Spring 2018

LAB 7 - STRUCTURES AND CELL ARRAYS

Objectives: To learn how to use MATLAB for data management, in particular:

- How to store your data efficiently
- How to create an array with different data types

Organizing data and moving it around efficiently is very helpful in MATLAB coding, and you will likely want to get in the habit of storing your data in structures or cell arrays. A cell array is different from a matrix in that you can store information of varying data types within the array. A structure takes your data and separates the information into fields embedded within an array.

```
% creates a structure named structureName with 3 fields
structureName = struct('field1',{},'field2',{},'field3',{});

% create a 3x3 empty cell array
a = cell(3)
```

Potentially useful functions:

```
strcmp
for
while
if
isvector
ismatrix
ischar
isinteger
isnumeric
iscell
struct
fprintf
```

1. **Boot It Up**
   (a) Start up MATLAB
2. **Warm Up**
   (a) If you want some experience with the coding, try these to get the hang of what you'll need to use in the problems below (not required, but some people like this...).
   - Try typing these into the command window:

```
x = cell(5)
x{1,2} = rand
x{3,4} = round(10*rand)
x{1,1} = 'this is a string'
x{5,4} = [1 2 4 5; 9 8 7 2]
x{3,2} = zeros(3)
x(3,2)
x{3,2}
x{3,2}(2,2) = rand
x{3,2}(1,3) = 25
x{1,4} = strcat(x{1,1},' and another string')
x{1,4}
x{1,4}(1,7:13)
clear all
ball = struct('mass',{},'position',{},'velocity',{})
```

```
ball(1).mass = 12
ball(2).position = [0 0 1]
ball.mass
ball.position
ball.velocity
ball(1).velocity = 35
ball(2).velocity = 24
[ball1Mass,ball2Mass] = ball.mass
[ball1_velocity,ball2_velocity] = ball.velocity
ball(2).mass = 12
totalMass = sum(ball.mass)
totalMass = sum([ball.mass])
```

3. **Data Management**

(a) Create a script (*buildACell.m*) that allows the user to create a cell array that has the dimensions of their choosing. Allow them to enter as many elements as they choose and any type they choose. After they have completed the array population, display the array to the user. Find all character arrays (words) and cell arrays then return the address (i,j) of the data types. Ask the user whether they would like to view the contents of the cell. Run your script.

(b) Create a script (*carInventory.m*) that builds a structure named *car* and include fields *make*, *model*, *year*, *mileage*, *class* (car, van, truck, etc) and *color*. Have you script include 3 vehicles of your choosing then have the user input their own vehicles (use a loop and continue asking until the user is done entering vehicles). When the user adds a specific make, model and year check the current array for duplicates. (The duplicate check needs to look for matches of make, model and year only.) If there is a duplicate, return a message to the user that says 'match found' then display their data and the data of the duplicate. If there is no match, return the user entered values only. Ask the user if they would like to view all vehicles, and if they choose *yes* display all vehicles in a table with appropriate headings. Run your script. Use *fprintf* to format your table. The following code could be handy for formatting:

```
fprintf('\%12s \%12s \%12s \%12s \%12s\n','make','model','year','mileage','class','color');

fprintf('--------------------------------------------------------------------------------\n');
```

Use the same notation in your loop for each vehicle to create a nicely formatted table.

(c) Once the TA has checked your files (*buildACell.m* and *carInventory.m*) you are free to go. Do not forget to upload your files to Canvas! Log off of your machine before you leave the lab.