

Name: _____

COSC 8, Fall 2007

Practice Midterm Exam

October 30, 2007

Problem	1	2	3	4	5	6	Total
Grade							
Points	20	15	15	15	15	20	100
Grader							

General Instructions

There are six (6) problems on this exam. Please check now that you have a complete exam booklet with eight (8) numbered pages. Please write your name on *each page* of the exam.

Be sure to look at *all* of the problems. Some are more difficult than others. Don't waste a lot of time on a problem that is giving you a hard time—if you get stuck, move on to another problem, and then return to it later.

There is space provided to answer each question, and you may request extra paper if you so desire. If you do use extra paper to record your solutions, be sure to write your name prominently on each extra page, and clearly indicate which problem is being answered there. You do not need to turn in any extra scratch-paper you use while working out your solutions; only the pages with your answers on them need to be submitted.

To help insure that you don't accidentally miss any of the questions, each section where an answer is requested has been marked with a \Leftarrow symbol in the right margin.

Please write neatly—the course staff reserves the right to ignore illegible answers. When writing solutions that involve program code, proper indentation of your code is important!

This examination covers material through Lecture 13.

You may not consult any books, notes, study-guides, or other outside material during the exam period, except for one sheet of 8.5 by 11 inch notebook paper which you are allowed to make hand-written notes on (both sides).

1. (20 Points) **Induction.** Given the following definitions:

```
data Tree a = Leaf a | Branch (Tree a) (Tree a)
```

```
listSum [] = 0
```

```
listSum (x:xs) = x + listSum xs
```

```
fringe (Leaf x) = [x]
```

```
fringe (Branch t1 t2) = fringe t1 ++ fringe t2
```

```
treeSum (Leaf x) = x
```

```
treeSum (Branch t1 t2) = treeSum t1 + treeSum t2
```

and the following lemma (which you need not prove, although you might want to for extra practice!):

Lemma: $\text{listSum } (xs ++ ys) = \text{listSum } xs + \text{listSum } ys$

prove the following by induction:

```
listSum . fringe = treeSum
```



2. (15 Points) Give the principal type for each of the following functions. Justify your answer.

(a) `map (foldr (+) 0)`

←←

(b) `zip [(+1), (+2), (+3)]`

←←

(c) `appendToHead x list = (x ++ head list) : list`

←←

3. (15 Points) **Functions.** Consider the following function:

```
thrice  :: (a -> a) -> a -> a
thrice f = f . f . f
```

For each of the following expressions, give the value that results from evaluating the expression, and justify your answer. You need not do the whole evaluation via substituting for quantities (although you may), but you should explain why your answer is correct.

(a) `(thrice (+1)) 0`

←←

(b) `((thrice . thrice) (+1)) 0`

←←

(c) ((thrice thrice) (+1)) 0



4. (15 Points) **Higher-Order Functions.** For each of the following exercises, write a function that solves the stated problem. For full credit, your solutions are subject to the following restrictions:

- Each solution must consist of a single function definition, and may not define any “helper” functions. You may construct anonymous functions, however.
- Your function may not call itself recursively, either directly or indirectly.

Apart from these restrictions, you may use any of the built-in functions described in the Haskell Quick Reference.

(a) (5 Points) Given a list of integers, compute the sum of the squares of all the even integers in the list. If there are no even integers in the list, the result should be zero. \leftarrow

(b) (5 Points) Write the `length` function. For this problem, obviously you may not use `length` in your answer. \leftarrow

(c) (5 Points) You are given an object x and a list $L = [s_1, s_2, \dots, s_n]$ in which each s_i is itself a list. Return a list of lists with all instances of x removed. \leftarrow

5. (15 Points) **Short Answer.**

(a) What is data abstraction and why is it useful?



(b) Why is `concat` defined as `foldr (++) []` as opposed to `foldl (++) []`?



(c) Why is normal induction (weak induction on the integers) less useful in Computer Science than strong structural induction?



6. (20 Points) The *transpose* of a matrix M is the matrix M^T obtained by flipping M about its main diagonal, so that its rows become columns, and its columns become rows. Formally, this means that if $a_{i,j}$ denotes the element of M in row i and column j , then the transpose takes $a_{i,j}$ to $a_{j,i}$.

For instance, the transpose of the matrix:

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

is the matrix:

$$M^T = \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix}$$

One way to represent a matrix in Haskell is as a list of rows, in which each row is a list of elements. This is the representation that Sudoku used. In this representation, the first matrix above would be represented:

```
[[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
```

and its transpose would be:

```
[[1,4,7,10],[2,5,8,11],[3,6,9,12]]
```

Note that the first list of the transpose consists of the first element of each list in the original matrix; the second list of the transpose consists of the second element of each list in the original matrix, *etc.*

There is a `transpose` function in the List module, and the Sudoku program used that. Here we want you to implement `transpose`. Write a Haskell function `transpose` which takes a matrix, represented as a list of lists, and returns its transpose represented as a list of lists. You may assume that all of the lists in the input matrix have the same number of elements. Feel free to use built-in Haskell functions such as `++`, `filter`, `map`, `foldr`, `reverse`, `zipWith`, *etc.* if they are useful.

