Homework 5

Due: October 7, 2003

Use the Advanced Student language in DrScheme for this assignment.

Problem 5.1

Write a recursive procedure called \textit{sum-list} that adds up the numbers in a list.

In addition, think of some test cases you might want to use to check that your code is working, and hand those in with your code. You’ll probably need at least four test cases, and maybe more.

Examples:

\begin{verbatim}
;; Tests for sum-list:
> (sum-list (list 3 4 5))
 12
> (sum-list (list))
 0
\end{verbatim}

Problem 5.2

Now write a recursive procedure \textit{multiply-list} that multiplies together the elements in a list. Again, also turn in test cases for your procedure.

Examples:

\begin{verbatim}
> (multiply-list (list 3 4 5))
 60
> (multiply-list (list))
 1
\end{verbatim}

Although \textit{multiply-list} and \textit{add-list} are fairly similar, there are some differences between them. What differences do you notice?
Problem 5.3

One way to represent a sentence in Scheme is as a list of symbols (words, in this case). In our primitive sentences, we’ll leave out punctuation and capitalization, but other than that they’ll look pretty normal.

Write a procedure `search` designed to help you look through your cs9 notes for sentences containing the word “recursion” (assuming you take your notes as lists in Scheme!). Your procedure should return `true` for sentences containing the symbol `recursion` and `false` otherwise.

As before, you’ll need to hand in test cases with your solution.

Examples:

```scheme
> (search '(i like recursive definitions))
false
> (search '(a recursive definition of recursion might use recursion to define recursion))
true
```

HINT: The Scheme built-in procedure `equal?` might be very useful here, in comparing one symbol to another.

Problem 5.4

Now assume that instead of figuring out whether a sentence contains the word “recursion,” you want to find out how many times it appears. Write a procedure `count` that takes a sentence in the form of a list of symbols and returns a number corresponding to the number of times the symbol `recursion` appears. Once again, hand in a good set of test cases as well.

Examples:

```scheme
> (count '(i like recursion))
1
> (count '(a recursive definition of recursion might use recursion to define recursion))
3
```
Problem 5.5

Write a clear, detailed explanation of the procedure `merge`. Your explanation should tell us what the procedure takes as input, what it returns as output, and what the intermediate steps are. You may find it useful to use a specific example of a possible input to support your response.

You may assume that `lst1` and `lst2` are lists of numbers ordered from least to greatest, for example, `(1 2 5 27 1269).

```
(define (merge lst1 lst2)
  (cond ((empty? lst1) lst2)
        ((empty? lst2) lst1)
        ((< (first lst1) (first lst2))
          (cons (first lst1)
                (merge (rest lst1) lst2)))
          (else
            (cons (first lst2)
                  (merge (rest lst2) lst1))))
```

When might you use a procedure like `merge`?