CS3 MIDTERM 1

Fall 2007

Read this page and fill in the left table now. Don't turn the page yet.

Name:	
Instructional login (eg, cs3-ab):	
UCWISE login:	
Lab section (day and time):	
T.A.:	
Name of the person sitting to your **left**:	
Name of the person sitting to your **right**:	

(1 pt)	Prob 0	
()	Prob 1	
()	Prob 2a	
()	2b	
()	Prob 3a	
()	3b	
()	3c	
()	3d	
()	3e	
()	Prob 4a	
()	4b	
()	Prob 5	
Raw Total (out of)		
Scaled Total (30)		

You have 90 minutes to finish this test, which should be reasonable. Your exam should contain XXX problems (numbered 0-6) on 12 total pages.

This is an open-book test. You may consult any books, notes, or other paper-based inanimate objects available to you. Read the problems carefully. If you find it hard to understand a problem, ask us to explain it.

Restrict yourself to Scheme constructs covered in chapters 3-6 and 11-13 of *Simply Scheme*, the *Difference Between Dates* case study, both the first and recursive versions, and the *Roman Numerals* case study. You can always use helper procedures and procedures from other questions you've answered, unless the question specifically disallows it.

Please write your answers in the spaces provided in the test; if you need to use the back of a page make sure to clearly tell us so on the front of the page.

Partial credit will be awarded where we can, so do try to answer each question.

Good Luck!

Problem 0. (1 point)

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

Problem 1. And the return value is... (9 points)

Write the result of evaluating the Scheme expression that comes before the →. If the Scheme expression will result in an error, write *ERROR* in the blank and describe the error. Some of these are tricky; please be careful.

1
```
(se 'heaven '(and the) earth) →
```

2
```
(word "" (word "" (word 'hello (word "" (word ""))))) →
```

3
```
(equal? 'lou (or 'mary 'lou 'retton)) →
```

4
```
(define (mystery wd)
  (cond (wd 'im-confused)
        (else 'ah-ha)))

(mystery '(you are not confused))  →
```

5
```
(or (if #f #t #f) 7) →
```

6
```
'(+ 3 5) →
```

7
```
(decimal-value 'vvv) →
```

8
```
(define (ends-with-prefix? number-sent)
  (starts-with-prefix? (reverse number-sent)))

(ends-with-prefix? _________________________________ ) → #t
```

9
```
(define (mystery2 sent)
  (if (not (number? (first sent)))
      0
      (+ (first sent) (mystery2 (bf sent)))))
```

```
(mystery2 '(1 2 3 four 5)) ➔
```

Problem 2. Rate my moves (points)

The Motion Picture Association of America rates movies with the following system:

Rating	**Description**
G	General Admission
PG	Parental guidance suggested
PG-13	Parents strongly cautioned. (Treat this as if under 13 not admitted)
R	Restricted
NC-17	No one under 17 admitted

Part A. Write the predicate `can-watch?`, which takes in an age and a rating and returns true if a person with a given age can be admitted to a movie with a given rating. Using excessive `cond` cases or excessive comparisons to specific rating values (e.g., `G`, `PG-13`, ...) will lose points; be judicious.

```
(define (can-watch? age rating)
```

Part B. Write a procedure `best-movie` which takes an age and two movie ratings, and returns the "highest" rating of the two (where the highest possible rating is NC-17). Part of your solution should use the form of the better solution in the *Difference between Dates* case study (the second, successful attempt). If the given age is not old enough to attend either movie, return the word `sorry`.

```
(define (best-movie age rate1 rate2)
```

Problem 3. Lights out (points)

This question concerns a simplified version of the game “lights out”. In this game, the goal is to get a 3x3 matrix of lighted buttons to be all off (unlit). Pressing a button will change the state (i.e., whether the light is on or not) of other buttons around it. The picture to the right shows the labels of the buttons, 1 to 9 starting in the upper left corner.

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

The state of the board—which lights are on and which are off—is represented by a sentence of 9 words. The first word in the sentence corresponds to the button 1 (and so on down the sentence).

If the word is **`lit`** or the number **`1`**, this implies that the corresponding button is lit; any other word indicates that the corresponding button is unlit. For instance, the sentence

```
   (lit 0 1 unlit unlit false 1 your-mama 1)
```

represents a board with only the corners lit.

Part A. The predicates `lit?` and `unlit?` take one of the button states (as a word), and return `#t` or `#f` if that button is lit or unlit, respectively.

Write procedure definitions for `lit?` and `unlit?`.

Part B. Pressing button 4, in this game, will toggle the state of buttons 1 and 7. Write the procedure `press4`, which takes a board state and returns the state of the board after button 4 is pressed.

Use helper procedures where appropriate. Do not use the procedure `item`.

Part C

The procedure `win?` takes a board state and returns #t if all of the lights are unlit. This is the winning position.

Write `win?` without using `or`, `and`, `appearances`, or `member?`. Assume you have properly working versions of `lit?` and `unlit?`.

Part D Write `win?` without using `if`, `cond`, `appearances`, or `member?`. Assume you have properly working versions of `lit?` and `unlit?`.

Part E. Consider the procedure `number-lit?`, which takes a number and a board state (as defined earlier) and returns `#t` if the number of lights lit on the board is equal to the number passed in.

`(number-lit? 3 '(1 0 0 lit 0 0 1 0 0))`	➔	`#t`
`(number-lit? 3 '(1 0 0 lit 0 0 0 0 0))`	➔	`#f`
`(number-lit? 3 '(1 0 0 lit 0 0 1 1 0))`	➔	`#f`

The following is a buggy version of number-lit?.

```
(define (num-lit? n state)
  (cond ((empty? state)
         (<= n 0))
        ((lit? (first state))
         (num-pegs-down? n (bf state)))
        (else
         (num-pegs-down? (- n 1) (bf state))) ))
```

<u>Briefly</u> describe the situations in which this version of `num-lit?` does not return the proper value or causes an error:

- Write a single (or so) sentence of english to describe the error(s). You can mark the above code if you want.
- Give examples of parameters (i.e., `n` and `state`) which will cause the error(s).
- Describe how to fix the error(s), with code or in text.

Problem 4. You'll be counting the days... (points)

Your well-meaning colleague decides to rewrite the recursive solution to day-span, but has run into trouble. He reasons that it should be easy to write a solution that counts each day one at a time. He calls this procedure `day-by-day`, to distinguish it from `day-span`, but the functioning should be identical. He needs help, however.

Part A. Fill in the blanks on his code below, using all of the framework code provided below (don't simply cross some out and ignore it). Be sure to use helper procedures where appropriate, and good procedure and parameter names. `cond` statements with 12 cases will be frowned upon. Feel free to use any procedure defined in the recursive version of `day-span` (Appendix C in the reader, also included at the end of this exam).

```
;; works as day-span
(define (day-by-day date1 date2)
  (if (equal? date1 date2)

      _______________

      ( + 1 (day-by-day (next-day date1) date2))) ))


(define (next-day date)
   (if (last-day-of-month? date)

       ____________________________________________________

       ____________________________________________________  ))


(define (last-day-of-month? date)
```

Part B. Provide test-cases for `next-day`. These tests will have to exercise code that `next-day` calls, to some extent. Be sure to include both the test call and the expected return value. Additionally, give a very brief explanation on what the test case is testing.

When testing a particular "issue" that can appear in each of the twelve months, refrain from including all twelve tests. Rather, consider how the months may form similar groups, with respect to the issue at hand, and provide a single test (or two) for each group. Don't forget to include a brief explanation of the nature of the group.

Problem 5. Any letters for me? (A: 2 points; B: XXX points)

Consider the procedure `get-2ltr-words` which takes a word as input, and `returns` a sentence of every common two letter word whose letters appear consecutively inside the argument.

`(get-2ltr-words 'sentence)`	➔	`()`
`(get-2ltr-words 'tommy)`	➔	`(to my)`
`(get-2ltr-words 'isolate)`	➔	`(is so at)`

Part A.

Complete the definition of the predicate `2ltr-word?`, which returns true if and only if the argument passed to it is one of the 18 simple simple two letter words included below. Use good parameter names.

```
(define (2ltr-word? ________________)

   (  ________________________________________________
     '(is it if in so no at an as to hi ok pi do my by be or)))
```

Part B.

Write `get-2ltr-words`. Assume you have a working version of `2ltr-word?`. Be <u>sure</u> to use helper functions where it will make your code more readable.

Difference between dates, recursive version (Appendix C in the reader)

```
; Return the number of days spanned by earlier-date and later-date.
; Earlier-date and later-date both represent dates in 2002,
; with earlier-date being the earlier of the two.
(define (day-span earlier-date later-date)
  (cond
    ((same-month? earlier-date later-date)
     (same-month-span earlier-date later-date) )
    ((consecutive-months? earlier-date later-date)
     (consec-months-span earlier-date later-date) )
    (else
     (general-day-span earlier-date later-date) ) ) )

; Access functions for the components of a date.
(define (month-name date) (first date))
(define (date-in-month date) (first (butfirst date)))

; Return true if date1 and date2 are dates in the same month, and
; false otherwise. Date1 and date2 both represent dates in 2002.
(define (same-month? date1 date2)
  (equal? (month-name date1) (month-name date2)))

; Return the number of the month with the given name.
(define (month-number month-name)
  (cond
    ((equal? month-name 'january) 1)
    ((equal? month-name 'february) 2)
    ((equal? month-name 'march) 3)
    ((equal? month-name 'april) 4)
    ((equal? month-name 'may) 5)
    ((equal? month-name 'june) 6)
    ((equal? month-name 'july) 7)
    ((equal? month-name 'august) 8)
    ((equal? month-name 'september) 9)
    ((equal? month-name 'october) 10)
    ((equal? month-name 'november) 11)
    ((equal? month-name 'december) 12) ) )

; Return true if date1 is in the month that immediately precedes the
; month date2 is in, and false otherwise.
; Date1 and date2 both represent dates in 2002.
(define (consecutive-months? date1 date2)
  (=
    (month-number (month-name date2))
    (+ 1 (month-number (month-name date1))) ) )

; Return the difference in days between earlier-date and later-date,
; which both represent dates in the same month of 2002.
(define (same-month-span earlier-date later-date)
  (+ 1
    (- (date-in-month later-date) (date-in-month earlier-date)) ) )
```

```
; Return the number of days in the month named month-name.
(define (days-in-month month-name)
  (cond
    ((equal? month-name 'january) 31)
    ((equal? month-name 'february) 28)
    ((equal? month-name 'march) 31)
    ((equal? month-name 'april) 30)
    ((equal? month-name 'may) 31)
    ((equal? month-name 'june) 30)
    ((equal? month-name 'july) 31)
    ((equal? month-name 'august) 31)
    ((equal? month-name 'september) 30)
    ((equal? month-name 'october) 31)
    ((equal? month-name 'november) 30)
    ((equal? month-name 'december) 31) ) )

; Return the number of days remaining in the month of the given date,
; including the current day. Date represents a date in 2002.
(define (days-remaining date)
  (+ 1 (- (days-in-month (month-name date)) (date-in-month date))) )

; Return the difference in days between earlier-date and later-date,
; which represent dates in consecutive months of 2002.
(define (consec-months-span earlier-date later-date)
  (+ (days-remaining earlier-date) (date-in-month later-date)) )

; Return the name of the month with the given number.
; 1 means January, 2 means February, and so on.
(define (name-of month-number)
  (item month-number
        '(january february march april may june
          july august september october november december) ) )

; Return the sum of days in the months represented by the range
;    first-month through last-month.
; first-month and last-month are integers; 1 represents January,
; 2 February, and so on.
; This procedure uses recursion.
(define (day-sum first-month last-month)
  (if (> first-month last-month)
      0
      (+
        (days-in-month (name-of first-month))
        (day-sum (+ first-month 1) last-month)) ) )

; Return the number of the month that immediately precedes the month
; of the given date.  1 represents January, 2 February, and so on.
(define (prev-month-number date)
  (- (month-number (month-name date)) 1) )

; Return the number of the month that immediately follows the month
; of the given date.  1 represents January, 2 February, and so on.
(define (next-month-number date)
  (+ (month-number (month-name date)) 1) )

; Return the difference in days between earlier-date and later-date,
; which represent dates neither in the same month nor in consecutive months.
(define (general-day-span earlier-date later-date)
  (+
    (days-remaining earlier-date)
    (day-sum
      (next-month-number earlier-date)
      (prev-month-number later-date) )
    (date-in-month later-date) ) )
```