**\*BE2080 Week#4**

**Submission:**

**1. You can still copy/paste your programs in a word document for submission.**

**2. Since the programs may become longer, you can also submit your Matlab programs (.m files). If you choose to do so, please name your files clearly, for example, Wk4\_Q1.m.**

**3. Please check and make sure all your files are submitted.**

1) (10 points) A file name is supposed to be in the form *filename.ext*. Write a **function** that will determine whether a string is in the form of a name followed by a dot followed by a three-character extension, or not. The function should return 1 for **logical** **true** if it is in that form, or 0 for **false** if not.

Some example filenames that will return true: data.txt, rain.dat, image.tif, data.dat.txt, ……

Some example filenames that will return false: movie.mpeg, img.tiff, data, ……

\*Doesn’t work for things like ‘data.txt.ex’ 🡪 check if filename(end-3) == "

function [output] = checkFilename(filename)

%{

This function determines if a string is in the form of a

name followed by a dot followed by a three-character extension.

%}

[~, rest] = strtok(filename, '.');

[extension, ~] = strtok(rest, '.');

num = sum(isletter(extension));

if isempty(extension)

output = false;

elseif num ~= 3

output = false;

else

output = true;

end

end

2) (10 points) Write a **function** *getstr* that prompts the user for a string, error-checking until the user enters something (the error would occur if the user just hits the Enter key without any other characters first). The function then returns the string as the output argument.

There should be no errors if you test your function in the Matlab main command window using the following expressions.

thestring = getstr();

fprintf('Thank you, your string is %d characters long\n', ...

length(thestring))

function [user\_str] = getstr()

%{

This function prompts user for string, error-checking until

user enters something. Function then returns string as output

argument.

%}

user\_str = input('Please enter a string: ', 's');

while isempty(user\_str)

user\_str = input('Please enter a string: ', 's');

end

end

3) (10 points) Write a **script** that will create x and y vectors. Then, it will ask the user for a color (‘red’, ‘blue’, or ‘green’) and for a plot style (circle or star). It will then create a string *pstr* that contains the color and plot style, so that the call to the **plot** function would be: **plot(x,y,pstr)**. For example, if the user enters ‘blue’ and ‘\*’, the variable *pstr* would be ‘b\*’.

%This script plots based on user choices.

%creation of vectors

x = 1:25;

y = sin(x);

%color

color = input('Enter a color (red, blue or green): ', 's');

if strcmpi(color, 'red') == 1

col = 'r';

elseif strcmpi(color, 'blue') == 1

col = 'b';

else

col = 'g';

end

%plot style

style = input('Enter a plot style (circle or star): ', 's');

if strcmp(style, 'circle') == 1

sty = 'o';

else

sty = '\*';

end

%create string that contains color & plot style

pstr = strcat(col, sty);

%plot function

plot(x,y,pstr)

4) (10 points) Two variables store strings that consist of a string of alphabets, a blank space, and a number (for example, in the form ‘AIR 14.3’). Write a **function** AddStrings that would receive two such strings as input arguments. Then, the function will use string manipulating functions to extract the numbers from the strings, add them together, and return the sum as the output argument.

For example, the following expressions can be used to test the function:

>>str1 = 'Door 8.23';

>>str2 = 'Window 24.4';

>> Total = AddStrings(str1, str2)

Total =

32.63

\*str1 and str2 should be input arguments

function [Total] = AddStrings()

%{

Functions receives two strings as input arguments, extracts

numbers from strings, adds numbers together, and returns sum

as output argument.

%}

str1 = input('Enter a string that consists of letters, a blank space, and a number (e.g. AIR 68): ', 's');

str2 = input('Enter a second string that consists of letters, a blank space, and a number (e.g. AIR 68): ', 's');

%extract numbers

[~, rest] = strtok(str1);

[nums1, ~] = strtok(rest);

nums1 = str2double(nums1);

[~, rest] = strtok(str2);

[nums2, ~] = strtok(rest);

nums2 = str2double(nums2);

%sum numbers

Total = nums1 + nums2;

end

5) (10 points) A file called *avehighs.dat* stores the average high temperatures for each month for a year (rounded to integers). The first column of the data stores the zip code, the 2nd to 13th columns store 12 temperatures. For example, the file might store:

65211 33 37 42 45 53 72 82 79 66 55 46 41

65201 29 33 41 46 52 66 77 88 68 55 48 39

65203 55 62 68 72 75 79 83 89 85 80 77 65

Write a **function** that will take in one input argument: the zip code. This function will first read in the data file, then find the line corresponding to the zip code, and plot the temperatures for all 12 months. The axis labels and titles should be as shown. The figure title should show the zip-code. For example, if the function is called as ShowTemp(65201), the following graph will be shown:

\*need to implement for-loop

function [] = AvgHighTemps(zipcode)

%{

Plots average high temperature for each month of a year

in a specific zipcode.

%}

load('avehighs.dat')

x = 1:12;

if zipcode == 65211

y = avehighs(1, 2:13);

plot(x, y, 'bo')

title('Location 65211')

elseif zipcode == 78201

y = avehighs(2, 2:13);

plot(x, y, 'bo')

title('Location 78201')

elseif zipcode == 65203

y = avehighs(3, 2:13);

plot(x, y, 'bo')

title('Location 65203')

elseif zipcode == 34203

y = avehighs(4, 2:13);

plot(x, y, 'bo')

title('Location 34203')

end

xlabel('Month')

ylabel('Avg High Temps')

end

6) (10 points) Write a function with no input argument and two output arguments. This function will keep prompting the user to input a string and read them in, store them in a cell array, until the user hits the return key without inputting a string. This function will return the resulting cell array as the second output argument, the number of strings as the first output argument.

function [num\_of\_str, c\_a] = Question6()

%{

Stores strings user inputs in cell array. Returns cell array

and number of strings after user hits return key

without inputting a string.

%}

num\_of\_str = 0;

c\_a = {};

i = 1;

str = input('Enter a string: ', 's');

while ~isempty(str)

c\_a{i} = str;

i = i + 1;

num\_of\_str = num\_of\_str + 1;

str = input('Enter a string: ', 's');

end

end

>> [num\_of\_str, c\_a] = Question6()

Enter a string: hey

Enter a string: hello

Enter a string: howdy

Enter a string: hi

Enter a string: hola

Enter a string:

num\_of\_str =

5

c\_a =

1×5 cell array

{'hey'} {'hello'} {'howdy'} {'hi'} {'hola'}

7) (10 points) Write a function with two input arguments and one output argument. The two input arguments are both structure variables representing the x-, y-, and z-coordinates of the 3D point in space. The function will then calculate the line distance between the two points and return the distance as the output argument. The distance between two 3D points (x1,y1,z1) and (x2,y2,z2) is:

Below is an example testing of this function:

>>p1=struct('x',0,'y',0,'z',0);

>>p2=struct('x',1,'y',1,'z',1);

>>dis = LineLength(p1, p2)

ans =

1.7321

function [distance] = LineLength(p1,p2)

%{

Calculates line distance between two points and

returns distance as output argument.

%}

distance = sqrt((p1.x-p2.x)^2 + (p1.y-p2.y)^2 + (p1.z-p2.z)^2);

end

>> p1 = struct('x', 0, 'y', 0, 'z', 0);

>> p2 = struct('x', 1, 'y', 1, 'z', 1);

>> dis = LineLength(p1, p2)

dis =

1.7321

8) (10 points) In chemistry, the pH of an aqueous solution is a measure of its acidity. A solution with a pH of 7 is said to be *neutral*, a solution with a pH greater than 7 is *basic*, and a solution with a pH less than 7 is *acidic*.

A structure variable is created to store the PH value of a solution. This structure variable has two fields: one is called ‘Item’, and the other is ‘PH’. An example of such a variable is created as:

sol1 = struct('Item', 'water', 'PH', 7);

Write a function that will take in one such a variable as the input argument, determine its acidity based on the pH value, add another field ‘acidity’ to the structural variable with the determined value (*neutral,*  *basic*, or *acidic* ), and return this new structure as the output variable.

For example, the function should work in the same way as below:

>>sol1 = struct('Item', 'water', 'PH', 7);

>>OutputVar = addAcidity(sol1)

>>OutputVar =

Item: 'water'

PH: 7

Acidity: 'neutral'

function [newStruct] = addAcidity(previousStruct)

%{

Determines acidity based on pH value, adds acidity field

with determined value, and returns new structure as output

variable.

%}

newStruct = previousStruct;

if previousStruct.pH < 7 && previousStruct.pH >= 0

newStruct.Acidity = 'acidic';

elseif previousStruct.pH > 7 && previousStruct.pH <= 14

newStruct.Acidity = 'basic';

else

newStruct.Acidity = 'neutral';

end

end

>> sol1 = struct('Item', 'water', 'pH', 7);

>> OutputVar = addAcidity(sol1)

OutputVar =

struct with fields:

Item: 'water'

pH: 7

Acidity: 'neutral'

9) (10 points) Download and read in the file ‘Patients.mat’ which contains 100 patients’ data stored in **a vector of structures** called *subjects*. Examine the structure using fieldnames():

>>load Patients.mat

>>fieldnames(subjects)

Write a function to select potential patients for a clinical study. The only patients who are eligible are those whose weight is above a minimal weight AND the systolic pressure is below a maximal pressure.

The function should take this struct vector subjects as the first input argument, a minimal weight as the second input argument, the maximal pressure as the third input argument. The function will examine all records and return the names of those who are eligible in the form of **a cell array as the output argument**. For example,

>> names=FindPatients(subjects, 175, 135)

names =

3×1 cell array

{'Baker' }

{'Turner'}

{'Butler'}

function [names] = FindPatients(subjects, min\_weight, max\_pressure)

%{

Function selects potential patients for clinical study

whose weight is above minimal weight and systolic

pressure is below a maximal pressure.

%}

names = {};

j = 1;

%load Patients.mat;

for i = 1:length(subjects)

if subjects(i).Weight > min\_weight && subjects(i).Systolic < max\_pressure

names{j} = subjects(i).LastName;

j = j + 1;

end

end

end

OR

function [names] = FindPatients(subjects, min\_weight, max\_pressure)

%{

Function selects potential patients for clinical study

whose weight is above minimal weight and systolic

pressure is below a maximal pressure.

%}

names = cell(100, 1);

j = 1;

%load Patients.mat;

for i = 1:length(subjects)

if subjects(i).Weight > min\_weight && subjects(i).Systolic < max\_pressure

names{j} = subjects(i).LastName;

j = j + 1;

end

names(cellfun('isempty', names)) = [];

names = names'; %to make column, not necessarily needed

end

end