Chapter 8
File Input and Output

Outline

8.1 Concept: Serial Input and Output (I/O)
8.2 Workspace I/O
8.3 High-level I/O Functions
8.4 Low-level File I/O
8.5 Engineering Example— Engineering Data

8.1 Concept: Serial Input and Output

- We refer to the process of reading and writing data files as Input/Output (I/O).
- When a program opens a file by name for reading, it continually requests block of data from the file stream until the end of the file is reached.
- As the data is received, the program must identify the delimiting characters and reformat the data as represented in the file.

8.1 Concept: Serial Input and Output

- Similarly, when writing data to a file, the program must serialize the data. To preserve the organization of the data, the appropriate delimiting characters must be inserted into the serial character stream.

- The purpose of the file I/O functions is to encapsulate them into *a single system function*.

8.2 Workspace I/O

- MATLAB defines the tools to save your complete workspace to a file with the *save* command, and reload it with the *load* command.

- If you provide a file name, i.e. my_filename, with the save command, MATLAB will save some variables or the entire workspace to my_filename.mat:

```
>> save mydata.mat a b c*
```

8.3 High-level I/O Functions

- Most programming languages require the programmer to write detailed programs to read and write files.

- Fortunately for MATLAB programmers, much of this work has been built into special file readers and writers.

8.3 High-level I/O Functions

```
x = 0:.1:1;
y = [x; exp(x)];                          %create a matrix with x, and f(x)
fid = fopen('exp.txt','w');          %open exp.txt file for writing
fprintf(fid,'%6.2f  %12.8f\n',y); %save the y matrix to file
fclose(fid);
```

The exp.txt file will contain the following values:

```
 0.00    1.00000000
 0.10    1.10517092
        ...
 1.00    2.71828183
```

8.3 High-level I/O Functions

- Suppose the file we are reading contains the string

'Blackbird singing in the dead of night'

- The following command returns only five characters of the first field:

```
C = textscan(fid, '%5s', 1);
C{:}
ans = 'Black'
```

8.3 High-level I/O Functions

- If we continue reading from the file, *textscan* resumes the operation at the point in the string where you left off:

```
C = textscan(fid, '%s %s', 1);
```

- The results are

```
C{:}
ans = 'bird‘
ans = 'singing'
```

8.4 Low-level File I/O

- Some text files contain data in mixed format that are not readable by the high-level file reading function.

- MATLAB provides a set of lower-level /O function that permit general-purpose text file reading and writing.

- In general:

  The file must be opened to be used by subsequent functions to identify its data stream.

8.4 Low-level File I/O

- We usually refer to this identifies as the “file handle”. After the file contents have been manipulated, the file must be closed to complete the activity.

- To open a file for reading or writing, use:

fh = fopen(<filename>, <purpose>);

<filename> is the name of the file to read/write.

<purpose> specifies the purpose which may be reading: ‘r’, writing ‘w’ (file contents will be ovewritten), or append, ‘a’ (to append new data).

8.4.2 Reading Text Files

- To read a file, three levels of support are provided: reading whole lines, parsing into tokens with delimiters, or parsing into cell arrays.
  - To read a whole line use str = fgets(fh) which will return each line as a string.
  - To parse each line into tokens separated by white space delimiters, using fgetl(…) and the tokenized function [<tk>,<rest>] = strtok(<ln>); where; where <tk> is a string token, <rest> is the remainder of the line, and <ln> is the original string.
  - MATLAB can parse a line into a cell array by using ca = textscan(fh, <format>; where <format> is a format control string.

8.4.4 Writing Text Files

- Once a file has been opened, the fprintf(…) function can be used to write to it. At the end every file needs to be closed.

```
oh = fopen(ofn, 'w');

...

fprintf(oh, ln);

...

fclose(oh);
```

Where oh is the file ID, and ln is a line of characters (a string)

8.5 Engineering Example—Reading Engineering Data File

- Problem:

Consider the problem where it is required to read a file of measurements. The data file includes repeated sets of times, dates, and measurements.

- The data file has the following format:
  - Number of measurements

  Time #1

  Date #1

  Measurements #1

  ....

8.5 Engineering Example—Reading Engineering DataFile

```
% Define the file name and size of record
filename='measurements.txt';
measrows=4;
measCols=4;

% Open the file
fid=fopen(filename);

% Read the file headers, find N (one value)
N=fscanf(fid,'%*s %*s\nN=%d\n\n',1);

% Read each set of measurements
for n=1:N

    % Read the time and date of each record
    struct(n).mtime=fscanf(fid,'%s',1);
    struct(n).mdate=fscanf(fid,'%s',1);

    % fscanf fills the array in COLUMN ORDER,
    % so transpose the results
    struct(n).meas =fscanf(fid,'%f',[measrows, measCols])';


end

% Close the file
fclose(fid);
```

8.5 Engineering Example—Reading Engineering Data File

```
>> struct(1)

ans =

    mtime: '12:00:00'
    mdate: '01-Jan-1977'
     meas: [4×4 double]

>> struct(1).mtime

ans =

12:00:00

>> struct(1).mdate

ans =

01-Jan-1977

>> struct(1).meas

ans =

    4.2100    6.5500    6.7800    6.5500
    9.1500    0.3500    7.5700       NaN
    7.9200    8.4900    7.4300    7.0600
    9.5900    9.3300    3.9200    0.3100

fx >>
```

8.5 Engineering Example—Reading Engineering Data File

- Problem:

Consider the problem where it is required to read a file of measurements. The data file includes repeated sets of times, dates, and measurements.

- Now consider using the EOF (End-Of-File). EOF is a condition in a computer operating system where no more data can be read from a data source. The data source is usually called a file or stream.

8.5 Engineering Example—Reading Engineering Data File

```
1     % Define filename and size of records
2 -   filename='measurements1.txt';
3 -   measrows=4;
4 -   meascols=4;
5
6     % Open the file
7 -   fid=fopen(filename);
8
9     % Read the file
10 -  recordn=1;
11 -  while ~feof(fid)
12 -    struct(recordn).mtime=fscanf(fid,'%s',1);
13 -    struct(recordn).mdate=fscanf(fid,'%s',1);
14
15      % fscanf fills the array in COLUMN ORDER,
16      % so transpose the results
17 -    struct(recordn).meas =fscanf(fid,'%f',[measrows, meascols])';
18
19 -  recordn=recordn+1;
20 -  end
```

8.5 Engineering Example—Reading Engineering Data File

```
>> struct(1)

ans =

    mtime: '12:00:00'
    mdate: '01-Jan-1977'
     meas: [4x4 double]

>> struct(1).meas

ans =

    4.2100    6.5500    6.7800    6.5500
    9.1500    0.3500    7.5700       NaN
    7.9200    8.4900    7.4300    7.0600
    9.5900    9.3300    3.9200    0.3100

fx >>
```

Homework on Chapter 8 is posted on the website:

http://www.ee.nmt.edu/~erives/289_F12/EE289.html

Homework is due within a week