Homework 7: Abstracting Procedures

Due:

Testing for the problems in this homework is important, as it shows that you can not only write code like in the book, but that you understand how it works. Be sure to include transcripts showing your test cases for all problems.

We continue suggesting some problems for additional practice. These have 0 points. You are not to hand these in to your TA. You should check your own work on them, and if you have questions, discuss them in recitation or with the staff, or with others in class.

1. (1 point; basic) Do exercise 7.1 on page 202. You can handwrite your answer.
2. (4 points; basic) Do problem 7.2.
3. (5 points) Do problem 7.3. As a base case, for all values x, `((compose-many)` x`)` $= x$; that is, `compose-many` with zero arguments returns the identity function.
4. (0 points; suggested practice) Do problem 7.4. You should not use the procedure `-`, but you may use `sub1` in your answer.
5. (9 points; basic) Do problem 7.5. You should handwrite this. Clearly label each part of your answer.

   In addition, write why each answer is what it is, in your own words. If you use the computer, only use it to check your work, not to think for you.
6. (0 points; suggested practice only) Do as many of the following until you get the idea of how to use `map` to solve problems.

   Use `map` to solve exercises 2.16, 3.4, 4.3, and 4.4 of the book.
7. (10 points; basic) Do problem 7.6.
8. (0 points; suggested practice only) Alonzo Typed has noticed that the `map` procedure of program 7.1, is not quite right for non-empty lists, even though it works for them. Define a procedure, `map-non-empty`, that is like `map`, but is tailored to work on non-empty lists. (That is, use the pattern for recursion over non-empty lists instead of recursion over possibly empty lists; for example, test for `(null? (cdr ls))` instead of `(null? ls)`.)
9. (3 points; extra credit only) Alonzo typed has noticed the same thing about `for-each`. Define a procedure, `for-each-non-empty`, that is like `for-each`, except that it is tailored to work on non-empty lists. (That is, use the pattern for recursion over non-empty lists instead of recursion over possibly empty lists.) You should notice a striking difference between the code for `for-each` and the code for `for-each-non-empty`. What is it?
10. (10 points; basic) Do problem 7.7. You should assume that the second argument, `ls`, is non-empty. Note that the last example returns `#t` as noted in the errata for the first printing.
11. (0 points; suggested practice only) Use `reduce`, from the previous problem, to solve problem 3.1 of the book. You may assume that this exercise is modified to take a non-empty list as an argument.
12. (5 points; extra credit only) Do exercise 7.8.
13. (6 points) Do problem 7.9.

14. (6 points; extra credit) Do problem 7.10. Note that the sentence following the faulty definition in the problem should read as follows, with the bracketed words inserted: "This program, as written, is incorrect, because [we would like] the two invocations of *map* within the definition ...." Note that there are three parts to this problem `map`, `ormap`, and the question at the end.

15. (4 points; extra credit only) Define a procedure, `map-poly`, that is takes a procedure `f` and a polynomial `poly` as arguments, and returns a polynomial. The procedure `f` must have type, term $\rightarrow$ polynomial, where a term is a monomial. The result is obtained by taking the results of applying `f` to each term of `poly`, and adding the resulting polynomials. For example,

    ```
    (map-poly (lambda (trm) (make-term (degree trm) (* 3 (leading-coef trm))))
              (poly-cons 3 12.0 (poly-cons 2 -5.1 (poly-cons 0 2.2))))
    ```

    returns the polynomial $36.0x^3 + -15.3x^2 + 6.6$. Similarly

    ```
    (map-poly (lambda (trm) (make-term (add1 (degree trm)) (+ 3 (leading-coef trm))))
              (poly-cons 3 12.0 (poly-cons 2 -5.1 (poly-cons 0 2.2))))
    ```

    returns the polynomial $15.0x^4 + -2.1x^3 + 5.2x$.

    Hint: you should pattern your answer after `map`. You may use `p+` in your solution.

16. (0 points; suggested practice) Do problem 7.11 and test your answer on the computer.

17. (4 points; basic) Do problem 7.12.

18. (0 points; suggested practice only) Do problem 7.13.

19. (3 points; basic) Do problem 7.14.

20. (4 points) Do problem 7.15.

21. (4 points) Do problem 7.16.

22. (5 points) Do problem 7.17.

23. (0 points; suggested practice only) Do problem 7.18.

24. (0 points; suggested practice only) Curry the procedures `map` and `for-each` in programs 7.1 and 7.2. Use your curried `map-c` procedure to solve exercises 2.16, 4.1, 4.2, and 4.3 of the book.

25. (8 points) Do problem 7.19. You can handwrite your answers. Use the computer only to check your work, not to think for you.

26. (1 point; extra credit only) Do problem 7.21.

27. (3 points; basic) Do exercise 7.22.

28. (5 points) Do exercise 7.23.

29. (5 points) Do exercise 7.24.

30. (0 points; suggested practice) Do exercise 7.25.

31. (12 points) Do exercises 7.26 and 7.27.

32. (0 points; suggested practice) Do exercise 7.28.

33. (5 points) Do exercise 7.29.

34. (5 points) Do exercise 7.30.

35. (5 points) Do exercise 7.31.

36. (0 points; suggested practice only) Fill out a table of types as suggested in the "Chapter 7" section of *Make Your Own Scheme Reference Card.*

37. (5 points; extra credit only) Do exercise 7.32.

38. (5 points; extra credit only) Do problem 17 from homework 3 using the techniques you have learned in this chapter.

39. (many points; extra credit only) We are not going to cover chapter 8 in class this semester. But there are many interesting problems in that chapter. Besides their intrinsic interest, these problems will increase your mathematical sophistication, and thus may help you in math and computer classes (such as Com S 330).

    We will give you 2 extra credit points per problem you do correctly, for all problems in chapter 8. (Note that some may be much harder than others.) You should be sure that your solutions are exceptionally clear and clearly tested (to be easier on your TA).