# CS3 Midterm 1

**Fall 2006 Titterton**

**Read and fill in this page 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**: | |

*Official Use Only! No suggestions!*

(1 pt) Prob 0 _____

(9) Prob 1 _____

(6) Prob 2a_____

(4) 2b_____

(10) Prob 3 _____

(9) Prob 4 _____

(9) Prob 5a _____

(4) 5b _____

Raw Total
(out of 52) _____

Scaled Total
(30) _____

You have 80 minutes to finish this test, which should be reasonable.
Your exam should contain 6 problems (numbered 0-5) on 9 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 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!

**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.
And, don't forget to read the "Roman Numerals" case study for the lab!

**1. (9 points). Fill in the blanks.**

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 | (count (word 'cs3 'roxor)) → |
| 2 | (count (sentence 'cs3 'roxor)) → |
| 3 | (member? 'dem '(texas democratice party)) → |
| 4 | (item 3 'a 'b 'c 'd) → |
| 5 | (appearances 'a '(alabama)) → |
| 6 | (equal? (or 'how 'about 'this)<br>(and 'how 'about 'this)) → |
| 7 | (equal? 'fred (or 'joe 'fred)) → |
| 8 | (or 'false<br>(equal? (quotient 7 7) 0)) → |
| 9 | (define (mystery wd)<br>(if (empty? Wd)<br>""<br>(word (first wd) 'cs3 (mystery (bf wd)))))<br><br>(mystery 'cs3) → |

**2. Those scheming doctors…. (A: 6 points, B: 4 points)**

Consider a predicate stressed? that could be used by doctors to determine whether a patient has encountered a jump in heart rate within a certain time period. stressed? takes a sentence of numbers representing heart rates taken each minute, and returns # t if two adjacent rates differ by more than 10 beats per minute, and # f otherwise.

| | | |
|---|---|---|
| (stressed? '(70 71 72 73 74)) | → | # f |
| (stressed? '(70 71 72 83 84)) | → | # t |
| (stressed? '(70 71 72 60 59)) | → | # t |

*Part A.* Below is a buggy version of stressed? with the conditional cases numbered:

```
     (define (stressed? rates)
1      (cond ((empty? rates) # t)
2            ((empty? (bf rates)) # f)
3            ((and (<=  (- (first rates) (first (bf rates))) 10)
                   (<=  (- (first (bf rates)) (first rates)) 10 ))
              (stressed? (bf (bf rates))))
4            (else  # f)))
```

For each of the conditional cases above, fill in the corresponding row on the table below:

| | **Base or recursive case?** | **Write OK if it is correct, otherwise write an argument (i.e., a sentence of rates) that will either return an incorrect answer or cause an ERROR *because of this condition.*** |
|---|---|---|
| **1** | | |
| **2** | | |
| **3** | | |
| **4** | | |

*Part B.* A fellow student wants to simplify the 3rd condition by writing a predicate called within-10?. This procedure takes two numbers, and returns true if there is a difference of 10 or less between the two numbers.

This student, although a very nice person, isn't the most careful scheme programmer. Write test cases for within-10? to fully test the procedure. Write enough test cases to catch all possible bugs, but don't write *too* many: i.e., don't write several test cases that test the same thing.

Make sure you include both the call to within-10? as well as the correct return value.

**3. Leaping from Tuesday to Tuesday… (10 points)**

Write a function named tuesday-span that, when given two dates in 2002, returns the number of Tuesdays in the period spanned by the two dates. January 1, 2002 was a Tuesday (as was December 31). Some examples appear below.

```
(tuesday-span  '(january 1)  '(January 3) )    →  1
(tuesday-span  '(january 2)  '(January 5) )    →  0
(tuesday-span  '(january 6)  '(January 9) )    →  1
(tuesday-span  '(january 1)  '(January 8) )    →  2
(tuesday-span  '(january 1)  '(January 31) )   →  53
```

You may assume that the first argument date is the same as or earlier than the second argument date. You may use any function appearing "Difference Between Dates" code, which works as well for dates in the 2002, without defining it (the "complete" version is included in Appendix A of this test). There are *both* recursive and non-recursive solutions to this problem.

**4. Put on your translation hat… (9 points)**

Write a procedure a12en to translate sentences from "Algolian" to English. Algolian sentences have the following rules, which differ from English:

- Sentences are either questions or statements. In Algolian questions, a "?" will be the first element of the sentence. In English questions, the "?" should go at the end. Statements have no punctuation.
- In an Algolian statement, the subject is in the second position. In English, the subject is in the first position, followed by the verb.
- In an Algolian question, the subject is in the third position. In an English question, the verb is in first position followed by the subject.
- In Algolian, verbs are always at the end of the sentence, and always have an extra "ing" at the end of the word. In English, verbs don't have the extra "ing", and they go first if the sentence is a question, or second if the sentence is a statement.
- The first word in Algolian (that isn't a "?") can't be translated into English. Simply remove it from the corresponding English sentence. All other words not covered by one of these rules should be copied verbatim.

Examples will make this easier to understand:

| | | |
|---|---|---|
| (a12en '(doying I friend with joe bob aming) ) | → | ( I am friend with joe bob) |
| (a12en '(? boing you to dinner cominging) ) | → | (coming you to dinner ?) |
| (a12en '(? pwing joe fish eating) ) | → | (eat joe fish ?) |

The resulting sentences aren't necessarily valid English, as the examples show. Use helper functions in order to make your code easier to read *and* write.

**5. Good help is hard to find… (A: 10 points, B: 4 points)**

This question concerns a database of *tutors*, and the use of good abstraction. A tutor entry is stored in scheme as a word, with 1-3 parts separated by asterisks (*).

- The first part is the tutor's name, and must be present. It can be any length, and will not contain an asterisk.
- The second part is the tutor's discipline, and may or may not be present. Disciplines are always non-numeric, and may be any length, and will not contain an asterisk. You won't have to work with this part, other than recognizing that it could be in a tutor entry.
- The third part is the tutor's rating, and may or may not be present. This is an integer between 1 and 6. If the rating is missing from the entry, that tutor is considered to have a rating of 0.

The following are all valid entries for tutors:

```
Andre_3000
joe*English
Peter_Jennings*2
Sade*Music*5
```

*Part A.* Write two accessors for tutor entries: one for the name and one for the rating. (Do not write one for discipline). Both procedures take a tutor entry as the single parameter. The accessor for name will return the name as a word; the accessor for rating will return an integer between 0 and 6. Be sure to use meaningful names for your procedures and arguments.

*Part B.* Write a predicate hire?, which takes a tutor entry and a sentence of "bab" tutor names. hire? Should return # f if the tutor should not be hired, and # t otherwise. Tutors should not be hired if their rating are below 4 or if their names appear within the sentence of bad tutor names.

| | | |
|---|---|---|
| (hire? 'Sade*music*5<br>'(peter_jennings andre_3000 joe) ) | → | # t |
| (hire? 'Sade*music*5<br>'(peter_jennings andre_3000 joe Sade) ) | → | # f |
| (hire? 'joe '( ) ) | → | # f |

Use proper abstraction.

# Appendix A: Difference between dates, complete version (Appendix B in the reader)

```
; Access procedures for the components of a date.

(define  (month-name  date)  (first  date))
(define  (date-in-month  date)  (first  (butfirst  date)))



; Return the number of days from January 1 to the first day of the named month.

(define  (days-preceding  month)
    (cond
       ( (equal?  month  'january)  0)
       ( (equal?  month  'february)  31)
       ( (equal?  month  'march)  59)
       ( (equal?  month  'april)  90)
       ( (equal?  month  'may)  120)
       ( (equal?  month  'june)  151)
       ( (equal?  month  'july)  181)
       ( (equal?  month  'august)  212)
       ( (equal?  month  'september)  243)
       ( (equal?  month  'october) 273)
       ( (equal?  month  'november)  304)
       ( (equal?  month  'december)  334)  )   )



; Return the number of days from January 1 to the given
; date, inclusive. Date represents a date in 2002.

(define  (day-of-year  date)
    (+
        (days-preceding  (month-name  date) )
        (date-in-month  date) )   )


; Return the difference in days between 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)
   (+
     1
       (-
         (day-of-year  later-date)
         (day-of-year  earlier-date)  )  ) )
```