CS 61A, Spring 97

Midterm 2

Professor Harvey

Problem #1 (6 points):

What will Scheme print in response to the following expressions? If an expression produces an error message, you may just say "error"; you don't have to provide the exact text of the message. If the value of an expression is a procedure, just say "procedure"; you don't have to show the form in which Scheme prints procedures. Also, draw a box and pointer diagram of the value produced by each expression.

(list (cons 2 3) (list 4 5))

(cons (cons 2 (cons 3 4)) '())

(append '() '(a b))

(cdar '((1 . 2) (3 4)))

Problem #2 (4 points):

True or false?

____Tagging complex numbers with rectangular or polar as in section 2.42 could be avoided if the different representation contents were distinguishable.

____In both its verions in section 2.4.1, (make-from-real-imag x y) always produces rectangular form.

____A generic add as in section 2.5.1 checks the types of all of its arguments.

____A generic add as in section 2.5.1 takes exactly two arguments.

Problem #3 (7 points):

Complete the following definitions of some basic list-manipulation operations as accumulations, filters, or maps. Fill in the table below from the numbers of selections offered. If no appropriate selection exists, write "none" and write out the solution. We give an example for part A.

```
(define (accumulate op init seq) ;this is our definition
       (if (null? seq) init (op (car seq) (accumulate op init (cdr seq)))))

(define (map f seq)
     (<A> (lambda (x y) <B>) nil seq)) ; <A> is accumulate

(define (append seq1 seq2)
    (accumulate cons <C> <D>))

(define (length seq)
     (accumulate <E> 0 seq))

(define (factorial n)
      (accumulate <f> <G> <H>))
```

Solutions
A 1; example

B
C
D
E
F
G
H

Suggested solutions

1. accumulate	2. cons
3. (car x)	4. (car y)
5. seq1	6. seq2
7. 0	8. *
9. (1+ x)	10. (lambda (x y) (1+ y))
11. (lambda (x y) (1+ x))	12.length
13. append	14. map
15. filter	16. (enumerate-interval 0 seq)
17. (cons (f x) y)	18. (enumerate-interval 1 n)

Problem #4 (10 points):

A monomial in one variable x is a power of that variable with a numerical coefficient. For example, 10x^2 is a monomial. A polynomial is a sum of monomials. Here is an example of a polynomial: 10x^2+4x+3. To be more complete, we could write it as 10x^2+4x+3x^0. We will represent a polynomial as an unordered list of monomials, each represented as shown below. Your task is to define certain extra functions, being sure to use appropriate abstractions!

A. To deal with monomials, we will use these function definitions:

(define coefficient car) (define exponent cdr) (define add-exponents +)

(define mult-coefficients *) (define add-coefficients +) (define expt-coefficients expt);

From the above, deduce the definition of make-monomial:

(define (make-monomial coefficient exponent)

B. Define a function (polyval p value) which returns the value of the polynomial p when x = value. Hint: first compute the value of each monomial, e.g. 10x^2 at value 3 is 90. You may assume, in this part of the answer, that x^n for any number n is of a numerical type compatible with *, and that a polynomial is a lisp list of monomials.

C. Define a function (mult-mon a b) that takes two monomials e.g. 3x^2 and 4x^3 and returns their product (e.g. 12x^5).

D. Answer the following question in one or more complete English sentences.

Assume now that coefficients and the values are rationals (constructed using make-rat). What must be changed and why?

Problem #5 (12 points):

As you know, an algebraic expression can be described by a tree that can be represented in much the same way as a lisp program. That is, one can take (+ (* a b) (expt d 2) 5) and draw it as a tree where each internal node is labeled by its operator (car), and cdr of each node is a list of children. Assume that you have a tree whose operators are +, * and expt, and whose leaves are names or numbers. In fact, anything not a pair is a leaf.

A. Draw the tree for the example expression above. This is not supposed to be a box-and-pointer diagram, but a tree.

B. Assume you have a set of pairs ((a . 3) (b . 4) (d . 2)) associating names a, b, d with their respective values, 3, 4, 2. This is sometimes called an association list or alist. Write a program (lookup-value var alist) which takes a name like a and returns a value like 3 from the set, if there is an appropriate pair, otherwise error. Except if var is already a number, just return it. That is, (lookup-value 3.14 alist) is 3.14.

```
(define (lookup-value x alist)



)
```

C. Next, write a program to evaluate a tree expression to a single number, when each name in tree is paired with a value by an association list alist. For example,

(tree-eval '(+ (* a b) (expt d 2) 5) '((a . 3) (b . 4) (d . 2)))

is 21. Start by defining abstractions as indicated. Use the reverse of this page if needed.

```
(define (leaf? node)

(define (datum node)

(define (children node)

(define (tree-eval tree alist)
```