CS3 MIDTERM 1 FALL 2007

STANDARDS AND SOLUTIONS

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

For problems 1-7, 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. The *Roman Numerals* case study code is at the end of this exam.

1	`(se 'heaven '(and the) earth)` ➔ ERROR – undefined variable: earth
2	`(word "" (word "" (word 'hello (word "" (word ""))))))` ➔ `hello`

3	`(equal? 'lou (or 'mary 'lou 'retton))` ➔ `#f`
	`(or 'mary 'lou 'retton)` returns `mary`, so the `equal?` becomes
	`(equal? 'lou 'mary)`

4	`(define (mystery wd)`
	`  (cond (wd 'im-confused)`
	`  (else 'ah-ha)))`
	
	`(mystery '(you are not confused))` ➔ `im-confused`
	we replace `wd` in the `cond` statement with the sentence `'(you are not confused)`, which evaluates to `#t` in scheme. Remember anything that is not `#f` in Scheme is true!

5	`(or (if #f #t #f) 7)` ➔ `7`
6	`'(+ 3 5)` ➔ `(+ 3 5)`
7	`(decimal-value 'vvv)` ➔ `15`

For problem 8, fill in the blank.

8	`(define (ends-with-prefix? number-sent)`
	`  (starts-with-prefix? (reverse number-sent)))`
	
	`(ends-with-prefix? '(10 5 1) )` ➔ `#t`
	Remember that starts-with-prefix takes in numbers, not roman numerals. Then you just need to reverse a sentence that returns #t when given to starts-with prefix.

For problem 9, write a meaningful comment for `mystery2`.

9	```;; mystery2 takes in a sentence of numbers
	;; and returns the sum of all the even numbers
	(define (mystery2 sent)
	  (cond ((empty? sent) 0)
	  ((even? (first sent))
	  (+ (first sent) (mystery2 (bf sent))))
	  (else
	  (mystery2 (bf sent)))))```

Problem 2. Rate my moves (A: 4 points, B: 6 points)

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

Rating	Description	Age
G	General Admission	any
PG	Parental guidance suggested	any
PG-13	Parents strongly cautioned.	13+
R	Restricted	13+
NC-17	No one under 17 admitted	17+

Part A. Write the predicate `can-watch?`, which takes in a numeric age and a rating (a word, as above) and returns `#t` if a person with a given age can be admitted to a movie with a given rating. Be judicious in your tests: using excessive `cond` cases or excessive comparisons to specific rating values (e.g., `G`, `PG-13`, ...) will lose points.

```
(define (can-watch? age rating)
  (cond ((>= age 17) #t)
      ((< age 13)
       (member? rate '(g pg)))
      (else
       (not (equal? rate 'nc17))) ))
```

Most students were okay with this problem. The only big issues were if students didn't know how to form a cond statement correctly or if they happened to check for the wrong ages in the `cond` statement (allowing 12 year olds to watch PG-13 and R movies for example).

Part B. A young person who is choosing between two movies to see needs your help: he wants to see the most mature movie possible for his age. Write a procedure `best-movie` which takes a numeric age and two movie ratings as words, and returns the highest of the two ratings that can be attended for that age. If the given age is not old enough to attend either movie, return the word `sorry`.

Part of your solution should use the form of the second, "better" solution in the first *Difference between Dates* case study. Assume you have a working version of `can-order?` to use.

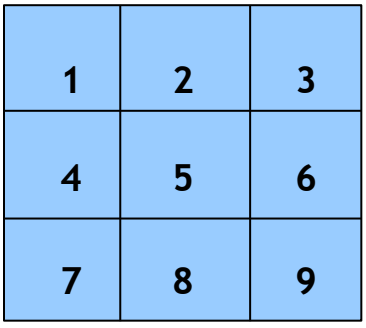
```
(define (best-movie age rate1 rate2)
  (cond
       ((and (not (can-watch? age rate1)) (not (can-watch? age rate2)))
        'sorry)
       ((and (can-watch? age rate1) (not (can-watch? age rate2)))
        rate1)
       ((and (not (can-watch? age rate1)) (can-watch? age rate2))
        rate2)
       ((> (value rate1) (value rate2))
        rate1)
       (else
        rate2)))

(define (value rating)
  (cond ((equal? rating 'g) 1)
       ((equal? rating 'pg) 2)
       ((equal? rating 'pg-13) 3)
       ((equal? rating 'r) 4)
       ((equal? rating 'nc-17) 5)
       (else
        'invalid-rating!)))
```

Common errors for this problem included:

- Not writing the solution in the dbd style, which required you to write a helper that assigned values to certain movie ratings, like in the value helper procedure.
- Incorrectly writing the value procedure, many students decided to assign values that related to the minimum age of a viewer for a certain movie rating (so 0 to both g and pg and 13 to pg-13 and r). By assigning the same value to different ratings, you could possibly return the lower rating if the values are the same.
- Several students forgot to use can-watch? again here and wrote overly complicated procedures to check if rate1 or rate2 can be watched.

Problem 3. Lights out (A: 3, B: 5, C: 4, D: 4, E: 6 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 some other buttons around it. The picture to the right shows the labels of the buttons, 1 to 9 starting in the upper left corner.



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?`.

```
(define (lit? wd) (member wd '(lit 1)))
(define (unlit? wd) (not lit))
```

Part B. Pressing button 4, in this version of the game, will toggle the state of buttons 1 and 7. (This may be different than other versions of the game). 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`.

```
(define (toggle wd) (if (lit? wd) 0 1))

(define (press4 state)
  (se (toggle (first state))
      (bf (bl (bl (bl state))))
      (toggle (last (bl (bl state))))
      (last (bl state))
      (last state)))
```

Many students also toggled the state of button 4 even though the problem specifically said it only toggles states 1 and 7.

If you tried to do this the non-recursive way, if you tried to use a cond, you had to make sure that you had to set both button 1 and 7 in the same conditional lines. If you checked for just lit1? or lit7? and it happened to be true, you would stop the cond statement right there and not go through the rest of the cond.

If you tried to do this recursively, you may have ended up toggling the entire board if you didn't keep track of what button you were currently looking at in your recursive call.

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?`.

```
(define (win? state)
  (cond ((empty? state) #t)
  ((lit? (first state))
   (win? (bf state)))
  (else #f)))
```

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

```
(define (win? state)
  (or (empty? state)
      (and (lit? (first state))
     (win? (bf state)))))
```

One common error for both problems was that students checked if the entire board was lit (as opposed to unlit, as the problem states). No points were taken off for non-recursive solutions, but you should have been able to write recursive solutions for both problems.

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 given.

`(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 (number-lit? n state)
  (cond ((empty? state)
         (<= n 0))

        ((lit? (first state))
         (number-lit? n (bf state)))

        ((unlit? (first state))
         (number-lit? (- n 1) (bf state))) ))
```

Briefly describe the situations in which this version of `number-lit?` does not return the proper value or causes an error:

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

Two errors could be found for this problem:

Incorrect base case: By checking for `(<= n 0)`, if too many lights are lit (more than the `n` argument) you will still return `#t`. `number-lit?` should only return `#t` when the number of lights lit on the board is exactly equal to `n`.

- For example, `(number-lit? 3 '(1 1 1 1 1 1 1 1))` would return `#t` (assuming we fixed the error below)
- How to fix: Turn `(<= n 0)` to `(= n 0)`

Incorrect recursive update: Notice that we subtract `1` from `n` when we find an `unlit?` state. This will find the number of unlit squares, rather than lit squares.

- For example, `(number-lit? 3 '(1 1 0 1 1 0 1 1 0))` returns #t
- How to fix: Switch the `number-lit?` recursive calls of the `lit?` and `unlit?` conditionals, so that we subtract `1` from `n` when we find a `lit?` state.

Problem 4. You'll be counting the days... (A: 8, B: 4 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 the number of days one at a time. He calls this procedure `day-by-day`, to distinguish it from `day-span`; it should function identically, however. He needs help.

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

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

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

       (se (next-month date) 1)

       (se (month-name date) (+ (date-in-month date) 1))  ))



(define (last-day-of-month? date)
   ;;finish this procedure...
  (cond
     ((equal? (month-name date) 'february)
        (equal? (date-in-month date) 28))
     ((member? (month-name date) '(january march may july
                                   august october december))
       (equal? (date-in-month date) 31))
     (else
       (equal? (date-in-month date) 30))))
```

Part B. Provide test-cases for `next-day`. The tests should include only calls to `next-day`, but should test the helper procedures that it uses. 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.

We were looking for 4 different tests, here:

- Testing the middle of the month: `(next-day '(january 12)) ;; (january 13)`
- Testing February 28th : `(next-day '(february 28)) ;; (march 1)`
- The last day of a 31-day month: `(next-day '(january 31)) ;; (february 1)`
- The last day of a 30-day month: `(next-day '(april 30)) ;; (may 1)`

Problem 5. Any letters for me? (A: 7 points)

Consider the procedure `get-2ltr-words` which takes a word as input, and `returns` a sentence of every common two-letter English 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)`

Write `get-2ltr-words`. Use helper functions where it will make your code more readable.

Assume you have a working version of `2ltr-word?`, which takes in a word and returns `#t` if and only if it is a two-letter English word. You do not need to write `2ltr-word?`.

```
(define (get-2ltr-words wd)
  (cond
     ((empty? wd) '())
     ((empty? (bf wd)) '())
     ((2-ltr-word? (word (first wd) (first (bf wd))))
       (se (word (first wd) (first (bf wd)))
           (get-2ltr-words (bf wd))))
     (else
       (get-2ltr-words (bf wd)))))
```