Homework 5: Local Definitions and Polynomials

Due:

To ease our grading and your testing, please continue to follow the conventions described in homework 4 for Scheme code.

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. (12 points) Write procedures to compute a baseball player's total average for a given year, `year-total-average`, and total average for all the given years, `career-total-average`. These were discussed in class. You should make your solutions elegant, concise and easy to understand, as these will count for most of the points. (Hint, revise your work at least once.) You may make reasonable assumptions as needed if the problem is still unclear; but be sure to state these clearly in comments to your code.

   Here is a description of the problem again. A baseball player's statistics are tabulated for several years, like Hank Aaron's below.

```
    YEAR  AB   H 2B 3B HR BB SB CS
Hank Aaron
    1959 629 223 46  7 39 51  8  0
    ...
    1971 495 162 22  3 47 71  1  1

where
  AB = At Bat (Chances)             H = Hits = 1B + 2B + 3B + 4B
  1B = singles = H - (2B+3B+HR)     2B = doubles (2 bases)
  3B = triples (3 bases)            HR = 4B = home runs (4 bases)
  BB = base on balls (walk, 1 base)
  SB = stolen base (1 base)         CS = caught stealing
```

   We define "Total Bases" and "Total Average" as follows.

$$\text{Total Bases} = 1B + 2*2B + 3*3B + 4*HR + SB + BB$$
$$\text{Total Average} = (\text{Total Bases}) / (AB - H + CS)$$

   Some clarifications follow.

   A player's record is given in a list of lists, like the following.

```
;   YEAR  AB   H 2B 3B HR BB SB CS
 '((1959 629 223 46  7 39 51  8  0)
   (1971 495 162 22  3 47 71  1  1)))
```

   Note that the first line is a comment. Thus the record has 0 or more sublists, and each of the sublists has 9 representing the year and the statistics for the player that year. There can be any number of year records in a player's record.

   If the year asked for is not in the player's record, give an error message. Give an error if the total average is not defined because AB-H+CS = 0.

   The year to select is the first argument to `year-total-average`. The year may be any of the year records in the list, not necessarily the first.

2. (7 points; Basic) Do 5.1. Use a squiggly arrow ($\leadsto$) going from a parameter to the value to which it is bound.

3. (0 points) Do 5.2. Handwrite this and check your answer on the computer.

4. (6 points) Do exercise 5.3. We suggest that you test your rewrite using the computer. Note the expressions you are to rewrite are *not* in exercise 5.2, but in exercise 5.1.

5. (0 points; suggested practice) Do exercises 5.4 and 5.5 on paper. Check your answer using the computer.

6. (4 points; extra credit only) Even experienced Scheme programmers sometimes leave out the parenthesis surrounding the list of *var* and *val* pairs in `let` and `letrec`. Perhaps it would be better to write

   ```
   (let x = (+ y z)
       (+ (* x 3) x))
   ```

   instead of

   ```
   (let ((x (+ y z)))
       (+ (* x 3) x))
   ```

   Why do you think the designers of Scheme chose their syntax instead of this alternative? Give at least 2 reasons. (Hint, you may want to consider the discussion of "implicit begin" on page 166.)

7. (5 points; Basic) Do exercise 5.6.

8. (3 points; Basic) Do exercise 5.7.

9. (4 points) Do exercise 5.8. Remember that you have to use the `error` procedure on page 199 of the book.

10. (0 points; Basic) Do exercise 5.9. The two ways to implement the algebra of polynomials are in Figures 5.14 and 5.15. (These figures are found in the files `polynomials1.ss` and `polynomials2.ss` in the directory `/home/cs227/lib`. This exercise asks you to test that they work.

11. (6 points) Do exercise 5.10.

12. (10 points) Do exercise 5.11. Low-level solutions will lose points.

13. (8 points) Do exercise 5.12. (You can handwrite this.) Be sure to consider efficiency and be specific. You may want to write out the code to help your thoughts.

14. (0 points; Suggested practice) Do exercise 5.13.

15. (0 points; Suggested practice) Do exercise 5.14.

16. (10 points) Do exercise 5.15.

17. (10 points; Extra credit only) The polynomials in Section 5.3 only have one variable (x). For this exercise, generalize polynomials to polynomials over 2 variables. You are to start by generalizing the "five basic definitions" to construct polynomials over x and y. Then define generalizations of `zero-poly?`, `make-term`, `leading-term`, `p*` and `poly-value`. You can change the names of these as you see fit, but if you do that tell us what the new names are. (You might want to change the names to use the original polynomial abstraction in building the new one.)

18. (10 points; Extra credit only) Generalize polynomials to an arbitrary number of variables. You will have to pass the names of the variables as parameters. The design of the "five basic definitions" and the generalizations of the other procedures (as in the previous problem) is up to you, but you should describe it to us, so we can understand your solution.

19. (2 points; Basic) Do exercise 5.16. You can handwrite this.

20. (2 points; Basic) Do exercise 5.17. You can handwrite this.

21. (0 points; Suggested practice) Try more exercises like 5.16 and 5.17.

22. (1 point; extra credit) Are there any numeral strings that represent the same number when interpreted in base 2 and base 10? If so, give all the examples you can find. If not, say why not.

23. (12 points) Do exercise 5.18.

24. (2 points; Extra credit only) Make a version of the `change-base` procedure described in the previous problem that works for any two bases less than 36.

25. (12 points) Do exercise 5.19.

26. (0 points; Suggested practise) Do exercise 5.20.

27. (3 points; Extra credit only) Given two polynomials p_1 and p_2, we say that $p_1 > p_2$ if and only if there is some number N such that for all $n > N$, `(> (poly-value` p_1 n`) (poly-value` p_2 n`))`. Implement this in an efficient manner! (That is, define a procedure `p>` that does this efficiently.)

28. (8 points; Extra credit only) Write a procedure `poly-factor` that takes a polynomial and returns a list of prime polynomials that when multiplied together, yield the given polynomial. For example,

    ```
    (poly-factor (poly-cons 2 1
    (poly-cons 0 -1 the-zero-poly)))
    ```

    returns a list containing `(poly-cons 1 1 (poly-cons 0 1 the-zero-poly))` and `(poly-cons 1 1 (poly-cons 0 -1 the-zero-poly))` in some unspecified order.

    Given polynomials f and g such that f is not the zero polynomial, and g has degree of at least 1, there are unique polynomials, q and r with real coefficients such that $f = gq + r$ and r is either the zero polynomial or has degree strictly less than the degree of g. The polynomial r is called the *remainder* of f divided by g, and q is called the *quotient* of f divided by g. We say that g *divides* f if r is zero. A polynomial f is reducible if there is some nonzero g of lesser degree that divides f. All polynomials of degree 1 are not reducible. A polynomial f is a *prime* if and only if it is not reducible.

    Warning: we're not sure how hard this problem is, it may even be impossible in its full generality. You can do a reasonable range of special cases and still get full credit, but you'll have to tell us what part of the problem you're solving. Do what you feel is 8 points worth or come talk to us if you're doing more.