CS 450 Homework 1

Carl Offner

Spring 2022

(Taken from a similar assignment due to Prof. Bolker)

Part I. Due Wednesday, January 26.

1. Apply for a department Unix account in the usual way. (And talk to the operators in the Unix lab if you are not familiar with this---they will show you how to do this.)

   Do that even if you already have a Unix account, so that you can tell the system that you are enrolled in this course. Do not **ever** change the protections on this account (or on any files or subdirectories in it).

   When I send email to the class, I will send it to the cs450-1 email list. This list consists of everyone who has signed up for the class in this way, and the email will go to your email address at umb.edu. So be sure to read your email there, or forward it to some place where you *will* read it.

2. Read pages 1-21 of the text (Abelson and Sussman's Structure and Interpretation of Computer Programs) once over lightly, and try not to worry about what doesn't seem clear. Answer the following 4 exercises:

   For this (Part 1 of homework 1) **only**, the work must be handed in in the class, before the class starts, on Wednesday, January 26.

   With a couple of very small exceptions, all the other homework assignments (including Part 2 below) will be collected electronically by me, as explained below and in the assignments.

   1. Problem 1.1 (page 20) Work this out with pencil and paper, thinking it through by yourself **before** you run Scheme to see what it really does. Then evaluate these expressions in Scheme to make sure you got them right. And if you get something wrong, make sure you understand the mistake you made. And explain that mistake in your work. (And I won't think badly of you if you do this—I'll be impressed, actually.)

   2. Problem 1.3 (page 21). When it says "Define a procedure...", what it means is that you have to write actual Scheme code. And you have to try it out to make sure it works!

3. (Call this **Problem A** on your paper, so I'll know which one it is.) Suppose we have this code:

```
(define (f x) (* x x))
```

   1. What is this turned into internally when the Scheme interpreter reads this definition?
   2. Given this definition, and given the following expression

```
(f (+ 3 5))
```

      1. Show how this expression is evaluated using applicative-order evaluation.
      2. Show how this expression is evaluated using normal-order evaluation.

4. For extra credit: Problem 1.5 (page 21). And notice that the problem asks for an explanation. If all you put is the answer but not a clear and convincing explanation for it, I'm not even going to read it. Please keep this in mind—it will apply to virtually everything we do.

To test your answers and your understanding you will want to be able to play with Scheme while you read the text. To run the UMB Scheme interpreter simply type

```
% scheme
```

at the Unix prompt. Then enjoy.

To work with a file of Scheme code rather than simply typing at the Scheme prompt you can redirect input (and output) with

```
% scheme < somefile.scm > somefile.out
```

or `(load "somefile.scm")` while in Scheme.

Note: Your answers to the exercises should come along with (or be incorporated in) a short essay in which you discuss what you discovered about Scheme. How does it compare to other languages you know? Did any of its responses surprise you? What is the largest integer Scheme will handle? What message appears when you type '+' at the Scheme prompt (and what does the message mean)? What other experiments did you perform? What do you like or dislike (so far) about Scheme?

This essay and the others I will regularly ask for are an important part of how you learn, and an important part of your grade. I usually read them **before** I read your code. So take them seriously.

Part II. Due Monday, January 31, 5:00 PM.

(Collected electronically.)

When your application for a Unix account for this course was processed, a `cs450` subdirectory of your home directory was created for you. You are to do all your work for this course there. Do not **ever** change the protections on this directory, or on any files or subdirectories in it. Sometimes I have seen students who think that others can see what is there, and try to read-protect it. Regardless of what it might look like to you, these directories are actually set up so that only you and I and the grader can see what is there. So don't worry, and don't change the permissions. If you do, lots of stuff will break. You don't want that to happen. (For one thing, it will make it impossible for me to collect your homework, and you won't get credit for it.)

Create a `hw1` subdirectory of your `cs450` directory. Do your work for this assignment there.

Emacs knows about Scheme. It edits files with the `.scm` extension in `Scheme mode`. In that mode, the `tab` key will indent code correctly. If you are not an emacs user you should definitely learn that editor now. It's much more powerful than vi; that power will save you lots of programming time.

One of the reasons I'm making a big point of this is that your code needs to be indented properly so that it can be understood. I (and the grader as well) have to be able to read your code and understand it. If the indentation is messed up, I can't do this, and I will definitely take off points. I'm not trying to be mean about this. This is just an important thing that you have to be able to do.

And by the way, I occasionally have students who try to do their work in Microsoft Word and then somehow turn it back into plain text. They end up wasting an incredible amount of time doing this, and their work looks like a mess. Don't fall into this trap. Use emacs, and if you need help, just ask me, or send me email.

Rapidly read through the Scheme language report ("Revised^5 Report on the Algorithmic Language Scheme"; this is usually referred to as "R5RS"). Do not try to learn everything that's there. At this point you won't really understand very much of it. Skip all the hard parts. Use this reading to learn the kinds of things that can be found there, so you can look things up when you need to. And you **will** need to later.

Put the answers to the exercises listed below in a file named `ASanswers.txt` in your `hw1` directory. Your answers should be written out in complete paragraphs. Your work should be sufficiently self contained so that three months from now you will be able to look at them and understand them without having to go back to the book to see what the question was, or the point of the question.

This file should be a plain ASCII text file. You will have to rely on white space for formatting. I am not going to deal with any files that contain extra or hidden formatting characters, whether it is Microsoft Word or HTML. And please make sure that all your lines in this file are 80 characters or less in length. If you put this line

```
(setq-default fill-column 80)
```

in your `~/.emacs` file, this will help you. But you still have to make sure that your lines haven't run over. (And yes, I really mean this. I'm taking off points if lines run over. If you don't know what I mean, ask me or send me email.)

Remember that unix file names are case sensitive. Here are three file names that are incorrect:

- `asanswers.txt` WRONG!
- `ASanswer.txt` WRONG!
- `ASanswers.doc` WRONG!

Do you see what is wrong with each of them? If you name your file even slightly incorrectly, my collecting script will not find it, and I will never see your homework. *It is not my responsibility to go rummaging around in your directory to try to figure out what you have named your files.* So please be careful with file names. It will save us both a lot of grief.

Here are the exercises you should do:

1. Problem 1.6 (page 25). Note that this exercise calls for an explanation, not a program.
2. Problem 1.12 (page 42). To be precise, let us define the function `p` for "Pascal" which takes two arguments:

   ```
   r -- the row number
   e -- the element number in the row
   ```

   So `(p r e)` is the value of element `e` in row `r`. And we number the rows starting from 0, and we also number the elements in each row starting from 0. (That is the usual convention.) Thus for instance,

   ```
   (p 0 0) evaluates to 1
   (p 2 1) evaluates to 2
   (p 4 2) evaluates to 6
   ```

   and so on. The problem then is to write a recursive Scheme procedure for the function `p`. And I really mean "recursive". For this problem, forget everything you ever learned about formulas for the elements of Pascal's triangle that involve factorials.

   And I shouldn't really have to say this, but you should make sure you test your function, and indicate to me how you did this.
3. For extra credit: Exercise 8.1 in the Lecture 2 handout.

And when I say, "Be sure to discuss design and testing", I really mean it. You should be able to write something interesting and meaningful, and it should be longer than just a few sentences. If you can't think of anything to write, then you probably haven't thought enough about these problems. In any case, please do feel free to contact me by email with any questions.