Previous Lecture:
- Characters and strings

Today’s Lecture:
- More on characters and strings
- Cell arrays
- File input/output

Announcements:
- Discussion this week in computer lab
- Project 4 due on Wednesday at 11pm
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.

A

Use MATLAB
in that lab.

censor ‘lab’

D

Use MATLAB
in that XXX

Function strcmpi does case-insensitive string comparison
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;
ns= length(str);
[nr,nc]= size(A);

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

% Traverse the matrix to censor string 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;
    ns= length(str);
    [nr,nc]= size(A);
    % Build a string of X's of the right length
    Xs= char( zeros(1,ns));
    for k= 1:ns
        Xs(k)= 'X';
    end

    % Traverse the matrix to censor string str
    for r= 1:nr
        for c= 1:nc-ns+1
            if strcmpi( str , A(r, c:c+ns-1) )==1
                D(r, c:c+ns-1)= Xs;
            end
        end
    end

Returns an array of type double
Changes the type to char
Case insensitive comparison of strings
Array vs. Cell Array

- **Simple array**
  - Each component stores one scalar. E.g., one char, one double, or one uint8 value
  - All components have the same type

- **Cell array**
  - Each cell can store something “bigger” than one scalar, e.g., a vector, a matrix, a string (vector of chars)
  - The cells may store items of different types
Vectors and matrices store values of the same type in all components.

Cell array: 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 = { 'Alabama', 'New York', 'Utah' }

C

'Alabama' 'New York' 'Utah'

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

C

'Alabama'

New York'

'Utah'

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

\[
m = \begin{bmatrix}
  5 & 4 \\
  1 & 2 \\
  0 & 8
\end{bmatrix}
\]

- Addressing

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

**Cell Array**

- Create

\[
C = \{ \text{ones}(2,2), 4 ; \ldots \\
\text{‘abc’}, \text{ones}(3,1) ; \ldots \\
9, \text{‘a cell’}
\}
\]

- Addressing

\[
C(2,1) = \text{‘ABC’} \\
C(3,2) = \pi \\
disp(C(3,2))
\]
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

\[ D\{1\} = 'A {Hearts}' ; \]
\[ D\{2\} = '2 {Hearts}' ; \]
\[ \ldots \]
\[ D\{13\} = 'K {Hearts}' ; \]
\[ D\{14\} = 'A {Clubs}' ; \]
\[ \ldots \]
\[ D\{52\} = 'K {Diamonds}' ; \]

But we don’t want to have to type all combinations of suits and ranks in creating the deck… How to proceed?
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'
suit = {'Hearts','Clubs','Spades','Diamonds'};
rank = {'A','2','3','4','5','6','7','8','9',... 
       '10','J','Q','K'};
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
function D = CardDeck()

% D is 1-by-52 cell array of strings that define a card deck

suit = {'Hearts','Clubs','Spades','Diamonds'};
rank = {'A','2','3','4','5','6','7','8','9','10','J','Q','K'};
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
Example: deal a 12-card deck

D:  

N: 1, 5, 9  \quad 4k-3

E: 2, 6, 10  \quad 4k-2

S: 3, 7, 11  \quad 4k-1

W: 4, 8, 12  \quad 4k
% 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

A B C D E F G H I J K L
Perfect Shuffle, Step 1: cut the deck

A B C D E F G H I J K L
A B C D E F
G H I J K L
Perfect Shuffle, Step 2: Alternate

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

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

2 4 6 8 10 12

k

2k
Perfect Shuffle, Step 2: Alternate

See function Shuffle
A 3-step process to read data from a file or write data to a file

1. (Create and ) open a file
2. Read data from or write data to the file
3. Close the file
Working with data files: Read the data in a file line-by-line and store the results in a cell array.

```
GATTTCGAG
GAGCCACTGGTC
ATAGATCCT
```

geneData.txt

```
‘GATTTCGAG’
‘GAGCCACTGGTC’
‘ATAGATCCT’
```

How are lines separated?
How do we know when there are no more lines?
In a file there are hidden “markers”

- Carriage return marks the end of a line
- `eof` marks the end of a file
Read data from a file

1. **Open** a file
2. **Read** it line-by-line until eof
3. **Close** the file
1. Open the file

\[
\text{fid} = \text{fopen}('\text{geneData.txt}', 'r');
\]

An open file has a file ID, here stored in variable \texttt{fid}.

Built-in function to open a file.

Name of the file opened. \texttt{txt} and \texttt{dat} are common file name extensions for plain text files.

'\texttt{r}' indicates that the file has been opened for reading.
2. Read each line and store it in cell array

```matlab
fid = fopen('geneData.txt', 'r');

k= 0;
while ~feof(fid)
    k= k+1;
    Z{k}= fgetl(fid);
end
```

- False until end-of-file is reached
- Get the next line
3. Close the file

```matlab
fid = fopen('geneData.txt', 'r');

k= 0;
while ~feof(fid)
    k= k+1;
    Z{k}= fgetl(fid);
end
fclose(fid);
```