Com S 227X — Introduction to Computer Programming May 17, 1996

Homework 5: Locally Defined Procedures

Due:

All the directions from homework 2 apply to this homework. In particular, remember to hand in a printout for procedures, and either a calculation or a transcript showing your testing.

We highly recommend that you put each problem in a separate file, and when making a transcript use the Scheme procedure `load` to read that file in to the Scheme interpreter.

1 Section 5.2, Let and Letrec

1. (4 points) When should one use `let` in a program? (You can handwrite your answer to this.)

For problems 2 to 4 you are to do the following: (i) write the local environments for each `let` expression, (ii) draw a smooth arrow from each variable use to where it is bound (i.e., the parameter) in a surrounding `lambda` or `let` expression, (iii) draw a squiggly arrow ($\leadsto$) from each such parameter to the value to which it is bound, and (iv) give a calculation of the value of the expression. For example,

```
(let ((a  7))
  (let ((doit (lambda (b) (list a b))))
    (let ((a  102)
          (b  29))
      (doit (+ a b)))))
```

For this example, the answer would look something like the following.

```
(i) Environment 1 (parent Environment 0):
       a ---> [7]
    Environment 2 (parent Environment 1):
       doit ---> [Procedure | (b) | (list a b) | Environment 1]
    Environment 3 (parent Environment 2):
       a ---> [102]
       b ---> [29]

(ii) and (iii)

  (let ((a  7))

    (let ((doit (lambda (b) (list a b))))

      (let ((a  102)

            (b  29))

        (doit (+ a b)))))

(iv) the value is the list (7 131), as shown by the following calculation.

    (doit (+ a b))
  = <by looking up the values of a and b in Environment 3>
    (doit (+ 102 29))
```

```
= <by addition>
  (doit 131)
= <by looking up the value of doit, starting in Environment 3>
  ((lambda (b) (list a b)) 131)
= <by definition of application>
  (list a 131)
= <variable a refers to 7 in Environment 1, the env of doit's birth>
  (list 7 131)
==> <by definition of the list procedure>
  (7 131)
```

2. (0 points; suggested practice) Do problem 5.1.

3. (7 points; basic) The directions for this problem are above.

```
(let ((x  5)
      (y  20))
         (let ((y  x))
            (* (+ y x) x)))
```

   You can handwrite your answer to this.

4. (7 points; basic) The directions for this problem are above.

```
(let ((x  100))
  (let ((g  (lambda (y) (list y x))))
    (let ((x  3))
       (list x (g (+ x 1))))))
```

   You can handwrite your answer to this.

5. (5 points) For this problem, you are to translate from `lambda` to `let`. First we present an example. The expression

```
((lambda (x)                        ;; example
   ((lambda (g)
      ((lambda (x)
         (list x (g (+ x 1))))
       3))
    (lambda (y) (list y x))))
   100)
```

   would be translated into

```
(let ((x  100))                     ;; example answer
  (let ((g  (lambda (y) (list y x))))
    (let ((x  3))
      (list x (g (+ x 1))))))
```

   Note that both expressions have the same result. You are to translate the following expression and replace each lambda by a `let` so that the value computed is the same.

```
((lambda (num)                      ;; this one's for you to do
   ((lambda (a b) (list a num b))
    10 20))
   55)
```

You can handwrite your answer to this.

6. (5 points) In this problem you will do the opposite of the previous problem. That is, you are to replace each use of `let` by `lambda` in the following expression. Your answer should be an expression that computes the same value as the one below.

```
(let ((three (+ 2 1))
      (add +))
  (let ((four (add 1 three)))
    (add four three)))
```

   You can handwrite your answer to this.

7. (0 points; suggested practice) Do exercise 5.3. Check on the computer that your answer evaluates to the same value as the original expressions.

8. (4 points) What's the use of `letrec` in a program? You can handwrite your answer to this.

9. (0 points; suggested practice) Do exercise 5.2. Check your answer on the computer.

10. (0 points; suggested practice) Do exercise 5.4. Check your answer on the computer.

11. (5 points; basic) Do exercise 5.5. You can handwrite your answer to this.

12. (0 points; suggested practice) Do exercise 5.6.

13. (5 points; basic) Do exercise 5.7.

14. (5 points) Do exercise 5.8.

15. (15 points; extra credit) Improve the clarity and efficiency of the Scheme code in the file `/home/cs227/lib/chaos.ss` by using iteration, `let`, and `letrec`. Your answer will be judged by how well you make use of these ideas in the code. Make a copy (using `cp`), change the copy to make improvements, and hand in a print out of your copy.

16. (20 points; extra credit) Write a program `let->lambda` that takes a quoted Scheme expression that uses `let`, and returns an equivalent expression that uses `lambda` instead (as in exercise 5.3). For example,

```
(let->lambda '(let ((x 3)) (+ x x))) ==> ((lambda (x) (+ x x)) 3)
(let->lambda '(let ((f (lambda (x) (let ((y 2)) (+ x y)))))
                (+ 3 (let ((x 4)) x))))
  ==> ((lambda (f)
                (+ 3 ((lambda (x) x) 4)))
       (lambda (x) ((lambda (y) (+ x y)) 2)))
```

   Your code should work for examples such as exercise 5.1 as well.

2 Section 5.3 Symbolic Manipulation of Polynomials

Like the rational numbers, polynomials can have several different representations. Be sure your code works for all representations of polynomials. You will lose points for low level code.

You can load the code for polynomials by executing `(load "/home/cs227/lib/ch5.ss")`. Another representation can be had by loading the file `/home/cs227/lib/polynomials1.ss`. Code for printing polynomials is found in `/home/cs227/lib/print-poly.ss`.

17. (5 points; basic) We can define the polynomial $5.2x^3 - 3.14x^2 + x - 17.3$ by writing

```
(poly-cons 3 5.2
  (poly-cons 2 -3.14
    (poly-cons 1 1.0 (poly-cons 0 -17.3 the-zero-poly)))))
```

Do the same for the following polynomials.

$$\begin{array}{ll} (a) & 4.3x \\ (b) & 3.87x^2 - 2.1x + 5.3 \\ (c) & x + 1.7 \\ (d) & x^3 - 1.0 \\ (e) & 5.1x^6 + 3.2x^5 - x^3 - 8.25x^2 + x - 5478.8 \end{array}$$

18. (0 points; suggested practice) Using the procedures `p+`, `p-`, etc. from `ch5.ss`, compute the sum of $4.3x$ and $3.87x^2 - 2.1x + 5.3$ on the computer. Try additional examples of arithmetic operations of the polynomials in the previous problem if you are unfamiliar with polynomial arithmetic.

19. (0 points; suggested practice) 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, and to compute using `poly-value` the value of the polynomials at various values for x.

20. (6 points) Do exercise 5.10.

21. (8 points) Write a procedure `increase-degrees` that takes a polynominal and increases the degree of each of its terms by 1. For example, passing the polynomial

    $$2x^3 + 9x + 7.2$$

    to `increase-degrees` results in the following polynomial.

    $$2x^4 + 9x^2 + 7.2x$$

    Passing the zero polynomial to `increase-degrees` results in the zero polynomial, since $0x = 0$.

    If your solution does not use the basic procedures for polynomials, you will lose points.

22. (15 points) For this problem you will write two procedures: `poly-quotient` and `poly-remainder`. The procedure `poly-quotient` finds the quotient polynomial when `poly1` is divided by `poly2`. The procedure `poly-remainder` finds the remainder polynomial when `poly1` is divided by `poly2`. (Hint: remember how to do long division? Note that because one can use real numbers for coefficients of terms, one can always cancel the leading term exactly when dividing.) For examples, consider the following transcript.

```
> (load "/home/cs227/lib/ch5.ss")
> (load "/home/cs227/lib/print-poly.ss")
> (load "poly-division.ss")
> (define p1 (poly-cons 3 5.2
              (poly-cons 2 -3.14
               (poly-cons 1 1.0 (poly-cons 0 -17.3 the-zero-poly)))))
> (define p2 (poly-cons 2 14.32 the-zero-poly))
> (print-poly p1)
5.2 x^3 + -3.14 x^2 + 1.0 x + -17.3
> (print-poly p2)
14.32 x^2
> (print-poly (poly-quotient p1 p2))
```

```
0.36312849162011174 x + -0.21927374301675978
> (print-poly (poly-remainder p1 p2))
1.0 x + -17.3
> (print-poly (poly-quotient p2 p1))
0
> (print-poly (poly-remainder p2 p1))
14.32 x^2
```

23. (8 points) Do problem 5.13. Write out the five basic definitions for polynomials using this representation and test your code on the computer.

24. (5 points; extra credit) Do exercise 5.14.

25. (5 points; extra credit) Write a procedure `big-O` that takes a polynomial representing the resources used by a computation and returns the polynomial that is the order of the polynomial. (See page 126 of the text.)

26. (8 points; extra credit) Given two polynomials p_1 and p_2, we say that $p_1 \leq p_2$ 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 terminates with a boolean result for all polynomial arguments.)

27. (20 points; Extra credit only) Generalize polynomials to an arbitrary number of variables. For example, your program should be able to work with $4.3x^2 + 3.4z^3 + 2.1y^2$. You will have to pass the names of the variables as parameters. The design of the "basic definitions" and the generalizations of the other procedures is up to you, but you should describe it to us, so we can understand your solution.

    As a simplier version of this problem, you can get up to 15 points for generalizing polynominals to work with 2 variables.

3 Binary Numbers

28. (0 points; suggested practice) Do exercises 5.16 and 5.17. Check your answers using the `binary->decimal` and `decimal->binary` procedures from `ch5.ss`.

29. (12 points) Do exercise 5.18.

30. (10 points; extra credit) Do problem 5.19.