Carleton University
Department of Systems and Computer Engineering
SYSC 3101 - Programming Languages - Winter 2019

Lab 6 - The Environment Model of Evaluation

References

- Evans, *Introduction to Computing*, http://computingbook.org
    - Chapter 9, http://computingbook.org/Mutation.pdf, Section 9.2, *Impact of Mutation*, up to the end of 9.2.2, *Evaluation Rules with State*
    - Chapter 10, http://computingbook.org/Objects.pdf, Section 10.1, *Packaging Procedures and State*, up to the end of Section 10.1.1, *Encapsulation*

Instructions

1. For each exercise, you don't have to draw a detailed sequence of diagrams that depict what happens, step-by-step, as each procedure definition and procedure call is evaluated. One or two diagrams, along with a few words of explanation, should be sufficient to demonstrate that you understand how environments are created and used.
2. To prepare the diagrams, neat hand-drawn diagrams are acceptable, but if you prefer, you can use the drawing tools in your favourite editor.

Exercise 1

This exercise deals with a solution to Question 6 from the midterm exam.

Draw a diagram that depicts the environment created when these expressions are evaluated:

```
(define dx 0.00001)

(define (deriv g)
  (lambda (x) (/ (- (g (+ x dx)) (g x)) dx)))

(define (cube x) (* x x x))
((deriv cube) 5)  ; Returns 75.00014999664018
```

Use the notation presented in Evans' book (see the sections listed in *References*, above).

Exercises 2 and 3 deal with a solution to Question 5 from the midterm exam.

Exercise 2

Recall that Racket's procedure `(build-list n f)` constructs a list by applying procedure `f` to the natural numbers between `0` and `n`-1, inclusive. In other words, `(build-list n f)` produces the same result as:

```
(list (f 0) (f 1) .. (f (- n 1)))
```

For example, when this expression is evaluated:

```
(build-list 5 (lambda (x) (* x x)))
```

`build-list` applies the `lambda` procedure to `0`, `1`, `2`, `3` and `4`, and returns `'(0 1 4 9 16)`.

Assume we are using a version of Racket that doesn't have `build-list`, and have therefore written our own implementation of this procedure:

```
(define (build-list n f)

  ; build-up creates a list by applying procedure f to the integers
  ; from m to n-1, m < n, in order. The first list element is the
  ; value produced by (f m), and the last list element is the value
  ; produced by (f (- n 1)).

  (define (build-up m)
    (if (= m n)
        '()
        (cons (f m) (build-up (+ m 1)))))

  (build-up 0))
```

Draw a diagram that depicts the environment created by:

```
(build-list 3 (lambda (x) (* x x)))
```

Use the notation presented in Evans' book. Your diagram doesn't have to show the frame created each time `cons` is called, because `cons` is a primitive procedure. Your diagram must show the frames created by the calls to `build-list`, `build-up` and the `lambda` procedure, as well as the procedure objects.

Exercise 3

Procedure `multiply-all` takes a list of numbers and returns the product of the numbers in the list:

```
(define (multiply-all nums)
  (if (empty? nums)
      1
      (* (car nums) (multiply-all (cdr nums)))))

> (multiply-all '(2 3 4))    ; returns 24
> (multiply-all '(3))        ; returns 3
> (multiply-all '())         ; returns 1
```

Procedure `factorial` takes a positive integer *n* and returns *n*!

```
(define (factorial n)
  (multiply-all (build-list n (lambda (x) (+ x 1)))))
```

Draw a diagram that depicts the environment created by: `(factorial 3)`

Don't duplicate your solution to Exercise 1. Your diagram should show the frames created when `factorial`, `multiply-all` and `build-list` are called, but you don't have to show the frames created when `build-up` and the `lambda` procedure are called.

Exercise 4

This example was presented in one of the lectures:

```
(define (make-upcounter counter)
  (lambda ()
    (set! counter (+ counter 1))
    counter))
```

(a) Draw a diagram that depicts the environment created by:

```
(define up-count1 (make-upcounter 10))
```

Use the notation presented in Evans' book.

(b) Draw a diagram that depicts the environment that is created the first time the expression `(up-count1)` is evaluated. Use the notation presented in Evans' book.

(c) Figure 10.1 in Evans' book depicts the environment created by a `make-counter` procedure that uses a `let` expression to initialize the variable that stores the counter value. Compare your diagrams from (a) and (b) to Figure 10.1. Ensure that you can explain the differences between that figure and your diagrams.