CS450: Structure of Higher Level Languages

Fall 2020 Assignment 1 part 1

Due: Sep. 23, 2020, before midnight on Gradescope

Taken from assignments by Profs. Carl Offner and Ethan Bolker

Intro and Goals

To test your answers and your understanding you will want to be able to play with Scheme while you read the text. I encourage you to do that, as the next assignments will involve a lot of coding. See class demo for DrRacket usage. The goal of this assignment is to get you used to the scheme syntax and help you master the basics.

Delivery

I will not run your code in this homework. Answers should be submitted to Gradescope. Notice that Gradescope only accepts pdf, so you have to convert any document you upload to pdf. Please make sure you know how. Google is your friend.

Questions

1. 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 (and note that the third one has a number of parts to it):

   (a) 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.

   (b) 1.6 (page 25). Note that this exercise calls for an explanation, not a program.

   (c) (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))
   ```

   i. What is this turned into internally when the Scheme interpreter reads this definition?

   ii. Given this definition, and given the following expression

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

   A. Show how this expression is evaluated using applicative-order evaluation.

   B. Show how this expression is evaluated using normal-order evaluation.

   (d) 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.

2. 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 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. The recursive procedure should be discussed in Discrete Math.

3. For extra credit: 1.17 (page 46). Be sure to discuss design and testing.

4. 1.20 (page 49). Be careful to read this problem carefully, to do exactly what it says, and to answer all the questions. I have to be able to easily understand what you have written.

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

   (a) How does it compare to other languages you know? Did any of its responses surprise you?

   (b) What message appears when you type '+' and run (and what does the message mean)?

   (c) What other experiments did you perform? What do you like or dislike (so far) about Scheme?