(B.1) Using fprintf( ) with arrays:

If you only specify one formatting command, all elements of an array will be printed on a single row (even multidimensional arrays). For multidimensional arrays, elements will be printed off one column at a time.

M-file:

```
x = (1 : 0.25 : 2)
y = [1,4,7 ; 9,8,6];
fprintf('%7.2f' , x)
fprintf('\n')
fprintf('%7.2f' , y)
```

Output:

```
   1.00   1.25   1.50   1.75   2.00
   1.00   9.00   4.00   8.00   7.00   6.00
```

In order to print off elements as columns, use the **\n** command. The format commands will be used over and over until all the elements are printed.

M-file:

```
x = (1 : 0.25 : 2);
y = [1,4,7 ; 9,8,6];
fprintf('%7.2f \n' , x)
fprintf('\n')
fprintf('%7.2f \n' , y )
fprintf('\n')
fprintf('%7.2f %7.2f %7.2f \n' , y )
fprintf('\n')
fprintf('%7.2f %7.2f \n' , y )
fprintf('\n')
fprintf('%7.2f %7.2f %7.2f \n' , y )
fprintf('\n')
fprintf('%7.2f %7.2f %7.2f %7.2f \n' , y )
```

After Execution:

```
  1.00
  1.25
  1.50
  1.75
  2.00

  1.00
  9.00
  4.00
  8.00
  7.00
  6.00

  1.00   9.00   4.00
  8.00   7.00   6.00

  1.00   9.00
  4.00   8.00
  7.00   6.00

  1.00   9.00   4.00
  8.00   7.00   6.00

  1.00   9.00   4.00   8.00
  7.00   6.00
```

If you try to print multiple arrays you may run into trouble. I would like to print off all **x**-values in the first column, all the corresponding **y**-values in the second column.

M-file:

```
x = (0:1:5);
y = x.^2;
fprintf('%4.1f  %6.1f \n' , x , y)
```

Output:

```
 0.0    1.0
 2.0    3.0
 4.0    5.0
 0.0    1.0
 4.0    9.0
16.0   25.0
```

Whoops. This didn't work because we printed off all the elements of **x** first, then the elements of **y**. Instead, we combine **x** and **y** into a single array and then print that. Remember, multi-dimensional arrays will be printed column by column.

M-file:
```
x = (0:1:5);
y = x.^2;
tablexy = [x;y];
fprintf('%4.1f  %6.1f \n' , tablexy)
```

Output:
```
 0.0    0.0
 1.0    1.0
 2.0    4.0
 3.0    9.0
 4.0   16.0
 5.0   25.0
```

(C) Writing to a file:

When writing to a file, you need to follow three steps:

(1) Open the file with the **fopen( )** command.

fileid = fopen('filename.txt' , 'w')

The **fileid** is a number that identifies the file you are opening. The **'w'** gives you permission to write to the file.

(2) Print to the file using **fprintf( )**.

fprintf(fileid, 'some text and formatting instructions' , variables)

Notice how we put the **fileid** inside the **fprintf( )** command. This is how Matlab/Octave knows where to write the information. If you left out the **fileid**, Matlab/Octave would write to the screen.

(3) Close the file using **fclose( )**.

fclose(fileid)

In mfile.m:

```
a = 50.5;
b = 21.2;
x = (0:0.25*pi:pi);
y = sin(x);
tablexy = [x;y];
file1 = fopen('sample.txt' , 'w');
fprintf(file1 , '%6.2f\n' , a , b);
fprintf(file1 , '%6.2f  %6.2f \n', tablexy);
fclose(file1);
```

Inside 'sample.txt':

```
> mfile
 50.50
 21.20
  0.00    0.00
  0.79    0.71
  1.57    1.00
  2.36    0.71
  3.14    0.00
```

(E) Reading from a file:

There are many commands that can be used to read data from a file depending on the situation. In this class we will learn one command, **fscanf( )**.

When reading data from a file, you need to follow a similar procedure as when writing data to a file.

(1) Open the file with the **fopen( )** command.

fileid = fopen('filename.txt')

The **fileid** is a number that identifies the file you are opening. Notice that we left off the **'w'** since we do not want to write to this file.

(2) Read from the file with the **fscanf( )** command.

output_array = fscanf(fileid, ‘format of data file’, size of data);

“size of data” refers to the number of rows and columns in the data file.
[2,10] means than 2 columns x 10 rows will be read in.

See example below for demonstration of **fscanf( )**.

(3) Close the file

fclose(fileid)

Example: Create a text file containing the hour, temperature , and relative humidity. Then read the data and store it in an array (which we can use later).

In mfile.m:

```
clear;clc;
% The first part of the program creates a text file containing the time and temperature.
time = [0,3,6,9,12,15];
temp = [55.3,54.1,54.0,56.7,62.9,63.1];
RH = [67,76,77,80,90, 93];
array = [time ; temp ; RH];
file1 = fopen('temp_RH.txt' , 'w');
fprintf(file1, '%2i \t %5.1f %5.1f \n' , array);
fclose(file1);

% now we read in the data
file2 = fopen('temp_RH.txt');
% This will read the time and temp data until the file is over
% (3 columns and an infinite number of rows... unless end of file is reached)
% [3,6] would do the same thing as [3,inf] in this case.
A = fscanf(file2, '%f' , [3, inf]);

%A = fscanf(file2, '%f' );
fclose(file2);

% Notice that the data is transposed when stored in A.
% fscanf( ) reads data in COLUMN ORDER.  Similar to how fprintf( ) writes in column order.
disp(A)

% However, this format is perfect for fprintf since it prints column by column
fprintf('%3i %7.2f %7.2f \n', array)

% We can manipulate the data in A
fprintf('The mean temperature is %5.1f \n' , mean(A(2,:)) )
fprintf('The mean RH is %5.1f \n' , mean(A(3,:)) )
```

In ‘temp.txt’:

```
 0        55.3  67.0
 3        54.1  76.0
 6        54.0  77.0
 9        56.7  80.0
12        62.9  90.0
15        63.1  93.0
```

After execution:

```
   0.00000    3.00000    6.00000    9.00000   12.00000   15.00000
  55.30000   54.10000   54.00000   56.70000   62.90000   63.10000
  67.00000   76.00000   77.00000   80.00000   90.00000   93.00000
  0   55.30   67.00
  3   54.10   76.00
  6   54.00   77.00
  9   56.70   80.00
 12   62.90   90.00
 15   63.10   93.00
The mean temperature is  57.7
The mean RH is  80.5
```

Comparing data in arrays:

I didn't have time to get to this material when discussing logical expressions.

M-file:

```
a = 1;
b = 2;
c = 4;
d = (a<3) & (b==c)
f = (a<3) | (b==c)
g = ~f
```

Output:

```
d = 0
f =  1
g = 0
```

You can compare entire individual elements in arrays too.

```
> quiz1 = [99,88,77];
> quiz2 = [90,90,89];
> quiz1 < quiz2
ans =   0   1   1
```

An array of true/false values is returned. These can be stored in a logical array.

```
> a = quiz1 < quiz2
a =   0   1   1
```

You can compare all elements in an array to a single value.

```
> b = quiz1 > 79
b =   1   1   0
```

```
> whos
Variables in the current scope:
  Attr   Name        Size              Bytes   Class
 ==== ====        ====              =====  =====
         a           1x3                   3    logical
         b           1x3                   3    logical
         quiz1       1x3                  24    double
         quiz2       1x3                  24    double
```

a and **b** are logical arrays.

The find( ) command:

This command will take as input a logical array and return the index of the array elements that are true.

```
> find(a)
ans =   2   3

> c = find(quiz1 > 79)
c =   1   2
```

You can use these array elements to find the value of the test scores that were higher than 79.

```
> quiz1(c)
ans =   99   88
```

Comparing character arrays:

You can compare character data too. Remember that characters are stored individual elements.

In M-file:

```
a = 'HTHHT'
b = 'TTTTH'
a == b
```

Output:

```
a = HTHHT
b = TTTTH
ans =   0   1   0   0   0
```

Only the second element of each array is the same. The number of elements in **a** and **b** must be the same.