CS450: Structure of Higher Level Languages

Assignment 2

Due: Oct. 2, 2020 on Gradescope

Taken from assignments by Profs. Carl Offner and Ethan Bolker

Guidelines

The purpose of the following problems is to become familiar with the quote special form, and to learn to think recursively about lists and to use the list primitives of Scheme: cons, car and cdr.

Important!: Put all your procedures in a single file named `ASanswers.scm` (this should be the exact name, including upper and lower cases). The order of the question Upload your code to Gradescope by midnight of the due date (that is, 11:59PM of that day). The code parts will be automatically graded. You will be able to re-upload your assignment as many times as you want. The last upload will be the one I'll see and grade.

Make sure your code works and gives exactly the expected results. The auto grader is not sensitive to even small variations. This will be true for all subsequent assignments. If you are having difficulty, you need to send me email before you pass in the homework, and preferably, the earlier the better. Notice, however, that I am not going to spend hours debugging your code for you. Learn how to use Racket's debugging tool. It's very useful.

Finally, a word of warning. Some people have posted answers to some of the problems from the book on the web. I strongly suggest that you not look for them and not use them. The reason is simple: whether those answers are right or wrong, you won't learn a thing. I know this. I've seen it happen. And I can guarantee that you won't find solutions to later assignments on the web, and if you have't learned this material, and learned it well, you won't pass this course. I'm happy to help you myself. I'm happy to have you get help from other students and other places, provided you acknowledge that help. I'm not at all happy to have you copy answers from anyone else or anyplace else, under any circumstance.

1. Write a Scheme procedure `is-list?` that tests to see if an object is or is not a list, according to the definition of a list in the Scheme language definition. The definition is on page 25, in Section 6.3.2, which is titled "Pairs and lists". It's the second paragraph in that section.

2. Exercise 2.18 (page 103). Call your procedure my-reverse. Now that you know about quote you can test with more than lists of integers.

3. Exercise 2.20 (page 104). (And remember what you learn in this exercise! You'll use it later in this course.)

4. Exercise 2.21 (page 106).

5. Exercise 2.23 (page 107). Call your procedure my-for-each.

6. Do exercises 2.24, 2.25, and 2.26 (page 110), and also 2.53 (page 144), but don't turn in the answers.

   Note: Many students (and I know you are all pressed for time) figure this means they can just skip these exercises. Of course I won't know if you don't do them. But I guarantee that you will need to understand these four exercises for a lot of the code that you will be writing in future assignments. So please be sure to do these and take them seriously. And if you don't understand something, please ask me.

7. Exercise 2.54 (page 145), with the following change: in the statement of the problem, substitute `eqv?` for `eq?`. This makes the definition closer to the Scheme language definition. (In fact, it would be even better to substitute "pair" for "list" in the recursive definition given in the problem. You can do this.) Call your procedure `my-equal?`.

8. (a) Define a procedure every? which takes two formal parameters:

   - `pred` is a predicate, and
   - `seq` (i.e., "sequence") is a list.

   `(every? pred seq)` evaluates to #t when every element of `seq` satisfies `pred`, and evaluates to #f otherwise.

   (b) What should happen if the list is empty? Justify your answer in a (clearly written) comment. Here's a hint: if `seq1` and `seq2` are lists, then it should be the case that

   ```
   (every? pred (append seq1 seq2))
   ```

   should be the same as

   ```
   (and (every? pred seq1) (every? pred seq2))
   ```

   Please note: I am asking you for what should happen. So in particular, if you write something like "I tried it out and this is what happened.", your answer is automatically incorrect. (On the other hand, you definitely should try some of this out. I have occasionally seen students hand in an explanation of what they thought should happen, but it was wrong, and if they had only tried it out, they would have seen that it was wrong. So you should definitely try things out. But just reporting what happened when you tried something is not what I am looking for.)

   I'm looking for clearly expressed reasoning here. You have all had a discrete mathematics course, so you should know what I mean. (Of course, as I explained above, if what does happen is not what should happen, something is wrong, either with your reasoning or with your code, right?)

9. Exercise 2.59 (page 153). Call your procedure unordered-union-set.

10. Exercise 2.62 (page 155). Call your procedure ordered-union-set.

11. Define a procedure remove-val which takes two parameters: a value and a list. It returns a list which is the same as the original list except that every element having the given value is deleted. If there are no such elements, the original list is returned. So for instance,

    ```
    (remove-val 3 '(4 5)) ==> (4 5)
    (remove-val 3 '(2 3 4 3)) ==> (2 4)
    ```

    This exercise will also be useful to you in a later assignment.