CS 61A Midterm #2

Your name _____

TA's name _____ Discussion section number _____

A random five-digit number: 31416

Circle the last two letters of your login (cs61a-xx)

a b c d e f g h i j k l m n o p q r s t u v w x y z 1 2 3

a b c d e f g h i j k l m n o p q r s t u v w x y z

This exam is worth 40 points, or about 13% of your total course grade. The exam contains seven substantive questions, plus the following:

Question 0 (1 point): Fill out this front page correctly and correctly copy your random five-digit number to the top of each of the following pages. (This is to make sure the pages of your exam stay together even if the staple comes out.)

This booklet contains ten numbered pages including the cover page. Put all answers on these pages, please; don't hand in stray pieces of paper. This is an open book exam.

When writing procedures, don't put in error checks. Assume that you will be given arguments of the correct type.

Our expectation is that many of you will not complete one or two of these questions. If you find a question especially difficult, leave it for later; start with the ones you find easier.

If you want to use procedures defined in the book or reader as part of your solution to a programming problem, you must cite the page number on which it is defined so we know what you think it does.

READ AND SIGN THIS:

I certify that my answers to this exam are all my own work, and that I have not discussed the exam questions or answers with anyone prior to taking this exam.

If I am taking this exam early, I certify that I shall not discuss the exam questions or answers with anyone until after the scheduled exam time.

_____

0	1 /1
1-2	6 /6
3	5 /5
4	5 /5
5	7 /8
6	11 /11
7	4 /4
total	39 /40

Question 1 (2 points):

What will Scheme print in response to the following expressions? If the expression produces an error message, you may just write "error"; you don't have to provide the exact text of the message. If the value of an expression is a procedure, just write "procedure"; you don't have to show the form in which Scheme prints procedures.

```
> (filter (not number?) '(3 hello (3 . 2) #t))
```

#f

error

```
> (load "~cs61a/lib/scheme1.scm")
> (apply-1 word '(a b c d))
```

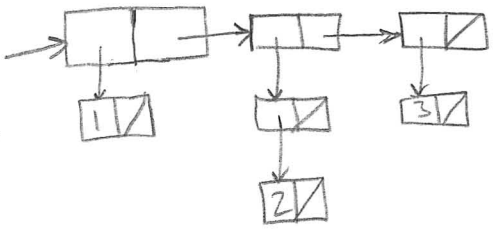
okay
abcd

Question 2 (4 points):

What will Scheme print in response to the following expressions? Also, draw the **box-and-pointer diagrams** for each result.

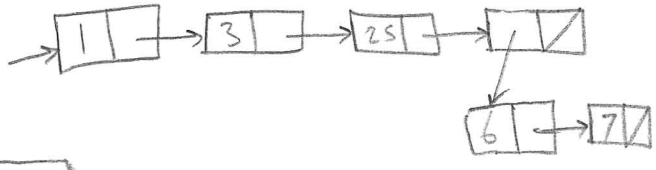
```
> (map list '(1 (2) 3))
```

((1) ((2)) (3))




```
> (map (lambda (x) (if (list? x) (car x) (* x x)))
       '(1 (3 4) 5 ((6 7) 8)) )
```

(1 3 25 (6 7))



Dislike

6

Your five-digit number: 31416

Question 3 (5 points):

What are the domain and range of the following procedures? Be specific, e.g. 'list', 'binary tree', 'procedure', 'Tree', 'number'.

`deep-map` Domain: procedure and list

Range: list

`forest-map` Domain: procedure and forest

Range: forest

`children` Domain: Tree Range: Forest

`left-branch` Domain: binary tree Range: binary tree

`segments->painter`
Domain: list Range: procedure

Assuming p. 156
(a.k.a. you're not talking about mobiles)

5

Question 4 (5 points):

We want to make the calculator program data-directed. It will use the `get/put` table to store all operations it knows how to perform. We can add operations like this:

```
> (put 'calc '+ (lambda (args) (accumulate + 0 args)))
okay
> (put 'calc '- (lambda (args)
                  (cond ((null? args) (error "Calc: no args to -"))
                        ((= (length args) 1) (- (car args)))
                        (else (- (car args)
                                 (accumulate + 0 (cdr args)))) )))
okay
> (put 'calc 'magic (lambda (args) (accumulate word "" args)))
okay
> (calc)
calc: (+ 2 (magic 3 4))
36
```

Here is the code for the calculator program:

```
1. (define (calc-eval exp)
2.   (cond ((number? exp) exp)
3.         ((list? exp) (calc-apply (car exp) (map calc-eval (cdr exp))))
4.         (else (error "Calc: bad expression:" exp))))
5.
6. (define (calc-apply fn args)
7.   (cond ((eq? fn '+) (accumulate + 0 args))
8.         ((eq? fn '-)
9.          (cond ((null? args) (error "Calc: no args to -"))
10.               ((= (length args) 1) (- (car args)))
11.               (else (- (car args) (accumulate + 0 (cdr args))))))
12.        ((eq? fn '*) (accumulate * 1 args))
13.        ((eq? fn '/)
14.         (cond ((null? args) (error "Calc: no args to /"))
15.               ((= (length args) 1) (/ (car args)))
16.               (else (/ (car args) (accumulate * 1 (cdr args))))))
17.        (else (error "Calc: bad operator:" fn))))
```

Question 4 continues on the next page.

Your five-digit number: 31416

Question 4 continued:

Change `calc-eval` and/or `calc-apply` to make this work. You do not need to worry about error handling.

For this problem, use the sections below to show which lines from the program on the previous page you are changing. If you want to replace a single line, write the same number in both spaces. You are not required to use all three sections.

Replace lines 7 *through* 17 *(inclusive) with the following:*

```
... ((get 'calc fn) args) )
```

Replace lines _____ *through* _____ *(inclusive) with the following:*

Replace lines _____ *through* _____ *(inclusive) with the following:*

(This is suspiciously short...)

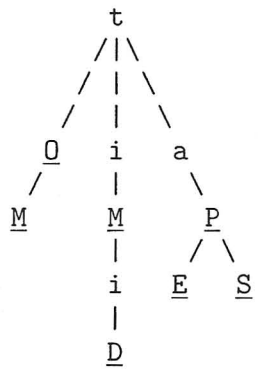
5

Question 5 (8 points):

A *trie* is a special Tree used to store sets of words, like a dictionary. Each datum in the Tree is a "trie entry" defined by the following ADT:

```
(define (make-trie-entry letter is-end-of-word)
  (cons letter is-end-of-word) )
(define (letter trie-entry)
  (car trie-entry) )
(define (end-of-word? trie-entry)
  (cdr trie-entry) )
```

Words in the trie are made up of the letters that stretch from the root to an "end-of-word" node. The trie below is for words starting with `t`. A node that is an end-of-word is shown below underlined and capitalized. (Note that end-of-word nodes are not necessarily leaves.)



The words in this trie are `(to tom tim timid tap tape taps)`. The order is not important. By our definition, `ta` and `timi` are *not* in the trie, since the last letter is not part of an "end of word" entry. `ape` and `mid` are also not part of the trie, since they do not start at the root of the Tree.

Question 5 continues on the next page:

Your five-digit number: 31416

Question 5 continued:

Write a procedure `all-words` that takes a trie and returns a list of all words in that trie (in any order).

Hint: One solution (though not the only one) requires you to keep track of the part of the word you've seen so far. You can use helper procedures for this!

```
(define (all-words trie)
  (define (all-words-helper trie word-part)
    (let* ((d        (datum trie))
           (combined (word word-part (letter d))))
      (accumulate append (if (end-of-word? d) (list combined) '())
                  (map (children trie) (λ (node) (all-words-helper node combined))))))
  (all-words-helper trie ""))
```


7

Question 6 (11 points):

A "chat bot" is a little program that sits online in a chat room and does something useful...or not. There are several kinds of chat bots. Some bots keep track of statistics, like how many people are in the chat room at certain hours. Others might provide definitions when asked. Still others might just insult whoever sends them a message.

We're going to write our own chat bots using object-oriented programming.

(a) Write a `chat-bot` class. Chat bots are given a `name` when they are created, and have a `greet` method and a `respond` method. The `greet` method returns a sentence introducing the chat bot, as shown below. The `respond` method takes an incoming chat message (a sentence). A basic chat bot's `respond` method should just return an empty sentence, no matter what the input is.

For this problem, **make your answer as short as possible.**

```
> (ask first-bot 'greet)
(hi my name is tommy)
> (ask first-bot 'respond '(tommy can you hear me))
()
> (ask second-bot 'greet)
(hi my name is sally)
> (ask second-bot 'name)
sally
```

```
(define-class (chat-bot name)
  (method (greet) (sentence '(hi my name is) name))
  (method (respond s) '())

  )
```

Question 6 continues on the next page.

Your five-digit number: 31416

Question 6 continued:

(b) Write a `grumpy-bot` class. A `grumpy-bot` is just like a `chat-bot`, but has an extra instantiation variable that says how many chat messages it can tolerate. At first, a `grumpy-bot` should return `(dont talk to me)` from its `respond` method. However, after the `grumpy-bot` has `respond`-ed to as many chat messages as its tolerance, it should always return the empty sentence from **both `respond` and `greet`.**

For this problem, **do not unnecessarily repeat code.**

```
; MARVIN is a GRUMPY-BOT with a tolerance of 2.
> (ask marvin 'greet)
(hi my name is marvin)
> (ask marvin 'respond '(where is ford?))
(dont talk to me)
> (ask marvin 'respond '(where is trillian?))
(dont talk to me)
> (ask marvin 'greet)
()
```

```
(define-class (grumpy-bot name tolerance)
  (parent (chat-bot name))
  (instance-vars (counter 0))
  (method (respond s) (set! counter (+ counter 1)) (if (> counter tolerance)
                                                       (usual 'respond s)
                                                       '(dont talk to me)))

  (method (greet) (if (>= counter tolerance)
                      '()
                      (usual 'greet)))

)
```

Question 7 (4 points):

Louis Reasoner wants to write `deep-squares` which takes a deep list of numbers and returns a list with each value squared. He writes the following procedure:

```
1. (define (deep-squares lol)
2.     (cond ((null? lol) '())
3.           ((list? (car lol))
4.            (cons (map square (car lol))
5.                  (deep-squares (cdr lol)) ))
6.           (else (cons (square (car lol))
7.                       (deep-squares (cdr lol)) ))))
```

For which of the following inputs will `deep-squares` not work as intended?

`(deep-squares '())`	Works: ✓	Broken: _____
`(deep-squares '(1 (2 3) 4))`	Works: ✓	Broken: _____
`(deep-squares '(1 (2 3) ((4)) 5))`	Works: _____	Broken: ✓

Which line contains the bug? 4

(needs to be (map deep-square (car lol)))

4