Lab 3: Recursion
12:00 PM, Sep 23, 2018

Contents

1 Intro to Recursion 1
2 Sum and Product 2
3 Spell Check 2
4 Double the Fun! 3
5 Candlestick Search 3
6 Reverse 4

Objectives

By the end of this lab, you’ll be able to:

- write procedures that recur on lists
- write procedures that retrieve the last element of a list

1 Intro to Recursion

In this exercise, we are going to practice visualizing recursive functions. Here is an example of a recursive function:

\[
f(n) = \begin{cases} 
0 & \text{if } n = 0 \\
1 + f(n-1) & \text{if } n > 0 
\end{cases}
\] (1)

We can use the rules to compute \(f(4)\)
\[
f(4) = 1 + f(3)
\]
\[
f(3) = 1 + f(2)
\]
\[
f(2) = 1 + f(1)
\]
\[
f(1) = 1 + f(0)
\]
\[
f(0) = 0
\]

From this we can build up to the solution as follows,
\[
f(1) = 1 + 0 = 1
\]
\[
f(2) = 1 + 1 = 2
\]
Problem:

\[ f(n) = \begin{cases} 
0 & \text{if } n = 0 \\
2 + f(n - 1) & \text{if } n > 0 
\end{cases} \]  

(2)

Task: What is \( f(4) \)?  
Note: Make sure to show your work as demonstrated in the example.

Problem: Suppose we know two things about a function that maps natural numbers to integers, \( f : \mathbb{N} \rightarrow \mathbb{Z} \):

\[ f(n) \leq \begin{cases} 
0 & \text{if } n = 0 \\
2 + f(n - 1) & \text{if } n > 0 
\end{cases} \]  

(3)

Task: What is the maximum possible value of \( f(4) \)?  
Task: What is the minimum value of \( f(4) \)?

You’ve reached a checkpoint! Please sign up to get a lab TA to review your work.

2 Sum and Product

Task: Write a procedure, called \textbf{sum-list}, that sums up the elements of a list of integers. The sum of an empty list should be 0. For example:

\begin{verbatim}
(sum-list (quote (3 4 5))) => 12
(sum-list empty) => 0
\end{verbatim}

Task: Write a procedure, called \textbf{prod-list}, that multiplies the elements of a list of integers. The product of an empty list should be 1. For example:

\begin{verbatim}
(prod-list (quote (3 4 5))) => 60
(prod-list empty) => 1
\end{verbatim}

You might notice striking similarities between the two procedures you wrote for this problem. What are the differences between them? Soon, you will learn how to write one procedure that can either sum up the numbers in a list or multiply them (or sum up their square roots, or anything the caller wants) depending on how it is invoked.
3 Spell Check

Colonel Mustard is teaching Mrs. Peacock to write English, allowing her to make the letter writing process even easier. Unfortunately, spelling is not Mrs. Peacock’s forte, so they have enlisted your help!

Your job in this problem is to write two spell-checkers, each of which catches one particular spelling mistake.

Task: Like many students, Mrs. Peacock thinks that the verb form of recursion is recurse. It’s not. According to Webster’s Dictionary, recurse is not even a word. Write a procedure called spell-check-recurs that replaces the non-word recurse with the word recur. It should run as follows:

```
(spell-check-recurs
 (quote (then recurse on the remainder of the list)))
=> (then recur on the remainder of the list)
(spell-check-recurs
 (quote (to recurse or not to recurse)))
=> (to recur or not to recur)
```

Task: Write a procedure called spell-check-alot that replaces the non-word alot with the words a and lot. Your procedure should run as follows:

```
(spell-check-alot
 (quote (i go skiing alot but only when there is alot of snow)))
=> (i go skiing a lot but only when there is a lot of snow)
(spell-check-alot
 (quote (i like you alot)))
=> (i like you a lot)
```

4 Double the Fun!

Task: Write a procedure (pairs alos) that takes as input a list of symbols and returns a list of pairs (two-element lists) with every element of the original list appearing twice. For example,

```
(pairs (quote (why do you say everything twice)))
=> ((why why) (do do) (you you) (say say) (everything everything) (twice twice))
```
You’ve reached a checkpoint! Please sign up to get a lab TA to review your work.

5 Candlestick Search

Colonel Mustard and Mrs. Peacock are looking for a candlestick but don’t know which room it’s in. Fortunately for Colonel Mustard, he has been keeping a list of rooms they have already visited, listed from the last room visited to the first. To start, Mrs. Peacock thinks they should check the innermost room, i.e. the first room that they visited.

Task: Help Mrs. Peacock by writing a procedure \texttt{last}, whose input is a list, which retrieves the very last (oldest) element in the list. You may assume all inputs are nonempty lists. In your Design Recipe, you can call this a NElist.

While Mrs. Peacock is checking the first room, Colonel Mustard wants to check the rest of the rooms.

Task: Write a procedure \texttt{all-but-last} whose input is a nonempty list and output is the list of elements, in order, excluding the last element.

6 Reverse

When you write a recursive procedure that processes a list, you have been taught that the recursive input is derived from the original input by dropping the first element; that is, the procedure is recursively applied to the \texttt{cdr} of the list (also called the \texttt{rest} of the list, for the less nostalgic among you). This works very well and is almost always the best choice.

However, now that we have an \texttt{all-but-last} procedure and a \texttt{last} procedure, we can consider a different form of recursion, in which the recursive input is derived from the original input by dropping the last element.

Task: Write a procedure \texttt{my-reverse} with the following spec:

- **input**: a list \texttt{L}
- **output**: a list with the same elements as \texttt{L} but in reverse order.

Example:

\begin{verbatim}
> (my-reverse (quote (I am Yoda)))
(Yoda am I)
\end{verbatim}

(The reason that the procedure is called \texttt{my-reverse} is that there is a built-in \texttt{reverse} procedure. Of course, you should not use \texttt{reverse} in \texttt{my-reverse}; you should use recursion.

Once a lab TA signs off on your work, you’ve finished the lab! Congratulations! Before you leave, make sure both partners have access to the code you’ve just written.
Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: http://cs.brown.edu/courses/csci0170/feedback.