Homework 2

You have to submit your solutions as announced in the lecture.
Unless mentioned otherwise, all problems are due 2017-03-02
There will be no deadline extensions unless mentioned otherwise in the lecture.

Problem 2.1 *Shellshock*

Points: 10

Consider the shellshock example from the lecture notes. In this problem, we implement a minimal shell that can exhibit shellshock.

You can use any programming language. However, because we want to be correct by design, it is best to use a good programming language such as SML or a very good programming language such as Scala.

1. Implement a data type for the following grammar, which represents the commands our shell can handle.

$$\begin{array}{lcl} COMM & ::= & fun\ NAME(NAME)\{COMM\} \\ & \mid & run\ NAME(SPACE\ EXPR)^* \\ & \mid & NAME(EXPR) \\ & \mid & COMM;COMM \\ EXPR & ::= & NAME \mid \text{"}\ (\backslash\backslash \mid \backslash\text{"} \mid [\hat{}\,\backslash\text{"}])^*\ \text{"} \\ NAME & ::= & \text{alphanumeric string} \end{array}$$

   where red color indicates BNF meta-symbols.

2. Implement a parser for your data type. It should be of the form

```
fun parseCommand(command : String) : Comm =
    ...
fun parseExpr(expr : String) : Expr =
    ...
```

3. Implement an interpreter for your data type. It should be of the form

```
abstract class Def()
class ValDef(name : String, value : String) extends Def
class FunDef(name : String, argName : String, body : Comm) extends Def

fun interpret(context : List[Def], command : Comm) : List[Def] =
    ...
fun evaluate(context : List[Def], expr : Expr) : String =
    ...
```

   and interpret commands as follows:

   - $fun\ f(x)\{C\}$: puts a $FunDef$ into the context
   - $run\ n\ arg_1\ \ldots\ arg_n$ call the shell function n with the listed arguments
   - $f(e)$ evaluates e to v, retrieves the $FunDef(f,x,b)$ from the context and executes b with an additional $ValDef(x,v)$ in the context
   - $c;d$ executes c first (which may return new definitions) and then executes d with those new definitions added to the context

   and evaluate expressions as follows

   - n: retrieve $ValDef(n,s)$ from the context and return s
   - "s": return the string s with the escapes removed

4. Your final program arises by taking a string s, calling $c = parseComm(s)$, then calling $interpret(Nil, c)$.

5. Now for the faulty functionality of bash, modify your program as follows:
   - At the beginning, try to parse every available environment variable that starts with fun into a $Comm$.
   - Execute all commands that parse successfully to obtain a list $defs : List[Def]$.
   - Call interpret with $defs$ instead of Nil as the initial context.
6. Fix your design so that function definitions in environment variables that are followed be commands cannot be executed.
7. Improve security further by considering only environment variables whose name begins with a certain prefix $SHELL\_FUNC$.