CS603 16 February 2005 μ-Scheme Metacircular Interpreter

```
(val emptystore '((next 0)))
(define find-c (key alist success-cont failure-cont)
   (letrec
       ((search (lambda (alist)
                   (if (null? alist)
                       (failure-cont)
                       (if (equal? key (caar alist))
                           (success-cont (cadar alist))
                           (search (cdr alist)))))))
     (search alist)))
(val sigma emptystore)
(define load  (l)   (find-c l sigma (lambda (x) x)
                            (lambda () (error (list2 'unbound-location: l)))))
(define store (l v) (begin (set sigma (bind l v sigma)) v))
(define allocate (value)
  (let*
    ((loc (load 'next)))
    (begin
       (store 'next (+ loc 1))
       (store loc value)
       loc)))
(define bindalloc (name v env)
  (bind name (allocate v) env))
(define bindalloclist (nl vl env)
  (if (and (null? nl) (null? vl))
    env
    (bindalloclist (cdr nl) (cdr vl) (bindalloc (car nl) (car vl) env))))
(define apply-prim (prim)
  (lambda (args)
    (if (= (length args) 1)
      (prim (car args))
      (if (= (length args) 2)
        (prim (car args) (cadr args))
        (error (list4 'all-primitives-expect-one-or-two-arguments---got (length args)
                      ': args))))))
(define primenv ()
  (let*
      ((env '())
       (env (bindalloc '+ (apply-prim +) env))
       (env (bindalloc '- (apply-prim -) env))
       (env (bindalloc '* (apply-prim *) env))
       (env (bindalloc '/ (apply-prim /) env))
       (env (bindalloc '< (apply-prim <) env))
       (env (bindalloc '> (apply-prim >) env))
       (env (bindalloc '= (apply-prim =) env))
       (env (bindalloc 'car        (apply-prim car)        env))
       (env (bindalloc 'cdr        (apply-prim cdr)        env))
       (env (bindalloc 'cons       (apply-prim cons)       env))
       (env (bindalloc 'print      (apply-prim print)      env))
       (env (bindalloc 'error      (apply-prim error)      env))
       (env (bindalloc 'boolean?   (apply-prim boolean?)   env))
       (env (bindalloc 'null?      (apply-prim null?)      env))
       (env (bindalloc 'number?    (apply-prim number?)    env))
       (env (bindalloc 'symbol?    (apply-prim symbol?)    env))
       (env (bindalloc 'procedure? (apply-prim procedure?) env))
       (env (bindalloc 'pair?      (apply-prim pair?)      env)))
    env))
(define find-variable (x env)
  (find-c x env (lambda (x) x) (lambda () (error (list2 'unbound-variable: x)))))
(define unary (name f rest)
```

CS603 16 February 2005 μ-Scheme Metacircular Interpreter

```
  (if (= (length rest) 1)
    (f (car rest))
    (error (list5 name 'expression-needs-one-argument,-got (length rest) 'in rest))))
(define binary (name f rest)
  (if (= (length rest) 2)
    (f (car rest) (cadr rest))
    (error (list5 name 'expression-needs-two-arguments,-got (length rest) 'in rest))))
(define trinary (name f rest)
  (if (= (length rest) 3)
    (f (car rest) (cadr rest) (caddr rest))
    (error (list5 name 'expression-needs-three-arguments,-got (length rest) 'in
                  rest))))
(define eval (env)
   (letrec
       ((ev (lambda (e) (if (symbol? e)
                            (load (find-variable e env))
                            (if (atom? e)
                              e
                              (let ((first (car e))
                                    (rest  (cdr e)))
                                (if (exists? ((curry =) first) '(set if while lambda quote begin))
                                    (if (= first 'set)    (binary  'set    ast-set    rest)
                                    (if (= first 'if)     (trinary 'if     ast-if     rest)
                                    (if (= first 'while)  (binary  'while  ast-while  rest)
                                    (if (= first 'lambda) (binary  'lambda ast-lambda rest)
                                    (if (= first 'quote)  (unary   'quote  ast-quote  rest)
                                    (if (= first 'begin)  (ast-begin rest)
                                    (error (list2 'this-cannot-happen---bad-ast first))))))))
                                    ((ev first) (map ev rest))))))))
        (ast-quote (lambda (e) e))
        (ast-if    (lambda (e1 e2 e3) (if (ev e1) (ev e2) (ev e3))))
        (ast-while (lambda (cond body) (while (ev cond) (ev body))))
        (ast-set   (lambda (v e)
                     (let ((loc (find-variable v env)))
                        (if (null? loc)
                           (error (list2 'set-unbound-variable v))
                           (store loc (ev e))))))
        (ast-begin (lambda (es) (foldl (lambda (e result) (ev e)) '() es)))
        (ast-lambda (lambda (formals body)
                      (if (all? symbol? formals)
                        (lambda (actuals)
                          ((eval (bindalloclist formals actuals env)) body))
                        (error (list2 'lambda-with-bad-formals: formals))))))
     ev))

(define ast-val (env)
  (lambda (v e)
    (if (symbol? v)
        (let* ((env
                  (if (exists? (lambda (pair) (= v (car pair))) env)
                    env
                    (bindalloc v '() env))))
          (begin
            ((eval env) (list3 'set v e))
            env))
        (error (list2 'val-tried-to-bind-non-symbol v)))))
(define ast-define (env)
  (lambda (name formals body)
    ((ast-val env) name (list3 'lambda formals body))))
(define ast-exp (e env)
```

CS603 16 February 2005 μ-Scheme Metacircular Interpreter

```
  (begin
    (print ((eval env) e))
    env))
(define topeval (e env)
  (if (pair? e)
    (let ((first (car e))
          (rest  (cdr e)))
      (if (= first 'val)
        (binary 'val (ast-val env) rest)
        (if (= first 'define)
            (trinary 'define (ast-define env) rest)
            (ast-exp e env))))
    (ast-exp e env)))
(define read-eval-print (el)
    (foldl topeval (primenv) el))
(define rep (el)
  (begin (read-eval-print el) 0))
```