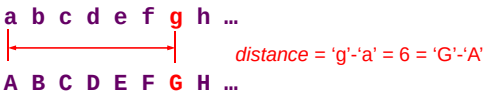
- Previous Lecture:
  - Characters and strings
- Today's Lecture:
  - More on characters and strings
  - Cell arrays
- Announcement:
  - Project 4 due Monday 4/4 at 11pm

Example: toUpper

Write a function toUpper(cha) to convert character cha to upper case if cha is a lower case letter. Return the converted letter. If cha is not a lower case letter, simply return the character cha.

Hint: Think about the distance between a letter and the base letter 'a' (or 'A'). E.g.,



Of course, do not use Matlab function upper!


```
function up = toUpper(cha)
% up is the upper case of character cha.
% If cha is not a letter then up is just cha.
```


Example: removing all occurrences of a character

- From a genome bank we get a sequence
  ATTG CCG TA GCTA CGTACGC AACTGG AAATGGC CGTAT…
- First step is to "clean it up" by removing all the blanks. Write this function:

```
function s = removeChar(c, s)
% Return string s with all occurrences
% of character c removed
```

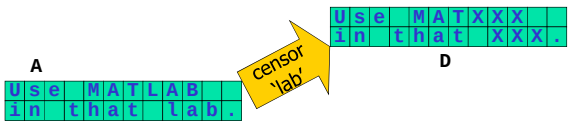

Example: removing all occurrences of a character

Can solve this problem using iteration—check one character (one component of the vector) at a time

```
function s = removeChar_loop(c, s)
% Return string s with all occurrences of
% character c removed.
```


Example: censoring words

```
function D = censor(str, A)
% Replace all occurrences of string str in
% character matrix A with X's, regardless of
% case.
% Assume str is never split across two lines.
% D is A with X's replacing str.
```

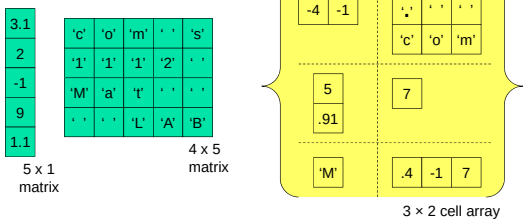


```
function D = censor(str, A)
% Replace all occurrences of string str in character matrix A,
% regardless of case, with X's.
% A is a matrix of characters.
% str is a string.  Assume that str is never split across two lines.
% D is A with X's replacing the censored string str.

D= A;
B= lower(A);
s= lower(str);
ns= length(str);
[nr,nc]= size(A);

% Build a string of X's of the right length



% Traverse the matrix to censor string str
```


Matrix vs. Cell Array

Vectors and matrices store values of the same type in all components

A cell array is a special array whose individual components may contain different types of data



Cell Arrays of Strings

```
C= { 'Alabama','New York','Utah'}
```

C | 'Alabama' | 'New York' | 'Utah'

```
C= { 'Alabama';'New York';'Utah'}
```

C
'Alabama'
'New York'
'Utah'

1-d cell array of strings

Contrast with 2-d array of characters

```
M= ['Alabama '; ...
    'New York'; ...
    'Utah    ']
```

M

'A'	'l'	'a'	'b'	'a'	'm'	'a'	' '	
'N'	'e'	'w'	' '	'Y'	'o'	'r'	'k'	
'U'	't'	'a'	'h'	' '	' '	' '	' '	

Use braces { } for creating and addressing cell arrays

Matrix

- Create

```
m= [ 5, 4 ; …
     1, 2 ; …
     0, 8 ]
```

- Addressing

```
m(2,1)= pi
```

Cell Array

- Create

```
C= { ones(2,2), 4          ; …
     'abc'    , ones(3,1) ; …
      9       , 'a cell'     }
```

- Addressing

```
C{2,1}= 'ABC'
C{3,2}= pi
disp(C{3,2})
```


Creating cell arrays…

```
C= {'Oct', 30, ones(3,2)};
```

is the same as

```
C= cell(1,3); % not necessary
C{1}= 'Oct';
C{2}= 30;
C{3}= ones(3,2);
```

You can assign the empty cell array: `D = {}`

Example: Represent a deck of cards with a cell array

```
D{1} = 'A Hearts';
D{2} = '2 Hearts';
          :
D{13} = 'K Hearts';
D{14} = 'A Clubs';
          :
D{52} = 'K Diamonds';
```

But we don't want to have to type all combinations of suits and ranks in creating the deck... How to proceed?

Make use of a suit array and a rank array …

```
suit = {'Hearts', 'Clubs', …
        'Spades', 'Diamonds'};

rank = {'A','2','3','4','5','6',…
    '7','8','9','10','J','Q','K'};
```

Then concatenate to get a card. E.g.,

```
str = [rank{3} ' ' suit{2} ];
D{16} = str;
```

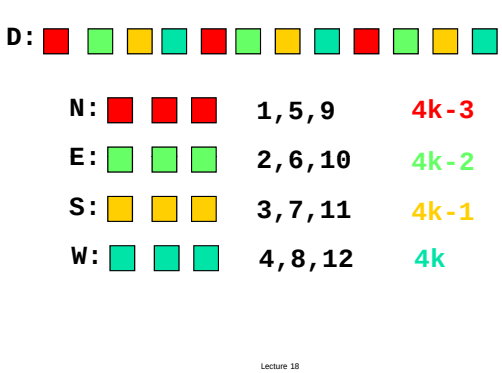
So **D{16}** stores **'3 Clubs'**

To get all combinations, use nested loops

```
i = 1;  % index of next card

for k= 1:4
   % Set up the cards in suit k
   for j= 1:13
      D{i} = [ rank{j} ' ' suit{k} ];
      i = i+1;
   end
end
```

See function **CardDeck**

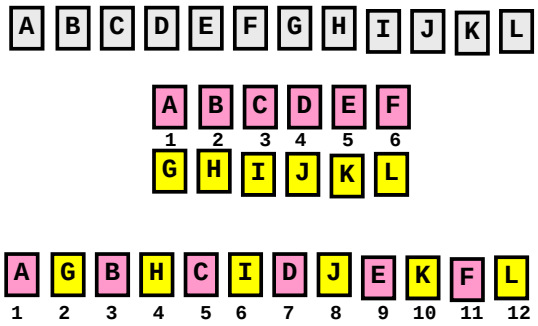
Example: deal a 12-card deck




```
% Deal a 52-card deck
N = cell(1,13); E = cell(1,13);
S = cell(1,13); W = cell(1,13);

for k=1:13
   N{k} = D{4*k-3};
   E{k} = D{4*k-2};
   S{k} = D{4*k-1};
   W{k} = D{4*k};
end
```

See function **Deal**

Step 2: Alternate



Example: Build a cell array of Roman numerals for 1 to 3999

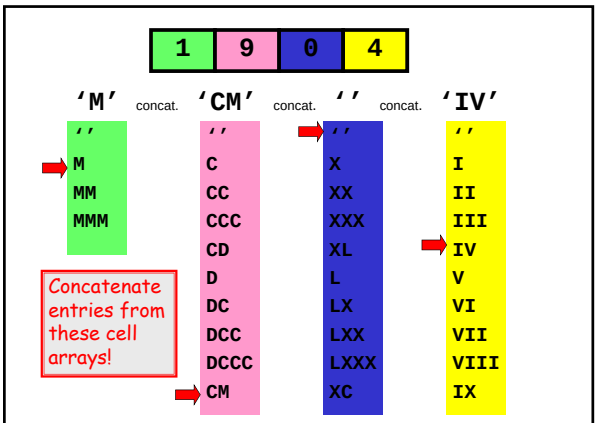
```
C{1} = 'I'
C{2} = 'II'
C{3} = 'III'
   :
C{2007} = 'MMVII'
   :
C{3999} = 'MMMXMXCIX'
```

Example

1904 = 1*1000 + 9*100 + 0*10 + 4*1

= M CM IV

= MCMIV



Ones-Place Conversion

```
function r = Ones2R(x)
% x is an integer that satisfies
%      0 <= x <= 9
% r is the Roman numeral with value x.

Ones = {'I', 'II', 'III', 'IV', ...
        'V', 'VI','VII', 'VIII', 'IX'};

if x==0
     r = '';
else
     r = Ones{x};
end
```


Similarly, we can implement these functions:

```
function r = Tens2R(x)
% x is an integer that satisfies
%      0 <= x <= 9
% r is the Roman numeral with value 10*x.
```

```
function r = Hund2R(x)
% x is an integer that satisfies
%      0 <= x <= 9
% r is the Roman numeral with value 100*x
```

```
function r = Thou2R(x)
% x is an integer that satisfies
%      0 <= x <=3
% r is the Roman numeral with value 1000*x
```


Now we can build the Roman numeral cell array for 1,…,3999

```
for a = 0:3
  for b = 0:9
    for c = 0:9
      for d = 0:9
        n = a*1000 + b*100 + c*10 + d;
        if n>0
          C{n} = [Thou2R(a) Hund2R(b)...
                            Tens2R(c) Ones2R(d)];
        end
      end
    end
  end
end
```


The reverse conversion problem

Given a Roman Numeral, compute its value.
Assume cell array **C(3999,1)** available where:

```
C{1} = 'I'
C{2} = 'II'
   :
C{3999} = 'MMMCMXCIX'
```

See script **RN2Int**