CS 3 (Clancy) **Solutions and grading standards for exam 1**
Spring 2004

Exam information

89 students took the exam. Scores ranged from 5 to 30, with a median of 23 and an average of just over 22. There were 52 scores between 23 and 30, 24 between 16 and 22, 11 between 8 and 15, and 2 between 5 and 7. (Were you to receive a grade of 23 on both your midterm exams, 46 on the final exam, plus good grades on homework and quizzes, you would receive an A–; similarly, a test grade of 16 may be projected to a B–.)

There were two versions of the exam, A and B. (The version indicator appears at the bottom of the first page.)

If you think we made a mistake in grading your exam, describe the mistake in writing and hand the description with the exam to your lab t.a. or to Mike Clancy. We will regrade the entire exam (even if the only error is a mistake in adding up your points).

Solutions and grading standards

Problem 0 (2 points)

You lost 1 point on this problem for each of the following:

- you earned some credit on a problem and did not put your name on the page,
- you failed to indicate who you were sitting next to, or
- you did not indicate your lab section or t.a.

The reason for this apparent harshness is that exams can get misplaced or come unstapled, and we would like to make sure that every page is identifiable. We also need to know where you will expect to get your exam returned. Finally, it is sometimes useful to know where particular students were sitting.

Problem 1 (6 points)

Part a in each version involved adding parentheses and quotes to produce a given value, while in part b you "played interpreter". The part a problems and solutions are given below.

Version A	Version B
`butfirst word last abc`	`butfirst word first xyz`
`(butfirst (word 'last 'xyz))`	`(butfirst (word 'first 'xyz))`

This part was worth 2 points. 1 point was deducted for each error. Most of you got this right.

In part b, you had to evaluate expressions involving sentence and butfirst. Version A's expressions were the following:

expression	value	explanation
`(butfirst '(gh))`	`( )` (the empty sentence)	(gh) is a one-word sentence; removing the first word leaves the empty sentence.
`(butfirst 'x)`	`""` (the empty word)	x is a one-character word; removing the first word leaves the empty word.
`(butfirst 'yz)`	`z`	yz is a two-character word; removing the first character leaves the second.
`(sentence 'ab '(cd ef) (butfirst '(gh)) (butfirst 'x) (butfirst 'yz) )`	`(ab cd ef "" z)`	The sentence procedure ignores arguments that are empty sentences. However, an empty word argument is included in the result sentence.

Version B's expressions were similar, in a slightly different sequence.

This part was worth 4 points. Again, 1 point was deducted per error. You weren't docked twice for making an error in one of the butfirst calls and then repeating it when evaluating the sentence expression.

By far, the most common error in this problem was to neglect to include the empty word in the resulting sentence.

Problem 2 (4 points)

For this problem, you were to describe all arguments for which the appropriate procedure below would not crash, and also describe what the procedure returns for an argument that doesn't cause it to crash.

Version A

```
(define (mystery x)
   (first (butfirst
      (last (butlast x)) )) )
```

Version B

```
(define (mystery x)
   (last (butlast
      (first (butfirst x)) )) )
```

Both procedures require a sentence of two or more words as an argument. Let's look at version A. The expression (last (butlast x)) returns the next-to-last word in a sentence or character in a word, but crashes if its argument is empty or contains only one word or character. Now suppose that x is a word. Then (last (butlast x)) is a character, butfirst of that character is the empty word, and first crashes.

Now suppose x is a sentence with at least two words. From (last (butlast x)) we get the next-to-last word. That word must contain at least two characters for (first (butfirst ...)) not to crash, for the same reason that x had to contain at least two words.

Thus x has to be a sentence with at least two words, whose next-to-last word contains at least two characters. (mystery x) then returns the second character in the next-to-last word of x.

Reasoning for version B's mystery is similar. It returns the next-to-last character in the second word of its argument, which must be a sentence with at least two words, whose second word contains at least two characters.

A description of the legal x values was worth 2 points, as was a description of the return value. Generally, you lost 1 point for an incomplete answer (e.g. "x must be a three-word sentence") and 2 points for an answer that included values for which mystery would crash. Another way to interpret deductions is that missing any of the following lost you 1 point:

- x is a sentence that has at least two words;
- the second/next-to-last word in x has at least two characters;
- the value returned is the second/next-to-last character ...
- in the next-to-last/second word of x.

The most common error on this problem was neglecting to specify that the second/next-to-last word had to contain two characters. It lost 1 point. Some of you attempted to echo the Scheme, saying something like "first you take the butfirst, then the first of that, then the butlast of that, and then the first of that, and that's what mystery returns." This answer earned 0 points.

Problem 3 (10 points)

Here, you were to write a procedure that translates a day of the year (between 1 and 365) into a date in the format accepted by the day-span code. This problem was the same on both versions. Here are two solutions:

```
(define (2003-date day-of-year)
   (sentence
      (number->name (2003-month-number day-of-year))
      (- day-of-year
         (days-preceding
            (number->name (2003-month-number day-of-year)) ) ) ) )

(define (number->name month-number)
   (item month-number '(january february ... december) ) )
```

```
(define (2003-date day-of-year)
   (sentence
      (to-name day-of-year)
      (- day-of-year (days-preceding (to-name day-of-year))) ) )

(define (to-name day-of-year)
   (item
      (2003-month-number day-of-year)
      '(january february ... december) ) )
```

Five features of your solution were evaluated.

a. construction of a two-element sentence;

b. computing the date in the month correctly;

c. using the days-preceding procedure from the case study code to do this computation (you were told to use procedures from the case study code wherever appropriate);

d. conversion of a number to a month name;

e. avoiding the use of cond in doing this conversion.

For each feature, you received 2 points for doing it perfectly, 1 point for doing it imperfectly, and 0 points for doing it badly or at all. Typical errors were as follows.

- Not using sentence lost 2 points. Calling sentence with a wrong argument type, for example by specifying only a procedure name, lost 1 point; doing this for both arguments lost you 2 points.
- Some students tried to use remainder to find the day of the month, apparently confusing dates in the Gregorian calendar with dates in the Islamic calendar. This error usually lost 4 points, 2 for feature b and 2 for feature c.
- Some students lost 2 points for feature c by inventing their own days-preceding procedure rather than using the one in the case study code. Many of you forgot that days-preceding takes a month name as argument; forgetting the call to 2003-month-number lost 1 point here. If you provided a procedure to translate numbers to month names but neglected to call it here, you also lost 1 point.
- People who lost the 2 points for feature d typically either didn't provide a number-to-name translation procedure, or provided an expression that looked exactly like a cond except for using and or or instead of the word cond. (It's possible to do this successfully, but none of you did.) Another way to lose these 2 points was to use the name-of procedure from part 2 of the "Difference Between Dates" case study (you were only allowed to use the code from part 1).
- Many solutions used cond to compute a month name, thereby losing these 2 points. Rewriting the month-number procedure from the case study didn't earn you any credit here. Other errors, each worth a 1-point deduction, were to reverse the arguments to item, to have an off-by-one argument to item, and to quote month names within a list.

Misparenthesizing somewhere cost you 1 point. Providing your own incorrect implementation of 2003-month-number lost you 2 points.

Problem 4 (8 points)

This problem was to provide two implementations of an is-special-day? procedure, one not using if or cond, the other not using and or or. Both your implementations had to avoid redundant uses of =, equal?, and member?. Moreover, you were not allowed to make any assumptions about the format of a date. The two exam versions differed only in the dates; the solutions below are for version A, in which the special dates were March 11, April 2, and April 27.

Here are some full-credit implementations. Solutions with four or fewer uses of =, equal?, or member? could earn full credit.

A procedure without and or or, with four uses of =, equal?, and member?:

```
(define (is-special-day? date)
   (cond
      ((equal? (month-name date) 'march)
       (= (date-in-month date) 11))
      ((equal? (month-name date) 'april)
       (member? (date-in-month date) '(2 27)))
      (else #f) )
```

A procedure without if or cond, again with four uses of =, equal?, and member?:

```
(define (is-special-day? date)
   (or
      (and
         (equal? (month-name date) 'march)
         (= (date-in-month date) 11) )
      (and
         (equal? (month-name date) 'april)
         (member? (date-in-month date) '(2 27)) ) )
```

Implementations with only three uses of =, equal?, and member?:

```
(define (is-special-day? date)
   (cond
      ((equal? (se (month-name date) (date-in-month date)) '(march 11))
       #t)
      ((equal? (se (month-name date) (date-in-month date)) '(april 2))
       #t)
      ((equal? (se (month-name date) (date-in-month date)) '(april 27))
       #t)
      (else #f) ) )

(define (is-special-day? date)
   (or
      ((equal? (se (month-name date) (date-in-month date)) '(march 11))
      ((equal? (se (month-name date) (date-in-month date)) '(april 2))
      ((equal? (se (month-name date) (date-in-month date)) '(april 27))
) )
```

Each implementation was worth 4 points. A solution that correctly identified special days earned you at least 2 points. Deductions for such a solution were as follows:

- 1 point for using two calls to = instead of one call to member? (this was quite common, and most people lost this point twice);
- 2 points for a redundant comparison, that is, re-testing a condition that had been earlier ruled out.

If you made both those errors, you lost 3 points.

The following solutions were categorized as incorrect, and lost 3 points:

- a solution with small logic errors;
- a solution that depended on a particular date representation rather than using month-name and date-in-month.

The latter solution occurred frequently. It received a relatively large deduction because it would fail completely for dates represented in a single word, e.g. may31.

Another error that appeared occasionally involved a misunderstanding of or:

```
(equal? date (or '(march 11) '(april 2) '(april 27)))
```

This was a serious logic error. In addition, you may have lost 1 point for misquoting, bad cond syntax, or misparenthesization.

Incidentally, an implementation that uses *neither* if or cond or and or or (and which thus could be used for both parts) is

```
(define (special-day? date)
   (member?
      (word (month-name date) (date-in-month date))
      '(march11 april2 april27) ) )
```