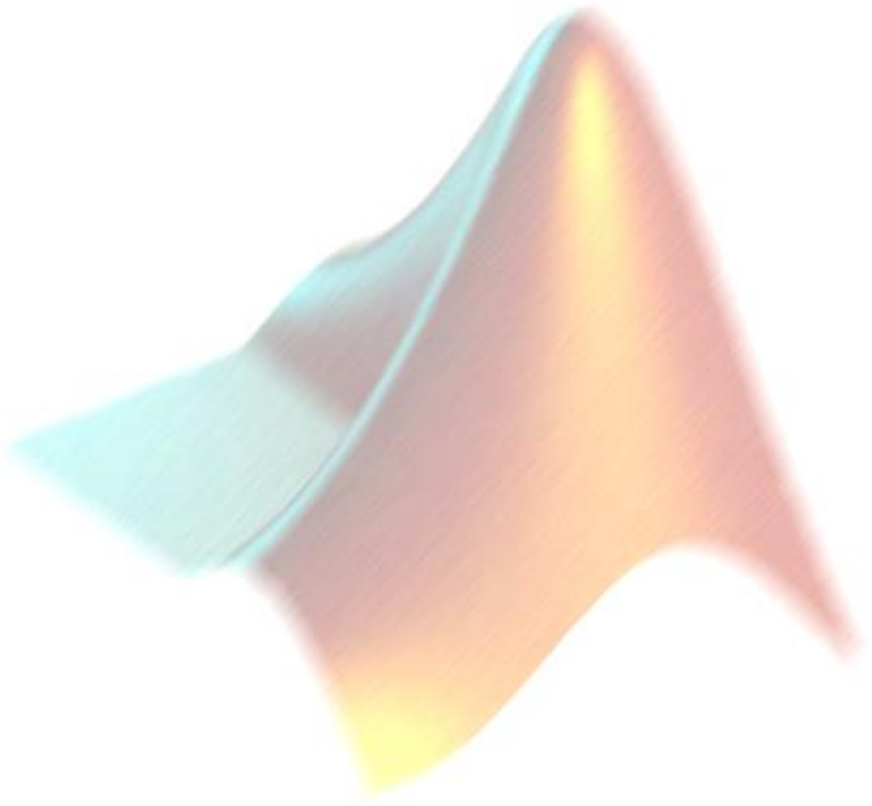
ENGGEN 131
Engineering Computation and
Software Development

Laboratory 5: Strings and Files



The Department of Engineering Science
The University of Auckland

Laboratory 5

TASK 5.0

Download the file “Lab_5.zip” from Canvas and extract the contents to an appropriate file directory. This contains files you may need for later tasks.

Then read the post “Lab 5 Notes” on the class Piazza discussion forum.

STRINGS

Recall that there are a number of useful functions for comparing strings.

strcmp	Compare two strings
strncmp	Compare the first n positions of two strings
strcmpi	Compare two strings ignoring case
strfind	Search for occurrences of a shorter string in a longer string

Try typing the following and check that you understand the result of each string comparison

```
str1 = 'banana'
str2 = 'baNANA'

strcmp(str1,str2)
strcmp(str1,lower(str2))

strncmp(str1,str2,2)
strcmpi(str1,str2)

strcmp(str1,'bandana')
strncmp(str1,'bandana',3)
strncmp('banana','bandana',4)
```

Stop ***Check your understanding***

Make sure you know why the above commands evaluated to either true or false. Ask a teaching assistant if you are unsure.

The `strfind` function is useful for searching a string for occurrences of a shorter string and returning the location(s) of that pattern. The `strfind` function returns an array listing the location(s) of that pattern. If the pattern is NOT found then the array will be empty.

To easily check if a pattern is found we can simply check if the length of the array containing the locations is 0 or not.

```
str1 = 'banana'

strfind(str1,'ana')
length(strfind(str1,'ana'))

strfind(str1,'skin')
length(strfind(str1,'skin'))
```

STRINGS AND USER INPUT/OUTPUT

By default the input command expects users to enter a numerical value. If you want to interpret the entered value as a string the input command needs to be passed the letter s as a second argument:

name = input('Enter your name:', 's')

The sprintf command is very useful for creating a nicely formatted string of characters which can then be displayed with the disp command. It supports a wider range of kinds of outputs.

Here is a reminder of some of the more common outputs.

specifier	**Output**	**example**
s	String	hello
c	Character	c
d or i	decimal integer	-23
e	scientific notation	1.2345e+10
f	decimal floating point	23.1234
g	The shorter of e or f	

The `sprintf` command takes a format string which usually includes one or more % signs followed by optional control flags and a specifier. Other arguments passed into the `sprintf` command are inserted in order in place of the specifiers, using the specified format.

Try typing in the following commands, leaving off the semi-colon so that you can see what string is created:

```
x = 10
sprintf('The value of x is %d and x cubed is %e',x,x^3)

name = input('Enter your name:','s');
sprintf('Hello %s, how are you?',name)

sprintf('The value of pi is: %f',pi)
sprintf('Pi with scientific notation is: %e',pi)
```

Inserting a decimal and a number between the % character and the specifier allows you to specify how many decimal places to use

```
sprintf('Pi with 2dp is: %.2f',pi)
sprintf('Pi with 8dp and scientific notation is : %.8e',pi)
```

Inserting just a number between the % character and the specifier allows you to specify the minimum width to reserve for displaying the value. This is handy when wanting to format output in columns.

Try typing the following and running it:

```
for i=0:10:100
     string = sprintf('The sqrt of %3d is %7.4f',i,sqrt(i));
     disp(string);
end
```

TASK 5.1 Formatting Strings

Often at times, we would prefer string output to look “pretty”. This requires the use of special string formatting functions like `sprintf` or `fprintf`.

Write a new script file that asks for Enggen131 percentage score for each of the **five** assessment categories (Labs, Assignments, Projects, Tests, Exam) using the function `input`. It should then display “pretty” formatted strings that show the contribution towards the final mark (based on the weighting of each assessment category). The script should at a minimum print out the name of each assessment, the contribution from that assessment and the final mark.

The final mark may be obtained by calling the function `FinalPercentage` with the correct inputs (see the header of the supplied script `FinalPercentage.m` to understand how it works. Here are some possible pretty outputs.

```
Labs                 24.0
Assignments           2.0
Projects             20.0
Tests                18.0
Exam                 15.0
-----------------------
Final Mark           79.0
```

```
ENGGEN 131 Marks
  Course Work
    Labs                    24
    Assignments              2
    Projects                20
    Tests                   18
  Final Exam
    Exam                    15

  Final Mark                79
```

```
*****************************
*****  ENGGEN131  Marks *****
**                         **
**   Labs         24/24    **
**   Assignments   2/2     **
**   Projects     20/20    **
**   Tests        18/24    **
**                         **
**   Exam         15/30    **
**                         **
**   Final Mark     79%    **
**                         **
*****************************
*****************************
```

Notice how in these examples, the columns of text and numbers line up and how the numbers are right justified whereas the text is left justified? This is an example of what people may consider “pretty”.

You may name each assessment appropriately and display grades as you wish (e.g. fractions, percentages, letters, weighted or raw etc.). The above three example outputs would all be acceptable. The important thing is that the data looks nice and lines up.

Show the code and output of the completed script to a Teaching Assistant.

TASK 5.2 Processing strings

You are provided with a file `DNAString.mat`. It contains two variables `dnaStringLong` and `dnaStringShort` which is a long and short string respectively of a DNA sequences gathered from a newly discovered species.

To analyse the genetic nature of this new species, you must compare its DNA string with a few known DNA strings. These strings are:

```
Sequence 1a: 'AGTCACT'
Sequence 1b: 'AcgT'
Sequence 2a: 'TACTga'
```

The underlined letters are case insensitive – so for example, an occurrence of `'taCTga'` would count as a match for sequence 2a, but `'tACTgA'` would not.

Write a function called `GetDnaMatches` that searches through the DNA strings and finds the number of matches to each sequence.

Input:

a character array containing a DNA string e.g. `dnaStringLong` or `dnaStringShort`.

Output:

A 1 by 3 array of the number of matches to each sequence 1a, 1b, 2a repectively.

You are also provided with a function `DnaOutputFormatter`. It accepts one array (three elements) of values indicating the number of DNA string matches (i.e the output of the function `GetDnaMatches`).

.

The output obtained from the function `GetDnaMatches(dnaStringShort)` combined with the function `DnaOutputFormatter` should look like:

```
     Sequence 1      |     Sequence 2
 a:                7 | a:               2
 b:               19 |
------------------------------------------
 Total:           26 | Total:           2
```

Test your function by possibly creating a new test script or using the console.

Submit the completed function `GetDnaMatches` to MATLAB Grader.

FILE I/O

MATLAB provides an import wizard which can be used to import data files. Download the gasData.txt file from Canvas. Now try using the file import command to import the data contained in the file (go to File > Import Data)

Sometimes the import wizard fails and we need to write code which will read in data from a file. Also if you wish to process many files at a time it is much more convenient to be able to use commands to read the contents of a file.

Instead of using the import wizard we will read in the contents of the gasData.txt file using file commands. See the chapter on file I/O for a reminder on how to do this.

Remember that you must open the file for reading using fopen before reading from it. You should close your file using fclose when finished.

TASK 5.3 Reading a file

Measurement data obtained from scientific experiments are often straight exported to files such as `.txt` and `.csv`. These raw measurements can be read into programs like MATLAB, then postprocessed to obtain values that we want, and can save by writing to files.

The file named `gasData.txt` contains experimental data on the volume (V) in m^3 and temperature (T) in °C of 2 moles (n) in mol of a particular gas.

Write a function called `ReadGasData` that reads data from a gas data file and returns the volume data and temperature data as separate array variables.

Input:
- A filename string of the gas data file,
- The first number of rows `m` of data to read (excluding the header),

Output:
- A `1` by `m` array of gas volume in m^3,
- A `1` by `m` array of gas temperature °C.

For example:

```
[Vol, Temp] = ReadGasData('gasData.txt', 1000);
```

reads the first 1000 rows of data from the file `gasData.txt`.

Make sure you test your function thoroughly by creating a test script.

Submit the completed function `ReadGasData` to MATLAB Grader.

TASK 5.4 Writing to a file

As from Task 5.3, the file named `gasData.txt` contains experimental data on the volume (V) in m^3 and temperature (T) in °C (**which will need to be converted to K by adding 273.15 to the temperature in °C**) of 2 moles (n) in mol of a particular gas. These measurements can be used to calculate the pressure (P) Pa of the gas (which is the data that is required in this situation), using the universal gas constant (R) of 8.314 m^3 Pa K^{-1} mol^{-1} and the relationship:

$$PV = nRT$$

Write a function called `WriteGasData` that writes an array of pressure data to a specific file and returns as output, the same array of pressure data.

Input:

A filename string of the output gas data file,
A `1` by `m` array of gas volume in m^3,
A `1` by `m` array of gas temperature °C.

Output:

A `1` by `m` array of gas pressure in Pa,

For example:

```
   Pressure = WriteGasData ('pressureData.txt', Vol, Temp);
```

calculates the pressure, then writes the pressure data (with a header line) to the file `pressureData.txt` and returns the pressure data (that can be used later on). Your file should look something like:

```
Pressure (Pa)
576.9813
581.7608
585.1694
581.2526
  ...
  ...
  ...
923.1845
927.1167
925.6272
928.9082
```

Make sure you test your function thoroughly by creating a test script (or extending and modifying the test script from Task 5.3).

Submit the completed function `WriteGasData` to MATLAB Grader.

TIP *If working on a windows machine use "Text" mode" when writing to a file*

A file can be opened for writing in text mode by adding a t next to the w, e.g.

```
myfid = fopen('squares.txt','wt');
```

If you use text mode on a windows machine, carriage return characters will automatically be inserted in front of any line feed (\n) characters, meaning your file will display correctly when opened in Microsoft Notepad. See **Using text mode (or a brief history of newlines)** at the end of Chapter 9 for more detail.

CELL ARRAYS

Sometimes we wish to work with a special kind of array where each element of the array is a string. These arrays have a special name in MATLAB, they are called Cell arrays. They are created and indexed with **curly braces**:

```
helloWorld = { 'hello', 'world' }
length(helloWorld)
disp( helloWorld{1} )
upper( helloWorld{2} )
```

When importing a file into MATLAB it is quite common for a cell array to be created.

TIP *Remember to use curly braces with cell arrays*

If you use square brackets to try and create a cell array you will not create a cell array, instead you will just end up with a concatenation of the individual strings.

If you use round brackets to try and index a cell array you will get a 1x1 cell array back rather than a string. This can be very confusing if you do not spot your error.

MATLAB string functions will also work on cell arrays, but the results can be rather confusing. If working with a cell array it is often a good idea to use a for loop to work through your cell array, allowing you to work on one string at a time.

Try saving the following code to a script file and running it

```
myMessage = { 'Remember', 'to', 'use', 'curly', 'braces' }
for i = 1:length(myMessage)
     str = myMessage{i};
     len = length(str);
     disp ( ['length of ' str, ' is ', num2str(len)] )
end
```

TASK 5.5 Debugging strings

The function `FindPhoneNumber` attempts to find phone numbers associated with a given name (or all matching names).

Open this function in the MATLAB Editor, read the header comment to understand what the function should do and then fix any bugs so that it runs correctly.

Remember you can use the debugger to step through line by line, so that you can see how each line of code changes the variables in the workspace.

Test your function by finishing the test script `TestFindPhoneNumber`.

Submit the contents of the debugged function `FindPhoneNumber` to MATLAB Grader.

LAB 5 Lab Task Submission Summary

□ Task 5.1
Show the code and output of the script to a Teaching Assistant.

□ Task 5.2
Submit the completed function `GetDnaMatches` to MATLAB Grader.

□ Task 5.3
Submit the completed function `ReadGasData` to MATLAB Grader.

□ Task 5.4
Submit the completed function `WriteGasData` to MATLAB Grader.

□ Task 5.5
Submit the debugged function `FindPhoneNumber` to MATLAB Grader.

COMPULSORY LAB TASKS END HERE

EXAM PRACTICE (PART A)

You are interested in catching people who have been copying each other's project files. Write a function called `CheckSimilarity` that will take as input two file names and then check to see how similar the first file is to the second file. Your function will do this by counting how many corresponding lines are identical. i.e. if line 2 of file1.txt matches line 2 of file2.txt then the count of identical lines is increased. Note that for simplicity we will only count matches for *corresponding lines*, so if line 2 from file1.txt matches line 3 of file2.txt, the count of identical lines is NOT increased. It will also calculate a percentage similarity figure using the formula:

*Number of identical lines / number of lines in shortest file * 100*

Your function should return two output values:
the number of identical lines,
the percentage similarity between the two files (a value between 0 and 100)

IMPORTANT: Remember to comment your file.

EXAM PRACTICE (PART B)

You are interested in catching people who have been copying each other's project files. Write a script file that will check to see how similar a given file is to all other files in the same directory. You should use the `CheckSimilarity` function you wrote in part A, which given two file names will return the number of identical lines and a similarity score.

You will need to download the function `GetFileNames` from Canvas. `GetFileNames` returns a cell array containing a list of all filenames in the current MATLAB directory.

Your script file should ask the user to enter a file name to check and then it should write out a file called similarityReport.txt

similarityReport.txt should contain a summary of how similar each file in the directory is to the entered file. It should use the format shown below:

```
Similarity Check for file: file1.txt

Filename     Lines the same      %similar
file1.txt          10                100.0
file2.txt           5                 50.0
file3.txt           5                100.0
file4.txt           1                  5.0
```

IMPORTANT: Remember to comment your file