Final Examination for Csci 3316 (Winter 1997)

Due: In 120 minutes.

INSTRUCTIONS

1. The main concepts in CSci 3316 are procedural abstraction, data abstraction, and analysis of programs. We learned about procedures, conditionals, recursion and tail-recursion in the area of procedural abstraction. We uncovered pairs, lists, mutation and abstract data type in the area of data abstraction. Droid models, space/time complexity, box-pointer diagrams and tracing represent the tools to analyze programs. This examination focuses on these concepts.
2. It is an open textbook examination. You are expected to refer to the Scheme Reference Manual (Appendix B, Mani/Little's textbook). Be careful not to spend too much time looking for other things in the textbook. No book other than the textbook is allowed.
3. You can use all primitives scheme functions in defining various functions, unless instructed otherwise in a question.

Score Table							
Question	Q0	Q1	Q2	Q3	Q4	Q5	Total
Score							

QUESTION 0 (1 points) Fill out the following table.

NAME:	
UMN Identification Number:	
Account on itlabs.umn.edu :	
Recitation Section:	1 / 2 / 3/ 4/ 5
TA:	Andrew Forester / Catherine Jacobs / Cecil Smith
Classification:	NTU / UNITE / Extension/ Regular

Q1a. (10 points)

Determine the output of the following scheme forms. Justify your answers via a trace of intermediate results. Use the definition of "reduce" from program 4-24 (pp 215) in the textbook.

(i) (reduce (lambda(a b)(- a b)) 5 '(1 2 3 4))

(ii) (map length '(() (0) ((1 2)) ((1)(2))))

Q1b. (10 points)

Define operations A and C and constant B, so that the following procedure computes the sum of the length of all sublists in a list L of lists:

```
(define sum-of-lengths-of-sublists
  (lambda ( L ) (reduce A  B (map  C L))))

(define A
  (lambda (x y)

        _____________________________________________
))

(define B _____________________ )

(define C
  (lambda (y)

        _____________________________________________
))
```

Trace your implementation using the test-input of L = '((1 2) (3)).

Q2. (20 points)

Consider the lists L1 and L2 defined below:

(i) (define L1 '(1 2 3 4 5))
(ii) (define L2 '((1) (cons 2 3) 4 (() 5)))

Q2a. Draw box-pointer diagrams for these lists.

Q2b. Provide scheme expressions using car and cdr functions to extract 5 from each list.

Q2b. Evaluate the following expressions using box-pointer diagrams:

(cons L1 L2)

(list L1 L2)

(append L1 L2)

Q3. (20 points)

Consider the two procedures, append and append!.

```
(define append
  (lambda (L1 L2)
        (if (null? L1)
           L2
           (cons  (car L1)  (append (cdr L1) L2) ) )))

(define append!
  (lambda (L1 L2)
        (if (null? L1)
           L2
           (begin
                (if (null? (cdr L1))
                   (set-cdr! L1 L2)
                   (append! (cdr L1) L2)  )
                L1 ))))
```

Draw the values of L1, L2, L3 and L4 using box-pointer diagrams after each of the scheme forms in the following sequence:

```
(define L1 '(2 3) )  ;;; draw L1

(define L2 '(4 5) )  ;;; draw L2

(define L3 (append L1 L2))  ;;; draw L3

(define L4 (append! L1 L2))  ;;; draw L4

(set-car! L1 6)  ;;; draw L1, L3 and L4

(set-car! L2 7)  ;;; draw L2, L3 and L4
```

Q4a. (10 points)

Draw the droid model for (mystery '(1 2 3)) where the function mystery is defined below:

```
(define mystery
 (lambda (L)
  (if (null? L)
      1
      (* (* 10 (car L)) (mystery  (cdr L)) ) )))
```

Q4b. (10 points)

Write mystery in tail_recursive form. Include comments to explain the design. Provide a trace with test-input '(1 2 3) to illustrate its correctness.

Consider the following code fragment from one of the the student's solution for Lab. 14:

```
(define all-record-get-number-of-minutes
 (lambda(call-rec)
  (if (>= (cadr (cddddr call-rec)) (car (cddddr call-rec)) )
    (if (> (quotient (cadr (cddddr call-rec)) 100)
                (quotient (car (cddddr call-rec)) 100) )
        (+ (- (remainder (cadr (cddddr call-rec)) 100)
                (remainder (car (cddddr call-rec)) 100) ) ;;; minutes
           (* 60
            (- (quotient (cadr (cddddr call-rec)) 100)
                (quotient (car (cddddr call-rec)) 100) ) ) ;;; hours
        )
        (- (cadr (cddddr call-rec)) (car (cddddr call-rec))  ) ;; same hour

        ;;; over midnight, hours overlap
        (+ (- (remainder (cadr (cddddr call-rec)) 100)
             (remainder (car (cddddr call-rec) 100))  ;;; minutes
           (* 60 (+ 24 (- (quotient (cadr (cddddr call-rec)) 100)
                          (quotient (car (cddddr call-rec)) 100))))
           )
        )
    )
   )
))
```

Recall that the input file has call-records in a specific format. It is represented as a list of six elements, the first three elements represent a phone number, the fourth element represents the start date of call, and the last two elements represent the start-time and end-time. Several examples are shown below:

```
(define call-record1 '(604 555 2953 "94.12.02" 1305 1317))
(define call-record2 '(808 555 3061 "94.12.05" 2357 0004))
```

It is assumed that a call does not last more than 24 hours. The format also ignores international calls which may require dialing of access code, country code and country specific phone number formats. Thus the format can change as the software evolves.

Q5. (19 points)

The procedure call-record-get-number-of-minutes, defined above, has a lot of repetitive code. For example, (car (cddddr call-rec)) is repeated a number of times. If the format of call-record changes, we will need to rewrite this expression in many places in the code, leading to significant effort and expenses (remember the Year 2000 problem!). USE BACK OF PREVIOUS PAGE SPACE TO WRITE ANSWER TO THIS QUESTION.

Q5a. Consider use of an abstract data type (ADT) named call-record-ADT to hide the data format. Design a constructor, namely make-call-record-ADT, and two accessors, namely, get-hours, get-minutes, for the call-record ADT to eliminate the redundant code in procedure all-record-get-number-of-minutes.

Q5b. Rewrite procedure all-record-get-number-of-minutes using the accessors.