MP7 – APL

CS 421 – Summer 2009
Revision 1.0

Assigned July 15, 2009
Due July 22, 2009, 1:00 PM
Extension 48 hours (penalty 20% of total points possible)
Total points 50

1 Change Log

1.0 Initial release.

2 Overview

In this MP you will learn to program in a high-level functional language without explicit recursion. Your assignment is to define several functions for matrix manipulation in APL.

In each case, you should define the function without using any conditionals or recursion. If you cannot see how to do that, then go ahead and define it using if; you will get partial credit. You may feel free to define local variables or functions using "let." You can also define non-local auxiliary functions, and you can use functions defined in previous problems.

3 Collaboration

Collaboration in two-person groups is allowed.

4 What to submit

You will submit your `mp7.ml` file via Compass. Rename `mp7-skeleton.ml` to `mp7.ml` and start working from there.

5 Instructions

Here are the instructions for this MP.

- Download `mp7grader.tar.gz`. This tarball contains all the files you need, including the APL implementation in OCaml.
- As always, extract the tarball, rename `mp7-skeleton.ml` to `mp7.ml`, and start modifying the file. You will modify only the `mp7.ml` file, and submit this file only.
- Compile your solution with `make`. Run the `./grader` to see how well you do.
- Make sure to add several more test cases to the `tests` file.

The following will allow you to run the solution interactively:

```
# #load "mp7common.cmo";;
# #load "solution.cmo";;
# open Mp7common;;
# Solution.upperones (newint 4);;
- : Mp7common.aplval =
AplArrI (4, 4, [1; 1; 1; 1; 0; 1; 1; 1; 0; 0; 1; 1; 0; 0; 0; 1])
```

If you replace "solution" with "student," you will be able to do the same for your own code. Note that in this case, each time you change your code, you will have to first make, then re-load the student.cmo file.

You may also use the `show` function to pretty-print the APL matrices, vectors and scalars. Some examples are given in the beginning of `mp7-skeleton.ml`.

6 Problems

1. (7 pts) Define `upperones n`, which produces an $n \times n$ matrix containing ones along the diagonal and upper right portion of the matrix, zeroes below.

   ```
   show (upperones four);;
   [ [ 1 1 1 1]
     [ 0 1 1 1]
     [ 0 0 1 1]
     [ 0 0 0 1]]
   ```

2. (7 pts) Define `sqmat n`, which takes a scalar value n and produces an $n \times n$ matrix with ones along the perimeter and zeros in the interior.

   ```
   show (sqmat (newint 5));;
   [ [ 1 1 1 1 1]
     [ 1 0 0 0 1]
     [ 1 0 0 0 1]
     [ 1 0 0 0 1]
     [ 1 1 1 1 1]]
   ```

3. (7 pts) Define `diagprod m`, which calculates the product of the elements along the diagonal of a square matrix m.

   ```
   let m1 = rho (newveci [4;4]) (indx (newint 20));;
   show m1;;
   [ [ 1 2 3 4]
     [ 5 6 7 8]
     [ 9 10 11 12]
     [ 13 14 15 16]]

   show (diagprod m1);;
   1056
   ```

(Hint: recall from class how to create the identity matrix.)

4. (7 pts) Define `occurs i v` which return one if i occurs in vector v, zero otherwise.

```
let v = newveci [2;4;6];;
show (occurs (newint 1) v);;
0
show (occurs (newint 2) v);;
1
```

5. (7 pts) Define `find i v` which returns the index of the first occurrence of i in v, if it occurs, and zero otherwise. Note that in APL arrays are indexed from 1. (Hint: It may seem difficult to do this without testing whether i is in v, but you can start by putting i in v, and then check where it was found.)

```
show (find (newint 2) v);;
1
show (find four v);;
2
show (find (newint 0) v);;
0
```

6. (7 pts) Define `plusscan v` which returns the cumulative sums of the elements in v—a vector of the same length as v, where the first element is the first element of v, the second element is the sum of the first two elements of v, the third element is the sum of the first three elements of v, *etc.*

```
show (plusscan v);;
[2 6 12]
```

7. (8 pts) Define `freqvec scores` which returns a vector $freq$ giving the frequency of occurrence of each value in $scores$. That is, suppose the lowest value in scores is lo and the highest value is hi. Then $freq$ has length $hi - lo + 1$, and the ith element is the frequency (number of occurrences) of $lo + i - 1$ in $scores$.

```
let scores = newveci [4; 3; 1; 5; 5; 4; 3; 5];;
show (freqvec scores);;
[1 0 2 2 3]
```

Hint: start by creating a matrix containing $(hi - lo + 1)$ copies of scores.