CS 3 (Clancy) **Exam 2**
April 12, 2004

Read and fill in this page now.
Do NOT turn the page until you are told to do so.

Your name: _____

Your login name: _____

Your lab section day and time: _____

Your lab t.a.: _____

Name of the person sitting to your left: _____

Name of the person sitting to your right: _____

Problem 0	____		Total:		____	/30	
Problem 1	____	____	Problem 3	____	____	____	
Problem 2	____		Problem 4	____			

This is an open-book test. You have approximately two hours to complete it; time estimates indicate a pace sufficient to finish in one hour. You may consult any books, notes, or other paper-based inanimate objects available to you. To avoid confusion, read the problems carefully. If you find it hard to understand a problem, ask us to explain it. If you have a question during the test, please come to the front or the side of the room to ask it.

Restrict yourself to Scheme constructs covered in chapters 3 through 14 of *Simply Scheme*, all parts of the "Difference Between Dates" case study, and the "Roman Numerals" case study. If you're asked to write a procedure, choose a good name for the procedure and its placeholders. You may use helper procedures.

This exam comprises 15% of the points on which your final grade will be based. Partial credit may be given for wrong answers. Your exam should contain five problems (numbered 0 through 4) on seven pages, along with the code from version 2 of the "Roman Numerals" program.

Please write your answers in the spaces provided in the test; in particular, we will not grade anything on the back of an exam page unless we are clearly told on the front of the page to look there. We believe we have provided more than enough space for your answers, so please don't try to fill it all up.

A few students are taking this exam late. Please refrain from discussing the exam with them and from positing any exam-related comments on discussions or newsgroups until Wednesday.

Relax—this exam is not worth having heart failure about.

Your login name: cs3-_____

Problem 0 (2 points, 1 minute)

Put your login name on each page. Also make sure you have provided the information requested on the first page.

Problem 1 (7 points, 10 minutes)

Part a

Write a procedure named winning-moves that, given a Tic-Tac-Toe board represented as a word of X's, O's, and underscores, along with a player, returns a sentence containing all the winning moves for the player. For example, given the argument _O_ _XXOOX representing the board below, along with player X,

	O	
	X	X
O	O	X

winning-moves should return a sentence containing 1, 3, and 4 (in any order). Use procedures already defined in the Tic-Tac-Toe program wherever possible. Use good placeholder names in your helper procedures (if any) and lambda expressions. Don't use recursion.

Your login name: cs3-_____

Part b (graded only if your answer to part a is essentially correct)

Circle the sentence that your winning-moves procedure returns given the board _O_ _XXOOX and the player X as arguments, and briefly explain your answer.

```
(1 3 4)        (3 1 4)        (4 1 3)

(1 4 3)        (3 4 1)        (4 3 1)
```

Explanation:

Problem 2 (8 points, 15 minutes)

Given below is the roman-sum procedure from version 2 of the Roman Numerals program. (The entire program appears at the end of this exam.)

```
(define (roman-sum number-sent)
   (cond
      ((empty? number-sent) 0)
      ((empty? (bf number-sent)) (first number-sent))              ; line 4
      ((not (starts-with-prefix? number-sent))
       (+ (first number-sent) (roman-sum (bf number-sent)) ) )     ; line 6
      ((starts-with-prefix? number-sent)
       (+
         (- (first (bf number-sent)) (first number-sent))          ; line 9
         (roman-sum (bf (bf number-sent))) ) ) ) )                ; line 10
```

Your clumsy lab partner may have accidentally deleted one of the five calls to bf. One could determine which bf was deleted by calling roman-sum four times, one to check for each possibility (removal of either of the bf calls in the last line produces identical behavior). However, it is possible to check for *all* four possibilities for the deleted bf in a *single* call to roman-sum.

Part a

Below, give a *single* Roman numeral that, when provided as input to decimal-value, will allow you to determine which call to bf in roman-sum—if any—was omitted.

Part b

Each of the versions of roman-sum below is missing a call to bf in the boldfaced line. Using your answer to part a as input to decimal-value, describe the result that decimal-value would return using the given roman-sum (and leaving all other parts of the program unchanged). If a crash would occur, indicate why and where it happens, as specifically as possible.

version of roman-sum	result of providing your answer to part a as input to decimal-value

`(define (roman-sum number-sent)`
`  (cond`
`  ((empty? number-sent) 0)`
`  **((empty? number-sent) (first number-sent))**`
`  ((not (starts-with-prefix? number-sent))`
`  (+ (first number-sent) (roman-sum (bf number-sent)) ) )`
`  ((starts-with-prefix? number-sent)`
`  (+`
`  (- (first (bf number-sent)) (first number-sent))`
` (roman-sum (bf (bf number-sent)))))))	

`(define (roman-sum number-sent)`
`  (cond`
`  ((empty? number-sent) 0)`
`  ((empty? (bf number-sent)) (first number-sent))`
`  ((not (starts-with-prefix? number-sent))`
`  **(+ (first number-sent) (roman-sum number-sent)) )**`
`  ((starts-with-prefix? number-sent)`
`  (+`
`  (- (first (bf number-sent)) (first number-sent))`
` (roman-sum (bf (bf number-sent)))))))	

`(define (roman-sum number-sent)`
`  (cond`
`  ((empty? number-sent) 0)`
`  ((empty? (bf number-sent)) (first number-sent))`
`  ((not (starts-with-prefix? number-sent))`
`  (+ (first number-sent) (roman-sum (bf number-sent)) ) )`
`  ((starts-with-prefix? number-sent)`
`  (+`
`  **(- (first number-sent) (first number-sent))**`
` (roman-sum (bf (bf number-sent)))))))	

Problem 3 (5 points, 14 minutes)

Given below is an implementation of sent-max, which returns the largest value in a sentence of numbers.

```
(define (sent-max sent)
   (accumulate max sent) )

(define (max x y)
   (if (> x y) x y) )
```

Suppose, when typing in the code for the max procedure, you mistakenly typed an x for a y in the if condition in max, so that max was defined as

```
(define (max x y)              ; incorrect version
   (if (> x x) x y) )
```

Part a

Using the incorrect max procedure above, give the result of evaluating

```
(sent-max '(1 3 2))
```

Part b

Give an example of a sentence of at least three elements for which sent-max, using the incorrect version of max, produces *the same output* as sent-max using the correct version of max.

Part c

Describe, as completely as possible, the collection of sentences for which sent-max using the incorrect version of max produces *the same output* as sent-max using the correct version of max. Your answer is a description of test cases that would *fail to reveal* the bug in the mistyped max procedure. Briefly justify your answer.

Problem 4 (8 points, 20 minutes)

Write a recursive procedure named split that, given a sufficiently long word named longWord and a sentence of nonnegative integers named lengths, returns a sentence of words determined as follows.

> Suppose lengths contains the integers k_1, k_2, k_3, ..., k_n. Then the first word in the sentence returned by split should contain the first k_1 characters of longWord, the second word in the result should contain the next k_2 characters in longWord, and so on. The last word in the result should contain whatever characters remain in longWord; it may be the empty word. (Thus the result should always contain n+1 words.)

Here's an example.

```
(split 'abcdefghijklmnopqrstuvwxyz '(3 0 1 5 4))
```

should return

```
(abc "" d efghi jklm nopqrstuvwxyz)
```

Assume that the number of characters in longWord is at least as large as the sum of the integers in lengths. You may use helper procedures. Don't use higher-order procedures.

```
(define (split longWord lengths)
```

Your login name: cs3-_____

Version 2 of the code from the "Roman Numerals" case study

```
;; Return the decimal value of the Roman numeral whose digits are contained in roman-numeral.
;; Roman-numeral is assumed to contain only Roman digits.
;; Sample call: (decimal-value 'xiv), which should return 14.
(define (decimal-value roman-numeral)
   (roman-sum (digit-values roman-numeral)) )

;; Return the decimal value of a Roman numeral. The decimal equivalents
;; of its Roman digits are contained in number-sent.
;; Sample call: (roman-sum '(10 1 5)), which should return 14.
(define (roman-sum number-sent)
   (cond
      ((empty? number-sent) 0)
      ((empty? (bf number-sent)) (first number-sent))
      ((not (starts-with-prefix? number-sent))
       (+ (first number-sent) (roman-sum (bf number-sent)) ) )
      ((starts-with-prefix? number-sent)
       (+
         (- (first (bf number-sent)) (first number-sent))
         (roman-sum (bf (bf number-sent))) ) ) ) )

;; Return a sentence containing the decimal values of the Roman digits in roman-numeral.
;; Roman-numeral is assumed to contain only Roman digits.
;; Sample call: (digit-values 'xiv), which should return (10 1 5).
(define (digit-values roman-numeral)
   (if (empty? roman-numeral) '( )
      (se
         (decimal-digit-value (first roman-numeral))
         (digit-values (bf roman-numeral)) ) ) )

;; Return the decimal value of the given Roman digit.
(define (decimal-digit-value roman-digit)
   (cond
      ((equal? roman-digit 'm) 1000)
      ((equal? roman-digit 'd)  500)
      ((equal? roman-digit 'c)  100)
      ((equal? roman-digit 'l)   50)
      ((equal? roman-digit 'x)   10)
      ((equal? roman-digit 'v)    5)
      ((equal? roman-digit 'i)    1) ) )

;; Return true if number-sent starts with a prefix, i.e. a number
;; that's less than the second value in the sentence.
;; Number-sent is assumed to be of length at least 2 and to contain
;; only positive numbers.
(define (starts-with-prefix? number-sent)
   (< (first number-sent) (first (bf number-sent))) )
```