Homework 06

● Graded

Student

Giancarlos Marte

Total Points

90 / 100 pts

Autograder Score

90.0 / 90.0

Passed Tests

Exercise 1. eval-exp apply (1) (6/6)
Exercise 1. eval-exp apply (2) (6/6)
Exercise 1. eval-exp apply+lambda (6/6)
Exercise 1. eval-exp lambdas (12/12)
Exercise 1. eval-exp values (0/0)
Exercise 1. eval-exp variables (12/12)
Exercise 2. eval-term all (26/26)
Exercise 2. eval-term define (10/10)
Exercise 2. eval-term seq + define (6/6)
Exercise 2. eval-term seq + value (6/6)

Question 2

Question 3

0 / 10 pts

– 0 pts Correct

✓ **– 10 pts** Incorrect

Autograder Results

Exercise 1. eval-exp apply (1) (6/6)

Exercise 1. eval-exp apply (2) (6/6)

Exercise 1. eval-exp apply+lambda (6/6)

Exercise 1. eval-exp lambdas (12/12)

Exercise 1. eval-exp values (0/0)

Exercise 1. eval-exp variables (12/12)

Exercise 2. eval-term all (26/26)

Exercise 2. eval-term define (10/10)

Exercise 2. eval-term seq + define (6/6)

Exercise 2. eval-term seq + value (6/6)

Submitted Files

hw6.rkt   3 Comments   Download

```
#|
   ===> PLEASE DO NOT DISTRIBUTE THE SOLUTIONS PUBLICLY <===

  We ask that solutions be distributed only locally -- on paper, on a
  password-protected webpage, etc.

  Students are required to adhere to the University Policy on Academic
  Standards and Cheating, to the University Statement on Plagiarism and the
  Documentation of Written Work, and to the Code of Student Conduct as
  delineated in the catalog of Undergraduate Programs. The Code is available
  online at: http://www.umb.edu/life_on_campus/policies/code/

|#
;; PLEASE DO NOT CHANGE THE FOLLOWING LINES
#lang typed/racket
(require "hw6-util.rkt")
(provide (all-defined-out))
;; END OF REQUIRES

;; Exercise 1
(: eval-exp (memory handle d:expression -> (eff memory d:value)))
(define (eval-exp mem env exp)
  ; mem is M
  ; env is E
  (match exp
    [(? d:value?)
     ; Return: v ▶ M
     (eff mem exp)]
    [(? d:variable?) ; exp is x
     ; Return: E(x) ▶ M
     (define temp (environ-get mem env exp))
     (eff mem temp)]
    [(d:lambda x t)
     ; Return: {E, λx.t} ▶ M
     (define close (d:closure env x t))
     (eff mem close)]
    [(d:apply ef ea)
     (match (eval-exp mem env ef)
       ;; ef □E {Ef, λx.tb} ▶ M1
       [(eff M1 (d:closure Ef x tb))
        ;; ea □E va ▶ M2
        (define va&M2 (eval-exp M1 env ea))
        (define va (eff-result va&M2))
        (define M2 (eff-state va&M2))

        ;; Eb ← Ef + [x := a] ▶ M3
        (define M3&Eb (environ-push M2 Ef x va))
        (define Eb (eff-result M3&Eb))
        (define M3 (eff-state M3&Eb))

        ;; tb Eb vb ▶ M4
        (define vb&M4 (eval-term M3 Eb tb))
        (define vb (eff-result vb&M4))
        (define M4 (eff-state vb&M4))

        ;; Return: vb ▶ M4
        vb&M4])]))

;; Exercise 2
(: eval-term (memory handle d:term -> (eff memory d:value)))
(define (eval-term mem env term)
  (match term
    [(d:define x e)
     ;; e E v ▶ M1
     (define v&M1 (eval-term mem env e))
     (define v (eff-result v&M1))
     (define M1 (eff-state v&M1))

     ;; E ← [x := v] ▶ M2
     (define M2 (environ-put M1 env x v))

     ;; Return: void ▶ M2
     (eff M2 (d:void))]

    [(d:seq t1 t2)
     ;; t1 E v1 ▶ M1
     (define v1&M1 (eval-term mem env t1))
     (define v1 (eff-result v1&M1))
     (define M1 (eff-state v1&M1))

     ;; t2 E v2 ▶ M2
     (define v2&M2 (eval-term M1 env t2))
     (define v2 (eff-result v2&M2))

     ;; Return: v2 ▶ M2
     v2&M2]
    [(? d:expression?)
     (eval-exp mem env term)]))

;; Exercise 3 (Manually graded)
#|
Racket returns #<procedure:funct_name> when you run a function name.
λd does not do this.
```

Instructor | 05/10 at 6:35 pm

This is not a difference in the *variable binding semantics*.

```
ex:
(define (f x) 10)
f

This returns #<procedure:f> when run on racket.
In λd it will most likely just return void.
```

Instructor | 05/10 at 6:38 pm

Does it? You have an interpreter for λD, so you don't have to guess, you can find out!

```

Also booleans are supported in racket, but not in λd.
```

Instructor | 05/10 at 6:37 pm

The exercise says "we are not interested in features that are implemented in one language but are not in another".

```
|#

```