P2 - EDL - 2019.2 - 90 minutos

Nome:

Cada questão inteira deixada em branco vale `1/5` da sua pontuação máxima.

Se for o caso, indique aqui as questões deixadas em branco: _____

1. [2.5]

Considere a seguinte representação para a memória, comandos e expressões:

```
type Mem = [(String,Int)]

data Cmd = Atr String Exp       -- atribuicao
         | Seq Cmd Cmd          -- sequencia
         | Cnd Exp Cmd Cmd      -- condicional

data Exp = Num Int              -- constante
         | Var String           -- variavel
```

Represente o trecho de código acima em uma outra linguagem *estática* de sua preferência (ex., *C* ou *Java*).

2. [2.0]

Considere as definições a seguir:

```
type Mem = [(String,Int)]
type Cod = [(String,Cmd)]
type Env = (Mem,Cod)

data Exp = ...              -- (constante, soma, variavel, etc)
         | App String       -- funcoes nao recebem ou retornam argumentos

data Cmd = ...              -- (atribuicao, sequencia, etc)
         | Fun String Cmd   -- definicao de funcao

avaliaCmd :: Env -> Cmd -> Env
...
avaliaCmd (mem,cod) (Fun id c)  = <...>

avaliaExp :: Env -> Exp -> Int
...
avaliaExp (mem,cod) (App id e)  = <...>
```

Complete os trechos em `<...>`, incluindo os acessos a memória. Note que aqui funções não recebem ou retornam valores (i.e., se comunicam por globais).

3. [3.0]

Considere a seguinte representação para comandos:

```
data Cmd = Atr String Exp
         | Dcl String
         | Seq Cmd Cmd
         | Cnd Exp Cmd Cmd
         | Fun String Cmd
```

Crie uma função `repetidas :: Cmd -> [String]` que receba um comando e retorne uma lista de variáveis que foram redeclaradas no programa:

Considere que variáveis globais (declaradas fora de funções) não conflitam com variáveis locais (declaradas dentro de funções). Apenas conflitos entre globais ou locais da mesma função devem ser considerados como redeclarações.

4. [2.5]

Considere a linguagem completa com todos os comandos vistos durante o curso:

```
data Exp = Num Int
         | Add Exp Exp
         | Sub Exp Exp
         | Var String
         | App String [Exp] -- funcoes aceitam multiplos parametros

data Cmd = Atr String Exp
         | Dcl String
         | Prt Exp
         | Seq Cmd Cmd
         | Cnd Exp Cmd Cmd
         | Rep Exp Cmd
         | Fun [String] Cmd -- funcoes aceitam multiplos parametros
```

Reescreva o programa a seguir em Python na linguagem descrita acima:

```
def calcula (x,y):
    soma = 0
    while x<y:
        soma = soma + x
        x = x + 1
    return soma

print(calcula(1,10))
```