CS1112 Lab Exercise 10

1 Cell array vs. vector

You already know that a vector is a collection of simple data. For example, you can have a vector of numbers (each component stores *a single number*) or a vector of characters (each component stores *a single character*). In a cell array, each cell can store an item that may be more complex than just a number or a character.

Type the following code in the command window and observe the output and the display in the *Workspace* pane. Also read the comments given below.

```
v= rand(1,4)  % a VECTOR of length four, each cell stores ONE number
v(3)          % Notice that you use PARENTHESES to access a cell in a VECTOR

c= cell(1,4)  % Use built-in function CELL to create a CELL ARRAY.  Note that its "class" in
              %   the Workspace pane is "cell."  Right now each cell is empty, therefore the
              %   screen output shows four empty vectors.

c{2}= v       % Put a VECTOR in the 2nd cell of the CELL ARRAY.  Notice that we use CURLY
              %   BRACKETS to access a cell in a CELL ARRAY.

c(3)= 1       % You get an error message: Must use curly brackets to access a cell in a
              %   CELL ARRAY; parentheses are for VECTORS.

c{2}          % Display what is in cell 2 of CELL ARRAY c:  a vector!

% So how do you display, say, the fourth value in the VECTOR in the 2nd cell of CELL ARRAY c?
c{2}(4)       % Once again, use curly brackets for the index of the CELL ARRAY; use
              %   parentheses for the index of the of VECTOR.

% Now put other things in the cell array.  Note that you can put different types of things
%   in a CELL ARRAY.  This is not possible in a VECTOR, whose cells must store the same
%   (simple) type of data.
c{1}= 'cat'
c{3}= 10
c{4}= ones(2,1)

% An alternate way to create a cell array is to specify all the contents inside CURLY
%   BRACKETS using spaces, commas, or semi-colons as the separator:
d= {'cat'; 10; v; ones(2,1)}  % A cell array of four cells
length(d)                     % The length function works for cell arrays as well.
```

2 Deck of cards

Download the functions `CardDeck` and `Shuffle` from the *Lecture Materials* page. Read the code and run the functions to make sure that you understand them. Ask if you have questions.

Now write the following function:

```
function DispCards(ca, p, q)
% Display the contents in cells p through q of cell array ca.
% ca is a 1-d cell array.
```

As you develop the next function, use `DispCards` to confirm that the shuffling is done correctly. For example, you can call `DispCards` *inside* the function `MyShuffle` to confirm that the intermediate steps are correct. Just remove the calls to `DispCards` after you've completed the functions.

```
function sd= MyShuffle(d)
% d is a one-dimensional cell array
% sd is the cell array after shuffling d
% The shuffle comprises two steps:
%  - randomly cut the deck into 2 parts. I.e., the position of the cut is random.
%  - interleave the cards from the two parts until the part with fewer
%    cards have been completely incorporated.  It is up to you whether
%    to start from the top or the bottom.
```

3 More card playing . . .

Implement the following function:

```
function sd = Cut3(d)
% d is a one-dimensional cell array whose length is a multiple of 4.
% sd is the cell array after cutting the deck (d) by taking half the cards from
%   the middle of the deck and putting that half on top.
```

Again use `dispCards` to confirm that `Cut3` is implemented correctly.

Write a script to find out whether the cards in the deck cycle back to the original arrangement after repeated cuts done by function `Cut3`. If so, how many cuts are needed to cycle back? You may use the function `strcmp` to compare two strings.

Please delete your files from the computer before you leave the lab.