

# Homework 10: Implementing the Environment Model

CIS 352: Programming Languages

28 March 2017, Draft

## Administrivia

- Turn in the Part II problems in the CIS 352 submissions box. If you trade ideas with another student, document it in your cover sheet.
- Turn in the Part I problems in via Blackboard. Include your:  
(i) source files, (ii) transcripts of test runs, and (iii) cover sheet.

## Background

In this assignment you will implement versions of LFP with

- call-by-value evaluation and dynamic scoping,
- call-by-name evaluation and lexical scoping, and
- call-by-name evaluation and dynamic scoping.

Figure 1 gives the big-step evaluation rules for each of these. Additionally, the version of LFP we'll be working with has two new commands:

1. The **return** command works pretty much as it does in C.

$$\text{Return: } \frac{\rho \vdash (e, s) \Downarrow (v, s')}{\rho \vdash (\text{return } e, s) \Downarrow (v, s')}$$

2. The **print** command does what you expect — sort of.

$$\text{Print: } \frac{\{ \rho \vdash (e_i, s_i) \Downarrow (v_i, s_{i+1}) \}_{i=1, \dots, n}}{\rho \vdash (\text{print } (e_1, \dots, e_n), s_1) \Downarrow (\text{skip}, s_{n+1})} \left( \begin{array}{l} \text{the values of } e_1, \\ \dots, e_n \text{ are printed} \end{array} \right)$$

The **print** command is implemented on the cheap (i.e., via Haskell's trace function). So because of Haskell's laziness, **print**'s can show up in the output in bizarre places and orders. (Try evaluating `et0a` and `et0b` to see what I mean.)

## Part I: Programming Problems

You'll need the files in <http://www.cis.syr.edu/courses/cis352/code/LFP2/>; `LFP2bs.hs` has the call-by-value/lexical-scoping version of LFP.

### ❖ Problem 1 (16 points) ❖

Change the version of LFP of `LFP2bs.hs` to *call-by-value/dynamic-scoping*. Test your program on `et1`, `et2`, `et3`, `et4`, `et5`, and `et6`.

`runAll` runs all these tests for you.

### ❖ Problem 2 (16 points) ❖

Change the version of LFP of `LFP2bs.hs` to *call-by-name/lexical-scoping*. Test your program on `et1`, `et2`, `et3`, `et4`, `et5`, and `et6`.

`runAll` runs all these tests for you.

## Grading Criteria

- The homework is out of 100 points.
- Running the required tests (for problems 1 and 2: `et1`, ..., `et6` and for problem 3: `et1`, ..., `et5`) suffices for testing.
- Omitting your name(s) in the source code loses you 5 points.

## Call-by-value, lexical-scoping

$$\begin{aligned}
 & \rho \vdash \langle e_1, s \rangle \Downarrow_V \langle (\lambda x. \hat{e}_1) \hat{\rho}_1, s' \rangle \\
 & \rho \vdash \langle e_2, s' \rangle \Downarrow_V \langle v_2, s'' \rangle \\
 \text{App: } & \frac{\hat{\rho}_1[x \mapsto v_2] \vdash \langle \hat{e}_1, s'' \rangle \Downarrow_V \langle v, s''' \rangle}{\rho \vdash \langle (e_1 \ e_2), s \rangle \Downarrow_V \langle v, s''' \rangle} \\
 \text{Var: } & \frac{}{\rho \vdash \langle x, s \rangle \Downarrow_V \langle v, s \rangle} (v = \text{lookup}(\rho, x)) \\
 \text{Fun: } & \frac{}{\rho \vdash \langle \lambda x. e, s \rangle \Downarrow_V \langle (\lambda x. e) \rho, s \rangle}
 \end{aligned}$$

## Call-by-name, lexical-scoping

$$\begin{aligned}
 & \rho \vdash \langle e_1, s \rangle \Downarrow_N \langle (\lambda x. \hat{e}_1) \hat{\rho}_1, s' \rangle \\
 & \hat{\rho}_1[x \mapsto e_2 \rho] \vdash \langle \hat{e}_1, s' \rangle \Downarrow_N \langle v, s'' \rangle \\
 \text{App: } & \frac{}{\rho \vdash \langle (e_1 \ e_2), s \rangle \Downarrow_N \langle v, s'' \rangle} \\
 \text{Var: } & \frac{\rho' \vdash \langle e, s \rangle \Downarrow_N \langle v, s' \rangle}{\rho \vdash \langle x, s \rangle \Downarrow_N \langle v, s' \rangle} \left( e \rho' = \text{lookup}(\rho, x) \right) \\
 \text{Fun: } & \frac{}{\rho \vdash \langle \lambda x. e, s \rangle \Downarrow_N \langle (\lambda x. e) \rho, s \rangle}
 \end{aligned}$$

## Call-by-value, dynamic-scoping

$$\begin{aligned}
 & \rho \vdash \langle e_1, s \rangle \Downarrow_V \langle (\lambda x. e'_1), s' \rangle \\
 & \rho \vdash \langle e_2, s' \rangle \Downarrow_V \langle v_2, s'' \rangle \\
 \text{App: } & \frac{\rho[x \mapsto v_2] \vdash \langle e'_1, s'' \rangle \Downarrow_V \langle v, s''' \rangle}{\rho \vdash \langle (e_1 \ e_2), s \rangle \Downarrow_V \langle v, s''' \rangle} \\
 \text{Var: } & \frac{}{\rho \vdash \langle x, s \rangle \Downarrow_V \langle v, s \rangle} (v = \text{lookup}(\rho, x)) \\
 \text{Fun: } & \frac{}{\rho \vdash \langle \lambda x. e, s \rangle \Downarrow_V \langle (\lambda x. e), s \rangle}
 \end{aligned}$$

## Call-by-name, dynamic-scoping

$$\begin{aligned}
 & \rho \vdash \langle e_1, s \rangle \Downarrow_N \langle \lambda x. e'_1, s' \rangle \\
 & \rho[x \mapsto e_2] \vdash \langle e'_1, s' \rangle \Downarrow_N \langle v, s'' \rangle \\
 \text{App: } & \frac{}{\rho \vdash \langle (e_1 \ e_2), s \rangle \Downarrow_N \langle v, s'' \rangle} \\
 \text{Var: } & \frac{\rho \vdash \langle e, s \rangle \Downarrow_N \langle v, s' \rangle}{\rho \vdash \langle x, s \rangle \Downarrow_N \langle v, s' \rangle} (e = \text{lookup}(\rho, x)) \\
 \text{Fun: } & \frac{}{\rho \vdash \langle \lambda x. e, s \rangle \Downarrow_N \langle \lambda x. e, s \rangle}
 \end{aligned}$$

**Note:** I've put closures in boxes. Also, the hat in  $\hat{\rho}$  is just a decoration.

Figure 1: Key operational semantics rules

❖ *Problem 3 (16 points)* ❖

Change the version of LFP of LFP2bs.hs to *call-by-name/dynamic-scoping*. Test your program on et1, et2, et3, et4, and et5.

runAllbut6 runs all these tests for you.

*Part II: Monadic IO Programming Problems*

Read the *Input and Output* chapter of (Lipovača, 2011) before working these problems. For testing for these problems all you need to do is run at least four sample runs with a boundary cases included.<sup>1</sup>

These problems are a warm up for the parsing code that is coming up. Use a fresh file (i.e., not one of the LFP2 files) for these problems.

<sup>1</sup> E.g., the empty list case for Problem 4.

❖ *Problem 4 (12 points)* ❖

Write a function

```
showHisto :: IO ()
```

that repeatedly reads Ints (one per line) until finding a negative value and then outputs a histogram of the (nonnegative) Ints values in the order they were entered. For example:

You may find the replicate function from Data.List handy. SUGGESTION: Break down the problem with helper functions.

```
*Main> showHisto
10
3
7
-1

*****
***
*****
```

❖ *Problem 5 (10 points)* ❖

Write a function

```
ask :: String -> IO Char
```

that writes its string argument as a user prompt, reads the entire next line of input, and returns (as in return) the character of the first line of that input. However, if the input line is empty, then ask should return the character 'n'. For example,

```
*Main> ask "Do you like rhubarb?"
Do you like rhubarb? no!!!
'n'
```

*Part III: Written Problems*

For each of the following, figure out by hand what the LFP program prints or returns.

**Important:** You can use your code to check your work, but working out these by hand will give you some practice for the next quiz.

❖ *Problem 6 (6 points)* ❖

Consider:

```

let x = 10
  in let f = ( $\lambda z. (x + z)$ )
    in let x = 100 in (f 4)

```

What is returned under:

- (a) (4 points) call-by-value/lexical scoping?
- (b) (4 points) call-by-value/dynamic scoping?

❖ *Problem 7 (6 points)* ❖

Consider:

```

let x = 100
  in let f = ( $\lambda y. (x * y)$ )
    in let g = ( $\lambda x. (f 3)$ )
      in print ((f 5), (g 10))

```

What is printed under:

- (a) (4 points) call-by-value/lexical scoping?
- (b) (4 points) call-by-value/dynamic scoping?

❖ *Problem 8 (6 points)* ❖Assume location  $X1$  (or  $\ell_1$  if you prefer) starts out with contents 0.

Consider:

```

let f =  $\lambda y. \{ X1 := !X1 + 50; \text{return } y \}$ 
  in let y = (f 3)
    in {  $X2 := y * y; \text{return } (!X1 + !X2)$  }

```

What is returned under:

- (a) (4 points) call-by-value/lexical scoping?
- (b) (4 points) call-by-name/lexical scoping?

❖ *Problem 9 (6 points)* ❖

Assume `X1` starts out with contents `0`. Consider:

```
let f = λy. { X1 := !X1 + 10; return y }
in let g = λz. 100
    in let w = (g (f 1000)) in (w + !X1)
```

What is returned under

- (a) (4 points) call-by-value/lexical scoping?
- (b) (4 points) call-by-name/lexical scoping?

❖ *Problem 10 (6 points)* ❖

Assume `X1` starts out with contents `0`. Consider:

```
let tick = 100
in let tock = λu. { X1 := !X1 + tick; return !X1 }
    in let tick = 10 in (tock (tock (tock 0)))
```

What is returned under:

- (a) (2 points) call-by-value/lexical scoping?
- (b) (2 points) call-by-value/dynamic scoping?
- (c) (2 points) call-by-name/lexical scoping?
- (d) (2 points) call-by-name/dynamic scoping?

## References

H. Abelson and G. J. Sussman. The environment model of evaluation. In *Structure and Interpretation of Computer Programs*, chapter 3.2. MIT Press, Cambridge, MA, USA, 2nd edition, 1996. ISBN 0262011530. URL <https://mitpress.mit.edu/sicp/full-text/book/book-Z-H-21.html>.

W. Cook. Anatomy of programming languages. Technical report, Department of Computer Science, UT Austin, 2013. URL <http://www.cs.utexas.edu/~wcook/anatomy/>.

M. Lipovača. *Learn You a Haskell for Great Good!* No Starch Press, 2011. URL: <http://learnyouahaskell.com>.

### Sample Problem

For the following program, what does it return under: **(a)** call-by-value/lexical-scoping? **(b)** call-by-value/dynamic-scoping?

Note: This is just Problem 4 with some numbers changed.

```

let x = 100
  in let f = ( $\lambda z. (x + z)$ )
    in let x = 20
      in (f 3)
    
```

#### An answer

| Part (a)   | ENVIRONMENT  | EXPRESSION   |
|------------|--|--|
| $\rho_0$ : | $\emptyset$  | <b>let</b> x = 100 <b>in</b> ...                     |
| $\rho_1$ : | $\begin{array}{c} \uparrow \\ x \mapsto 100 \end{array}$                       | <b>let</b> f = ...                                   |
| $\rho_2$ : | $\begin{array}{c} \uparrow \\ f \mapsto \lambda z. (x + z) \rho_1 \end{array}$ | <b>let</b> x = 20 <b>in</b> ...                      |
| $\rho_3$ : | $\begin{array}{c} \uparrow \\ x \mapsto 20 \end{array}$                        | (f 3)  |
| $\rho_4$ : | $\begin{array}{c} \uparrow \\ z \mapsto 3 \end{array} \rightarrow \rho_1$      | $x + z$<br>$= \rho_1(x) + \rho_1(z) = 100 + 3 = 103$ |

| Part (b)   | ENVIRONMENT   | EXPRESSION   |
|------------|---|--|
| $\rho_0$ : | $\emptyset$   | <b>let</b> x = 100 <b>in</b> ...                   |
| $\rho_1$ : | $\begin{array}{c} \uparrow \\ x \mapsto 100 \end{array}$                | <b>let</b> f = ...                                 |
| $\rho_2$ : | $\begin{array}{c} \uparrow \\ f \mapsto \lambda z. (x + z) \end{array}$ | <b>let</b> x = 20 <b>in</b> ...                    |
| $\rho_3$ : | $\begin{array}{c} \uparrow \\ x \mapsto 20 \end{array}$                 | (f 3)  |
| $\rho_4$ : | $\begin{array}{c} \uparrow \\ z \mapsto 3 \end{array}$                  | $x + z$<br>$= \rho_4(x) + \rho_4(z) = 20 + 3 = 23$ |