CS3 Spring 05 – Midterm 1
Standards and Solutions

Problem (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 *2-parts* (6 / 4 points, 10 minutes) : Make test cases

A procedure called `starts-or-ends-with?` (which you can abbreviate as `soew?`) was written by a classmate:

Specification `(soew? letter wd)`
Input: `letter` is a word of length 1; `wd` is a word of any length
Output: `#t` if the letter is at the beginning or at the end of the word, `#f` otherwise

(a) Provide enough test cases that will assure that it works correctly, given the specification above. Do not provide too many test cases, though! Note that you don't know how your classmate wrote `soew?`, so minimize the assumptions you make about how the code is written (although, you will have to assume some things).

You don't need to test for situations outside of the specification (e.g., where the second parameter is a sentence, or not given in a call). You are testing that the *logic* of the program is right.

```
(soew? 'a "")         ; test wd is empty
(soew? 'a 'abcde)     ; test when letter is only at front
(soew? 'a 'edcba)     ; test when letter is only at end
(soew? 'a 'abcdea)    ; test when letter is at both ends
(soew? 'a 'bcdef)     ; test when letter isn't present,
                      ;  or is only in middle

(soew? 'a 'a)         ; one of these three also
(soew? 'a 'b)         ;  necessary
(soew? 'a 'bcade)
```

You got 1 point each for tests in the first group (there was some flexibility on the last, or failure, test). You got one point for any of the tests in the second group. By *far* the most common mistake that you made was forgetting to test for whether `wd` was empty; remember, a length of 0 is most certainly part of "any length". Emptiness is a very important state to consider in Scheme.

(b) Now write `starts-or-ends-with?`

```
(define (soew? letter wd)
  (and (not (empty? wd))
       (or (equal? (first wd) letter)
           (equal? (last wd) letter))))
```

The majority of you forgot to check for emptiness of the `wd` parameter, which lost a point. Other than that, most of you did this correctly. Conditionals were the most common solution—`if` or `cond`—and most wrote code where you did something redundant like this:

```
;; [[wrong, in that it doesn't check for (empty? wd)
(define (soew? Letter wd)
  (if (or (equal? letter (first wd))
          (equal? letter (last wd)))
      #t
      #f)
```

Remember, you don't ever *ever* need to write an `if` like that; the `or` statement returns either `#t` or `#f` itself, and the `if` adds nothing. Well, it does add confusion, in that writing a conditional statement like that makes your code harder to read, after you gain a little experience with reading code. You lost no points for this, but it is a style issue that became quite apparent after reading all of these exams!

Problem (6 points, 8 minutes): Make the expressions correct

Fill in the blanks such that the expression evaluates to the specified value (i.e., what is shown after the arrow). You can leave some blanks unfilled, but cannot ignore text that is present. If no such entry will create the proper evaluation, write the word IMPOSSIBLE to the right of the question.

1]	`(day-span '(january 3) __'(january 3)_____ )` ➔ `1`
2]	*[[for this problem, do not use quotes in text you enter]]* `(__(bf (first (bf__ '(can you hear me) ___) ) )_____)` ➔ `ou`

3]	`(and (equal? 1 _____)`
	`(or (equal? 1 2) (equal? 1 3) (equal? 1 4))`
	`(equal? 1 1))`
	➔ `#t` IMPOSSIBLE

4]	`(define (vowel? ltr)` ;; these functions
	`(member? ltr '(a e i o u)))` ;; are define for the
	;; next three
	`(define (mystery wd)` ;; questions
	`(if (empty? wd)`
	`#t`
	`(if (vowel? (first wd))`
	`#f`
	`(mystery (bf wd)))))`
	
	`(mystery 'strength)` ➔ `_____#f_____`

5]	`(mystery "")` ➔ `_____#t_____`
6]	`(mystery '(this is scheme fosho))` ➔ `_____#t_____`

Each of these responses was worth 1 point, and there was no partial credit available: either you got it totally correct or you got 0 points. Many of you wrote `"item 2"` instead of `"(first (bf"` for the 2nd question, which works just fine.

Problem (6 points, 8 minutes): Count adjecent duplicates

Write a procedure `count-adjacent-duplicates` (`cad`) that takes as input a sentence of any length, and returns a numeric count of the how many adjacent members of the sentence are the same.

`(cad '(I had had enough!))`	➔	1
`(cad '(a a a a))`	➔	3
`(cad '(1))`	➔	0

```
(define (cad sent)
  (cond ((empty? sent) 0)
        ((empty? (bf sent)) 0)
        ((equal? (first sent) (first (bf sent)))
         (+ 1 (cad (bf sent))))
        (else (cad (bf sent)))))
```

-1 point per missing either base case (empty or just one word)
-1 point for any incorrect second recursive case (the else case)

In the main recursive case:
-2 points for (member? (first s) (bf s))
-2 points for bad recursive case
-1 point for minor problems

- The most common problem was missing a base case. Most people were missing the one-word base case.
- Some people tried to use `(member? (first s) (bf s))` to see if two words were next to each other, but that doesn't test adjacency.
- Some people used `(<= (count s) 1)` or `(< (count s) 2)` to handle both base cases at once. This works (although, JJ wants to say that it makes the code much less efficient, since count has to check the whole sentence passed in!)

Problem (7 points, 10 minutes): Backwards day-span…

Write a function `date-in-year` which, in some ways, does the reverse of `day-span`. `date-in-year` takes a numeric span which is the number of days after January 1st in a non-leap year, and returns the date that is defined by that number of days.

`(date-in-year 5)`	➔	`(january 5)`
`(date-in-year 34)`	➔	`(february 3)`
`(date-in-year 325)`	➔	`(november 21)`

You may find the code to the Difference between Dates version 2 case-study useful; it is reproduced in an appendix at the end of this exam. Note that you will receive partial credit for solutions that work for some of the months.

```
(define (date-in-year span)
  (cond ((> span 334)
         (se 'december (- span 334)))
        ((> span 304)
         (se 'novemeber (- span 304)))
        ((> span 273)
         (se 'october (- span 273)))
        ((> span 243)
         (se 'september (- span 243)))
        ((> span 212)
         (se 'august (- span 212)))
        ((> span 181)
         (se 'july (- span 181)))
        ((> span 151)
         (se 'june (- span 151)))
        ((> span 120)
         (se 'may (- span 120)))
        ((> span 90)
         (se 'april (- span 90)))
        ((> span 59)
         (se 'march (- span 59)))
        ((> span 31)
         (se 'february (- span 31)))
        (else
         (se 'january span))))
```

(No login needed here)

The biggest issue on problem 4 was confusing >, <, <=, and >=.

Some of you skipped most of the code. The problem with this is that others who went through the effort of writing it out also risked making some sort of mistake, like getting off by a month or forgetting December (which happened once or twice).

A couple of you tried recursive solutions. In the end, this isn't any easier than the long way shown above, and the code isn't pretty.

Several of you reversed the order of tests in the `cond`, so they first used `(> day 0)` to check for January and then `(> day 31)` for February. Since every day is greater than 0, that code will always return January, and never even check for February!

Finally, some of you tried using difference-between-dates procedures that worked with dates. They don't work well when you just give them a day, and so quite a bit had to be changed. Again, the code above was the easiest.

Problem *2-parts* (8 / 4 points, 13 minutes):
Transform song titles: help a friend with your new powers

A friend knows that you have extensive programming skills, having taken 6 weeks of CS3, and asks if you can help with a problem. He had a big list of song titles on his computer and they somehow got corrupted; he wants you to fix the titles. The titles are words in scheme, without spaces, which is the way your friend wants it. However, there are problems with some of the titles:

- It seems that, sometimes, a title starts with `%20`: that should be removed if present.
- Also, if the title doesn't end in a period, something was cut off: you should add three periods `"..."` at the end to indicate that.
- Finally, some titles have garbage letters in them – either `$`, `&`, or `@` -- if the title has that then the whole thing should be thrown away (turned into an empty word).

The shortest title is 10 letters long, but most are much longer.

`(fix-title '%20JimmyHasAF)`	➔	`JimmyHasAF...`
`(fix-title 'WhenI$WasAChild.)`	➔	`""`
`(fix-title '%20Tom@and^^zaaz)`	➔	`""`
`(fix-title 'SouthMainStreet.)`	➔	`SouthMainStreet.`

(a) Write a procedure `fixed-title` that takes the title (as a word) and returns a fixed version of it, as specified above. You should define the helper procedures that are described below, using good names for the procedures and parameters, and use the helper procedures in `fixed-title`.

```
;;returns true if the title should be thrown away
(define (garbage-title? title)
    (or (member? '@ title)
        (member? '$ title)
        (member? '& title)))
```

Points were taken off for incorrect usage of `or` and `member?`, such as `(equal? (first title) (or '@ '$ '&))` or `(member? title '(@ $ &))`. Please remember to use the single quotes. A few students checked if all the characters in the title were "letters," but this is wrong because it returns false for legal titles such as `hello*every#body^`.

```
;; returns true if the front of the title has a problem
(define  (bad-title-front? title)
    (and (equal? (first title) '%)
         (equal? (first (bf title)) '2)
         (equal? (first (bf (bf title))) '0)))
```

For `bad-title-front?`, 2 points were taken off if only compared `(equal? (first title) '%)` instead of the entire `'%20`. This is not an assumption you can make; always ask us if you are not sure. Some people had an interesting way of doing this by wording together the first three characters of the title `(word (item 1 title) (item 2 title) (item 3 title))` and comparing it to `'%20`. 0.5 point was taken off for using = to compare `%`.

```
;; returns true if the end of the title was cut off
(define bad-title-end? title)
    (not (equal? (last title) '.)))
```

Most students got this one right. 1 point was taken off if the opposite value was returned, i.e. returning true if the end of the title was NOT cut off. A lot of people did this:

```
     (if (not (equal? (last title) '.))
        #t
        #f )
```

Please try to avoid doing this. The logic behind this is basically to return whatever the testing condition returns. It would be stylistically better to lose the `if`:

```
     (not (equal? (last title) '.))
```

```
;; returns the fourth element and on of the word
(define (but-first-three x)
   (bf (bf (bf x))))
```

```
;; fixed-title
(define (fixed-title title)
  (if (garbage-title? title)
    ""
    (if (bad-title-front? title)
        (but-first-three
           (if (bad-title-end? title)
               (word title '...)
               title))
        (if (bad-title-end? title)
           (word title '...)
           title))))
```

1 point was taken off for not handling the case `(fixed-title '%20hello)` when both the front and the end of the title need to be fixed. 1 point was taken off for incorrectly handling `(fixed-title '%20hey&hey@hey$.)` This should have returned `""` because of the garbage. Rather than using nested `if`s, some students accomplished the same thing with a `cond`.

Generally, 1 point was taken off for returning the wrong value, such as a fixed-title instead of a boolean value. 1 point was taken off for bad coding style, such as the poor naming of the procedures or placeholder names that are not descriptive. 1 point was taken off for wrong syntax or usage of procedures. 1 point was taken off for invalid arguments, such as `(word title '(...))`. `word` does not take a sentence as its argument. Another note: some students did `(word test '. '. '.)`; this could be simplified to be `(word test '...)`.

Several of you wrote a recursive solution, (you got full credit):

```
(define (fixed-title title)
   (cond ((garbage-title? title)
          "")
         ((bad-title-front? title)
          (fixed-title (but-first-three title))
         ((bad-title-end? title)
          (word (bl title) '...))
         (else title)))
```

Note, however, this solution will remove `%20` multiple times from front of title, which the specification doesn't explicitly forbid.

(b) Your friend has lots and lots of titles, though; you will need to automate things. Write a procedure `all-titles-fixed` which takes a sentence of titles (each as a word), of any length, and returns a sentence where each title is fixed.

(You can still get full credit on this problem if you haven't answered part (a) correctly)

```
(define (all-titles-fixed sent)
   (if (empty? sent)
      '()
      (se (fix-title (first sent))
          (all-titles-fixed (bf sent)))))
```

Easy! Most of you got this one correct. 1 point was taken off for an incorrect base case. Some returned `""` if `sent` were empty, rather than '(); remember what the range of `all-titles-fixed` is. 2 points were taken off for an incorrect recursive call.