Lab: Recursive procedures and the LETREC language
CSC 312, “Programming language implementation”
Department of Computer Science · Grinnell College
September 16–18, 2015

Today’s pair-programming partnerships are:

• Mr. Zahid and Ms. Byrne
• Mr. Nordin and Mr. Mataire
• Mr. Boateng Asante and Ms. DeWitt
• Mr. Champagne and Mr. Schlager
• Mr. Dewey and Mr. Mulhall
• Ms. Jervis and Ms. Kyaruzi
• Ms. Liddell

In each pair, the partner listed first should log in and manage the keyboard and screen.

Exercises

A. Create a subdirectory called LETREC in the directory you’re using for labs in this course and copy my implementation of the LETREC programming language into it. This implementation in /home/stone/courses/languages/code/LETREC comprises six language-specific modules:

• expvals-and-environments.scm
• tokens.scm
• syntax-trees.scm
• scanner.scm
• parser.scm
• interpreter.scm

B. Consider this LETREC program, from page 82 of the textbook:

```letrec double(x)
  = if zero?(x) then 0 else -((double -(x, 1)), -2)
in (double 6)
```

(1) In the course of the execution of this program, how many times will the expression (double 6) be evaluated? In what environment will the evaluation(s) occur? (List the variables that will be bound in that environment and the values to which they will be bound.)

(2) In the course of the execution of this program, how many times will the if-expression be evaluated? In what environment will the first of these evaluations occur?

C. Study the definitions of the environment data type and the apply-env procedure in the expvals-and-environments.scm module. In the apply-env procedure, the call to the constructor proc-val to build and return a completely new expressed procedure value with a freshly built closure inside it seems a little surprising. Why is it necessary for this lookup procedure to create such values on the fly like this?

D. Exercise 3.35 on pages 85 and 86 of the textbook describe an alternative implementation of environments that avoids the inefficiency of re-creating the closure each time it is needed. It creates the closure only once and stores it as the sole element of a length-1 vector (sometimes called a “box”).
(1) Why don’t we just use the implementation that the authors suggest, but use the procedure value itself as the value of the procedure name, rather than a vector containing that value? What’s the benefit of storing the closure in a vector?

(2) Modify the implementation of LETREC so that environments containing bindings for the names of recursive procedures are implemented in the way that the authors suggest. Test your modified implementation.

E (if time permits): Use your implementation of the DYNAMIC-PROC language, from the September 14 lab, to do exercise 3.37 from the textbook (page 87), which explores a different way to get recursive procedures in a language.