APS 105S – Computer Fundamentals

Assignment #6: Arrays

Fall 1996

(To be completed on Nov. 7/8)

Objective: To implement a Turing function that performs matrix inversion (by reduction) and to understand sorting.

1 Matrix Inversion

For this assignment you will construct a modular program that will calculate the inversion of a matrix by reduction. Recall that to solve a matrix by reduction you perform "side-by-side" calculations on the matrix A and the identiy matrix I. You will require at least two functions and two procedures as follows:

```
function Reduce(A,I: Matrix) : Matrix
  % function to reduce Matrix
  % First reduce matrix below the diagonal, then above the diagonal
  var AInv : Matrix
  var pivot : real
  AInv := A

  % Reduce lower diagonal of A and I, simultaneously
  for k: 1..lim

    for i:2..lim

      for j:1..lim

                AInv( , ) :=
                I( , ) :=
      end for
```



```
    end for
  end for

  % Reduce upper diagonal of A and I, simultaneously
  .
  your code goes here
  .
end Reduce


        function Initalize_I (N : int): Matrix
          var I: Matrix
          .
          your code goes here
          .
          result I
        end Initialize_I

procedure Get_I(N : int, var A: Matrix)
  put "Please enter the values of the ",N, " x ", N, " matrix"
  .
  your code goes here
  .
end Get_I

procedure SortMatricies(row:int, var A, I: Matrix)
  % Code to sort (simultaneously) arrays A and I
  % so that the current diagonal element of matrix A
  % is not a zero element
  .
  your code goes here
  .
end SortMatricies
```

The program should accept data from the user for a matrix and and should return to the user the inverted matrix. The program should prompt the user for the values of the elements of the matrix.

Your program should make use of the user-defined type *Matrix*. This is to be defined at the top of the program as shown here:

```
const lim := 3
type Matrix: array 1..lim,1..lim of real
```

The constant *lim* defines the bounds of the matrix and vector to be 3, but your program should not make use of this prior knowledge.

2 QuickSort

The intent of this and the following 2 assignments is help you understand sorting algorithms by visualizing their behavior. In this assignment, you are to implement a QuickSort procedure that is applied to an array with 10 elements that initially have random values between 1 and 10. Much of the code you need is available in `/usr/copy/aps105/QSort.t`, and you generally only need to add the procedure `SelectAndShuffle`. To visualize the effects of the sorting algorithm, we assume that each value between 1 and 10 represents a block with a different gray scale.

If you are having problems debugging your program, then it may be beneficial to comment out the visualization code and instead add put statements at strategic points in the program.

3 BubbleSort

Implement 2 variants of BubbleSort, one that sorts 10 numbers between 1 and 10 that were initially random, and one that sorts 10 numbers that are initially in decreasing order. The algorithm is also to be visualized, as QuickSort was. Fragments of code in `/usr/copy/aps105/BSort.t` may be of use to you. Once you have everything working, optimize BubbleSort so that you stop as soon as everything is sorted.

4 Optional Section — A Hangman Game

Devise and implement a hangman game program that should require two players. The first player should supply a word or phrase to the program, and the second player should try to guess the word or phrase before s/he runs out of the allotted number of wrong guesses. This should be a good exercise in working with character arrays, in searching arrays and in working with array indices.

The first player should supply a word or phrase to the computer while the second player does not look at the screen. The phrase should not be more than a few words long. The first player may also supply a clue to go along with the phrase to be guessed.

The program should then display the structure of the phrase with underscores in place of the actual characters and spaces in between words. If a clue was supplied, this could be displayed below. For example if the first player supplied the phrase "The Wizard Of Oz" along with the clue "Movie", the program would display:

```
_ _ _   _ _ _ _ _ _   _ _   _ _
             Movie
```

The second player, or the guessing player, should then try to reveal the letters of the phrase by choosing letters that s/he feels are likely to be within the phrase. When a letter is chosen that is contained within the phrase, all instances of that letter are revealed. If that letter is not contained in the phrase, it is counted as a wrong guess.

At any time, the guessing player has the option to try to guess the entire phrase. If s/he is successful, s/he wins the game, otherwise, it is counted as a wrong guess. The program is

to allow a certain amount of wrong guesses. The number of allowed wrong guesses remaining should be displayed after each guess. You may display this information numerically, or with an illustrative hangman icon, such as:

```
   ----
   |  o
   | /|\
   | / \
   |
-------
```

If the number of allowed guesses reaches zero, the second player loses. The program should also display a simple menu for the guessing player, and a list of the letters that have not yet been chosen. Here is an example of what the program might display for the first few turns of the game with the phrase and clue written above:

```
                     HANGMAN

             8 Wrong Guesses Remaining

         _ _ _   _ _ _ _ _ _   _ _   _ _
                      Movie

       Letters remaining: abcdefghijklmnopqrstuvwxyz
Select a small letter or capital G to guess phrase: t

             8 Wrong Guesses Remaining

         T _ _   _ _ _ _ _ _   _ _   _ _
                      Movie

       Letters remaining: abcdefghijklmnopqrs uvwxyz
Select a small letter or capital G to guess phrase: o

             8 Wrong Guesses Remaining

         T _ _   _ _ _ _ _ _   O _   O _
                      Movie

       Letters remaining: abcdefghijklmn pqrs uvwxyz
Select a small letter or capital G to guess phrase: s
```