1. Suppose Names is a \( N \times 1 \) cell array of character strings, such as

```matlab
names = cell(10,1);
names{1} = 'Color';
names{2} = 'Shape';
names{3} = 'Direction';
names{4} = 'Intensity';
names{5} = 'Precedence';
names{6} = 'Seniority';
names{7} = 'Density';
names{8} = 'Target';
names{9} = 'Antagonist';
names{10} = 'Friend';
```

What does the following code do?

```matlab
plist = [names cell(length(names),1)']'; % don't miss the transpose
A = struct(plist{:});
```

Explain your answer in terms of

- how the \texttt{struct} constructor works,
- horizontal concatenation of cells,
- transpose for cells,
- multiple-contents referencing in cells, and
- comma-separated lists.

2. Write a function (call it \texttt{propmatch}) that has the following behavior: 2 input arguments, \texttt{PropertyNames} and \texttt{singleprop}; 1 output argument, called \texttt{Idx}. Input argument \texttt{PropertyNames} is a \( N \times 1 \) cell-array of strings (ie., \texttt{char}). Input argument \texttt{singleprop} is a single string (of size \( 1 \times P \), say). \texttt{Idx} is an integer, which indicates which entry of \texttt{PropertyNames} contains the string in \texttt{singleprop} by the following rule: First, see if any of the strings in \texttt{PropertyNames}, match with the string in \texttt{singleprop}, on the first \( P \) characters, and independent of uppercase/lowercase.

(a) If there are no matches, return \texttt{Idx} as an empty matrix

(b) If there is one match, return \texttt{Idx} as the integer location in \texttt{PropertyNames} of the match.

(c) If there are multiple matches, check further as follows:
i. If the length of one of the multiple matches in PropertyNames matches
the length of singleprop, return this index.
ii. Otherwise, return [].

Try is out on

PropertyNames = {'CloseRequestFcn';
    'Color';
    'Colormap';
    'CurrentAxes';
    'CurrentCharacter';
    'CurrentObject';
    'Pointer';
    'PointerShapeCData';
    'PointerShapeHotSpot';
    'Renderer';
    'RendererMode';
    'Resize';
    'ResizeFcn'};

and illustrate all of the cases with examples. Hint: you probably should use
strncmp, strncmpi and/or strcmp. Read the help if these are unfamiliar.

3. Write a simple class with one trivial field that does not allow concatenation (hor-
izontal, vertical, or in higher dimensions) or assignment of anything non-scalar.
Call the class noconcat and the field Value. The following should work.

    >> A = noconcat(7);
    >> A.Value % should be 7
    >> get(A,'Value') % should be 7
    >> set(A,'Value',8);
    >> get(A,'Value') % should be 8
    >> AA = set(A,'Value',9);
    >> get(A,'Value') % should be 8
    >> get(AA,'Value') % should be 9
    >> B = [A];
    >> isequal(B,A) % should be TRUE
    >> C = B(1);
    >> D = B(1,1,1);
    >> A(1,1) = D;
    >> A(1,1).Value % should be 8

The following should not work.
You will need an `@noconcat` folder with a constructor, and (at least) `horzcat`, `vertcat`, `get`, `set`, `subsref`, `subsasgn` methods.

4. Show by example that class precedence is not transitive. In other words, suppose that (by declaration within constructors) objects of class A are superior to objects of class B, and that objects of class B are superior to objects of class C. Assuming no precedence is declared between objects of class A and C, show that no precedence is inferred/implied through their individual relationships to B. Explain in the commentary of the script file how the code you run illustrates these points. You will need to create 3 classes, `classA`, `classB` and `classC`. Each class will have a constructor (obviously) and enough (a few) very simple methods to prove the point.