____
NAME

CSci 555: Functional Programming
Fall 2010, Examination #1

1. **(8 points)** Match language paradigm terms from the following list with the statements given below. For each statement write the letter of the *best* matching term in the blank. *A term may be used more than once.*

A.	dysfunctional	B.	assertive	C.	declarative	D.	referential
E.	imperative	F.	implicative	G.	opaque	H.	transparent

___ Any needed program state must be handled explicitly (no implicit state)
___ Programs made up of statements that are sequenced, denoting a corresponding sequence of actions
___ Repetitive execution is accomplished by recursion
___ Programs express *how* something is to be computed

2. **(20 points)** The following table has columns of Haskell parameter patterns and corresponding arguments. As shown in the example line, indicate whether each match succeeds or fails and, for successful matches, indicate the bindings of values to identifiers.

Pattern	**Argument**	**Succeeds (Yes/No)**	**Bindings**
`x`	`0`	yes	`x` ← `0`
`x`	`[ ]`		
`[x]`	`[ ]`		
`x`	`[1,2,3]`		
`[x]`	`[1,2,3]`		
`(x:y)`	`[ ]`		
`(x,y)`	`[4,3]`		
`(w:x@(y:z))`	`[2,4,6]`		
`((x,y):z)`	`[(1,7)]`		
`(x:_:y)`	`"hugs"`		
`((w:x):y:z)`	`["one", "bites", "the dust"]`		

3. **(15 points)** Show appropriate polymorphic type declarations for the following Haskell functions.

   (a) `head` as in the standard prelude

   (b) `++` (append) as in the *Notes*

   (c) `filter` as in the *Notes*

   (d) functional composition `(.)` as in the standard prelude

   (e) `h` defined as `h x = (flip (:)) x`

4. **(15 points)** Let `xsss = [["hugs","and"],["kisses"],["from","Hersheys"]]`. (Careful: This is a list of lists of lists of characters.) What are the values of the following Haskell expressions? If the expression contains an error, write "error".

   (a) `head xsss`

   (b) `foldr (++) [] xsss`

   (c) `map (map length) xsss`

   (d) `filter (==1) (map length xsss)`

   (e) `(head . tail . head . tail . tail) xsss`

5. **(4 points)** What is meant by the term *referential transparency*? Why is this considered an important property of functional programming languages? Are variable names in Java (or C++ or Fortran) programs referentially transparent? Why or why not?

6. **(4 points)** What is meant by the term *higher order function*? Which of the following Haskell standard prelude functions are higher order?

   `head, foldr, ++, reverse, flip, length, map, tail`

7. **(17 points)** Consider a Haskell function `floorList` that takes a value `b` from any totally ordered set and a list of values `xs` from the same set and returns `xs` except that any value that is less than `b` is replaced by `b`.

   For example. `floorList 5 [1, 2, 6, 7, 3, 9]` yields `[5, 5, 6, 7, 5, 9]`.

   (a) Give an appropriate polymorphic type signature for this function.

   (b) Define a version of the function that uses (backward) recursion directly.

   (c) Define a version of the function that uses the `map` function.

   (d) Define a function `countFloor b xs` that takes a value from any totally ordered set and a list of values `xs` from that same set and returns the count of the number of values greater than equal to `b`. (Hint: Maybe use `filter`.)

8. **(5 points)** Give the type signature and defining equations for a polymorphic Haskell function `applyEach` that, given a list of functions, applies each function to some given value.

   For example, `applyEach [(*3), (+2)] 5` yields `[15, 7]`.

9. **(5 points)** Give the type signature and defining equations for a Haskell function that takes a nonempty list of integers and returns a pair (i.e., a 2-tuple) containing the minimum and maximum values of the list. The solution should use a tail recursive auxiliary function with accumulating parameters.

10. (**10 points**) Define the following set of text-justification functions. You may want to use standard prelude functions like `take`, `drop`, and `length`.

    `spaces' n` returns a string of length `n` containing only space characters (i.e., the character `' '`).

    `left' n xs` returns a string of length `n` in which the string `xs` begins at the head (i.e., left end). Examples: `left' 3 "ab"` yields `"ab "` (ending with one blank) and `left' 3 "abcd"` yields `"abc"`.