Previous Lecture:
- Characters and strings

Today’s Lecture:
- More on characters and strings
- Cell arrays

Announcement:
- Project 4 due tonight at 11pm
Example: removing all occurrences of a character

- From a genome bank we get a sequence
  ATTG CCG TA GCTA CGTACGC AACTGG AAATGGGC CGTAT...

- First step is to “clean it up” by removing all the blanks. Write this function:

```matlab
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

```matlab
function s = removeChar_loop(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

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

t = '';  % initialize empty string
for k = 1:length(s)
    t = t + s(k);
end
s = t;
```
Example: removing all occurrences of a character

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

```matlab
function s = removeChar_loop(c, s)
    % Return string s with all occurrences of character c removed.
    t = '';  % initialize empty string
    for k = 1:length(s)
        if s(k) ~= c
            t = [t s(k)];
        end
    end
    s = t;
end
```
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.

5 x 1 matrix

4 x 5 matrix

3 x 2 cell array
Cell Arrays of Strings

$C = \{ '\text{Alabama}', '\text{New York}', '\text{Utah}' \}$

$C = \{ '\text{Alabama}'; '\text{New York}'; '\text{Utah}' \}$

Contrast with 2-d array of characters

$M = [ '\text{Alabama }'; ... '\text{New York}'; ... '\text{Utah }' ]$
Use braces \{\} for creating and addressing cell arrays

Matrix

- Create
  
  \[
  \begin{bmatrix}
  5 & 4 \\
  1 & 2 \\
  0 & 8
  \end{bmatrix}
  \]

- Addressing
  
  \[m(2,1) = \pi\]

Cell Array

- Create
  
  \[
  C = \{
  \begin{bmatrix}
  1 & 2 \\
  0 & 8
  \end{bmatrix},
  4,
  'abc',
  \begin{bmatrix}
  1 & 2 \\
  0 & 8
  \end{bmatrix}
  \}
  \]

- Addressing
  
  \[
  C_{2,1} = 'ABC'
  \]
  
  \[
  C_{3,2} = \pi
  \]
  
  \[
  \text{disp}(C_{3,2})
  \]
Creating cell arrays...

\[
C= \{ \text{'Oct'}, 30, \text{ones}(3,2) \};
\]
is the same as

\[
C= \text{cell}(1,3); \ % \text{ not necessary}
C\{1\}= \text{'Oct'};
C\{2\}= 30;
C\{3\}= \text{ones}(3,2);
\]

You can assign the empty cell array:

\[
D = \{ \}
\]
Example: Represent a deck of cards with a cell array

\[
\begin{align*}
D\{1\} &= 'A\ Hearts'; \\
D\{2\} &= '2\ Hearts'; \\
&\vdots \\
D\{13\} &= 'K\ Hearts'; \\
D\{14\} &= 'A\ Clubs'; \\
&\vdots \\
D\{52\} &= 'K\ Diamonds';
\end{align*}
\]

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 ...

\[
\text{suit} = \{\text{‘Hearts’}, \text{‘Clubs’}, \ldots \text{‘Spades’}, \text{‘Diamonds’}\};
\]

\[
\text{rank} = \{\text{‘A’}, \text{‘2’}, \text{‘3’}, \text{‘4’}, \text{‘5’}, \text{‘6’}, \ldots \text{‘7’}, \text{‘8’}, \text{‘9’}, \text{‘10’}, \text{‘J’}, \text{‘Q’}, \text{‘K’}\};
\]

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

\[
\text{str} = [\text{rank}\{3\} \ ‘ ’ \ \text{suit}\{2\} ];
\]
\[
\text{D}\{16\} = \text{str};
\]

So \( \text{D}\{16\} \) stores ‘3 Clubs’
To get all combinations, use nested loops

```matlab
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

D:

N: 1, 5, 9

E: 2, 6, 10

S: 3, 7, 11

W: 4, 8, 12
% 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
The “perfect shuffle” of a 12-card deck
Perfect Shuffle, Step 1: cut the deck
Perfect Shuffle, Step 2: Alternate

A B C D E F G H I J K L

A B C D E F G H I J K L

A B C D

G H I J K L

A G B H C I D J E K F L

1 2 3 4 5 6 7 8 9 10 11 12
Perfect Shuffle, Step 2: Alternate

A B C D E F G H I J K L

A B C D E F G H I J K L

A B C D E F

1 2 3 4 5 6

G H I J K L

k

2k

A G B H C I D J E K F L

2 4 6 8 10 12
Perfect Shuffle, Step 2: Alternate
Shuffle.m
Example: Build a cell array of Roman numerals for 1 to 3999

\[
\begin{align*}
C\{1\} &= \text{‘I’} \\
C\{2\} &= \text{‘II’} \\
C\{3\} &= \text{‘III’} \\
&\vdots \\
C\{2007\} &= \text{‘MMVII’} \\
&\vdots \\
C\{3999\} &= \text{‘MMMIXMXCIX’}
\end{align*}
\]
Example

$1904 = 1\times1000 + 9\times100 + 0\times10 + 4\times1$

$= M \quad CM \quad IV$

$= MCMIV$
Concatenate entries from these cell arrays!
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
Ones-Place Conversion

```matlab
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:

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

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

```matlab
function r = Thou2R(x)
  % x is an integer that satisfies
  %    0 <= x <= 3
  % r is the Roman numeral with value 1000*x
```
We want all the Roman Numerals from 1 to 3999. We have the functions Ones2R, Tens2R, Hund2R, Thou2R.

The code to generate all the Roman Numerals will include loops—nested loops. How many are needed?

A: 2   B: 4   C: 6   D: 8
Now we can build the Roman numeral cell array for 1,…,3999

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

```matlab
for a = 0:3 % possible values in thous place
    for b = 0:9 % values in hundreds place
        for c = 0:9 % values in tens place
            for d = 0:9 % values in ones place
                n = a*1000 + b*100 + c*10 + d;
                if n>0
                    \textbf{C\{n\} = \{Thou2R(a) Hund2R(b)…
                        \ \ \ \ \ \ \ \ \ Tens2R(c) Ones2R(d)\}];
                \end{highlight}
            \end{highlight}
        \end{highlight}
    \end{highlight}
\end{highlight}
```

Four strings concatenated together

The \(n^{th}\) component of cell array \(C\)
The reverse conversion problem

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

\[
\begin{align*}
C\{1\} &= \text{`I'} \\
C\{2\} &= \text{`II'} \\
\vdots \\
C\{3999\} &= \text{`MMMCMXCIX'}
\end{align*}
\]

See script RN2Int
Example: subset of clicker IDs

IDs

['d091314'; ... 'h134d83'; ... 'h4567s2'; ... 'fr83209']

Find subset that begins with ‘h’

L

L= 

L= 

L= {};
k= 0;
for r=1:size(IDs,1)
  if IDs(r,1)=='h'
    k= k+1;
    L{k }= IDs(r,:);
  end
end

Directly assign into a particular cell—good!

Concatenate cells or cell arrays—prone to problems!