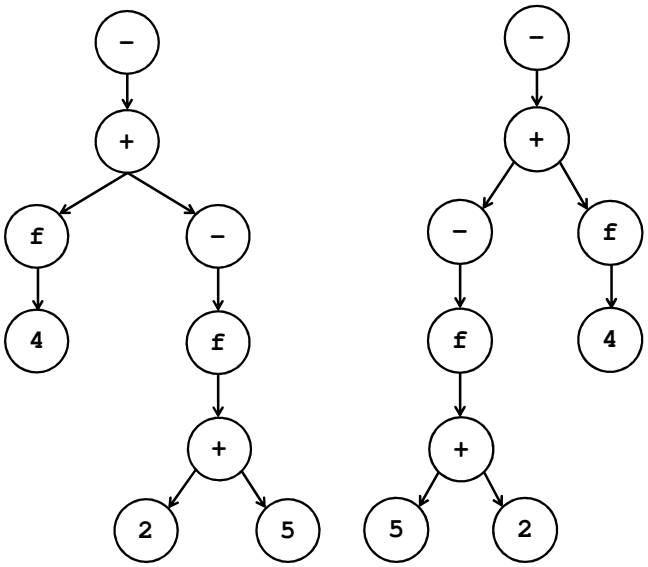
Exam in
Declarative Languages

Course code:	D7012E
Time:	4 hours
Number of assignments:	5
Total number of points:	
Date of exam:	2015-12-19
Teacher:	Fredrik Bengtsson, tel. 0920492431, 0738166670
Allowed aiding equipment:	Dictionary

Good luck!

Assignment 1: Data structure (8p)

a (2p): Declare a data type `ETree`, in Haskell that are able to represent a special type of expressions. Our expressions are much like arithmetic expressions, but contains additions, unary minus (a single minus sign in front of an expression), a function application, and numbers. Only one type of function can be represented.
The data type should represent the expression in a tree-like fashion (sometimes called parse-tree or abstract syntax tree). The operations should be called `Add`, `Minus`, `Function` and `Number` in the tree. The function takes only one argument. An example of such an expression is "-(f(4)+-f(2+5)) and its corresponding tree would look like the *left* tree below:



b (1p): State the haskell expression that would correspond to the above tree.

c (2p): Declare a function that takes an `ETree` and a function from `int` to `int` as argument and evaluates the expression represented by the `ETree` and returns the result of the evaluation. The function should be used for all function evaluations in the tree.

d (3p): Declare a function, `mirror`, that use the fact that addition is commutative, in order to change the operand order for all additions while maintaining the arithmetic value of the represented expression. For example, all `a+b` chould be changed to `b+a`. The example from the *left* figure in (a) would look like the *right* figure in (a).

Assignment 2 (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 3 (6p):
a: In prolog, declare a predicate `count(+E,+L,-N)`, that counts the number of occurrences of the element `E` in the list `L` and binds the result to `N`. Make sure the predicate always binds the correct number of elements to `N`, even when backtracking over the predicate. You are allowed to write a helper function that performs the actual work, but you are not allowed to use built-in predicates that performs the same or similar operations.

b: Using a correct implementation of count, what would happen if you call

```
count(A, [a,b,a,a,c,d,a], N).
```

Show what would be printed on screen, including attempts to perform backtracking until "false". (exact what characters are printed is not important, but you need to show what names are bound and to what value they are bound).

Assignment 4 (6p):
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 5: Higher order functions (6p)

a: Declare a function in haskell,

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

, that takes as arguments a binary tree, a function and a value. Also, declare a datatype for a binary tree. `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 arguments 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.