membership degree of each element. For example, the following Haskell code defines such a characteristic function for the fuzzy set of the "mature" ages of people.

```
> mature age = if age <= 19 then max 0 (1 - 0.5 * (19 - age)) else 1.0
```

   This says that the membership degree in the fuzzy set of "mature" people's ages is function of the age. For example, at age 17 the degree is 0.0, at age 18 it's 0.5, at age 18.5 it's 0.75, and at 19 it's 1.0. (Thus, age 18 is somewhat mature, and 18.5 is more mature, but not fully mature, according to this definition.)

   As another example, one can represent measurements of a point on some scale (of say, temperature) with a fuzzy set. The following is useful

```
> mymeasurement = fuzzyPoint 0.6 40
> fuzzyPoint c x = (\y -> max 0 (1 - c * abs(y - x)))
```

   So `mymeasurement 7` is 0.0, and thus the membership degree of `7` in mymeasurement is 0.0. For the fuzzy set `mymeasurement`, the membership degree of 39 is 0.4, and it's 0.7 for 39.5, 1.0 for 40, and 0.7 for 40.5.

   In this problem, you are to, in Haskell,

   (a) define a polymorphic type generator `FuzzySet`,

   (b) define a function

```
> fuzzySetSuchThat :: (a -> Double) -> FuzzySet a
```

   so that `fuzzySetSuchThat f` is a fuzzy set with membership degree given by the function `f`.

(c) define a function

```
> membershipDegree :: FuzzySet a -> a -> Double
```

that takes a fuzzy set of some type `a` and an element `x` of type `a`, and returns the membership degree of `x`, which should be a number between 0.0 and 1.0 (inclusive).

(d) define a function

```
> fuzzyUnion :: FuzzySet a -> FuzzySet a -> FuzzySet a
```

that takes two fuzzy sets `s1` and `s2` and returns a fuzzy set `s`, such that for any `x` of type `a`, the membership degree of `x` is the maximum of the membership degrees of `x` in `s1` and `s2` so that `fuzzySetSuchThat f` is a fuzzy set with membership degree given by the function `f`.

As examples, consider the following, with `fuzzyPoint` as defined above.

```
membershipDegree (fuzzySetSuchThat (fuzzyPoint 0.6 40)) 40
   = 1.0
membershipDegree (fuzzySetSuchThat (fuzzyPoint 0.6 40)) 40.5
   = 0.7
let s1 = (fuzzySetSuchThat (fuzzyPoint 0.6 40))
    s2 = (fuzzySetSuchThat (fuzzyPoint 0.6 42))
in membershipDegree (fuzzyUnion s1 s2) 40.5
  = 0.7
```

6. (10 points) Explain how Haskell's type classes are different from Smalltalk's classes.

7. (10 points) What is the difference between the pattern matching in Haskell and that in λProlog? Give an example.

8. (10 points) Which language helps achieve more reuse of code: Haskell, Smalltalk, or λProlog? Pick just one language and explain your answer.

9. (10 points) Which language would be better for rapid prototyping: Haskell, Smalltalk, or λProlog? Pick just one language and explain your answer.

10. (10 points) Specify the relation `subseteq` in λProlog such that

```
type subseteq  (list T) -> (list T) -> o.
```

and `subseteq L1 L2` holds if each element of `L1` is an element of `L2`. (Write any auxiliary relations that you need.)