**Report of Structure and Interpretation of Computer Programs**

Chonbuk National University

Department of Computer Science

201514768

임유택

**Problem 1**

**code;**

#lang racket

;;

;; eval.scm - 6.037

;;

(require r5rs)

(define first car)

(define second cadr)

(define third caddr)

(define fourth cadddr)

(define rest cdr)

;; Tell DrRacket to print mutable pairs using the compact syntax for

;; ordinary pairs.

(print-as-expression #f)

(print-mpair-curly-braces #f)

(define table-tag 'table)

(define (make-table) (cons table-tag null))

(define table (make-table))

(define (find-assoc key table)

(cond

((null? table) 'ERROR)

((equal? key (caar table)) (cadar table))

(else (find-assoc key (rest table)))))

(define (add-assoc key val alist)

(cons (list key val) alist))

(define (table-get tbl key)

(if (table? tbl)

(find-assoc key (rest tbl))

#f))

(define (table? table1)

(if (equal? (first table1) 'table)

#t

#f))

(define (table-put! tbl key val)

(if (table? tbl)

(set-cdr! tbl (add-assoc key val (rest tbl)))

#f))

;; mutable cons cell version of map

(define (mmap f lst)

(if (null? lst)

'()

(cons (f (car lst)) (mmap f (cdr lst)))))

(define (tagged-list? exp tag)

(and (pair? exp) (eq? (car exp) tag)))

(define (self-evaluating? exp)

(cond ((number? exp) #t)

((string? exp) #t)

((boolean? exp) #t)

(else #f)))

(define (quoted? exp) (tagged-list? exp 'quote))

(define (text-of-quotation exp) (cadr exp))

(define (variable? exp) (symbol? exp))

(define (assignment? exp) (tagged-list? exp 'set!))

(define (assignment-variable exp) (cadr exp))

(define (assignment-value exp) (caddr exp))

(define (make-assignment var expr)

(list 'set! var expr))

(define (definition? exp) (tagged-list? exp 'define))

(define (definition-variable exp)

(if (symbol? (cadr exp)) (cadr exp) (caadr exp)))

(define (definition-value exp)

(if (symbol? (cadr exp))

(caddr exp)

(make-lambda (cdadr exp) (cddr exp)))) ; formal params, body

(define (make-define var expr)

(list 'define var expr))

(define (lambda? exp) (tagged-list? exp 'lambda))

(define (lambda-parameters lambda-exp) (cadr lambda-exp))

(define (lambda-body lambda-exp) (cddr lambda-exp))

(define (make-lambda parms body) (cons 'lambda (cons parms body)))

(define (if? exp) (tagged-list? exp 'if))

(define (if-predicate exp) (cadr exp))

(define (if-consequent exp) (caddr exp))

(define (if-alternative exp) (cadddr exp))

(define (make-if pred conseq alt) (list 'if pred conseq alt))

(define (cond? exp) (tagged-list? exp 'cond))

(define (cond-clauses exp) (cdr exp))

(define first-cond-clause car)

(define rest-cond-clauses cdr)

(define (make-cond seq) (cons 'cond seq))

(define (let? expr) (tagged-list? expr 'let))

(define (let-bound-variables expr) (mmap first (second expr)))

(define (let-values expr) (mmap second (second expr)))

(define (let-body expr) (cddr expr)) ;differs from lecture--body may be a sequence

(define (make-let bindings body)

(cons 'let (cons bindings body)))

(define (begin? exp) (tagged-list? exp 'begin))

(define (begin-actions begin-exp) (cdr begin-exp))

(define (last-exp? seq) (null? (cdr seq)))

(define (first-exp seq) (car seq))

(define (rest-exps seq) (cdr seq))

(define (sequence->exp seq)

(cond ((null? seq) seq)

((last-exp? seq) (first-exp seq))

(else (make-begin seq))))

(define (make-begin exp) (cons 'begin exp))

(define (application? exp) (pair? exp))

(define (operator app) (car app))

(define (operands app) (cdr app))

(define (no-operands? args) (null? args))

(define (first-operand args) (car args))

(define (rest-operands args) (cdr args))

(define (make-application rator rands)

(cons rator rands))

(define (time? exp) (tagged-list? exp 'time))

(define (and? exp) (tagged-list? exp 'and)) ;; 2번 문제

(define (until? exp) (tagged-list? exp 'until)) ;; 3번 문제

(define (current-env? exp) (tagged-list? exp 'current-env)) ;; 5번 문제

(define (procedure-env? exp) (tagged-list? exp 'procedure-env)) ;; 5번 문제

;;

;; this section is the actual implementation of meval

;;

(define (m-eval exp env)

(cond ((self-evaluating? exp) exp)

((variable? exp) (lookup-variable-value exp env))

((quoted? exp) (text-of-quotation exp))

((assignment? exp) (eval-assignment exp env))

((unset? exp) (eval-unset exp env)) ;; 4번 문제

((definition? exp) (eval-definition exp env))

((if? exp) (eval-if exp env))

((lambda? exp)

(make-procedure (lambda-parameters exp) (lambda-body exp) env))

((begin? exp) (eval-sequence (begin-actions exp) env))

((cond? exp) (m-eval (cond->if exp) env))

((let? exp) (m-eval (let->application exp) env))

((time? exp) (time (m-eval (second exp) env)))

((and? exp) (eval-and exp env)) ;; 2번 문제

((until? exp) (m-eval (until->transformed exp) env)) ;;3번 문제

((current-env? exp) (eval-current-env env)) ;;5번 문제

((procedure-env? exp) (eval-procedure-env exp env)) ;;5번 문제

((time? exp) (time (m-eval (second exp) env)))

((application? exp)

(m-apply (m-eval (operator exp) env)

(list-of-values (operands exp) env)))

(else (error "Unknown expression type -- EVAL" exp))))

(define (m-apply procedure arguments)

(cond ((primitive-procedure? procedure)

(apply-primitive-procedure procedure arguments))

((compound-procedure? procedure)

(eval-sequence

(procedure-body procedure)

(extend-environment (make-frame (procedure-parameters procedure)

arguments)

(procedure-environment procedure))))

(else (error "Unknown procedure type -- APPLY" procedure))))

(define (list-of-values exps env)

(cond ((no-operands? exps) '())

(else (cons (m-eval (first-operand exps) env)

(list-of-values (rest-operands exps) env)))))

(define (eval-if exp env)

(if (m-eval (if-predicate exp) env)

(m-eval (if-consequent exp) env)

(m-eval (if-alternative exp) env)

))

(define (eval-sequence exps env)

(cond ((last-exp? exps) (m-eval (first-exp exps) env))

(else (m-eval (first-exp exps) env)

(eval-sequence (rest-exps exps) env))))

(define (eval-assignment exp env)

(set-variable-value! (assignment-variable exp)

(m-eval (assignment-value exp) env)

env))

(define (eval-definition exp env)

(define-variable! (definition-variable exp)

(m-eval (definition-value exp) env)

env))

(define (let->application expr)

(let ((names (let-bound-variables expr))

(values (let-values expr))

(body (let-body expr)))

(make-application (make-lambda names body)

values)))

(define (cond->if expr)

(let ((clauses (cond-clauses expr)))

(if (null? clauses)

#f

(if (eq? (car (first-cond-clause clauses)) 'else)

(sequence->exp (cdr (first-cond-clause clauses)))

(make-if (car (first-cond-clause clauses))

(sequence->exp (cdr (first-cond-clause clauses)))

(make-cond (rest-cond-clauses clauses)))))))

;;6번 문제

(define (until-test exp) (cadr exp))

(define (until-exps exp) (cddr exp))

(define (until->transformed exp) ;; 6번 문제

(make-let

'()

(list

(make-define

'(loop)

(make-if

(until-test exp)

#t

(make-begin (append (until-exps exp) (list '(loop))))))

'(loop))))

(define (last-pair? lst)

(null? (cdr lst)))

(define (and-clauses exp) (cdr exp))

(define (eval-and exp env) ;; 2번 문제

(define (and-helper clauses)

(let ((val (m-eval (car clauses) env)))

(cond ((last-pair? clauses)

val)

(val

(and-helper (cdr clauses)))

(else

#f))))

(if (last-pair? exp)

#t

(and-helper (and-clauses exp))))

(define input-prompt ";;; M-Eval input level ")

(define output-prompt ";;; M-Eval value:")

(define (driver-loop) (repl #f))

(define (repl port)

(if port #f (prompt-for-input input-prompt))

(let ((input (if port (read port) (read))))

(cond ((eof-object? input) 'meval-done)

((eq? input '\*\*quit\*\*) 'meval-done)

(else

(let ((output (m-eval input the-global-environment)))

(if port #f (begin

(announce-output output-prompt)

(pretty-display output)))

(repl port))))))

(define (prompt-for-input string)

(newline) (newline) (display string) (display meval-depth) (newline))

(define (announce-output string)

(newline) (display string) (newline))

;;

;;

;; implementation of meval environment model

;;

; double bubbles

(define (make-procedure parameters body env)

(list 'procedure parameters body env))

(define (compound-procedure? proc)

(tagged-list? proc 'procedure))

(define (procedure-parameters proc) (second proc))

(define (procedure-body proc) (third proc))

(define (procedure-environment proc) (fourth proc))

; bindings

(define (make-binding var val)

(list 'binding var val))

(define (binding? b)

(tagged-list? b 'binding))

(define (binding-variable binding)

(if (binding? binding)

(second binding)

(error "Not a binding: " binding)))

(define (binding-value binding)

(if (binding? binding)

(third binding)

(error "Not a binding: " binding)))

; frames

(define (make-frame variables values)

(define (make-frame-bindings rest-vars rest-vals)

(cond ((and (null? rest-vars) (null? rest-vals))

'())

((null? rest-vars)

(error "Too many args supplied" variables values))

((symbol? rest-vars)

(list (make-binding rest-vars rest-vals)))

((null? rest-vals)

(error "Too few args supplied" variables values))

(else

(cons (make-binding (car rest-vars) (car rest-vals))

(make-frame-bindings (cdr rest-vars) (cdr rest-vals))))))

(make-frame-from-bindings (make-frame-bindings variables values)))

(define (make-frame-from-bindings list-of-bindings)

(cons 'frame list-of-bindings))

(define (frame? frame)

(tagged-list? frame 'frame))

(define (frame-variables frame)

(if (frame? frame)

(mmap binding-variable (cdr frame))

(error "Not a frame: " frame)))

(define (frame-values frame)

(if (frame? frame)

(mmap binding-value (cdr frame))

(error "Not a frame: " frame)))

(define (add-binding-to-frame! binding frame)

(if (frame? frame)

(if (binding? binding)

(set-cdr! frame (cons binding (cdr frame)))

(error "Not a binding: " binding))

(error "Not a frame: " frame)))

(define (find-in-frame var frame)

(define (search-helper var bindings)

(if (null? bindings)

#f

(if (eq? var (binding-variable (first bindings)))

(first bindings)

(search-helper var (rest bindings)))))

(if (frame? frame)

(search-helper var (cdr frame))

(error "Not a frame: " frame)))

; environments

(define the-empty-environment '(environment))

(define (extend-environment frame base-env)

(if (environment? base-env)

(if (frame? frame)

(list 'environment frame base-env)

(error "Not a frame: " frame))

(error "Not an environment: " base-env)))

(define (environment? env)

(tagged-list? env 'environment))

(define (enclosing-environment env)

(if (environment? env)

(if (eq? the-empty-environment env)

(error "No enclosing environment of the empty environment")

(third env))

(error "Not an environment: " env)))

(define (environment-first-frame env)

(if (environment? env)

(second env)

(error "Not an environment: " env)))

(define (find-in-environment var env)

(if (eq? env the-empty-environment)

#f

(let ((frame (environment-first-frame env)))

(let ((binding (find-in-frame var frame)))

(if binding

binding

(find-in-environment var (enclosing-environment env)))))))

; name rule

(define (lookup-variable-value var env)

(let ((binding (find-in-environment var env)))

(if binding

(binding-value binding)

(error "Unbound variable -- LOOKUP" var))))

(define (set-variable-value! var val env)

(let ((binding (find-in-environment var env)))

(if binding

(set-binding-value! binding val)

(error "Unbound variable -- SET" var))))

(define (define-variable! var val env)

(let ((frame (environment-first-frame env)))

(let ((binding (find-in-frame var frame)))

(if binding

(set-binding-value! binding val)

(add-binding-to-frame!

(make-binding var val)

frame)))))

;; 5번 문제

(define (env-variables boxed-env)

(frame-variables

(environment-first-frame

(unbox-env boxed-env))))

(define (env-parent boxed-env)

(box-env (enclosing-environment (unbox-env boxed-env))))

(define (env-value sym boxed-env)

(if (symbol? sym)

(let ((binding (find-in-environment sym (unbox-env boxed-env))))

(if binding

(binding-value binding)

#f))

(error "Not a symbol: " sym)))

; primitives procedures - hooks to underlying Scheme procs

(define (make-primitive-procedure implementation)

(list 'primitive implementation))

(define (primitive-procedure? proc) (tagged-list? proc 'primitive))

(define (primitive-implementation proc) (cadr proc))

(define (primitive-procedures)

(list (list 'car car)

(list 'cdr cdr)

(list 'cons cons)

(list 'set-car! set-car!)

(list 'set-cdr! set-cdr!)

(list 'null? null?)

(list '+ +)

(list '- -)

(list '< <)

(list '> >)

(list '= =)

(list 'display display)

(list 'not not)

; ... more primitives ;; 1번 문제

(list '\* \*)

(list '/ /)

(list 'list list)

(list 'cadr cadr)

(list 'cddr cddr)

(list 'newline newline)

(list 'printf printf)

(list 'length length)

(list '<= <=)

(list '>= >=)

(list 'empty? empty?)

(list 'list? list?)

(list 'not not)

(list 'null null)

(list 'eq? eq?)

(list 'append append)

(list 'env-variables env-variables) ;; 5번 문제

(list 'env-parent env-parent)

(list 'env-value env-value)

(list 'caddr caddr) ;; 6번 문제

(list 'cadddr cadddr)

(list 'caadr caadr)

(list 'cdadr cdadr)

(list 'symbol? symbol?)

(list 'pair? pair?)

(list 'number? number?)

(list 'string? string?)

(list 'boolean? boolean?)

))

(define (primitive-procedure-names) (mmap car (primitive-procedures)))

(define (primitive-procedure-objects)

(mmap make-primitive-procedure (mmap cadr (primitive-procedures))))

(define (apply-primitive-procedure proc args)

(apply (primitive-implementation proc) args))

; used to initialize the environment

(define (setup-environment)

(extend-environment (make-frame (primitive-procedure-names)

(primitive-procedure-objects))

the-empty-environment))

(define the-global-environment (setup-environment))

;;;;;;;; Code necessary for question 6

;;

;; This section doesn't contain any user-servicable parts -- you

;; shouldn't need to edit it for any of the questions on the project,

;; including question 5. However, if you're curious, comments provide a

;; rough outline of what it does.

;; Keep track of what depth we are into nesting

(define meval-depth 1)

;; These procedures are needed to make it possible to run inside meval

(define additional-primitives

(list (list 'eof-object? eof-object?)

(list 'read read)

(list 'read-line read-line)

(list 'open-input-file open-input-file)

(list 'this-expression-file-name

(lambda () (this-expression-file-name)))

(list 'pretty-display pretty-display)

(list 'error error)

(list 'apply m-apply))) ;; <-- This line is somewhat interesting

(define stubs

'(require r5rs mzlib/etc print-as-expression print-mpair-curly-braces))

(define additional-names (mmap first additional-primitives))

(define additional-values (mmap make-primitive-procedure

(mmap second additional-primitives)))

(require mzlib/etc)

(define (load-meval-defs)

;; Jam some additional bootstrapping structures into the global

;; environment

(set! the-global-environment

(extend-environment

(make-frame stubs

(mmap (lambda (name)

(m-eval '(lambda (x) x) the-global-environment)) stubs))

(extend-environment

(make-frame additional-names

additional-values)

the-global-environment)))

;; Open this file for reading

(let ((stream (open-input-file (this-expression-file-name))))

(read-line stream) ;; strip off "#lang racket" line

(repl stream)) ;; feed the rest of the definitions into meval

;; Update the meval-depth variable inside the environment we're simulating

(set-variable-value! 'meval-depth (+ meval-depth 1) the-global-environment)

'loaded)

;;4번 문제

(define (one-binding-value? binding) (null? (cdddr binding)))

(define (set-binding-value! binding val)

(if (binding? binding)

(set-cdr! (cdr binding) (cons val (cddr binding)))

(error "Not a binding: " binding)))

(define (unset-binding-value! binding)

(cond

((not (binding? binding)) (error "Not a binding: " binding))

((one-binding-value? binding) (void))

(else

(set-cdr! (cdr binding) (cdddr binding)))))

(define (reset-binding! binding val)

(if (binding? binding)

(set-cdr! (cdr binding) (cons val '()))

(error "Not a binding: " binding)))

(define (unset? exp) (tagged-list? exp 'unset!))

(define (unset-variable exp) (cadr exp))

(define (eval-unset exp env)

(let ((var (unset-variable exp)))

(let ((binding (find-in-environment var env)))

(if binding

(unset-binding-value! binding)

(error "Unbound variable -- UNSET" var)))))

(define (boxed-env? boxed-env) ;;5번 문제

(and

(box? boxed-env)

(environment? (unbox boxed-env))))

(define (box-env env)

(if (environment? env)

(box-immutable env)

(error "Not an environment: " env)))

(define (unbox-env boxed-env)

(if (boxed-env? boxed-env)

(unbox boxed-env)

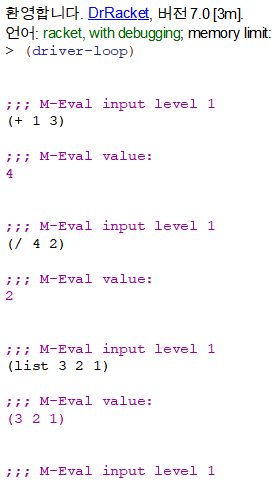
(error "Not an environment: " boxed-env)))

(define (eval-current-env env) (box-env env))

(define (eval-procedure-env exp env)

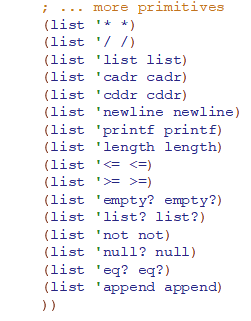
(box-env (procedure-environment (m-eval (second exp) env))))

Problem1

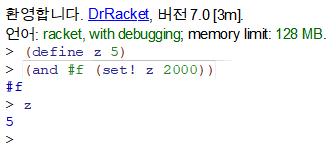


Problem1 추가사항

1.



Problem2



Problem2 추가사항

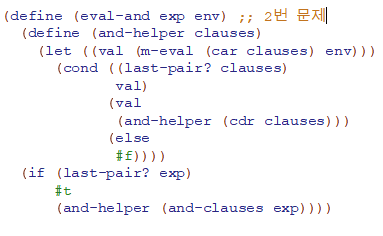
1.



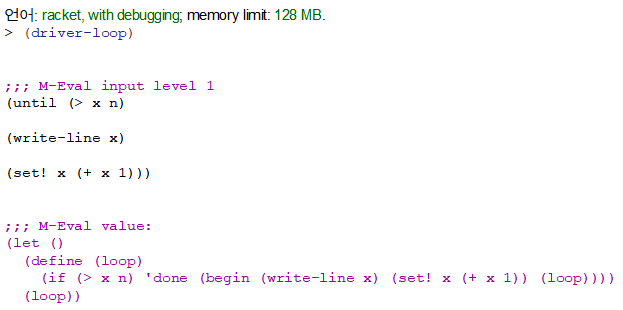
2.



3.



Problem3



Problem3 추가사항

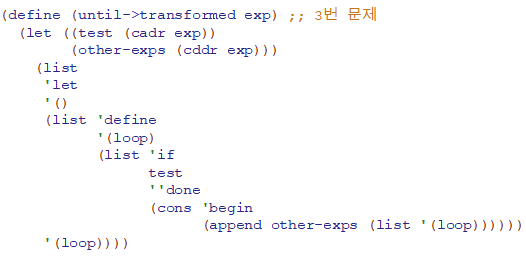
1.



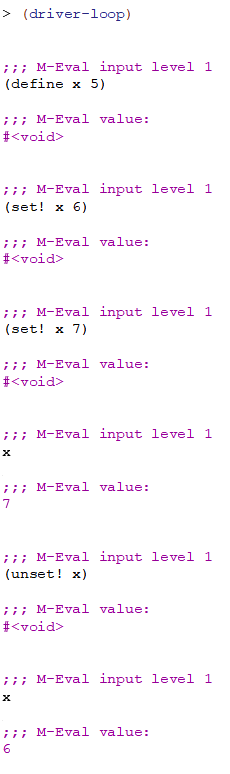
2.



3.



Problem4

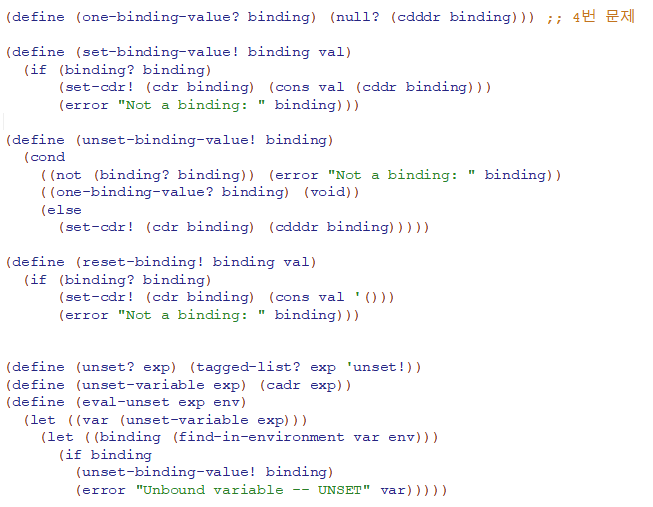


Problem4 추가사항

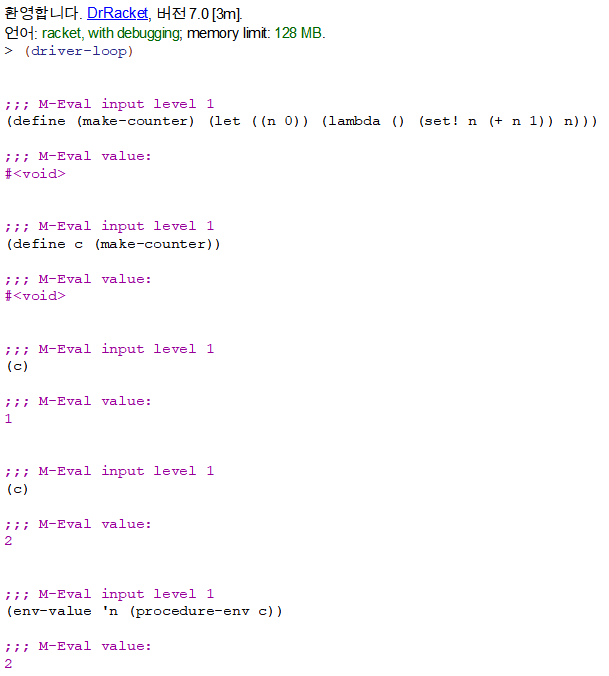
1.



2.

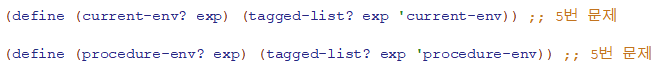


Problem5



Problem5 추가사항

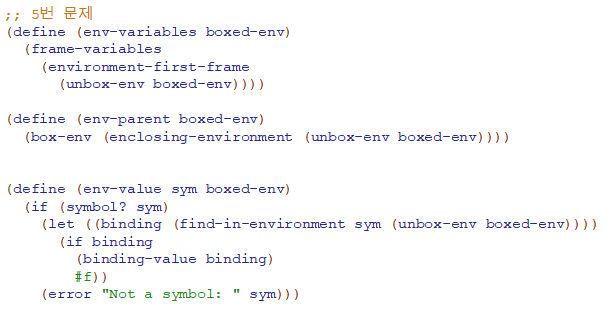
1.



2.



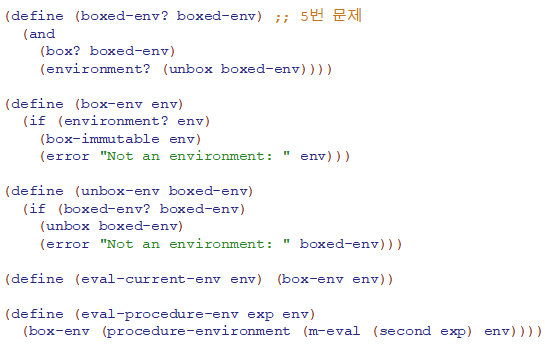
3.



4.



5.



Problem6

