Exam in
Declarative Languages

Course code:	D7012E
Time:	4 hours, 09:00-13:00
Number of assignments:	6
Total number of points:	32
Date of exam:	2015-08-19
Teacher:	Fredrik Bengtsson, tel. 0920492431, 0738166670
Allowed aiding equipment:	Dictionary

Good luck!

Assignment 1 (8p)

a (2p): Declare a data type `BTree`, in Haskell that represents trees with information in each node and any branch-factor (This could be used to represent B-Trees, hence the name). The branch-factor of a tree (and node) is the number of child-nodes to that node. (A binary tree has a branch-factor of 2, for instance). This means that each node should be able to contain any number of child-nodes. Each node should also contain a piece of information which should be a type variable for the type `BTree`. For full credit, the representation should be unique, which means that there should be only one way of representing a particular tree.

b (3p): Declare a function `bsum`, that computes the sum of all elements in a `BTree`. You may use standard auxiliary functions without declaring them.

c (3p): Declare a function, `gt`, that takes a `BTree` and a number as an argument and returns a `BTree` of booleans where each number is replaced by a boolean value `True` if the corresponding element in the argument `BTree` is greater than the number argument and `False` otherwise.

Assignment 2 (6p)

a (3p): Write a predicate in prolog, `exchange(Vlist, V, Rlist)`, that, given a list, `Vlist`, of values (numbers), and a value, `V`, computes a list, `Rlist`, of numbers from `Vlist`, which sum is `V`. If no such combination can be found, `Rlist` should be bound to the empty list. All possible combinations of numbers from `Vlist` should be considered through backtracking.

b (3p): Write a predicate, `bestXchange(Vlist, V, Rlist)`, in prolog that computes the shortest list, `Rlist`, from assignment (a). You have to use the predicate `exchange` from (a).

Assignment 3 (6p)

Consider the following standard Haskell-functions:

```
foldr :: (a -> b -> b) -> b -> [a] -> b
filter :: (a -> Bool) -> [a] -> [a]
```

a (3p): Implement the function `filter` by using `foldr` (you may not use direct recursion).

b (3p): Implement a function

```
split :: [a] -> ([a],[a])
```

that takes a list, `L`, and returns a pair of lists containing the elements at even and odd positions in `L`, respectively. Consider the first element to be at position 1. Use `foldr` and do not type a recursive implementation directly.

Assignment 4 (5p)

In haskell, write a simple calculator program that can do sum calculations over lists (this could be used by an exam-grader that is otherwise unable to correctly sum the credits of the assigments of the exam).
The program should prompt for a numbers, one at a time, and place the numbers in a list. The list should be printed on screen as part of the prompt. When the number 0 is entered, the sum of the previously entered numbers should be computed and printed on screen.

Assignment 5 (4p)

Consider predicate `delall(X,L,L1)`, for deleting all occurences of X from list L obtaining L1:

```
delall(_, [], []).
delall(X,[X|L],L1):-delall(X,L,L1).
delall(X,[A|L],[A|L1]):-delall(X,L,L1).
```

a (2p): When backtracking the predicate `delall`, not all elements will be deleted in all solutions, as intended. Only the first solution will be correct. Modify the code by placing a cut somewhere, so that only the correct solution is retained.

b (2p): Modify the code as in (a), but <u>without using cut</u>, in order to achieve the same result.

Assignment 6 (3p)

What is the logical equivalent of the following programs:

a (1,5p)

```
p :- a,b; c; d.
p :- e,f.
```

b (1,5p)

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

State a logical (boolean) expression equivalent to `p`.