"Difference Between Dates" Case Study (Recursive Version)
© 1996 M. J. Clancy and M. C. Linn

Preparation

The reader should have been introduced to recursion.

A recursive solution

How can recursion be used to implement the day-span **procedure?**

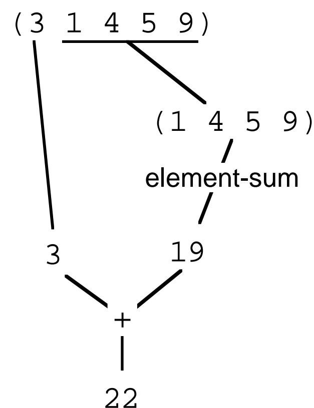
An alternative solution to the day-span procedure may be designed using recursion, a procedure's call of itself. Recall the dead end encountered in the original design: given the start and end of a month range, we were unable to determine how many days the months in that range contain.

How can recursion be used to add values in a sentence?

One defines a recursive computation in terms of a simpler yet similar computation. This allows the computation to be repeated and its results accumulated. For example, recursion for sentence processing typically involves regarding a sentence as the combination of its first word and the sentence of remaining words; thus a procedure that returns the sum of all the numbers in a sentence would be coded as

```
(define (sum-of-all sent)
    (if (empty? sent) 0
        (+ (first sent) (sum-of-all (bf sent))) ) )
```

A diagram that represents the computation of the sum of all the words in the sentence (3 1 4 5 9) is



Any recursion involves a *base case*, the situation(s)—there may be more than one—that represents the simplest possible computation. A base case for a recursion involving a sentence is typically the empty sentence, as in the sum-of-all procedure.

How can recursion be used without a sentence?

Values to be added don't necessarily have to be already collected in a sentence. One may use recursion to find the sum of all integers in a given range, say from 22 to 500. The approach here is to break the range down into a value—we'll call it v—combined with a smaller range, find

the sum of all the integers in the smaller range, then add it to v. The simplest possible range of integers to add is the empty one, and that provides the base case. Here are two procedures that find the sum of integers in a range whose endpoints are called left and right.

```
(define (range-sum1 left right)
   (if (> left right) 0
      (+ left (range-sum1 (+ left 1) right)) ) )

(define (range-sum2 left right)
   (if (> left right) 0
      (+ right (range-sum2 left (- right 1))) ) )
```

Stop and consider ➨ *Which of the two* range-sum *procedures do you prefer? Why?*

The two procedures break down the range of integers in different ways. In range-sum1, the range of integers is broken down into its first element and the range of remaining integers; in range-sum2, the range is broken down into the range of integers from the first up to but not including the last, and the last integer itself. For example, the range of integers between 22 and 500 would be broken down in one of two ways:

22 and 500 is …

> 22 combined with the range of integers between 23 and 500, inclusive; or
>
> the range of integers between 22 and 499, inclusive, combined with 500.

How can range-sum **be tailored to compute the number of days in a range of months?**

A range of months isn't much different from a range of integers. Instead of the integers in the range, the month lengths for months in the range are to be added. We modify one of the range-sum procedures as follows:

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

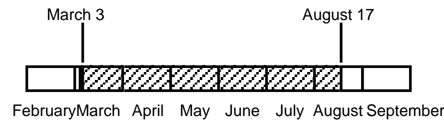
What extra code is needed?

An expression computing the number of days in first-month must still be designed. Reviewing the code in part I of the case study, we notice the days-in-month procedure; it seems to provide exactly what's needed, except that its argument is a month name rather than the integer representing that month. What's needed is the inverse of the month-number procedure that, given a month name, returns its numeric index—but such a procedure will be easy to provide. We'll call it name-of. Substituting the appropriate calls to these procedures gives

```
(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)) ) )
```

Stop and help ➨ *Write the* name-of *procedure.*

The last step is to determine how to use day-sum**. What was missing from the dead-end code was the** general-day-span **procedure, which was to handle the case where the two argument dates were neither in the same month nor in consecutive months, for example, March 3 and August 17. In part 1 of this case study, we represented this in diagram form:**



The day-sum **procedure just designed allows the computation of the number of the days of the complete months shaded in the diagram. The days in the partial months at the ends of the shaded region can be counted as in** consec-months-span**. Required in addition is code to find the first complete month and the last complete month in the range; we'll call these procedures** next-month-number **and** prev-month-number**.**

Stop and help ➨ *Write the* next-month-number *procedure.*

Stop and consider ➨ *Why should the procedure be named* next-month-number *instead of* next-month *or* next-month-name*?*

The following procedure results. The complete program appears in Appendix C.

```
(define (general-day-span earlier-date later-date)
    (+
        (days-remaining earlier-date)
        (day-sum
            (next-month-number earlier-date)
            (prev-month-number later-date) )
        (date-in-month later-date) ) )
```

Outline of design and development questions

A recursive solution

How can recursion be used to implement the day-span procedure?

- How can recursion be used to add values in a sentence?
- How can recursion be used without a sentence?
- How can range-sum be tailored to compute the number of days in a range of months?
- What extra code is needed?

Exercises

Application 1. Write a procedure called total-days that, given a sentence of month names, returns the total number of days in those months.

Application 2. The following procedure is intended to compute the difference between its argument dates recursively, reducing the day span between the argument dates by exactly one day at each recursive call. Supply the code it's missing, along with any auxiliary procedures that are necessary.

```
(define (day-span earlier-date later-date)
   (if (equal? earlier-date later-date) 1
      (+ 1 (day-span _____ _____ ) ) ) ) )
```

Analysis 3. Are the first two cases in day-span still necessary? That is, can day-span now be coded merely as a call to general-day-span? Explain why or why not.

Analysis 4. What happens if the dates provided to general-day-span are out of order, that is, earlier-date is later than later-date?

Debugging 5. Your programming partner, in a late-night coding session, changes a line in the Scheme code, with the result that day-span's return values are now too large:

call to day-span	*returned value*
`(day-span '(january 1) '(december 31))`	*396*
`(day-span '(february 1) '(december 31))`	*362*
`(day-span '(january 1) '(november 30))`	*365*

What line could your partner have changed to produce this behavior?

Application 6. Write a procedure average that, given a sentence of numbers, returns the average of the values in the sentence.

Application 7. Write a procedure standard-deviation that, given a sentence of numbers, returns their standard deviation. If the numbers in the sentence are x_1, x_2, …, x_n and m is their average, then the standard deviation of the numbers is

$$\sqrt{\frac{(x_1-m)^2 + (x_2-m)^2 + \ldots + (x_n-m)^2}{n}}$$

Analysis 8. Consider a procedure product-of-all that, given a sentence of numbers, returns their product. Thus

```
(product-of-all '(2 3 5))
```

should return 30. What would be the base case for product-of-all, and what value should be returned in this case?

Application 9. Code the days-preceding procedure as a single call to day-sum.

Reflection 10. What's your "mental image" of recursion? That is, when you picture a recursive computation in your head, what does it look like?

Modification 11. Modify one of the range-sum procedures so that it returns the sum only of the integers in the given range that satisfy the predicate good?.