COMP 50 Lab: Lists

Fall 2013

This lab gives you practice working with lists. When you complete the lab, you should understand several data definitions based on lists, as well as how to design functions for them.

The lab is configured as a long series of short problems. Plan on finishing about half of them. All problems use the design recipe for lists, and most (but not all) problems can be solved using the simple template for natural recursion.

Example problem

Suppose you are given the problem

> Define function `sum`, which consumes a list of numbers and produces the sum of the numbers in the list.

Your solution should look something like this:

```
; A list-of-numbers is a one of:
; - empty
; - (cons number list-of-numbers)

; sum : list-of-numbers -> number
; return the sum of the given list of numbers
(define (sum nums)
  (cond [(empty? nums) 0]
        [(cons? nums) (+ (first nums) (sum (rest nums)))]))
(check-expect (sum (cons 1 (cons 2 (cons 3 empty)))) 6)
(check-expect (sum (cons 1 (cons 5 (cons 6 (cons 3 empty))))) 15)
```

The problems

1. Make a data definition for a list of movies, where each movie has a title and a duration in minutes. Show at least three examples.
2. Define a function that consumes a list of movies and returns a Boolean telling whether the movies can be shown consecutively in a single 24-hour period.
3. Design a function `join-strings` which consumes a list of strings and produces a string which is the concatenation of each string in the list. For example,

```
(define a-list-of-strings
  (cons "A"
        (cons "List"
              (cons "Of"
                    (cons "Strings" empty)))))

(check-expect (join-strings a-list-of-strings) "AListOfStrings")
```

4. Design a function `join-strings-with-space` which consumes a list of strings and produces a string which is the concatenation of the strings in the list interspersed with a space (`" "`). For example,

```
(check-expect (join-strings-with-space a-list-of-strings) "A List Of Strings ")
```

5. Design a function `join-strings-with-string` which consumes a string and a list of strings and behaves like `join-strings-with-space`, except it uses a given string instead of a space. For example,

```
(check-expect (join-strings-with-string "!!!" a-list-of-strings)
              "A!!!List!!!Of!!!Strings!!!")
(check-expect (join-strings-with-string " wheeeeeee " a-list-of-strings)
              "A wheeeeeee List wheeeeeee Of wheeeeeee Strings wheeeeeee ")
(check-expect (join-strings-with-string " " a-list-of-strings)
              "A List Of Strings ")
(check-expect (join-strings-with-string "" a-list-of-strings)
              "AListOfStrings ")
```

   Would you agree that `join-strings-with-string` is "more powerful", "more general", or "more flexible" than `join-strings-with-space`? Why or why not?

6. Here is a data definition of a list of stock trades:

```
; A `list-of-trades` is one of:
; - empty
; - (cons (make-stock-sale     symbol shares pps) list-of-trades)
; - (cons (make-stock-purchase symbol shares pps) list-of-trades)
; where symbol is the symbol under which the company trades
;       shares is the number of shares traded
;       pps is a number representing the price per share of the given trade
(define-struct stock-sale     (company shares price-per-share))
(define-struct stock-purchase (company shares price-per-share))
```

   Design a function that takes a list of stock trades and returns a list of the traded companies (that is, their stock symbols).

7. Design a function that takes a list of stock trades and a symbol and returns the total values of traded shares (both bought and sold) for the company represented by the symbol.

8. In BSL, as in almost all computer languages, a string is a sequence of *characters*. Breaking a string into characters can be useful when we want to rearrange characters or interrogate individual characters. In the BSL and Racket documentation, as in many other languages, character is abbreviated as `char`.

   Here are a couple of BSL functions on strings and characters:

```
string->list : string -> list-of-chars
convert a string to an equivalent list of characters

char-lower-case? : char -> boolean
tell if a character represents a lower-case letter
```

   Design a function that takes a string and returns a Boolean that tells whether the string is composed entirely of lower-case letters.

   ...

9. Design a function `keep-lower-case` that takes a list of strings and returns a new list containing only those strings which are composed entirely of lower-case letters. Example

```
(check-expect (keep-lower-case (cons "Able" (cons "was" (cons "I" empty))))
              (cons "was" empty))

(check-expect (keep-lower-case
                   (cons "It" (cons "wasn't" (cons "me" (cons "really" empty)))))
              (cons "me" (cons "really" empty)))
```

10. Design the function `nth` which takes a list of symbols and a number `n`, and returns the `n`th symbol of the list, counting from *zero*.[1]

```
(check-expect (nth (cons 'a (cons 'b (cons 'c (cons 'd empty)))) 0)
              'a)
(check-expect (nth (cons 'a (cons 'b (cons 'c (cons 'd empty)))) 2)
              'c)
(check-error (nth (cons 'a (cons 'b (cons 'c (cons 'd empty)))) 99))
```

11. Design the function `replace-nth` which takes a list of symbols, a symbol `s`, and a number `n`, and returns a list that is exactly like the original list, except that the `n`th symbol is now `s`.

```
(check-expect (replace-nth (cons 'a (cons 'b (cons 'c (cons 'd empty)))) 'new 2)
              (cons 'a (cons 'b (cons 'new (cons 'd empty)))))

(check-expect (replace-nth (cons 'a (cons 'b (cons 'c (cons 'd empty)))) 'yay 0)
              (cons 'yay (cons 'b (cons 'c (cons 'd empty)))))

(check-expect (replace-nth (cons 'a (cons 'b (cons 'c (cons 'd empty)))) 'end 3)
              (cons 'a (cons 'b (cons 'c (cons 'end empty)))))
```

Submitting the lab

Ten minutes before the end of the lab, put the following text at the beginning of a DrRacket file. You may use an empty file or the source code you have been working on:

```
#|
What I did during this lab:
   (you fill in this part)

What I learned during this lab:
   (you fill in this part)

|#
```

The lab staff will help you articulate what you learned.

Finally, *submit this file through the handin server* as `lab-lists`. You will need to submit it using *two* usernames connected by a + sign, as in

```
Jane.Doe+Richard.Roe
```

You submit using Jane Doe's password.

[1] Counting from zero is a strange thing that computer scientists learn to do because it makes the math and the code come out well.