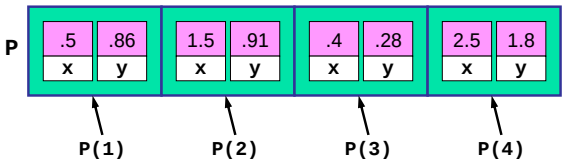
- Previous Lecture:
  - Structure & structure arrays
- Today's Lecture:
  - Structure arrays
  - Working with large data files
  - Built-in `sort` function
  - Read Chapter 11 to learn about the .bin file format
- Announcement:
  - Project 5 due Thursday, Apr 14th, at 11pm
  - Prelim 3 Tuesday, Apr 19th

Structure Arrays

- An array whose components are structures
- All the structures must be the same (have the same fields) in the array
- Example: an array of points (point structures)



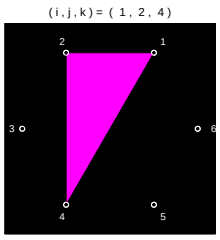
Function returning an array of points (point structures)

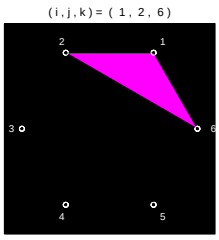
```
function P = CirclePoints(n)

theta = 2*pi/n;
for k=1:n
   c = cos(theta*k);
   s = sin(theta*k);
   P(k) = MakePoint(c,s);
end
```


Example: all possible triangles

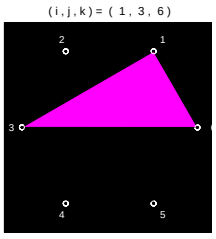
- Place n points uniformly around the unit circle.
- Draw all possible unique triangles obtained by connecting these points 3-at-a-time.

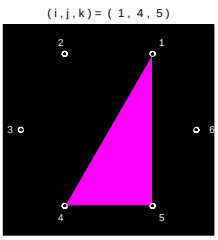





```
function DrawTriangle(P,Q,R,c)
% Draw c-colored triangle;
% triangle vertices are points P,
% Q, and R.

fill([P.x Q.x R.x], ...
     [P.y Q.y R.y], c)
```



The following triangles are the same: (1,3,6), (1,6,3), (3,1,6), (3,6,1), (6,1,3), (6,3,1)

Bad! i, j, and k should be different, and there should be no duplicates

```
for i=1:n
  for j=1:n
    for k=1:n
       % Draw a triangle with vertices
       %   P(i), P(j), and P(k)
    end
   end
end
```


All possible (i,j,k) combinations but avoid duplicates.
Loop index values have this relationship i < j < k

i j k

i = 1: 1 2 3, 1 2 4, 1 2 5, 1 2 6, 1 3 4, 1 3 5, 1 3 6, 1 4 5, 1 4 6, 1 5 6

i = 2: 2 3 4, 2 3 5, 2 3 6, 2 4 5, 2 4 6, 2 5 6

i = 3: 3 4 5, 3 4 6, 3 5 6

i = 4: 4 5 6

```
for i=1:n-2
  for j=i+1:n-1
    for k=j+1:n
      disp([i j k])
    end
  end
end
```


All possible triangles

```
% Drawing on a black background
for i=1:n-2
  for j=i+1:n-1
    for k=j+1:n
      DrawTriangle( P(i),P(j),P(k),'m')
      DrawPoints(P)
      pause
      DrawTriangle(P(i),P(j),P(k),'k')
    end
  end
end
```


Structures with array fields

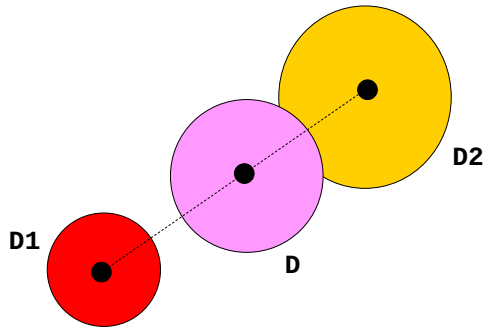
Let's develop a structure that can be used to represent a colored disk. It has four fields:

```
xc:  x-coordinate of center
yc:  y-coordinate of center
r:   radius
c:   rgb color vector
```

Examples:

```
D1 = struct('xc',1,'yc',2,'r',3,…
            'c',[1 0 1]);
D2 = struct('xc',4,'yc',0,'r',1,…
            'c',[.2 .5 .3]);
```


Example: Averaging two disks

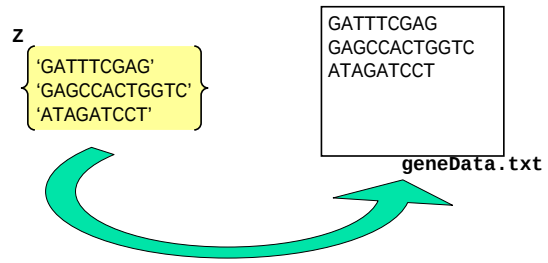


Example: compute "average" of two disks

```
% D1 and D2 are disk structures.
% Average is:
r  = (D1.r  + D2.r) /2;
xc = (D1.xc + D2.xc)/2;
yc = (D1.yc + D2.yc)/2;
c  = (D1.c  + D2.c) /2;

% The average is also a disk
D = struct('xc',xc,'yc'yc,'r',r,'c',c)
```

Example: Write a cell array of gene sequences to a file



Lecture 20 31

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

Lecture 20 32

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

Lecture 20 33

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**

Lecture 20 35

3. Close the file

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

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

fclose(fid);
```

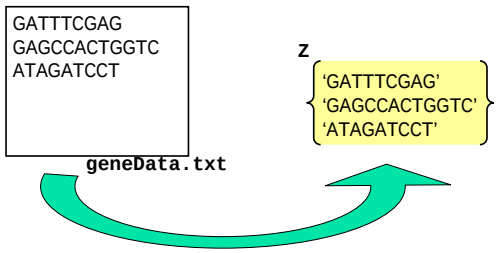
Lecture 20 36

```
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);
```

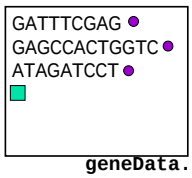
Lecture 20 37

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 **r**eading

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
```

x y z

Signal: Lines that begin with 'ATOM'

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);
```

Get the next line from 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

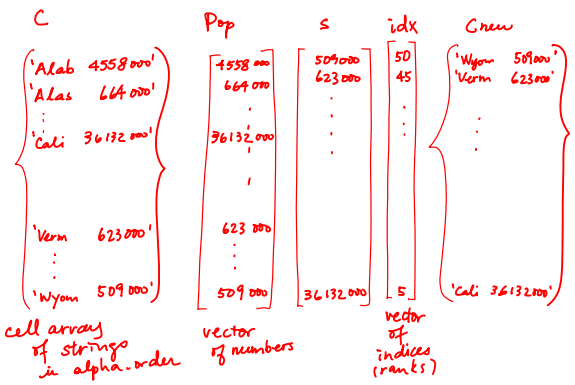
```
C = file2cellArray('StatePop');
n = length(C);
pop = zeros(n,1);
for i=1:n
  S = C{i};
  pop(i) = str2double(S(16:24));
end
```


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')
```