課題 8.1

1.

```
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;;

let rec exp2string e =
    match e with
          INTexp i             -> string_of_int i
        | FLOATexp f           -> string_of_float f
        | VARexp v             -> v
        | LETexp (str, e1, e2) -> "let " ^ str ^ " = " ^ (exp2string e1) ^ " in " ^ (exp2string e2)
        | PRIMexp(pri, e1, e2) -> "(" ^ (exp2string e1) ^ (if pri = PLUSop then " + " else " * ") ^
(exp2string e2) ^ ")";;

exp2string (PRIMexp (PLUSop, INTexp 1, FLOATexp 2.0));;
exp2string (LETexp ("x", INTexp 1, PRIMexp (MULop, VARexp "x", VARexp "x")));;
```

実行結果

```
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
val exp2string : exp -> string = <fun>
- : string = "(1 + 2.)"
- : string = "let x = 1 in (x * x)"
```

2.

```
type value = INTval of int | FLOATval of float;;

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

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

let rec eval env exp =
    match exp with
          INTexp i             -> INTval i
        | FLOATexp f           -> FLOATval f
        | VARexp v             -> lookup env v
     (* | LETexp (str, e1, e2) ->  わからん!(^^)! *)
        | PRIMexp(pri, e1, e2) ->
            let (val1, val2) = (eval env e1, eval env e2) in
            if pri = PLUSop then
                match (val1, val2) with
                      (INTval m, INTval n)      -> INTval (m + n)
                    | (INTval m, FLOATval r)    -> FLOATval (float_of_int m +. r)
                    | (FLOATval r, INTval n)    -> FLOATval (float_of_int n +. r)
                    | (FLOATval r, FLOATval r') -> FLOATval (r +. r')
            else
                match (val1, val2) with
                      (INTval m, INTval n)      -> INTval (m * n)
                    | (INTval m, FLOATval r)    -> FLOATval (float_of_int m *. r)
                    | (FLOATval r, INTval n)    -> FLOATval (float_of_int n *. r)
                    | (FLOATval r, FLOATval r') -> FLOATval (r *. r');;

eval [] (PRIMexp (PLUSop, INTexp 1, FLOATexp 1.5));;
(* eval [] (LETexp ("x", INTexp 2, PLUSexp (VARexp "x", VARexp "x")));; *)
```

実行結果

```
type value = INTval of int | FLOATval of float

File "8.ml", line 27, characters 4-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
[]

val lookup : ('a * 'b) list -> 'a -> 'b = <fun>
val extend : ('a * 'b) list -> 'a * 'b -> ('a * 'b) list = <fun>

File "8.ml", line 33, characters 4-1036:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
LETexp (_, _, _)

val eval : (string * value) list -> exp -> value = <fun>
- : value = FLOATval 2.5
そふと
```