CS3 Midterm 1

Spring 2007

Read this page and fill in the left table now.

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	
(10)	Prob 1	
(6)	Prob 2a	
(4)	2b	
(5)	Prob 3	
(5)	Prob 4a	
(8)	4b	
(5)	Prob 5a	
(2)	5b	
(5)	5c	
Raw Total (out of 51)		
Scaled Total (30)		

You have 80 minutes to finish this test, which should be reasonable. Your exam should contain 6 problems (numbered 0-6) on 11 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. You can always use helper procedures, and procedures from other questions you've answered.

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... (10 points [#9 is worth 2 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.

1	`(se 'I (word 'am (se 'sam)))` ➔
2	`(quote (word sam I am))` ➔

3	`(appearances`
	`   (bf 'sam)`
	`   (se 'that 'sam 'I 'am 'sam 'a 'a 'a))`
	
	➔

4	`(cond ('false 'is-true)`
	`      ('lies 'are-truth)`
	`      (else 'im-confused))` ➔

5	`(equal? 3 (or 1 2 3))` ➔
6	`(and #t (and #t (and #t (and #t (and #t (and #t #f))))))` ➔
7	`(member? 'I (word 'you 'and 'I))` ➔
8	`( 3 + 5 )` ➔

9	`(define (mystery sent)`
	`   (or (empty? sent)`
	`       (and (+ 1 (first sent))`
	`            (mystery (bf sent)) ) ) )`
	
	`(mystery '(2 3 4))` ➔

Problem 2. Whatever floats your boat (A: 6 points, B: 4 points)

This problem involves a procedure `can-order?`, which takes two ranks in the United States navy and returns `#t` if and only if the first rank is "above" the second and can, therefore, order the other one around. The following table lists the ranks:

rank	**Explanation**
5	*5 star admiral*
3	*3 star admiral*
1	*1 star admiral*
cpn	*captain*
cmd	*commander*
ltn	*lieutenant*
en	*ensign*

Part A:

Write `can-order?` in the form of the "better solution" in the *Difference Between Dates* case study (the second attempt that successfully wrote `day-span`, after the dead end was reached in the first attempt). You can assume that the ranks passed to `can-order?` are valid.

<u>Chooose good names for your parameters and helper procedures, and add relevant comments above every procedure</u>. Partial credit will be awarded for solutions that don't follow the form of the better solution in *Difference Between Dates.*

Part B:

There are many possible valid calls to `can-order?`. For this problem, you will write test cases for the procedure, but we don't want a large list. Instead, we want you to describe what the general classes of tests cases are. That is, think about how the test cases can be grouped, such that the cases in a group are similar in how they check for errors or otherwise test the program.

There are not many groups. For each group, briefly describe what the similarity is and provide a single test case. Be sure to include the correct result of the test case.

Problem 3. And and or were walking down the street... (5 points)

A friend tells you that `and` always returns the last argument given to it, unless there is an error. He types:

`(and 'joe 'bob)` and it returns `bob`

`(and 'joe 'bob 'briggs)` and it returns `briggs`

Give an example showing that he is wrong.

____

He then tells you that `or` always returns the first argument it is passed, typing:

`(or 3 4 5)` which returns `3`

Again, can you prove him wrong by providing a counter example?

____

Finally, your friend wrote a procedure `not-in-order?`, which takes three numeric arguments and returns true if and only if they are neither in ascending or decending order. It might not work!

```
(define (not-in-order? a b c)
  (or (not (or (> a b) (< b c)))
      (not (or (< a b) (> b c)))))
```

What will this procedure return for three ascending arguments? ____________________

What will this procedure return when the middle argument is larger than the other two? ____________________

Will this procedure always return the same thing in calls where exactly two of the arguments are the same? ____________________

Problem 4. Recursive `general-day-span` (A: 5 points, B: 8 points)

These questions concern the case study *Difference between dates, recursive version.* The code for this case study is included at the end of this exam (which is Appendix C in your reader).

Part A:

Correctly define the following functions by filling in the blanks. Solutions should fit easily! (Points will be deducted, otherwise).

You can (and should) use procedures defined in the case study!

```
;; takes a month-name and a date-in-month, and returns a valid date.
;;  e.g., (make-date 'january 3) ==> (january 3)
(define (make-date month-name date-in-month)

     ________________________________________ )
```

```
;; takes a month-name and a number representing days, and returns true
;; if the month has that many days in it.
;;  e.g., (has-days? 'january 31) ==> #t
(define (has-days? month-name num-days)

   ____________________________________________ )
```

```
;; Takes a non-december date, and return the name of the following
;; month.
;;  e.g., (next-month-name '(january 3)) ==> february
(define (next-month-name date)

   ___________________________________________________________  )
```

```
;; Takes two dates, and returns the number of months spanned by those
;; dates (including the months for each of the dates).  Assume the
;; first date is earlier than the second.
;;  e.g., (month-span '(june 16) '(july 4)) ==> 2
(define (month-span earlier-date later-date)

    ___________________________________________________________  )
```

Part B:

In the case study, recursion was used in the `day-sum` procedure, which was called by `general-day-span`. For this problem, you will write a recursive `general-day-span-r` which, in a sense, combines the functionality of `day-sum` and `general-day-span` into one procedure.

You can use helper procedures from part A or from the case study, except for `day-sum`. (Assume the procedures from part A work correctly). You can also write your own helper procedures, but they must be non-recursive.

Your solution must use recursion—that is, your definition of `general-day-span-r` must include a meaningful call to `general-day-span-r`.

```
(define (general-day-span-r earlier-date later-date)
```

Problem 5. Scrambled and confused (A: 5 points, B: 2 points, C: 5 points)

Consider the procedure `scramble` defined below:

```
;; takes a word and a sentence of numeric positions, and returns the
;; word formed by the letters (of the original word) pointed to by each
;; position, in the order defined by the sentence of positions.
(define (scramble wd positions)
  (if (empty? positions)
      ""
      (word (item (first positions) wd)
            (scramble wd (bf positions)))))
```

Part A: For the the expression

```
   (scramble 'fred '(4 3 1 2))
```

list all of the calls to `scramble` that will result, in order, and what the return value of each call will be. This is the same information that tracing `scramble` will provide.

Part B: Do a particular scramble on any word using sentence operators. Define the procedure `scramble-532`, using only the procedures `word`, `first`, `last`, `butfirst`, and `butlast`, which takes a <u>five-letter word</u> and scrambles it with the position sentence `'(5 3 2)`. For example,

```
(scramble-532 'abcde) ➔ ecb
```

```
(define (scramble-532 wd)
```

Part C:

Consider a better version of `scramble` (called `better-scramble`) in which the sentence of positions can be more complicated. For `better-scramble`, each position can be

- a positive number, which references the letter at that position.
- a negative number, which references the letter at the position counting from the end of the word towards the front
- the word `first`, which references the first letter of the word.
- the word `last`, which references the last letter of the word.

For example,

`(better-scramble 'abcde '(-2 first 3 last))`	➔	`dace`
`(better-scramble 'abcde '(-4 first -2))`	➔	`bad`

Fill in the blanks to complete `better-scramble`. Assume that the sentence of positions will always be valid (i.e., won't contain positions out of range, etc.).

```
(define (better-scramble wd positions)
   (if (empty? positions)
       ""
       (word (item (real-position ___________________________________ ) wd)
             (better-scramble wd (bf positions)) ) ) )
```

```
(define (real-position ____________________________________ )
```

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) ) )
```