Exam in
Declarative Languages

Course code:	D7012E
Time:	4 hours, 9:00-13:00
Number of assignments:	6
Total number of points:	30
Date of exam:	2010-03-15
Teacher:	Fredrik Bengtsson, tel. 0920492431, 0738166670
Allowed aiding equipment:	None
Result announced:	2010-04-05

Assignment 1: Binary tree

8p
A binary tree is a data structure of nodes, where each node contains an element, and two nodes.

a (1p), Declare an algebraic data type, `BinTree a`, for binary trees in Haskell.

b (1p), Implement a function

```
countnodes :: BinTree a -> Int
```

that counts the number of nodes in a binary tree.

c (4p), Implement a function

```
issearchtree :: Ord a => BinTree a -> Bool
```

that checks if a binary tree is a binary *search* tree. A binary *search* tree is a tree where all elements in the left subtree is smaller than all the elements in the right subtree. This condition holds for all nodes.

d (2p), Implement a function

```
insert :: Ord a => BinTree a -> a -> BinTree a
```

that inserts an element into a binary search tree.

Assignment 2: One-corner block support

4p
We are faced with the problem of constructing support for a building where one of four corner-pillars have disintegrated (rämnat!). We want to replace the disintegrated pillar with one of the same height. However; we only have a bunch of already-built blocks of different thickness which we are about to combine in order to get a combined thickness as close as possible to the height of the disintegrated pillar.

In prolog, define a predicate

```
combinepads(+PadList, +PillarHeight, -ResultList)
```

where `PadList` is a list of values of the thickness of each block (one value for each pad) and `PillarHeight` is a value bound to the height of the disintegrated pillar. The predicate should compute the best possible combination of blocks such that their combined thickness is as close as possible to the height of the pillar. The thickness of the blocks chosen should be in `ResultList`.

Assignment 3: Monads

3p
Implement a user interface for multiplying numbers in haskell. The interface should prompt the user for numbers, one at a time, until the user enters 1. The program should then print the product of the numbers and the total number of numbers on screen.

Assignment 4: Logic

3p
What is the logical equivalent of the following programs:

a:
```
p :- c.
p :- a,!,b.
```

b:
```
p :- a,!,b.
p :- c.
```

c:
```
p :- a,b.
p :- c.
```

State a logical (boolean) expression equivalent to p.

Assignment 5: Higher order functions

6p

a: Declare a function in haskell,

```
collapse :: BinTree a -> (a -> b -> b -> b) -> b -> b
```

, that takes a function and a binary tree (as defined in assignment 1), a function and a value. `collapse` "combines" the values, of type `a`, from a tree into a single value of type `b`. The function argument `(a->b->b->b)` is used for the actual combining of values. The first argument to this function is the value from the node, whereas the second and third argument are combined results from the left and right subtree, respectively. The last argument to collapse is the start value for the computation used at leaf nodes.

b: Define a function

```
treetolist :: BinTree a -> [a]
```

, that returns the node-values of a tree according to a pre-order traversal of the tree. Pree-order traversal is when you visit the node first, then all nodes in the left subtree then all nodes in the right subtree. Here, you *must* use collapse from (a) in order to perform most of the work.

Assignment 6:

6p

a: In prolog, declare a predicate `findfirst(+E,+ L, -F)` that binds the first occurrence of `E` in the list `L` to `F`. The predicate should only bind the first occurrence to `F` and return no more answers regardless of the number of occurences of `E` in `F`. You may not use any "helper-functions" such as member and similar.

b: What is the difference between a red and a green cut? Explain.

c: Using findfirst from assignment **a**, declare a predicate `multiples(+L1, -L2)` that binds `L2` to a list containing all elements from `L1` that appears more than once in `L1`.