CSL665/CS765: Introduction to (Logic and Functional) Programming
I semester 2003-04

Assignment 2: The Untyped Lambda calculus

Program a structure `Lambda` which implements the following signature of the untyped lambda-calculus. Please read the comments carefully to understand what each function is supposed to do. But they should be self-explanatory and are based entirely on lecture material.

```
signature LAMBDA =
sig
    datatype lambda = LVAR of string
                    | LABS of string * lambda
                    | LAPPLY of lambda * lambda

    datatype deBruijn = BVAR of int
                      | BABS of deBruijn
                      | BAPPLY of deBruijn * deBruijn

    val lambda2deBruijn : lambda -> deBruijn
        (* convert to deBruijn notation *)

    val deBruijn2lambda : deBruijn -> lambda
        (* convert to lambda notation *)

    val outer1 : deBruijn -> deBruijn
        (* 1-step leftmost-outermost beta-reduction *)

    val inner1 : deBruijn -> deBruijn
        (* 1-step leftmost-innermost beta-reduction *)

    val outerReduce : deBruijn -> deBruijn
        (* full computation with each step visible *)

    val innerReduce : deBruijn -> deBruijn
        (* full computation with each step visible *)
end;

structure Lambda: LAMBDA = ...
```

The functions `outerReduce` and `innerReduce` will naturally not terminate for many lambda-expressions. However, they must keep computing and outputting each step of their computation. This would require a function such as `showDeBruijn` to be defined internally, which outputs each step as a deBruijn term.

The function `deBruijn2lambda` requires the generation of strings representing variables. Think carefully how you will generate variable names.

Note:

1. You are *not* allowed to change any of the names given in the signature. You are not even allowed to change upper-case letters to lower-case letters or vice-versa.

2. You may define any new functions you like in the structure besides those mentioned in the signature.

3. Your program should work with the given signature.

4. You need to think of the *most efficient way* of implementing the various functions given in the signature so that the function results satisfy their definitions and properties.

5. Since the evaluator is going to look at your source code before evaluating it, you must explain your algorithms in the form of ML comments, so that the evaluator can understand what you have implemented.

6. Do *not* add any more decorations or functions or user-interfaces in order to impress the evaluator of the program. Nobody is going to be impressed by it.

7. There is a serious penalty for copying. If it is felt that there is too much similarity in the code between any two persons, then both are going to be penalized equally. So please set permissions on your directories, so that others cannot copy your programs.