Com Sci 221 — Programming Languages
Assignment #6 — Winter 1993
Due Wednesday, 10 March

1. Given an array $A[bottom..top]$ of integers and an integer v, the *three-way partitioning problem* is to reorder the entries in the array into three ranges: $A[bottom\ ..\ low-1]$ containing values less than v, $A[low\ ..\ high]$ containing values equal to v, and $A[high+1\ ..\ top]$ containing values greater than v. This is sometimes called the *Dutch national flag problem,* because it can be viewed as sorting three colors into the three stripes of the flag. Write an elegant program fragment, using only assignments, conditionals, and **while**-loops, to solve the three-way partitioning problem. Look at problems 3.13 and 3.14 on p. 114 of the text for inspiration. Give preassertion, postassertion, and loop invariant(s) sufficient to show the correctness of your program fragment. You may use whatever mixture of mathematical notation, pictures, and English you find most convenient, but your assertions must refer *only* to the memory state of the computation at a single control point, and must be precise, unambiguous, and sufficiently strong.

2. Demonstrate that Scheme uses static scope by running a program that would produce different results under dynamic and static scope rules. Do a hand calculation for the same program from McCarthy's equations for the *eval* function to show that the original LISP has dynamic scope.

3. Use stream operations in Scheme to compute the stream *hamming* described in problem 9.5 on p. 376 of the text. Notice that MacScheme has the stream primitives already defined for you in the file **Scheme Folder:Compatibility:streams.sch**, but *qcons* of the text is called *cons-stream* on the MacScheme disk. Discuss the difference between your stream–oriented program for the Hamming sequence and the sort of program that you would have written in a language without streams.

4. Do problem 7.3 on p. 292 of the text in ML. Do part (b) first, then apply it to solve part (a), and also to sort lists of words into alphabetical order.

5. A word x precedes a word y in *anagram order* if, after sorting the letters in x and y into alphabetical order, the resulting scrambled version of x precedes the scrambled version of y. For example, "zebra" precedes "accordion", because the scrambled form of "zebra" is "aberz", and the scrambled form of "accordion" is "accdinor". Notice that anagrams are equal in this ordering — for example "pots", "stop", and "tops" all have the scrambled form "opst". Build a program to sort lists of words in anagram order, using the solution to the previous problem for the sorting of letters within a word, and also for the sorting of scrambled words. The final result, however, should contain the *original* words from the input in anagram order, *not* the scrambled forms. Comment on the difference between your ML solution and the sort of solution that you would use in a language without functions as first-class objects. Comment also on the types of the functions you defined.

6. Define interesting control structures as higher-order functions in ML. Include at least

   (a) **While**-loops in the style of Pascal,

   (b) **For**-loops in the style of C,

   (c) **Case**-commands in the style of Pascal.

   For extra credit, add more sophisticated control structures. For ideas, look at the treatment of exceptions in ML. The goal is not to get a precise implementation of C or Pascal, but rather to reproduce the essential utility of their control structures within ML. Discuss the differences between the ML functions and the procedural-language control structures that they imitate.