- Previous Lecture:
  - Characters and strings
- Today's Lecture:
  - More on characters and strings
  - Cell arrays
  - File input/output
- Announcement:
  - Project 4 due tonight at 11pm. Late penalty reduced to 1 point for submission within 24 hrs after deadline
  - Prelim 2 on Nov 6th (Tues) at 7:30pm. Email Randy Hess (rbh27) ASAP about any conflict and include information on the conflicting event (course number, instructor name and email, etc.)

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
```

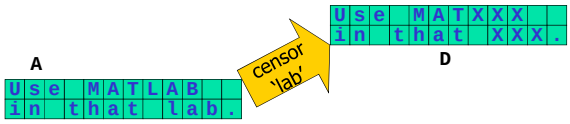

Example: removing all occurrences of a character

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.

t= '';  % initialize empty string
for k= 1:length(s)



end
s= t;
```


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.
```

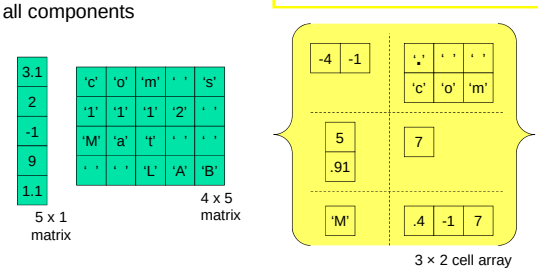



```
function D = censor(str, A)
% Replace all occurrences of string str in character matrix A,
% regardless of case, with X's.
% A is a matrix of characters.
% str is a string.  Assume that str is never split across two lines.
% D is A with X's replacing the censored string str.

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

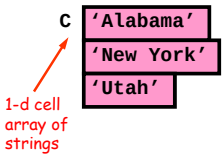
A cell array is a special array whose individual components may contain different types of data



Cell Arrays of Strings

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


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



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	Cell Array
▪ Create	▪ Create
m= [5, 4 ; … 1, 2 ; … 0, 8]	C= { ones(2,2), 4 ; … 'abc' , ones(3,1) ; … 9 , 'a cell' }
▪ Addressing	▪ Addressing
m(2,1)= pi	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;
```

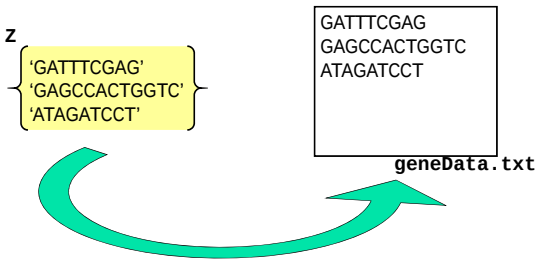
So `D{16}` stores `'3 Clubs'`

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: Write a cell array of gene sequences to a file



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

1. Open a file

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

An open file has a file ID, here stored in variable **fid**

Built-in function to open a file

Name of the file (created and) opened. **txt** and **dat** are common file name extensions for plain text files

'**w**' indicates that the file is to be opened for **w**riting

Use '**a**' for **a**ppending

2. Write (print) to the file

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

for i=1:length(Z)
    fprintf(fid, '%s\n', Z{i});
end
```

Printing is to be done to the file with ID **fid**

Substitution sequence specifies the *string* format (followed by a new-line character)

The **i**th item in cell array **Z**

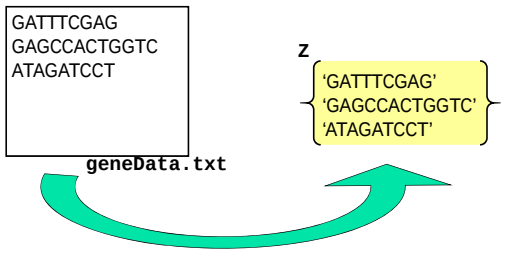
3. Close the file

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

for i=1:length(Z)
    fprintf(fid, '%s\n', Z{i});
end

fclose(fid);
```



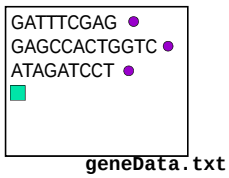
```
function cellArray2file(CA, fname)
% CA is a cell array of strings.
% Create a .txt file with the name
% specified by the string fname.
% The i-th line in the file is CA{i}

fid= fopen([fname '.txt'], 'w');
for i= 1:length(CA)
    fprintf(fid, '%s\n', CA{i});
end
fclose(fid);
```

Reverse problem: Read the data in a file line-by-line and store the results in a cell array



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

1. Open the file

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

An open file has a file ID, here stored in variable **fid**

Name of the file opened. **txt** and **dat** are common file name extensions for plain text files

'**r**' indicates that the file has been opened for reading

Built-in function to open a file

2. Read each line and store it in cell array

```
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

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

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



```
function CA = file2cellArray(fname)
% fname is a string that names a .txt file
%   in the current directory.
% CA is a cell array with CA{k} being the
%   k-th line in the file.

fid= fopen([fname '.txt'], 'r');
k= 0;
while ~feof(fid)
   k= k+1;
   CA{k}= fgetl(fid);
end
fclose(fid);
```

A Detailed Read-File Example

From the protein database at

http://www.rcsb.org

we download the file **1bl8.dat** which encodes the amino acid information for the protein with the same name. We want the xyz coordinates of the protein's "backbone."

The file has a long "header"

```
HEADER    MEMBRANE PROTEIN                        23-JUL-98   1BL8
TITLE     POTASSIUM CHANNEL (KCSA) FROM STREPTOMYCES LIVIDANS
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: POTASSIUM CHANNEL PROTEIN;
COMPND   3 CHAIN: A, B, C, D;
COMPND   4 ENGINEERED: YES;
COMPND   5 MUTATION: YES
SOURCE    MOL_ID: 1;
SOURCE   2 ORGANISM_SCIENTIFIC: STREPTOMYCES LIVIDANS;
```

Need to read past hundreds of lines that are not relevant to us.

Eventually, the xyz data is reached…

```
MTRIX1   2 -0.736910 -0.010340  0.675910      112.17546    1
MTRIX2   2  0.004580 -0.999940 -0.010300       53.01701    1
MTRIX3   2  0.675980 -0.004490  0.736910      -43.35083    1
MTRIX1   3  0.137220 -0.931030  0.338160       80.28391    1
MTRIX2   3  0.929330  0.002860 -0.369240      -33.25713    1
MTRIX3   3  0.342800  0.364930  0.865630      -31.77395    1

ATOM      1  N   ALA A  23      65.191  22.037  48.576  1.00181.62        N
ATOM      2  CA  ALA A  23      66.434  22.838  48.377  1.00181.62        C
ATOM      3  C   ALA A  23      66.148  24.075  47.534  1.00181.62        C
```

Signal: Lines that begin with 'ATOM'

x y z

Where exactly are the xyz data?

```
1-4     14-15          33-38 41-46 49-54
```

Column nos. of interest

```
ATOM     14  N   HIS A  25      68.656  24.973  44.142  1.00128.26           N
ATOM     15  CA  HIS A  25      69.416  24.678  42.939  1.00128.26           C
ATOM     16  C   HIS A  25      68.843  23.458  42.227  1.00128.26           C
ATOM     17  O   HIS A  25      68.911  23.354  41.007  1.00128.26           O
ATOM     18  CB  HIS A  25      70.881  24.416  43.300  1.00154.92           C
ATOM     19  CG  HIS A  25      71.188  22.977  43.573  1.00154.92           C
ATOM     20  ND1 HIS A  25      71.886  22.184  42.689  1.00154.92           N
ATOM     21  CD2 HIS A  25      70.877  22.182  44.625  1.00154.92           C
ATOM     22  CE1 HIS A  25      71.993  20.963  43.183  1.00154.92           C
ATOM     23  NE2 HIS A  25      71.388  20.935  44.356  1.00154.92           N
ATOM     24  N   TRP A  26      68.271  22.546  43.005  1.00 87.09           N
ATOM     25  CA  TRP A  26      67.702  21.311  42.475  1.00 87.09           C
ATOM     26  C   TRP A  26      66.187  21.378  42.339  1.00 87.09           C
ATOM     27  O   TRP A  26      65.577  20.508  41.718  1.00 87.09           O
```

x y z

Just getting what you need from a data file

- Read past all the header information
- When you come to the lines of interest, collect the xyz data
  - Line starts with '**ATOM**'
  - Cols 14-15 is '**CA**'


```
fid = fopen('1bl8.dat', 'r');
x=[];y=[];z=[];
while ~feof(fid)
   s = fgetl(fid);
   if strcmp(s(1:4),'ATOM')
       if strcmp(s(14:15),'CA')
          x = [x; str2double(s(33:38))];
          y = [y; str2double(s(41:46))];
          z = [z; str2double(s(49:54))];
       end
   end
end
fclose(fid);
```

Iterate Until End of File

A detailed sort-a-file example

Suppose each line in the file

`statePop.txt`

is structured as follows:

Cols 1-14: State name
Cols 16-24: Population (millions)

The states appear in alphabetical order.

A detailed sort-a-file example

Create a new file

`statePopSm2Lg.txt`

that is structured the same as **`statePop.txt`** except that *the states are ordered from smallest to largest according to population.*

```
Alabama          4557808
Alaska            663661
Arizona          5939292
Arkansas         2779154
California      36132147
Colorado         4665177
   :                :
   :                :
```

- Need the pop as *numbers* for sorting.
- Can't just sort the pop— have to maintain association with the state names.

First, get the populations into an array

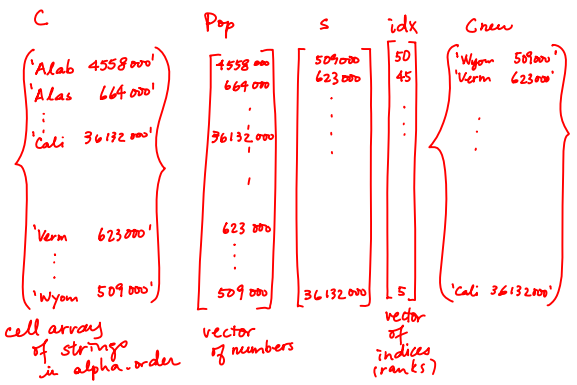
```
C = file2cellArray('StatePop');
n = length(C);
pop = zeros(n,1);
for i=1:n
  S = C{i};
  pop(i) = str2double(S(16:24));
end
```


Built-In function `sort`

Syntax: `[y,idx] = sort(x)`

x:	10	20	5	90	15
y:	5	10	15	20	90
idx:	3	1	5	2	4

`y(k) = x(idx(k))`



Sort from little to big

```
% C is cell array read from statePop.txt
% pop is vector of state pop (numbers)
[s,idx] = sort(pop);
Cnew = cell(n,1);
for i=1:length(C)
    ithSmallest = idx(i);
    Cnew{i} = C{ithSmallest};
end

cellArray2file(Cnew,'statePopSm2Lg')
```