Local Procedures and Recursion

**Summary:** In this laboratory, we consider the various techniques for creating local recursive procedures, particularly `letrec` and named `let`. We also review related issues, such as husk-and-kernel programming.

**Contents:**

- Exercises
  - Exercise 1: The Last Element
  - Exercise 2: Alternating Lists
  - Exercise 3: Iota, Revisited
  - Exercise 4: Taking Some Elements
  - Exercise 5: Taking Some More Elements
  - Exercise 6: Reflection

**Exercises**

**Exercise 1: The Last Element**

a. Define a recursive procedure, `last-of-list`, a recursive procedure that returns the last element of a list.

b. Using that procedure, compute the sum of the last elements of the lists `(3 8 2)`, `(7)`, and `(8 5 9 8)`.

Note that you will probably need to make three calls to `last-of-list`.

c. Rewrite your solutions to the previous two problems using a `letrec`-expression in which
   - the identifier `last-of-list` is locally bound to a recursive procedure that finds and returns the last element of a given list, and
   - the body of the expression computes the sum of the last elements of the lists `(3 8 2)`, `(7)`, and `(8 5 9 8)`.

The body of your expression should invoke `last-of-list` three times.

Note that you are to write an expression and not a procedure (other than the local `last-of-list`) for part c of this exercise.
Exercise 2: Alternating Lists

A non-empty list is an \textit{s-n-alternator} if its elements are alternately symbols and numbers, beginning with a symbol. It is an \textit{n-s-alternator} if its elements are alternately numbers and symbols, beginning with a number.

Write a \texttt{letrec} expression in which

- the identifiers \texttt{s-n-alternator?} and \texttt{n-s-alternator?} are bound to \textit{mutually recursive} predicates, each of which determines whether a given non-empty list has the indicated characteristic, and
- the body invokes each of these predicates to determine whether the list \((2 \ a \ 3 \ b \ 4 \ c \ 5)\) fits either description.

Your \texttt{letrec} expression should have the form

\begin{verbatim}
(letrec
  ((s-n-alternator? ...)
   (n-s-alternator? ...))
  ...)
\end{verbatim}

\textit{Note: By “mutually recursive”, we mean two procedures that call each other.}

Exercise 3: Iota, Revisited

As you may recall, the \texttt{iota} procedure takes a natural number as a parameter and returns a list of all the lesser natural numbers in ascending order. For example,

\begin{verbatim}
> (iota 5)
(0 1 2 3 4)
\end{verbatim}

a. Define and test a version of the \texttt{iota} procedure that uses \texttt{letrec} to pack an appropriate kernel inside a husk. The husk should do precondition testing and the kernel should build the list. This version of \texttt{iota} should look something like

\begin{verbatim}
(define iota
  (lambda (num)
    (letrec ((kernel (lambda (...) ...)))
      (cond
        ((fails-precondition) (error ...))
        ...
        (else (kernel num))))))
\end{verbatim}

b. Define and test a version of the \texttt{iota} procedure that uses a named \texttt{let}. This version of \texttt{iota} should look something like
(define iota
  (lambda (num)
    (cond
      ((fails-precondition) (error ...))
      ...
      (else
       (let kernel (...)
         ...)))))))

Exercise 4: Taking Some Elements

Define and test a procedure, (take n lst), returns a list consisting of the first n elements of the list, lst, in their original order. You might also think of take as returning all the values that appear before index n.

For example,

> (take 3 (list 'a 'b 'c 'd 'e))
(a b c)
> (take 2 (list 2 3 5 7 9 11 13 17))
(2 3)
> (take 0 (list "here" "are" "some" "words"))
()
> (take 8 (string->list "triskadecaphobia"))
(#\t #\r #\i #\s #\k #\a #\d #\e)
> (take 2 (list null null))
(() ())

The procedure should signal an error if lst is not a list, if n is not an exact integer, if n is negative, or if n is greater than the length of lst.

Note that in order to signal such errors, you may want to take advantage of the husk-and-kernel programming style.

Exercise 5: Taking Some More Elements

Rewrite take to use whichever of named let and letrec you didn’t use in the previous exercise.

Exercise 6: Reflection

You’ve now seen two examples in which you’ve written two different solutions, one using letrec and one use named let. Reflect on which of the two strategies you prefer and why.

Copyright © 2006 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.