Lab: Environments and the LET language
CSC 312, “Programming language implementation”
Department of Computer Science · Grinnell College
September 11, 2015

Today’s pair-programming partnerships are:

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

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

Exercises

A. Copy my implementation of the LET programming language into the directory you’re using for labs in this course. You’ll need ten .scm files altogether: the three modules test.scm, character-sources.scm, and natural-numbers.scm from the /home/stone/courses/languages/code directory, which will be useful in all of our language implementations, and seven language-specific modules from the LET subdirectory of that directory: expval.scm, tokens.scm, syntax-trees.scm, environments.scm, scanner.scm, parser.scm, and interpreter.scm. You’ll need to create a LET subdirectory of your own for these last seven files. The other three should be one level up in the structure, in the directory containing the LET subdirectory.

B. Load the interpreter into DrRacket and run it. If no output (specifically, no error messages) appear in the Interactions window, you have a working implementation of LET. Otherwise, figure out what the problem is and correct it.

C. The interpreter imports the scan&parse procedure from parser.scm, making it available in the Interactions window as well. Use it to construct the abstract syntax tree for the LET program

```
let counter = 100
    in let index = -(counter, i)
        in if zero?(counter)
            then x
            else -(-(counter, index), x)
```

D. Use the LET interpreter to compute the value of the program in exercise C.

E. LET’s initial environment contains bindings for the Roman numerals i, v, and x. Change the initial environment so that it also binds l to 50, c to 100, d to 500, and m to 1000. Test the interpreter to make sure that your changes had the intended effects.

F. What happens if a let-expression in LET is the body of another let-expression that attempts to bind the same variable to a different value? Which binding, if either, takes precedence? Should the language implementation signal an error if it encounters a construction of this sort?
G. What happens if a let-expression in LET attempts to bind a variable that was already bound in the initial environment to a different value? For instance, in

    let x = 3142
    in -(x, v)

does the initial binding of 10 to x take precedence over the local binding of x to 3142, or vice versa? Should the language implementation signal an error if it encounters a construction of this sort?