Exam in
Declarative Languages

Course code:	D7012E
Time:	4 hours
Number of assignments:	6
Total number of points:	31
Date of exam:	2016-05-28
Teacher:	Fredrik Bengtsson, tel. 0920492431, 0738166670
Allowed aiding equipment:	Dictionary

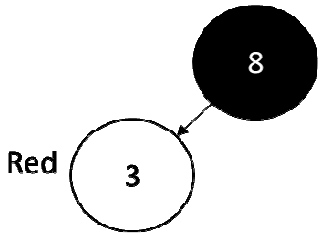
Please fill out the course evaluation form and hand in in a *separate bin*.

Good luck!

Assignment 1: Data structure (8p)

a (2p): Declare a data type that are able to represent two colors in Haskell: Red and Black. The type constructors should have intuitive names. Also, declare a datatype for red-black trees. A red-black tree is a binary search tree where information is stored in the nodes, as in a regular binary search tree, but where a color also is associated with each node. The color can be either red or black. Use the declared data type for the color. The data type of the information stored in each node should be a parameter to the type for the tree. The leafs does not store information or color.

b (1p): What expression, using the type from (a), corresponds to the following tree (the node denoted "red" is to be considered red):



c (3p): Implement a function `rbCheckRb` that takes a red-black tree as parameter and returns true if the red-black tree has the red-black balancing property (this is not the only property of a real red-black tree). The property to be checked is as follows: If a node is red then both it's children are black. All leaves are considered black. A black node can have any children. You are allowed to implement helper functions as you desire.

d (2p): Implement a function `rbToList` that, given a red-black tree, returns a list of all values for all nodes, in increasing order. In a binary search tree, for a particular node (any node), all nodes to the left have smaller value than the node itself, which is smaller than the values of the nodes to the right (traverse the tree in-order).

Assignment 2 (3p):

Using a predicate `prime(N)` that binds `N` to a prime number, starting with the smallest prime number, and a predicate and `goldbach(N,X,Y)`, that computes the goldbach decomposition, `X+Y`, of `N`, declare a predicate `strange(+N)`, that succeeds if and only if `N` is a prime number smaller than 100 *or* a prime number larger than 1000, but always fails if `N` is a goldbach number.

Assignment 3 (4p):

a: How many different solutions would the following goal yield on backtracking:

```
X=3,Y=4,N=3,length(L,N),member(X,L),member(Y,L).
```

State the binding for `L` for each solution.

b: Same question as (a), but with the following goal:

```
X=3,Y=4,N=3,length(L,N),member(X,L),!,member(Y,L).
```

Assigment 4 (4p):

Implement a simple cash register in Haskell: The cash register should repeatedly wait for number inputs as long as numbers are entered and, when an empty line is entered, the sum of the previous numbers should be printed. When “e” is entered, the program should exit.
Example of session:

```
Main> cashreg
5
7
3

15
-------
3
8

11
-------
e
```

Observe that the entered numbers are printed as they are entered by Haskell and need not be printed by other means. Helpful functions:

```
putStrLn :: String -> IO ()
getLine :: IO String
read :: Read a => String -> a
print :: Show a => a -> IO ()
```

Assignment 5 (6p):

a (3p): In prolog, given a predicate `prime(N)`, that will bind `N` to a prime number, starting with the smallest prime number, write a predicate `prime2(M,N)`, that will bind `N` to a prime number strictly smaller than `M`, starting with the smallest number and generating all prime numbers up to, but not including `M`.

b (3p): Using `prime2` from (a), write a predicate `goldbach(+N,-X,-Y)`, that will compute the goldbach decomposition of `N`. The goldbach decomposition of a number, N, is two prime numbers, X and Y, such that N=X+Y. (Goldbachs conjecture is the conjecture that such numbers always exists – still unproven).

Assignment 6: Higher order functions (6p)

a (3p): Declare a function `foldl`

```
foldl :: (a -> b -> a) -> a -> [b] -> a
```

in Haskell that folds a list from the left, instead of from the right, as `foldr`. A direct implementation without auxiliary helper functions is required.

b (2p): Declare, using `foldl` from (a), a function `length` that computes and returns the length of a given list. The binary operator for `foldl` should be declared using lambda-function (anonymous function).

c (1p): Would `foldr` work equally well for computing list length as `foldl` (possibly using a different binary operator)?