Lab 3: Recursion
12:00 PM, Sep 23, 2019

Contents

1 Intro to Recursion 1
2 Sum and Product 2
3 Spell Check 2
4 Double the Fun! 4
5 Just for Fun: Address Book 5

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,

\[ f(1) = 1 + 0 = 1 \]
\[ f(2) = 1 + 1 = 2 \]
\[ f(3) = 1 + 2 = 3 \]
\[ f(4) = 1 + 3 = 4 \]
Problem:

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

**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 \( f : \mathbb{N} \rightarrow \mathbb{Z} \)

\[ f(n) \leq \begin{cases} 
0 & \text{if } n = 0 \\
2 + f(n-1) & \text{if } n > 0 
\end{cases} \quad (3) \]

**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 call over a lab TA to review your work.

## 2 Sum and Product

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

```scheme
(sum-list (cons 3 (cons 4 (cons 5 empty))))
=> 12
```

```scheme
(sum-list empty)
=> 0
```

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

```scheme
(prod-list (cons 3 (cons 4 (cons 5 empty))))
=> 60
```

```scheme
(prod-list empty)
=> 1
```

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.

You’ve reached a checkpoint! Please call over a lab TA to review your work.
3 Spell Check

Betty is teaching her farm dog Spot to write, allowing her to make the letter writing process even easier. Unfortunately, spelling is not Spot’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 aspiring mail-carrier dogs, Spot 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-recur that replaces the non-word “recurse” with the word “recur”. It should run as follows:

```
(spell-check-recur
 (cons "then"
 (cons "recurse"
 (cons "on"
 (cons "the"
 (cons "remainder"
 (cons "of"
 (cons "the"
 (cons "list" empty))))))))
=> (cons "then"
 (cons "recur"
 (cons "on"
 (cons "the"
 (cons "remainder"
 (cons "of"
 (cons "the"
 (cons "list" empty))))))))
```

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

```
(spell-check-alot
 (cons "to"
 (cons "recurse"
 (cons "or"
 (cons "not"
 (cons "to"
 (cons "recurse" empty)))))))
=> (cons "to"
 (cons "recur"
 (cons "or"
 (cons "not"
 (cons "to"
 (cons "recur" empty))))))
```

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

---

1. The OED entry for “recurse” calls the word obsolete and refers you to “recur”.
2. I love this alot: hyperbolean and a half
4 Double the Fun!

**Task:** Write a procedure `(double alos)` that takes as input a list of strings and returns a list in which every element of the original list appears twice. For example,

```
(double (cons "im" (cons "seeing" (cons "double" empty))))
```
Task: Write a procedure (pairs alos) that takes as input a list of strings and returns a list of pairs (two-element lists) with every element of the original list appearing twice. For example,

\[
\text{pairs (cons "why" (cons "do" (cons "you"
\text{ (cons "say" (cons "everything" (cons "twice" empty))))))})
\]

=> (cons
    (cons "why" (cons "why" empty))
    (cons
        (cons "do" (cons "do" empty))
        (cons
            (cons "you" (cons "you" empty))
            (cons
                (cons "say" (cons "say" empty))
                (cons
                    (cons "everything" (cons "everything" empty))
                    (cons
                        (cons "twice" (cons "twice" empty))
                        empty))))))

You’ve reached a checkpoint! Please call over a lab TA to review your work.

5 Just for Fun: Address Book

As winter approaches and photosynthesis slows, Rahul, Spike’s pig, wants to go to the locations where he has been collecting and burying truffles, to obtain the oldest one to put on his pizza. Fortunately for Spike, he has been keeping a list of places where he has been keeping his truffles, listed from newest to oldest. To start, Spike wants to retrieve the innermost, or oldest, truffle in his collection, before it spoils.

Task: Help Spike by writing a procedure, last, whose input is a list of strings (places where Rahul buried the truffles), 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.

Spike is having guests over, and wants to dig up bones to serve to them. He does not want to serve them old truffles, so he decides to harvest every truffle except the last truffle from his list of places.

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

Note: Spike realizes that these procedures are highly inefficient compared to their analogues, first and rest. Can you see why?
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/cs017/feedback](http://cs.brown.edu/courses/cs017/feedback)