Programming Language Concepts, CS2104
Tutorial 1 (27-31 August 2007)
(All students must prepare/attempt in advance.)

Exercise 1. **(Variables and Cells)** This exercise compares variables and cells. We give two code fragments. The first uses variables:

```
local X in
X=23
local X in
X=44
end
{Browse X}
end
```

The second uses a cell:

```
local X in
X={NewCell 23}
X:=44
{Browse @X}
end
```

In the first, the identifier `X` refers to two different variables. In the second, `X` refers to a cell. What does `Browse` display in each fragment? Explain.

Exercise 2. **(Accumulators)** This exercise investigates how to use cells together with functions. Let us define a function `{Accumulate N}` that accumulates all its inputs, that is: it adds together all the arguments of all calls. Here is an example:

```
{Browse {Accumulate 5}}
{Browse {Accumulate 100}}
{Browse {Accumulate 45}}
```

This should display `5`, `105`, and `150`, assuming that the accumulator contains zero at the start. Here is a wrong way to write `Accumulate`:

```
declare
fun {Accumulate N}
Acc in
Acc={NewCell 0}
Acc:=@Acc+N
@Acc
end
```

What is wrong with this definition? How would you correct it?

Exercise 3. (Values as trees) Given is the following declaration and assignment.

```
declare
R=r(1:[a b c] 4:[d [e [f]]] z:q(g h [10 11 [12 13]]))
```

Draw the value as a tree. In the following you have to give an expression composed of dot, width, and label functions that return the desired value. For example, for the value `g` the expression is `R.z.1` and for `r` it is `{Label R}`. If there is more than one possibility, give at least two expressions.

1. `b` 2. `q` 3. `12` 4. `nil`
5. `'|'` 6. `3` 7. `h` 8. `2`

Exercise 4. (Traversing trees) We have seen a way to traverse in preorder (first the root, then the left child, followed by the right child). It is:

```
declare
Root=node(left:X1 right:X2 value:0)
X1=node(left:X3 right:X4 value:1)
X2=node(left:X5 right:X6 value:2)
X3=node(left:nil right:nil value:3)
X4=node(left:nil right:nil value:4)
X5=node(left:nil right:nil value:5)
X6=node(left:nil right:nil value:6)
{Browse Root}
proc {Preorder X}
 if X \= nil then {Browse X.value}
   if X.left \= nil then {Preorder X.left} end
   if X.right \= nil then {Preorder X.right} end
 end
end
{Preorder Root}
```

Design the other known strategies, namely traverse in inorder (first the left child, then the root, followed by the right child) and postorder (first the left child, then the right child, followed by the root).

Exercise 5. (Pattern Matching for Head and Tail) Give definitions for `Head` and `Tail` that use pattern matching.

Exercise 6. (Length of a List) Try the version of `Length` as presented in the lecture. Since `X` from the pattern is not used in the right-hand side of the case, it can be replaced with a universal don't-care variable "_" that will not be bound to anything. Write another version, using the equality operator and the conditional construct. Is there any other way to implement `Length`?