CS 457/557 Homework 4 – due 2pm, Tuesday, Oct. 25, 2005

Hand in your solutions on paper *and* email them to `cs457acc@cs.pdx.edu`. All the programs should be placed in a single `.hs` file, which can be the body of your email message or an attachment. It is *not* necessary to show evidence that you have loaded and tested your programs, but this is of course the only sensible way to make sure that you have found correct answers!

1. Write a function

```
subStrings :: String -> [String]
```

that returns a list containing all the substrings of its argument. For example, the substrings of `"abc"` are `"abc"`, `"ab"`, `"bc"`, `"a"`, `"b"`, `"c"`, and `""`. Each distinct substring must appear exactly once in the result list, but the order does not matter.

(But, for extra credit, write a version whose result lists the substrings in decreasing order by length, e.g. for input `"abc"`, output should be in the order listed above.)

2. Newton's method says we can calculate $\sqrt{x}$ by taking the limit of the following sequence of approximations: $a_0 = 1.0, a_1 = (a_0 + x/a_0)/2.0, \ldots, a_n = (a_{n-1} + x/a_{n-1})/2.0, \ldots$. Implement a function

```
squareRoot :: Float -> Float
```

that returns the best possible approximation to the square root of its argument using Newton's method. (Note: You can stop taking approximations when the difference between successive terms is less than ϵ, for a suitable small value of ϵ.)

3. Do Hudak Exercise 7.1. Hint: Give your function the type

```
foldTree :: (a -> a -> a) -> (b -> a) -> Tree b -> a
```

4. Do Hudak Exercise 7.4, using `InternalTree`.

5. Do Hudak Exercise 7.5. You'll want to extract and modify the existing code from `Trees.lhs` (rather than trying to import it). Hint: Define your revised version of `evaluate` in terms of an auxiliary function

```
evaluate' :: [(String,Float)] -> Expr -> Float
```

where the first argument is a list of variable bindings to be used in evaluating the second argument. For example,

```
evaluate' [("x", 1.0),("y",2.0)] (V "x" :+ V "y")
```

should yield `3.0`. You may find the function `Prelude.lookup` to be useful.

6. Do Hudak Exercises 8.5 and 8.6. You'll want to extract and modify the existing code from `Region.lhs` (rather than trying to import it). Assume that the list of vertices passed to `polygon` is in counter-clockwise order.

7. Just as a set containing elements of type `a` can be represented by a function of type

```
a -> Bool
```

so a dictionary (finite map) with keys of type `k` and values of type `v` can be represented by a function of type

```
k -> Maybe v
```

Suppose we want to define an abstract data type of such dictionaries. Complete the following implementation by giving definitions of `find` and `insert` . (Hint: Let the types be your guide!)

```
type Dict a b = a -> Maybe b
empty :: Dict a b
empty a = Nothing
find :: Dict a b -> a -> Maybe b
insert :: Eq a => Dict a b -> a -> b -> Dict a b
```