Exercice 2.5

Patrick Marchand

18-05-13

Ce module est ecrit en haskell litteraire et decrit un petit langage avec des Integers, des variables et des lambdas.

```
module Main where
```

1 Grammaire

L 'on definit en premier des types qui decrivent les elements de notre grammaire.

```
type Var = String

data Exp = Enum Int
  | Evar Var
  | Elet Var Exp Exp
  | Ecall Exp Exp
```

E nsuite on rajoute une expression pour representer les lambdas, ceci est une fonction anonyme qui prend un nom de variable et une expression a laquelle l'appliquer.

```
  | Elambda Var Exp
```

L a valeur Vprim represente des fonctions qui existent deja dans le contexte, pour simplifier je m'en suis debarasser et j'utilise les lambdas pour toute formes de fonctions.

```
data Val = Vnum Int
  | Vlambda (Val -> Val)
```

2 Evaluation

P ar apres nous avons besoin d'une fonction pour rechercher les valeurs des variables dans un contexte donner.

E lle trouve une clee dans une liste de paires et retourne la valeur associer. J'ai changer l'ordre des arguments dans la signature afin d'etre plus consistant avec la fonction eval.

```
elookup :: Eq a => [(a, b)] -> a -> b
elookup ((x1, v1):env) x =
 if x == x1 then v1 else elookup env x
```

L a fonction principale, qui s'occupe d'evaluer les expressions et retourner leur valeur, en se basant sur le contexte fournie.

```
eval :: [(Var, Val)] -> Exp -> Val
```

1. Il n'y a rien a faire d'autre avec un chiffre que le retourner.
2. Si l'on recoit une variable, l'on retourne sa valeur
3. Une expr let permet de definir des nouvelles variables
4. Les fonctions sont elles meme lier a des variables et peuvent etre appeler par une expr call.
5. Dans le cas d'un lambda, nous retournons sa valeur

```
eval env (Enum n) = Vnum n

eval env (Evar x) = elookup env x

eval env (Elet x e1 e2) =
  let v = eval ((x,v):env) e2 in eval ((x,v):env) e2

eval env (Ecall fun actual) =
  case eval env fun of
    Vlambda f -> f (eval env actual)

eval env (Elambda v e) =
  Vlambda (\value -> eval ((v, value) : env) e)
```

Le main permet tout simplement de taire le compilateur, nous ne pouvons directement utiliser notre evaluateur sans definir des instances Show pour nos valeurs, ce qui est un peu difficile avec la definition de Vlambda.

```
main = do print "done"
```