Exam in

Declarative Languages

Course code:	D7012E
Time:	4 hours, 13:00-17:00
Number of assignments:	7
Total number of points:	32
Date of exam:	2008-01-18
Teacher:	Fredrik Bengtsson, tel. 0920492431, 0738166670
Allowed aiding equipment:	None

Note: Don't forget to fill out the course evaluation and hand it in a separate bin.

Assignment 1: The knapsack problem **5p**

The knapsack problem is the problem of, given a knapsack of volume i and set of items (s,v) of size s and value v, maximize the total value of items in the knapsack. That is, you should pick a subset of items (s,v) such that the sum of values, v, from each item in the knapsack is maximized. The total size (sum of s) of the items must always be less than or equal to the size of the knapsack, i.

The assignment should be solved in prolog. Declare a predicate

```
knapsack(Size, List, Subset)
```

, where `Size` is the size of the knapsack, `List` is a list of items of the form `s/v`, and Subset is a list of items of the same form that represents the solution set. The predicate should only give one solution and should fail on backtracking.

Assignment 2: Binary search tree **5p**

A binary search tree is a data structure of nodes, where each node contains an element, and two nodes. All elements in the left subtree should be smaller than the element in the node and all elements in the right subtree should be larger. You may assume that all elements are distinct.

Declare an algebraic data type, `BinTree`, for binary search trees in Haskell.

Implement a function

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

that inserts an element into the tree

Implement a function

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

that checks if an element is present in the tree

Assignment 3: Quicksort **3p**

Quicksort is a sorting algorithm that works as follows: Pick a pivot element (can be chosen as first element of the sequence), partition the elements around the pivot (smaller than the pivot respective larger than the pivot. Sort each partition using quicksort. Put together the sorted sequence of smaller numbers, the pivot, and the sorted sequence of larger numbers. Done!

Implement quicksort as a function

```
quicksort :: Ord a => [a] -> [a]
```

in Haskell.

Assignment 4: Logic **3p**

What is the logical equivalent of the following programs:

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

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

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

State a logical (boolean) expression equivalent to p.

Assignment 5: Negation **6p**

a: In prolog, declare a predicate `not(P)`, which succeeds if and only if `P` fails.

b: Consider the following code:
```
notmember(X,L):-not member(X,L)

member(X,[X,_]).
member(X, [_|Rest]):-
  member(X, Rest).
```

In the following, assume no other bindings than the above two predicates.
Now, will `notmember(a, [b, c, d, e])` succeed? Explain why.
Will `notmember(A, [b,c,d,e])` succeed? Explain why.

Assignment 6: Monads **3p**

Implement a user interface for sorting numbers in Haskell. The interface should prompt the user for numbers, one at a time, and then sort the numbers. The input is ended by entering the number 0. After sorting, the sorted sequence should be printed on screen.

Assignment 7: Higher order functions **6p**

a: Declare a function in haskell,

```
comp :: [a -> a] -> a -> a
```

, that takes a list of functions `a -> a` and returns a composed function `a -> a`, where functions from the list are successively applied to the argument, from right to left. That is, given list `[f1, f2, f3, ... ]`, `comp` should return a function computing `f1(f2(f3(...)))`.

b: A repeat-loop is a construction that can perform the same actions repeatedly, each time updating a state. Declare a function

```
repeat :: (a -> a) -> Int -> a -> a
```

, where is any type, but can be thought of as a state of the computation. The integer is the number of iterations in the repeat-loop. The semantics of the repeat -loop is to compose the function `(a -> a)` repeatedly (as in exercise a) as many times as the Int specifies. The Int is a natural number (starting from 0). You are required to use the function `comp`, declared in a. Even if you did not solve problem a, you may use comp in the solution to this problem.

c: Declare a function

```
pow :: Floating a => a -> Int -> a
```

that computes the first argument raised to the power of the second argument. That is, if the first argument is `a` and the second is `b`, `pow a b` is a^b. You have to use the repeat -loop from exercise b. Even if you did not solve problem b, you may use the for-loop in the solution to this problem.

D7012E 2007: Course Evaluation

	not at all	to a small extent	Agrees to some extent	to a large extent	completely
General					
I think I have learnt declarative programming during the course (to the extent the course intends to)	☐	☐	☐	☐	☐
Overall impression from the course is good.	☐	☐	☐	☐	☐
Course literature is good (haskell)	☐	☐	☐	☐	☐
Course literature is good (prolog)	☐	☐	☐	☐	☐
Lectures					
	too low	low	just ok	high	too high
The pace of the lectures has been	☐	☐	☐	☐	☐
	useless	bad	ok	good	great!
The lecturer has been	☐	☐	☐	☐	☐

Comment:____

Labs					
	too low	low	just ok	high	too high
Complexity of the labs has been	☐	☐	☐	☐	☐
Workload of the labs has been	☐	☐	☐	☐	☐
	useless	bad	ok	good	great!
Quality of lab supervision has been	☐	☐	☐	☐	☐

Comment:____

Continue on next page!

	not at all	to a small extent	Agrees to some extent	to a large extent	completely
There has been enough time for lab assistance	☐	☐	☐	☐	☐

Additional comments:____

____

____

____

Hand in evaluation in a <u>separate</u> bin!