Fundamentals of Computer Science I (CS151.01 2006F)

Homework 3: Reconfiguring Lists

Assigned: Friday, 1 September 2006
Due: Tuesday, 5 September 2006
No extensions!

Summary: In this assignment, you will use list operations to reconfigure the elements of lists. You will both use list operations to extract elements from lists and find expressions for building lists that are equivalent to one another.

Purposes: To give you experience with list operations. To help you consider the relationships among list operations. To give you experience writing compound expressions.

Expected Time: One hour.

Collaboration: You should first do the homework by yourself. After doing the best job you can on your own, find a few other people in class and compare answers, extending your list of solutions as appropriate. Make sure to cite those partners. (You may interpret "few" as any number between one and fifty.)

Submitting: Email me your work, using a subject of *CSC151 Homework 3*. More details below.

Warning: So that this exercise is a learning assignment for everyone, I may spend class time publicly critiquing your work.

Assignment

Part A. Decomposing Lists

Citation: The exercise is based on Exercise 1.13 of Springer and Friedman (1989), *Scheme and the Art of Programming*

If we define the list breakfast thus

```
(define breakfast (list 'spam (list 'eggs 'spam) (list 'spam 'spam)))
```

then we might figure out how to extract the symbol `eggs` from `breakfast` as follows:

```
> breakfast
(spam (eggs spam) (spam spam))
> (cdr breakfast)
((eggs spam) (spam spam))
> (car (cdr breakfast))
(eggs spam)
> (car (car (cdr breakfast)))
eggs
```

Note that the first two operations were exploratory; only the final expression is necessary to extract `eggs` from `breakfast`.

For each of the expressions below, write at least one expression that extracts the symbol `eggs`. You may use the following procedures from the lab as you see fit: `append`, `car`, `cdr`, `cons`, `length`, `list`, `list-ref`, `nil`, and `reverse`.

1. `(define super-spam-plate (list 'spam 'spam 'spam 'spam 'spam 'eggs 'spam))`

2. `(define spam-and (list (list 'spam 'eggs) (list 'toast 'spam)))`

3. `(define artery-buster (list (list 'spam 'bacon) (list 'eggs) 'spam))`

4. `(define nested-spam (list (list (list 'eggs 'spam))))`

Part B. Alternate Constructions

Suppose we've defined two lists as follows:

```
(define abc (list 'ape 'baboon 'chimp))
(define yz (list 'yak 'zebra))
```

Given these two lists, there are (at least) two different ways to construct the list `(chimp yak zebra)` using Scheme list procedures.

You could use `append` and `cdr`:

```
> (cdr abc)
(yak zebra)
> (cdr (cdr abc))
(chimp)
> (append (cdr (cdr abc)) yz)
(chimp yak zebra)
```

Alternately, you could use `cons`, `car`, and `reverse` to get exactly the same list as a result:

```
> (reverse abc)
(chimp baboon ape)
> (car (reverse abc))
chimp
> (cons (car (reverse abc)) yz)
```

You could even use just `list-ref` and `list`.

```
> (list (list-ref abc 2) (list-ref xy 0) (list-ref xy 1))
```

Now you get to try the same. For each of the following expressions using the lists abc and yz and the Scheme list procedures, give at least two other expressions that result in the same list. (Note that what we wrote is not necessarily the most elegant way to do it!)

1. `(list (list-ref yz 0) (list-ref abc 1) (list-ref abc 2))`
2. `(reverse (append abc yz))`
3. `(list abc yz)`
4. `(cons abc (append (list) (list)))`

Part C. The Power of A Few Operations

Suppose you have only the operations `reverse` and `append` and the lists `abc` and `yz` as defined above.

How many different lists can you make with expressions that use each of `abc` and `yz` at most once? (You can use `append` and `reverse` as many times as you want.)

For each list in this part, show both the list-building expression and the result of that expression.

Important Evaluation Criteria

Students who provide correct expressions for each question will earn a check.

Students who provide expressions that are incorrect will earn a lower grade.

Students who provide more expressions than required may earn a higher grade.

Students who use expressions that I failed to think of may earn a higher grade.

Students who demonstrate a particularly elegant or insightful approach to this assignment may earn a higher grade.

Submitting Your Homework

I'd like to see the code for creating the lists and the result lists. Hence, type each expression in the interactions window and then cut and paste the results into an email message with a subject of an email message with a subject of *CSC151 Homework 3*. Make sure to include the list of students with whom you've collaborated.
