Programming Language Concepts, CS2104, 3rd Sept 2007
Tutorial 2 (Please attempt questions before coming to tutorial)

Exercise 1. (Finding an Element in a List) Give a definition of `{Member Xs Y}` that tests whether `Y` is an element of `Xs`. For this assignment you have to use the truth values `true` and `false`. The equality test (that is `==`) returns truth values and a function returning truth values can be used as condition in an `if`-expression. For example, the call `{Member [a b c] b}` should return `true`, whereas `{Member [a b c] d}` should return `false`.

Exercise 2. (Taking and Dropping Elements) Write two functions `{Take Xs N}` and `{Drop Xs N}`. The call `{Take Xs N}` returns the first `N` elements of `Xs` whereas the call `{Drop Xs N}` returns `Xs` without its first `N` elements. For example, `{Take [1 4 3 6 2] 3}` returns `[1 4 3]` and `{Drop [1 4 3 6 2] 3}` returns `[6 2]`.

Exercise 3. (Zip and UnZip) The operation `a # b` constructs a tuple with label `'#'` and fields `a` and `b` which is also known as a pair. We can use it to implement lists-of-pairs, e.g `[a#1 b#2 c#3]`. A different view of this data structure is known as a pair-of-lists, e.g `[a b c]#[1 2 3]`. Two important functions that convert list-of-pairs to pair-of-lists and vice versa are `Zip` and `UnZip`.

a) Implement a function `Zip` that takes a pair `Xs#Ys` of two lists `Xs` and `Ys` (of the same length) and returns a pairlist, where the first field of each pair is taken from `Xs` and the second from `Ys`. For example, `{Zip [a b c]#[1 2 3]}` returns the pairlist `[a#1 b#2 c#3]`.

b) The function `UnZip` does the inverse, for example `{UnZip [a#1 b#2 c#3]}` returns `[a b c]#[1 2 3]`. Give a specification and implementation of `UnZip`.

Exercise 4. (Finding the Position of an Element in a List) Write a function `{Position Xs Y}` that returns the first position of `Y` in the list `Xs`. The positions in a list start with `1`. For example, `{Position [a b c] c}` returns `3` and `{Position [a b c b] b}` returns `2`.
Try two versions:
1) one that assumes that `Y` is an element of `Xs` and
2) one that returns `0`, if `Y` does not occur in `Xs`.

Exercise 5. (Arithmetic Expressions Evaluation) Suppose that you are given an arithmetic expression described by a tree constructed from tuples as follows:

1. An integer is described by a tuple `int(N)`, where `N` is an integer.
2. An addition is described by a tuple `add(X Y)`, where both `X` and `Y` are arithmetic expressions.
3. A multiplication is described by a tuple `mul(X Y)`, where both `X` and `Y` are arithmetic expressions.

Implement a function `Eval` that takes an arithmetic expression and returns its value. For example, `add(int(1) mul(int(3) int(4)))` is an arithmetic expression and its evaluation returns `13`.

Exercise 6. Abstract Machine Concepts : Lecture 3 cover the definitions of the following declarative programming concepts: statement, value expression, environment, semantic statement, semantic stack, single-assignment store, execution state, and computation. There exists a visual abstract machine, called **VamOz** (Visual Abstract Machine for Oz), which can be freely downloaded from http://www.imit.kth.se/~schulte/misc/vamoz.html, which is executing kernel language programs as defined in the book Concepts, Techniques and Models of Computer Programming by Peter Van Roy and Seif Haridi. VamOz has been developed by Frej Drejhammar and Dragan Havelka with contributions from Christian Schulte. The idea is to give students a tool with which they can increase their understanding of how the abstract machine computes. VamOz has been used successfully in 2003 in the Datalogi II course taught by Christian Schulte at KTH. The language supported by VamOz is mainly the kernel language as described in Section 2.3 together with threads as introduced in Section 4.1 of the above mentioned book. Oz files loaded in the evaluator are automatically converted into kernel syntax. The following primitive operations are supported by VamOz: record operations (`Arity`, `.`, `Label`), equality tests (`==`, `\=`), order tests (`>`, `<`, `>=`, `=<`), operations on numbers (`+`, `-`, `*`, `/`, `div`, `mod`), type tests (`IsProcedure`).

Using the visual abstract machine, execute the following Oz programs. Note the semantic stack size (called in VamOz, the "thread stack") during the execution of some specific calls. Compare and explain the thread stack size in case of `FactProc` and `FastFact`.

a)
```
local X in
   X=1
   local X in X=2 end
   X=1
end
```

b)
```
local B in
   if B then skip else skip end
end
```

c)
```
local B in
   B = false
   if B then skip else skip end
end
```

d)
```
local FactProc R in
   proc {FactProc N ?Res}
      if N==0 then Res=1
      else local M in
             {FactProc N-1 M}
             Res=N*M
           end
      end
   end
   {FactProc 3 R}
end
```

e)
```
local FactAux FastFact X in
   fun {FactAux N M}
     if N==0 then M
     else {FactAux N-1 M*N}
     end
   end
   fun {FastFact N}
     if N>=0 then {FactAux N 1}
     end
   end
   X = {FastFact 3}
 end
```