Read and fill in this page now

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: _____

Prob 0: _____ Prob 1: _____ Prob 2: _____

Prob 3: _____ _____ _____

Prob 4: _____ _____ Prob 5: _____

Prob 6: _____ Total _____/28

You have one hour to finish this test, which should be reasonable; there will be approximately 20 minutes of leeway given past one hour. Your exam should contain six problems (numbered 0-5) on six pages, along with the code from the "Difference Between Dates" program at the end.

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. 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 3-6 and 11 of *Simply Scheme* and the "Difference Between Dates" case study, part 1.

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. We believe we have provided more than enough space for your answers, so please don't try to fill it all up.

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

Relax!

Problem 0 (1 point, 1 minute)

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

Problem 1 (6 points, 10 minutes): Make the expressions correct

Add parentheses and procedures in the underlined areas to make these expressions return the value specified. Do not define any new procedures. Be careful with parentheses and quotes!

You can leave the underlined areas blank, but all the explicit text must be included in your expression—you can't just leave something out. If it is impossible to create an expression that returns the value specified, write IMPOSSIBLE somewhere (to the right of the return value, say).

1]
```
(___________ 'scheme 'is 'a 'state 'of 'mind __________)
➔ (scheme is a state)
```

2]
```
(sentence __________  (item 3 _____________________))
➔ (a mind)
```

3]
```
(_____________________ '(expression) _________________)
➔ i
```

4]
```
(_________________ '(go cal bears) _________________)
➔ gocalbears
```

5]
```
(count _____________)
➔ 0
```

6]
```
(if (or #f (and ___________________ #f))
   'yep 'nope)
➔ yep
```

7]
```
(define (what-the sent)
   (if (empty? sent)
       ""
       (word "-um-" (first sent) (what-the (bf sent)))))

(what-the ________________________________________________)
➔ "-um-i-um-like-um--um-cheese"
```

Note for problem 1: in the exam, students could also add quoted words and sentences into the blanks.

Problem 2 (5 points, 10 minutes) : Validating dates for `day-span`

Write a procedure called `day-span-validated` which takes the same arguments as `day-span` and returns the same value as `day-span` when the argument dates are valid (i.e., legal). If either of the dates is invalid, however, `day-span-validated` should return `(invalid date)`.

For instance:

`(day-span-validated '(february 30) '(june 3))`	➔	`(invalid date)`
`(day-span-validated '(june 4) '(may 15))`	➔	`-19`
`(day-span-validated '(january 19) '(flugalmonth 5))`	➔	`(invalid date)`
`(day-span-validated '(january three) '(may five))`	➔	`(invalid date)`

You can assume that each argument to `day-span-validated` is a sentence containing two words. You can (and should) use procedures from the "Difference Between Dates" case study, part I, that will help you. This code is included in an appendix at the end of the exam. In particular, consider the procedures `day-span`, `days-in-month`, `month-name`, and `date-in-month`. Assume that this is not a leap year.

Problem 3 (6 points, 10 minutes): Data Abstraction in School

You are part of a team working on a project to create, among other things, a database system for schools. One of your tasks is to implement the school datatype in Scheme. This question does not require recursion.

In the project, a *school* will have three pieces of information:

- A short name, which will be a single word,
- A full name, which will be one or more words,
- Enrollment, a number

For instance, here are some possible *schools*:

full name	*short name*	*enrollment*
University of California Berkeley	Cal	23000
Rice	Rice	2822
California Institute of Technology	Caltech	900

Part A
Make a constructor procedure for the *school* datatype. Choose a reasonable name for the procedure, and include good comments. (You will, of course, need to decide how the data is stored; there is more than one correct way to do this.) Include a call to your constructor.

Part B
Make the three selector (accessor) procedures, with reasonable names and comments. DO NOT use recursion when writing these.

Part C
What is a reasonable name of the boolean test procedure for this datatype? (That is, the procedure that somebody can use to test whether a Scheme value is a *school* or not.) Don't write this procedure, just name it.

Problem 4 (5 points, 9 minutes): Argue

Write a function called `argue-response` which takes a sentence and returns a sentence in which the meaning is reversed. Perhaps the easiest way to explain how it reverses meaning is through some examples:

`(argue-response '(I hate the opera))`	➔	`(you love the opera)`
`(argue-response '(you love cookies))`	➔	`(I hate cookies)`
`(argue-response '(you hate everything that I cook))`	➔	`(I love everything that you cook)`

Note that the third example in this first set of test cases above contains an error: the second to last "I" in the argument was transformed to a "you" by `argue-response`, which would require recursion It should read

`(I love everything that I cook)`

If the first two words in the sentence are not of the form you've seen, `argue-response` takes a different tack, adding `(all you ever say is)` to the input:

`(argue-response '(lets have dinner))`	➔	`(all you ever say is lets have dinner)`
`(argue-response '(what))`	➔	`(all you ever say is what)`

Part A:
Fill in the blanks below, assuming that the code implied by *[…]* is written correctly.

```
(define (argue-response remark)
  (cond
    ((starts-with? remark '(i hate))
      _____________________________________________________
      )
    ((starts-with? remark '(i love))    […] )
    ((starts-with? remark '(you love)) […] )
    ((starts-with? remark '(you hate)) […] )

    (else _____________________________________________________
      )
   ) )
```

Part B:
Write the function `starts-with?` so that the above code works correctly.

Problem 5 (5 points, 10 minutes): Count-vowels

Write a recursive procedure named `count-vowels` which takes a single word as input, and returns the number of vowels in the word.

Assume that the function `vowel?` has been defined. You may define and use any other helper functions that make sense to you.

For instance,

`(count-vowels 'fred)`	➔ `1`
`(count-vowels 'frederick)`	➔ `3`
`(count-vowels 'happy)`	➔ `1`

Appendix A: Difference Between Dates, part I, code. This is not a question!

```
; 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 dates.
 (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 procedures 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)
  (cond
    ((equal? month 'january) 1)
    ((equal? month 'february) 2)
    ((equal? month 'march) 3)
    ((equal? month 'april) 4)
    ((equal? month 'may) 5)
    ((equal? month 'june) 6)
    ((equal? month 'july) 7)
    ((equal? month 'august) 8)
    ((equal? month 'september) 9)
    ((equal? month 'october) 10)
    ((equal? month 'november) 11)
    ((equal? month '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.
(define (days-in-month month)
  (cond
    ((equal? month 'january) 31)
    ((equal? month 'february) 28)
    ((equal? month 'march) 31)
    ((equal? month 'april) 30)
    ((equal? month 'may) 31)
    ((equal? month 'june) 30)
    ((equal? month 'july) 31)
    ((equal? month 'august) 31)
    ((equal? month 'september) 30)
    ((equal? month 'october) 31)
    ((equal? month 'november) 30)
    ((equal? month '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 number of days from January 1 to the first day
; of the month 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.  Note: because day-span and general-day-span are equivalent,
; sometimes this function is written as day-span.
(define (general-day-span earlier-date later-date)
  (+ 1 (- (day-of-year later-date)
          (day-of-year earlier-date) ) ) )
```