CS450: Structure of Higher Level Languages

Spring 2018 Assignment 4, part 1

Due: Wednesday, February 28

Taken from assignments by Profs. Carl Offner and Ethan Bolker

Guidelines

This part is about Assignment, local state and the environment model.

There is a lot of new material here. It will take getting used to. You will find this material in Sections 3.1 and 3.2 of the text.

For this first part of the assignment, you will hand in problems 2 and 5 on paper (in class), and the rest of the problems electronically:

- Problems 2 and 5 should be written out on paper and passed in at the beginning of class on Wednesday, February 28. Handwriting is fine, but please make them neat, clear, and easy to read. If you're proficient with a graphics software that's fine, but don't bother too much. I will not accept papers after the class has started. That means 5:30 PM sharp – not 5:35 PM. If for some reason you cannot be present when the class starts, make sure that these problem are in my mailbox in the Math/CS office no later than 5:00 PM, or alternatively send a scanned copy to my email by that time (I'm usually leaving for class at 5:20).

  Please note that problems 2 and 5 are exercises that show that you understand the environment model. To do this, you have to go back and do exactly what we discussed in class. If you just think you sort of understand things and write down what seems to be correct, you will almost certainly get these problems wrong. Many students in the past have done just that, and then they are astonished to find out that what they did makes no sense at all.

- Put your answers to the rest of the problems in file **ASanswers.scm** in your new project directory .../cs450/hw4, with discussion when required as Scheme comments. I will load and test your file, so be sure it's bug free. Anything that isn't yet working should be a stub or a comment. Any tests that you ran that are in this file should also be commented out.

Questions

1. Rewrite the make-account procedure on page 223 so that it uses lambda more explicitly. Create several versions, as follows. (Important: please do exactly what each of the three following versions specifies; no more and no less.)

   (a) First version: First, replace

   ```
   (define (make-account balance)
    ...)
   ```

   by

   ```
   (define make-account-lambda
     (lambda ...
      ...
   ```

```
        )
      )
```

Then, for the first two internal procedure definitions, replace `(define (proc args) ...)` with `(define proc (lambda (args) ...))`.

Finally, replace the

```
(define (dispatch ...)
...
...)
dispatch)
```

construction with a lambda expression which is evaluated and returned (but of course not called). As indicated above, call this procedure make-account-lambda.

(b) Second version: Start with a copy of the first version. Then inline the internal procedures `deposit` and `withdraw`. That is, replace references to them by the bodies of the procedures. Then you can eliminate the definitions of those procedures. Call this procedure make-account-inline.

(c) Third version (A little extra credit): Start with a copy of the second version. I don't know how to say this without doing it for you, but you might then notice that a lambda can be factored out of the cond in your last version. (If you don't know what I'm talking about here, just ignore this part of the problem.) If you do this, call this new version `make-account-inline-factored`.

Note that none of these three versions of `make-account` contains a `dispatch` procedure.

2. Consider the procedure new-withdraw which we talked about in Lecture 6 and which we discussed again in Lecture 7. The implementation of that procedure is sketched in Figure 10 on page 8 of Lecture 7 (the figure numbers refer to the class notes at lec07.pdf).

(a) I want you to draw what that picture looks like after

```
(new-withdraw 25)
```

is evaluated.

(b) Then draw a second picture that shows the situation after

```
(new-withdraw 30)
```

is subsequently evaluated.

3. Exercise 3.2 (page 224). Please read the statement of the problem very carefully. It tells you precisely what you should do. Please do exactly what it says. And do not use any global variables in doing this problem.

4. Exercise 3.3 (page 225). In doing this problem, build on your solution to Problem 1. In fact, see if you can use one of your solutions to Problem 1 as a "black box" – that is, make the solution to this problem a "wrapper" procedure that just invokes one of the versions of `make-account` from Problem 1, after handling password checks. In this way, you don't have to copy any of the body of the original `make-account` procedure. (It isn't necessary that you do it this way – this is just a suggestion.)

   Call your new function `make-pw-account`. And yes, I really mean this – note that this is different from what the book says.

5. Exercise 3.9 (page 243). Let's make it simpler, however: show how to compute (factorial 3) (rather than (factorial 6)).

   Be sure to read the footnote. And follow the construction that I gave in class exactly. You may think the pictures should look different. And if something bothers you about how the pictures look, you should write about it in your notes.txt file. But the construction that I specified is actually what happens internally. You will have to understand it for later assignments.