ENCMP 100 – Computer Programming for Engineers
Assignment #5
Due: Monday, Dec. 2, 2013 at 6:00pm MST

Objective
To gain programming experience with MATLAB cell arrays, structures and structure arrays. In addition, to get some experience with inputting data using a simple graphical user interface (GUI).

Marking Scheme
The assignment is worth 3% of your final mark. You will get a total of 50 points for completing the following:

Task	**Points**
Part A – Correct code and demonstration of the initial GUI	14
Part B – Area calculations with structure arrays	20
Overall quality of the code (8 points for each part)	16
TOTAL	50

Points for the Quality of the Code
- Complete file headers, following the same format as in previous assignments (2 points)
- Clarity and logical simplicity of the designs (2 points)
- Appropriate comments in the body of the code (2 points)
- Clarity and tidy layout of the code (2 points)

Submission
- Submit your m-files for Parts A and B to eClass/Moodle.
- This assignment is due on Monday, Dec. 2, 2013 at 6:00pm MST.

Background
MATLAB was originally intended to be an easy-to-use “matrix laboratory” programming environment. Two-dimensional matrices can be manipulated and operated upon by functions in MATLAB just as easily as scalar constants and variables. However, for many problems, the matrix data type is overly restrictive because the matrix elements must all be of the same type (e.g., all integers, all doubles, all characters). For that reason MATLAB, like most other programming languages, provides additional multi-element data structures that can be used in addition to matrices. In MATLAB we are provided with “cell arrays”, “structures” and “structure arrays”.

Cell arrays are similar to matrices in that they are a data type that is an arrangement of elements that can be identified by index values (e.g., row and column index). Individual cell array elements, or specified ranges of cell array elements, can be used as data destinations on the left side of assignment statements, or used as data sources that appear in expressions. However, unlike in matrices, the elements in cell arrays do not have to be of the same type. Some cell array elements can be integers, while other elements can be doubles, and yet other elements can be characters. In addition, cell array elements can themselves be composite data values such strings and matrices.

A *structure*, which is usually slightly shortened and spelled *struct* in MATLAB, is similar to a one-dimensional cell array in that a structure contains one or more elements of possibly different types. However, in a structure the elements of a structure are identified using string tags and not by integer index values. The use of string tags has the advantage of being more readable and self-documenting. However, the string tags of a structure can become awkward to use when the number of structure fields becomes large. MATLAB structures are often used to record the various different properties of a physical object or data item. For example, one could use a structure to record the first and last name of a student, the student's ID number, e-mail address, and the name of the student's degree program. In MATLAB, many software objects (e.g., figures, options to functions, etc.) are expressed as structures containing multiple tagged fields.

A *structure array* is simply a matrix in which all of the elements are structures of the same type, having exactly the same set of string tags. For example, one could record information about all of the students in a class by using a structure array that contains elements that were each student structures of the kind described in the previous paragraph. The structures within a structure array are identified using a numerical index; however, the fields within a particular structure in a structure array are identified using the appropriate string tags. Thus there are two kinds of dimensions in a structure array: one (or more) outer integer indexes for identifying one (or a range) of structures of the same type, and a string tag for identifying one field within the structure(s).

Assignment Description

Part A: Extend a simple user interface

Create a new m-file called `Assign5PartA_<UofA_ID>.m` and enter the following MATLAB code fragment:

```
while 1
    shapeList = {'Circle','Square'};
    [selection,ok] = listdlg('PromptString','Select the next
shape:',...
                         'SelectionMode','single',...
                         'OKString','Enter',...
                         'CancelString','No more',...
                         'ListString',shapeList);
    % selection now holds the index of the selected shape
    % ok is 1 if a shape was selected; otherwise, ok is 0
    if ~ok break; end

    prompt={'Enter shape dimension 1','Enter shape dimension 2'};
    title = 'Shape dimension dialog box';
    numlines = 1;
    defaults = {'0','0'};
    options.Resize = 'on';
    options.WindowStyle = 'normal';
    options.Interpreter = 'none';
    inputvalues = inputdlg(prompt,title,numlines,defaults,options);
    % str2num(inputvalues{1}) is the value of the first dimension
    % str2num(inputvalues{2}) is the value of the second dimension
end
```

The code fragment implements a simple graphical user interface (GUI) that queries the user to select shapes from a menu, and then allows the user to enter two numerical shape dimensions for each shape. Note that the number of shape dimensions is fixed at two, regardless of the number of dimensions that are actually required. A circle only requires a diameter dimension; the second dimension is not required. A square also only requires one dimension that gives the common length of all sides of the square.

Note also that the code fragment already uses cell arrays. In your submitted code, identify all of the cell arrays and structures that are used, and briefly explain in comments what they are used for.

Consult the on-line documentation for the function `listdlg()` and then modify the given code fragment to extend the capabilities of the user interface in the following ways.

1) Increase the number of object shapes to include ellipses, triangles and rectangles by making suitable modifications to the initialization of the cell array `shapeList`. The two dimensions of an ellipse are to be major and minor diameters. The two dimensions of a triangle are to be the length of a base side, and the corresponding height to the third corner of the triangle. The dimensions of a rectangle are to be the height and width.
2) Provide an additional pull-down menu that allows the use to enter a shape colour. The available shape colours should be 'red', 'yellow', 'blue', 'green', 'orange' and 'violet'. The two buttons at the bottom of the colour select menu should be labelled "Enter" and "No colour".
3) Provide a loop counter `numShapes` that keeps track of the number of shape instances that have been entered by the user.
4) When the user selects the button labelled "No more", enhance the program so that it prints out the message "The number of entered objects was xxx" where "xxx" is replaced by the integer count of the number of entered objects.

Part B: Shape database and area calculation

The user interface that you developed in Part A is now to be modified to store the entered shape information into a single structure array called shape. The `n`'th shape you enter should be the structure `shape(n)`. The structure array should have sub-fields of 'ID', 'color', 'dimensions' and any other fields you wish. Your code should calculate the areas of the shapes you enter and print a summary table of results, an example of which is shown below.

```
The number of entered objects was 2
No.          ID          Color              Area
1            Circle      red                3.141593
2            Square      yellow             1.000000
```

Enter your solution in a new file called: `Assign5PartB_<UofA_ID>.m`. Your code should use at least one sub-function that accepts as a single argument the structure array `shape`.