ソフトウェア技法　第 8 回　201311367　佐藤良祐 2014 年 12 月 20 日

課題 8.1

(1), (2) を両方まとめてのせる.

```
type primop = PLUSop | MULop;;

type exp = INTexp of int
        | FLOATexp of float
        | VARexp of string
        | LETexp of string * exp * exp
        | PRIMexp of primop * exp * exp;;

(* 上のデータ構造で表されたプログラムを文字列に変換する *)
let rec exp2string exp =
  match exp with
    INTexp x -> string_of_int x
  | FLOATexp x -> string_of_float x
  | VARexp x -> x
  | PRIMexp (primop, exp1, exp2) ->
     if primop = PLUSop then
       "(" ^ exp2string exp1 ^ "+" ^ exp2string exp2 ^ ")"
     else
       "(" ^ exp2string exp1 ^ "*" ^ exp2string exp2 ^ ")"
  | LETexp (str, exp1, exp2) ->
     "let " ^ str ^ "=" ^ exp2string exp1 ^ " in " ^ exp2string exp2;;


(*
 * 上のデータ構造で与えられたプログラムを評価するインタープリタ
 * データ型value で示し 整数と実数の演算時は整数を実数に変換して計算する
 *)
type value = INTval of int | FLOATval of float;;

let extend env (x, v) = (x, v)::env;;

let rec lookup l x =
  match l with
    (y, v)::rest -> if x = y then v else lookup rest x;;

let plusop exp1 exp2 =
  match (exp1, exp2) with
    (INTval x, INTval y) -> INTval (x + y)
  | (INTval x, FLOATval y) -> FLOATval (float_of_int x +. y)
  | (FLOATval x, INTval y) -> FLOATval (x +. float_of_int y)
  | (FLOATval x, FLOATval y) -> FLOATval (x +. y);;

let multiop exp1 exp2 =
  match (exp1, exp2) with
    (INTval x, INTval y) -> INTval (x * y)
  | (INTval x, FLOATval y) -> FLOATval (float_of_int x *. y)
  | (FLOATval x, INTval y) -> FLOATval (x *. float_of_int y)
  | (FLOATval x, FLOATval y) -> FLOATval (x *. y);;


let rec eval env exp =
  match exp with
  INTexp n -> INTval n
 | FLOATexp n -> FLOATval n
 | VARexp n -> lookup env n
 | LETexp (x, exp1, exp2) ->
   let v1 = eval env exp1 in eval (extend env (x, v1)) exp2
 | PRIMexp (primop, exp1, exp2) ->
   if primop = PLUSop then
     plusop (eval env exp1) (eval env exp2)
   else
     multiop (eval env exp1) (eval env exp2);;
```

(1)

```
# exp2string (PRIMexp (PLUSop, INTexp 1, FLOATexp 2.0));;
- : string = "(1+2.)"
# exp2string (LETexp ("x", INTexp 1, PRIMexp (MULop, VARexp "x", VARexp "x"
  )));;
- : string = "let x=1 in (x*x)"
```

(2)

```
# eval [] (PRIMexp (PLUSop, INTexp 1, FLOATexp 1.5));;
- : value = FLOATval 2.5
# eval [] (LETexp ("x", INTexp 2, PRIMexp (MULop, VARexp "x", VARexp "x")));;
- : value = INTval 4
```

課題 8.2

実行サンプルもソースコードに埋め込んであるので以下のようにすぐ実行できる.

```
(* オプション課題 *)
(* クロージャ *)
(* 以下の 2つのデータ型で表される *)
type primop = PLUSop | MINUSop;;

type exp =
  | INTexp of int
  | VARexp of string
  | FNexp of string * exp
  | PRIMexp of primop * exp * exp
  | IFZEROexp of exp * exp * exp
  | APPexp of exp * exp;;

(* 値はこのデータ型で表す*)
type value = INTval of int | CLOSUREval of string * exp * (string * value) list;;

let extend env (x, v) = (x, v)::env;;

let rec lookup l x =
  match l with
    (y, v)::rest -> if x = y then v else lookup rest x;;

let rec eval env exp =
  match exp with
    INTexp x -> INTval x
  | VARexp x -> lookup env x
  | FNexp (str, exp) -> CLOSUREval (str, exp, env)
  | IFZEROexp (exp1, exp2, exp3) ->
     if (eval env exp1) = INTval 0 then (eval env exp2) else (eval env exp3)
  | PRIMexp (primop, exp1, exp2) ->
     let calc x y =
       match (x, y) with
         (INTval a, INTval b) ->
         if primop = PLUSop then INTval (a + b) else INTval (a - b)
     in calc (eval env exp1) (eval env exp2)
  | APPexp (exp1, exp2) ->
     match (eval env exp1) with
       CLOSUREval (m, n, a) -> let arg = (eval env exp2) in eval (extend a (m, arg
           )) n;;

(* サンプル実行 *)
let exp1 =
  APPexp (APPexp (FNexp ("x",
                         FNexp ("y", PRIMexp (PLUSop, VARexp "x", VARexp "y"))),
                      INTexp 1),
        INTexp 2);;

let fixsub = FNexp ("x", APPexp (VARexp "f",
                                 FNexp ("y",
                                        APPexp (APPexp (VARexp "x", VARexp "x"),
                                                VARexp "y"))));;

```

```
let fix = FNexp ("f", APPexp (fixsub, fixsub));;

let sum =
  FNexp ("g", FNexp ("x", IFZEROexp (VARexp "x",
                                     INTexp 0,
                                     PRIMexp (PLUSop,
                                              VARexp "x",
                                              APPexp (VARexp "g",
                                                      PRIMexp (MINUSop,
                                                               VARexp "x",
                                                               INTexp 1)))))));;

let sum n = APPexp (APPexp (fix, sum), INTexp n);;

  eval [] exp1;;
  eval [] (sum 4);;
```

```
# eval [] exp1;;
- : value = INTval 3
# eval [] (sum 4);;
- : value = INTval 10
```