Problem Solving For Engineering Transfer

ENS 1300 - Spring 2020

Due Date	Lab 4: Logicical Operators, Conditional Statements, Input Validation	Last Modified
24 Febuary 2020		26 February 2020

Important

Create a new directory for this lab, `ENS1300/Labs/Lab04`. Set this as your working directory in MATLAB before beginning. Upon completion: (1) be sure that all the deliverable(s) specified at the end of each problem are contained within the `Lab04` folder, (2) zip `Lab04`, name it `Lab04_LastnameFirstname.zip`, and (3) upload `Lab04_LastnameFirstname.zip` to the brightspace dropbox.

Main deliverable: `Lab04_LastnameFirstname.zip`

Graded Item	Point Value
Zip file named as specified and contains all required files and directories	5
Total	5

Note: You'll need to download `data.zip` and `converter.zip` from the course website before beginning problems 2-6.

Problem 1

Write a function called `doubleFunc` that accepts a single input and returns no output.

- If the input is a double, print the size (dimensions) to the command window.
- If the input is not a double, exit execution with an error message
- If no input is provided, exit execution with an error message
- If more than one input is provided, exit execution with an error message

Useful functions: `class`, `strcmp` `isa` `fprintf`, `string`, `size`, `error`

Function Specifications:

Function Name	Purpose	Input(s)	Output(s)
`doubleFunc`	1: perform input validations 2: print an appropriate message	any	none

Deliverable(s): `doubleFunc.m`

Graded Item	Point Value
Adequate comments and organization	2
Correct input/output	3
Correct input validations	5
Total	10

Problem 2

Make a copy of the `calcGrade2.m` function file and the `calcGrade2test.m` script file from Lab 3 and name them `calcGrade3.m` and `calcGrade3test.m`, respectively.

(a) Modify the function to also determine the letter grade. Assign the string to a field called `letterGrade`.

$90 \leq \text{Grade} \leq 100 \quad \Rightarrow \quad \text{A}$
$85 \leq \text{Grade} < 90 \quad \Rightarrow \quad \text{B+}$
$80 \leq \text{Grade} < 85 \quad \Rightarrow \quad \text{B}$
$75 \leq \text{Grade} < 80 \quad \Rightarrow \quad \text{C+}$
$70 \leq \text{Grade} < 75 \quad \Rightarrow \quad \text{C}$
$60 \leq \text{Grade} < 70 \quad \Rightarrow \quad \text{D}$
$0 \leq \text{Grade} < 60 \quad \Rightarrow \quad \text{F}$

(b) Modify the **script file** to print the letter grade to the command window.

Function Specifications:

Function Name	Purpose	Input(s)	Output(s)
`calcGrade3`	1: calculate average 2: determine letter grade	structure with 4 fields (`exams`, `labs`, `quizzes`, `projects`)	structure with 6 fields (`exams`, `labs`, `quizzes`, `projects`, `avgGrade`, `letterGrade`)

Deliverable(s): `calcGrade3.m`, `calcGrade3test.m`

Graded Item	Point Value
Adequate comments and organization	5
Correct input/output	5
Total	10

Problem 3

In this problem, flexibility will be added to the cartesian to spherical coordinate conversion functions written in the previous lab. We will validate the input argument(s) in order to combine the functionality of `ENS1300/Labs/Lab03/cart2sphere_uno.m` and `ENS1300/Labs/Lab03/cart2sphere.m`

Make a copy of the `ENS1300/Labs/Lab03/cart2sphere.m` function file and name it `ENS1300/Labs/Lab04/cart2sphere_flex.m`.

- If the user provides no inputs, the program should work exactly like Lab 3, Problem 3.
- If the user provides a single input, ensure that it is a double with 3 columns, then convert to spherical coordinates. Otherwise, use the `error()` function to display a message and terminate execution.
- If the user provides two inputs, ensure that they are **both** either of class char or string, then proceed to run the program exactly like Lab 3, Problem 4. Otherwise, use the `error()` function to display a message and terminate execution.
- If the user provides more than two inputs, use the `error()` function to display a message and terminate execution.

Useful functions: `strcmp`, `isa`, `error`

Deliverable(s): `cart2sphere_flex.m`

Graded Item	Point Value
Adequate comments and organization	5
Correct input validations	10
Correct/appropriate program behavior	5
Total	20

Note: Use the `cartesianCoordinates.csv` file for testing.

Problem 4

Write a function called `logicalPract1` that accepts a single input and returns no outputs. The input should be the full path to a CSV file. The function should perform the following tasks:

(a) Import data from a CSV file.
(b) Determine how many NaN values are contained in the data set and print the result to the command window.
(c) Eliminate all NaN values from the data set.
  - Compare the dimensions of the original matrix to the matrix resulting from this operation. Describe why this happened in the header section of your file.

(d) Determine the following and print the results to the command window:
  - Mean of the largest 1000 elements.
  - Mean of the smallest 1000 elements.
  - Sum of the 150th, 200th, 1,000th, 20,000th, and 100,000th elements.
  - Standard deviation of all the values contained in even elements.
  - Square root of the sum of the squares of all values contained in odd elements.
  - Number of values that are greater than or equal to the mean.
  - Number of values that are less than the median.

Useful functions: `fullfile`, `csvread`, `dlmread`, `isnan`, `sort`, `mean`, `median`, `std`, `size`, `norm`, `fprintf`

Function Specifications:

Function Name	Purpose	Input(s)	Output(s)
`logicalPract1`	1: import data from CSV file 2: perform logical operations	full path to csv file (string or char)	none

Deliverable(s): `logicalPract1.m`

Graded Item	Point Value
Adequate comments and organization	5
Correct input/output	5
Correct logical operations	5
Total	15

Note: Use `LP1data.csv` for testing.

Problem 5

Write a function called `logicalPract2` that accepts a single input and returns a single output. The input should be the full path to a CSV file and the output should be a structure containing results. The function should perform the following tasks:

(a) Import data from a csv file.
- Assume that the first column is time and the other two columns contain sensor readings (sensor 1 and sensor 2).

(b) Determine how many NaN values are contained in the data set and print the result to the command window.

(c) Eliminate all bad/missing sensor readings (NaN elements) from the data set.
- NOTE: The columns will need to be separated before doing this. Since the sensor 1 and sensor 2 vectors will be different lengths after this operation, they will each need their own respective time vectors for plotting.

(d) Determine the following for both sensor 1 and sensor 2 and store the results in the output structure.
- Maximum values and times at which they occur (vectors [time, maximum value]).
- Minimum values and times at which they occur. (vectors [time, minimum value]).
- Number of readings above 1000 (scalars).
- Number of readings below -1000 (scalars).
- Number of readings between -100 and 100 (scalars).

(e) Create an appropriately annotated plot including both sensor readings. Use blue dots for sensor 1 and red dots for sensor 2.

It is recommended that you organize your code by using several subfunctions, but you are free to approach this problem however you like. The output structure should be defined **exactly** as follows:

```
results
|-- sensor1
|   |-- Neg100toPos100
|   |-- maxVals
|   |-- minVals
|   |-- over1000
|   |-- underNeg1000
|-- sensor2
    |-- Neg100toPos100
    |-- maxVals
    |-- minVals
    |-- over1000
    |-- underNeg1000
```

For example, the maximum value and time for sensor 1 would be accessed like this:
`results.sensor1.maxVals`

Function Specifications:

Function Name	Purpose	Input(s)	Output(s)
`logicalPract2`	1: import data from CSV file 2: perform logical operations 3: plot data	full path to csv file (string or char)	results structure (struct)

Useful functions: `fullfile`, `csvread`, `dlmread`, `isnan`, `fprintf`

Deliverable(s): `logicalPract2.m`

Graded Item	Point Value
Adequate comments and organization	5
Correct logical operations	10
Correct input/output	5
Total	20

Problem 6

For this problem, we work as a team to create a unit conversion function. I will write the main function, a time conversion function, and organize everyone's contributions into a single package. Make a selection from below:

- **Length** - `lengthConverter()`
  - Meter - (`Meter`)
  - Mile - (`Mile`)
  - Yard - (`Yard`)
  - Foot - (`Foot`)
  - Inch - (`Inch`)
  - Astronomical Unit - (`AU`)
  - Lightyear - (`Lightyear`)
- **Area** - `areaConverter()`
  - Square meter - (`SqMeter`)
  - Square mile - (`SqMile`)
  - Square yard - (`SqYard`)
  - Square foot - (`SqFoot`)
  - Square inch - (`SqInch`)
  - Acre - (`Acre`)
- **Volume** - `volumeConverter()`
  - Cubic meter - (`CuMeter`)
  - Cubic mile - (`CuMile`)
  - Cubic yard - (`CuYard`)
  - Cubic foot - (`CuFoot`)
  - Cubic inch - (`CuInch`)
  - Gallon - (`Gallon`)
- **Velocity** - `velocityConverter()`
  - Meter per hour - (`MeterPerHour`)
  - Meter per second - (`MeterPerSecond`)
  - Foot per second - (`FootPerSecond`)
  - Inch per second - (`InchPerSecond`)
  - Mile per hour - (`MilePerHour`)
- **Acceleration** - `accelerationConverter()`
  - Meter per second squared - (`MeterPerSecondSq`)
  - Foot per second squared - (`FootPerSecondSq`)
  - Inch per second squared - (`InchPerSecondSq`)
- **Temperature** - `tempConverter()`
  - Degrees Celcius - (`DegC`)
  - Degrees Farenheight - (`DegF`)
  - Kelvin - (`Kelvin`)
  - Rankine - (`Rankine`)
- **Energy** - `energyConverter()`
  - Joule - (`Joule`)
  - Foot Pound - (`ftlb`)
  - Inch Pound - (`inlb`)
  - Calorie - (`Calorie`)
  - Watt-Hour - (`WattHour`)
  - British Thermal Unit - (`btu`)
- **Power** - `powerConverter()`
  - Watt - (`Watt`)
  - Horsepower - (`horsepower`)
  - BTU per hour - (`BTUperHour`)

    - BTU per minute - (`BTUperMinute`)
    - BTU per second - (`BTUperSecond`)
- **Mass** - `massConverter()`
    - Gram - (`gram`)
    - Slug - (`slug`)
- **Force** - `forceConverter()`
    - Newton - (`Newton`)
    - Pound - (`Pound`)
- **Mass Density** - `mdensityConverter()`
    - Gram per cubic meter - (`GramPerCubicMeter`)
    - Slug per cubic foot - (`SlugPerCubicFoot`)
- **Weight Density** - `wdensityConverter()`
    - Newton per cubic meter - (`NewtonPerCubicMeter`)
    - Pound per cubic foot - (`PoundPerCubicFoot`)
- **Pressure** - `pressureConverter()`
    - Pascal - (`Pascal`)
    - Pound per square inch - (`psi`)
    - Pound per square foot - (`psf`)
    - Kip per square inch - (`ksi`)
    - Kip per square foot - (`ksf`)
    - Atmosphere - (`atm`)
    - Inches of mercury - (`inmg`)
- **Volume Flow** - `volflowConverter()`
    - Cubic foobisk per second - (`CuMeterPerSecond`)
    - Cubic meter per minute - (`CuMeterPerMinute`)
    - Cubic meter per hour - (`CuMeterPerHour`)
    - Cubic foot per second - (`CuFootPerSecond`)
    - Cubic inch per second - (`CuInchPerSecond`)
    - Gallon per second - (`GallonPerSecond`)
    - Gallon per minute - (`GallonPerMinute`)
- **Mass Flow** - `massflowConverter()`
    - Gram per second - (`GramPerSecond`)
    - Gram per minute - (`GramPerMinute`)
    - Gram per hour - (`GramPerHour`)
    - Gram per day - (`GramPerDay`)
    - Slug per second - (`SlugPerSecond`)
    - Slug per minute - (`SlugPerSecond`)
- **Angle** - `angleConverter()`
    - Degree - (`Degree`)
    - Radian - (`Radian`)
    - Gradian - (`Gradian`)
    - Quadrant - (`Quadrant`)
    - Angular Mil - (`Mil`)
- Electric Charge - `chargeConverter()`
    - Coulomb - (`Coulomb`)
    - Franklin - (`Franklin`)
    - Franklin - (`Franklin`)
- Electric Current - `currentConverter()`
    - Ampere - (`Amp`)
    - Franklin per second - (`FranklinPerSecond`)
    - Gilbert - (`Gilbert`)

Your set of functions should convert between all of the units listed under your selection. I have already accounted for the most common prefixes (Giga, Mega, Kilo, ect), so there's no need to include them in your functions. Follow the `timeConverter.m`, `convertTest.m`, and `notFound.m` files for guidance.

Useful functions: `string`, `fprintf`, `isa`, `strcmp`

Function Specifications:

Function Name	Purpose	Input(s)	Output(s)
select from above	1: validate units 2: convert units	1: value(s) to convert 2: from unit (string) 3: to unit (string)	converted value(s)

Deliverable(s): Your conversion function file

Graded Item	Point Value
Adequate comments and organization	10
Correct input/output	5
Correct conversions	5
Total	20