Reading and writing data in files

It is often very useful to store data in a file on disk for later reference. But how does one put it there, and how does one read it back? Each programming language has its own peculiar convention. Scilab provides several different functions that can be used to write and read data, each of which is designed for a particular situation. In this lesson, I will describe one method: not the most efficient one, certainly, but one which will look very similar in several other programming languages. If you learn to use this style, you can very easily adapt it to other environments.

In the examples below, the data file is a simple one: it contains several columns of ASCII characters, with the same number of columns on each line, like this:

```
1       2       3.5
2       4       7
3       6       10.5
4       8       14
```

This multi-column format is very common -- even spreadsheets can read and write it.

Reading multicolumn data

Let's break the task down into several steps.

1. Open the file for reading
2. Read the next line
    - If there are no more lines, quit
3. Close the file

Step 1 typically occurs near the start of a program, step 2 inside a loop, and step 3 at the very end. Let's look at each one in some detail.

- **Step 1: Open the file** Before we can read anything from a file, we need to open it via the *mopen* function. We tell Scilab the name of the file, and it goes off to find it on the disk. If it can't find the file, it returns with an error; even if the file does exist, we might not be allowed to read from it. So, we need to check the value returned by *mopen* to make sure that all went well. A typical call looks like this:
- fid = mopen(filename, "r");

```
if (fid == -1)
  error("cannot open file for reading");
end
```

There are two input arguments to *mopen*: the first is a string with the name of the file to open, and the second is a short string which indicates the operations we wish to undertake. The string "r" means "we are going to read data which already exists in the file."

We assign the result of *mopen* to the variable *fid*. This will be an integer, called the "file descriptor," which we can use later on to tell Scilab where to look for input.

- **Step 2: Read the next line** One usually enters a loop in which one reads data from the file repeatedly. We can call the *mfscanf* function to read a specific number of items from the next line (4 items, in the example below):

```
while (done_yet == 0)

  [num_read, val(1), val(2), val(3), val(4)] = mfscanf(fid, "%f %f %f %f");
  if (num_read <= 0)
    done_yet = 1;
  end

```

We need to pass to the *mfscanf* function the file descriptor of the file from which we wish to read, and then a template for the number and type of items that we wish to read. In the example above, I am expecting to see **four floating-point values** at the start of the next line in the file. There could be additional items after these four -- they will be ignored.

The function returns two types of information. First, how many of the requested items was it able to read properly?

- -1, if there are no more lines to read, or
- 0, if the line didn't have any such items, or
- some number between 1 and 4 (in this example), for the number of floating point values which it did read properly

If there are no more lines to read, one usually needs to break out of a loop, or set a flag to do so.

Next, the *mfscanf* function places the individual items of the line into its second (and third, and fourth, etc.) output arguments.

Given this line:

```
1   2   3.5   4.9
```

the above call to *mfscanf* would result in

```
num_read  =  4
val(1)    =  1
val(2)    =  2
val(3)    =  3.5
val(4)    =  4.9
```

But given this line:

```
1   2   3.5   xyz
```

the above call to *sscanf* would result in

```
num_read  =  3
val(1)    =  1
val(2)    =  2
val(3)    =  3.5
val(4)    =   is undefined
```

You should always check to make sure that *mfscanf* was able to read the number of items expected.

- **Step 3: Close the file** At the very end of the program, after all the data has been read, it is good form to close a file:
- fclose(fid);

If, in the middle of a program, you need to start reading data from the beginning of a file you've already opened and used, you can close the file and then open it again.

Writing multicolumn data

Let's break the task down into several steps.

1. Open the file for writing
2. Write the next line
3. Close the file

Step 1 typically occurs near the start of a program, step 2 inside a loop, and step 3 at the very end. Let's look at each one in some detail.

- **Step 1: Open the file** Before we can write anything into a file, we need to open it via the *mopen* function. We tell Scilab the name of the file, and give the second argument 'w', which stands for 'we are about to write data into this file'.
- fid = mopen(filename, "w");
- if (fid == -1)
- error('cannot open file for writing');
- end

  When we open a file for reading, it's an error if the file doesn't exist. But when we open a file for writing, it's not an error: the file will be created if it doesn't exist. If the file does exist, all its contents will be destroyed, and replaced with the material we place into it via subsequent calls to *mfprintf*. Be sure that you really do want to destroy an existing file before you call *mopen*!

- **Step 2: Write the next line** Placing data into the file is easy: call the *mfprintf* function, just as you would call *mprintf* to print values to the screen; but include as the first argument the file descriptor
- mfprintf(fid, "%f %f %f \n", time, height, velocity);

  Don't forget to place a newline character '\n' to mark the end of each line. If you don't supply newline characters, all the data will be placed on a single, very long line.

- **Step 3: Close the file** At the very end of the program, after all the data has been written, it is good form to close a file:
- mclose(fid);

Examples

You can find some examples of short, simple Scilab programs which read and write information from and to files in the examples section of the course WWW site. Specifically,

- read_words.sci
- write_file.sci
- plot_from_file.sci

Practice in class

- Write a program which prints lines to a file called "powers.txt". The file should have 10 lines; in the first column is the line number (1, 2, 3, ..., 10), in the second column the square of the line number (1, 4, 9, ..., 100), and in the third column the cube of the line number (1, 8, 27, ..., 1000). So the first two lines would look like this:
-
- 1 1 1
- 2 4 8

  When done, open your data file in a browser window and call me over so that I can see it and mark it off.

- Write a program which reads numbers from the file "powers.txt" and calculates, for each line, the sum of the numbers. So, the first line would add up to 3. Your program should print one line per sum, something like this:
-
- 1 + 1 + 1 = 3
- 2 + 4 + 8 = 14

Source: http://spiff.rit.edu/classes/phys317/lectures/readwrite.html