University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Summer 2001 Instructor: Clint Ryan 2001-08-16

CS 3 Final

Personal Information

Last name	
First Name	
Student ID Number	
When you started this test	
All the work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS3 who have not taken it yet. (please sign)	

Instructions

- Question 0 (worth 1 point) Please fill in the front and write your name on every page!
- You have at least two hours to complete this exam. It is open book and open notes, but not open computer or open mouth.
- Partial credit will be given for incomplete / wrong answers, so please write down as much of the solution as you can**.**
- You may always write auxiliary functions for a problem unless they are specifically prohibited in the question.
- Feel free to use any Scheme function that was described in sections of the textbook we have read without defining it yourself. Do not use functions or constructs that we did not cover this semester.
- You do not need to write comments for functions you write unless you think the grader will not understand what you are trying to do otherwise.
- **Use good style!** You will lose points if you do not.
- Please comment on the exam on the right. Rate its difficulty (0 = too easy, 5 = impossible), fairness (0 = unfair, 5 = fair), and feel free to add any other comments that come to mind.

Grading Results

Question	*Max. Points*	*Points Given*
0	**1**	
1	**21**	
2	**18**	
3	**20**	
4	**20**	
Total	**80**	

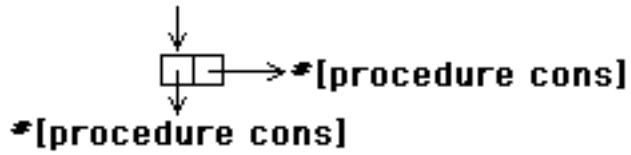
Comments:

- Difficulty (0=easy, 5=hard):
- Fairness (0=unfair, 5=fair):
- Other thoughts?

Name: _____

Question 1: Short Questions (21 points)

a) What Scheme expression does this box and pointer diagram represent?



_____ (2 points)

b) Draw the box and pointer diagram for the list I get from
`(cons (list 1 2) (append '(3) '(4 5)))`. (3 points)

c) Write the specs for the following function: (6 points)

```
(define (show-me-the-truth bool)
   (begin
      (if bool
          (show 'true)
          (show 'false))
    #f))
;;Name:

;;Inputs:

;;Requires:

;;Side Effects:

;;Returns:

;;Examples:
```

Name: _____

```
(define (mystery x)
    (if (null? (children x))
        (begin (show (datum x))
               'done)
        (begin (mystery (car (children x)))
               (mystery (cadr (children x)))
               (show (datum x))
               'this-is-hard)))

(define thistree '(a (b) (c (d) (e))))
```

d) Draw `thistree`. (3 points)

e) Are there any trees for which `mystery` will return an error? If so, what kind of trees? (3 points)

f) If I type `(mystery thistree)`, what return value do I get? _____ (1 point)
Write all of the side effects of `(mystery thistree)`? (3 points)

Name: _____

Question 2: Slightly Short Questions (18 points)

Part A: (6 points)

If I want to change the definition of triples (Tic Tac Toe, Chapter 10) from three digit words to three element lists, list all of the functions in `ttt.scm` that you would have to change.

Part B: (6 points)

Now I want to change the way a date is represented in the Difference Between Dates case study (`datesv1.scm`). Instead of a date being a list of a month name and a day, like (january 3), I want it to be a single word, like `"January 3"`. List all of the functions in `datesv1.scm` that you would have to change to do this.

Part C: (6 points)

Explain why, in a few words, it would be easier to change a date than a triple.

Name: _____

Question 3: Because Johnathon Insisted… (20 points)

Johnathon was not happy with the version of `deeper-count` that I put on the last midterm (Problem 2a). Write a function called `cooler-deeper-count`, which takes a one-character word and a sentence and counts how many times that one-character word appears in the words of the sentence. You may not use `appearances` or `word`. Also, do not use recursion.

Examples:

```
(cooler-deeper-count 'e '(this is a test)) ➔ 1
(cooler-deeper-count 'w '(who what when where why)) ➔ 5
(cooler-deeper-count 'z '(this sentence lacks that letter)) ➔ 0
```

Name: _____

Question 4: Symbolic Differentiation (20 points)

If you have ever taken a calculus class, you have taken the derivative of a function. To find the derivative of a term, like $5x^3$, multiply the coefficient (5) by the exponent (3), and then subtract 1 from the exponent. Thus, the derivative of $5x^3$ is $(5*3)x^{3-1}$, which is $15x^2$. The derivative of a number, like 5, is just 0.

<u>If you don't understand some part of this (like all of it), please ask us!</u>

In Scheme, we would represent a term as a word. The term $3x^5$ would be written as `'3x^5`. In this problem, all exponents and coefficients will be non-negative integers.

Write a function called `deriv-term`, which takes a term like `3x^5` and takes its derivative (to get `15x^4`). Don't forget that the derivative of any ordinary number (like `18`) is `0`.
`deriv-term` should be able to handle the following special cases:

1) Instead of `x^1`, all you should see is `x` (and something like `5x^1` should be `5x`)
2) Instead of `x^0`, all you should see is `1` (and something like `3x^0` should be `3`)
3) Instead of `1x`, all you should see is `x` (and something like `1x^7` should be `x^7`)

If you can't get these to work, write your code without them. You will lose 3 points for each special case your function won't handle.

Examples:

```
(deriv-term '4x^5) ➔ 20x^4    |    (deriv-term '3x^2) ➔ 6x
(deriv-term '3x) ➔ 3          |    (deriv-term 9) ➔ 0
(deriv-term 0) ➔ 0            |
```

Do not write any function over a billion lines! In fact, try to avoid writing one of more than 10.

Write constructors and selectors first.
Write the constructor, `make-term`. `make-term` should take a coefficient and an exponent and return a term.

Name: _____

Now write the selector `get-coefficient`. This should take a term and return the coefficient. Remember that `(get-coefficient 'x^3)` should give me `1`. `(get-coefficient 5)` should return `5`.

Now write the selector get-exponent. This should take a term and return the exponent. Remember that `(get-exponent '3x)` should return `1` and `(get-exponent 7)` should return `0`.

Finally, write `deriv-term`.