Fall 05 CS3 Midterm #1, question 3c
Standards and Solutions

The following are buggy versions of the recursive procedure `day-sum`, defined in the cases study *Difference between dates, recursive solution*. (The code for the case study is included as an appendix). The bugs result from small changes which are underlined.

For each version, note whether the bug creates a problem in the

- conditional ,
- the base case,
- making the problem smaller,
- calling the function recursively, or
- combining the recursive calls.

Also briefly describe in English the effect of the bug on the operation of `day-span` as a whole (*not just on `day-sum`)*—this should take between 1 and 2 sentences for each case. You might include an example call to day-span illustrating the problem, although this isn't necessary with a sufficient explanation (and, might be wrong!).

Don't be too verbose! We may deduct points if our eyes start to bleed.

Each part was worth 3 points: 1 point for noting where the problem was, and 2 points for your description. In general, too low level of a description lost you 1 or maybe 2 points. We didn't want an english description of how the procedure worked, but a description of what its implications were.

```
(define (day-sum first-month last-month)
  (if (>= first-month last-month)
    0
    (+ (days-in-month (name-of first-month))
       (day-sum (+ first-month 1) last-month))))
```

Conditional. [Note: this bug changes the test in the conditional, it doesn't change the value of the base case!]

The bug will cause `day-sum` to drop out of the recursion before it adds in the number of days in the last month. This will cause day-span to return a too-small number when called with non-equal or consecutive months.

Some of you wrote that this would add an extra month, or wrote that it would lose a middle month rather than the last month.

```
(define (day-sum first-month last-month)
  (if (> first-month last-month)
    0
    (+ (days-in-month (name-of first-month))
       (day-sum first-month (+ last-month 1)))))
```

Making the problem smaller. [The recursive call is made in such a way that the problem

doesn't get simpler each time]

The bug will cause day-sum to loop infinitely, and cause calls to `day-span` to never return a value (until scheme crashes). Those of you that said, simply, `day-span` will return an error only received one point for your explanation.

Some of you simply said that `day-sum` would cause an error, which wasn't enough to get full credit here. If you emphasized that "all of scheme would crash", we did give you full points, but the much preferable answer included some notion of a never-ending situation.

```
(define (day-sum first-month last-month)
  (if (<= first-month last-month)
    0
    (+ (days-in-month (name-of first-month))
       (day-sum (+ first-month 1) last-month))))
```

Conditional.

With normal arguments to `day-span` (where the month of the earlier date is not later than the month of the later date), `day-sum` will return `0` every time. This will cause `day-span` to fail to add any months between dates with non-consecutive or equal months (essentially, treating these situations as if the months were consecutive).

Some of you noted that in situations where the first month was greater than the last month this would recurse forever, which was OK but not necessary (actually, the `item` call in `(name-of first-month)` will cause an error when first-month grows past `12`).

```
(define (day-sum first-month last-month)
  (if (> first-month last-month)
    1
    (+ (days-in-month (name-of first-month))
       (day-sum (+ first-month 1) last-month))))
```

Base case. [The value that `day-sum` returns for the base case is wrong.]

`day-sum` will return a number `1` greater than it should, because of the erroneous base case. This will cause `day-span` to return a number `1` greater than it should for dates that are non-equal or consecutive.

Some of you wrote that this buggy version would return extra day for each month, which is wrong.