

# Week 7 Homework Assignments: Advanced Data Types - Structures, Cell Arrays, and Tables

## Global Requirements

- All deliverables shall be added, committed, and pushed to your **Week7** folder in your repository.
  - Include your name and the names of anyone who assisted you in the following format:  

```
% Student: Firstname Lastname  
% Assisted by: Firstname Lastname, etc.
```
  - Ensure your **scripts** (**Not functions**) include the following to clear the workspace and command window:  

```
clc;  
clear;  
close all;
```
  - Provide comments explaining each part of your code.
  - Use appropriate data types (structures, cell arrays, tables) as specified.
  - Ensure your programs handle input and output properly, and test them with different scenarios.
  - Any data files required are provided. If not, instructions are given to create them.
  - Outputs should resemble the examples shown; however, they do not need to be exact, as long as sufficient data is presented.
- 

## 1: Component Database Management

### Task

Create a MATLAB program to manage a database of engineering components using **structures** and **tables**. The database will store information about various components, such as their ID, name, dimensions, material, and weight.

### Function 1: addComponent

This function will simply add a struct object to an existing struct array and return the new struct array.

#### Requirements:

- **File Name:** addComponent.m
- **Inputs:**
  - structArray: Existing array of components (structure array).
  - componentStruct: A structure representing a new component.
- **Outputs:**
  - structArray: Updated structure array with the new component added.

### Function 2: struct2Table

This function will be used to convert a structure array into a MATLAB table.

#### Requirements:

- **File Name:** struct2Table.m
- **Inputs:**
  - structArray: Structure array of components.

- **Outputs:**
  - `table`: MATLAB table containing the component data.

### Instructions:

Write a MATLAB function named `struct2Table` that takes a struct array and converts it to a MATLAB table and returns the table. The table should have columns for `ID`, `Name`, `Length`, `Width`, `Height`, `Material`, and `Weight`.

**Script:** `componentDatabase.m`

### Requirements / Instructions:

1. **Define a Structure for Components:**
  - Each component should be represented as a structure with the following fields:
    - `ID` (integer)
    - `Name` (string)
    - `Dimensions` (structure with fields `Length`, `Width`, `Height` in meters)
    - `Material` (string)
    - `Weight` (double, in kilograms)
2. **Create an Empty Structure Array:**
  - Initialize an empty structure array to hold multiple components.
  - Remember: You can use the `struct` function.
3. **Populate the Database:**
  - Use the `addComponent` function to add at least **three** components to the database.
  - Save the table to `componentsDatabase.csv`.
  - Load the table back from `componentsDatabase.csv`.
  - Display the contents of the database in a **readable** format.
    - Note that you should not just print the table object, use output commands to print a readable format with units.

### Example Output

After running your script, the command window should display the components:

ID	Name	Length (m)	Width (m)	Height (m)	Material	Weight (kg)
1	Beam	2.5	0.3	0.5	Steel	150.0
2	Column	3.0	0.4	0.4	Concrete	200.0
3	Bolt	0.1	0.02	0.02	Alloy	0.5

### Testing

You can test your functionality running the following test files:

- `testAddComponent`
- `testStruct2Table`
- `testComponentDatabase`

### Deliverables

1. Submit the script `componentDatabase.m`.
2. Include comments explaining each part of your code.
3. Submit the functions `addComponent.m` and `struct2Table.m`.

## 2: Recipe Manager

### Task

Create a MATLAB program to manage recipes and ingredients using **cell arrays** and **tables**. The program will allow you to store recipes with varying numbers of ingredients and manage an inventory of ingredients.

### Function 1: `updateInventory`

Write a function to update inventory when a recipe is used. Subtract the quantities of the ingredients in the recipe from the inventory. If an ingredient is not available or insufficient, display a warning message.

#### Requirements:

- **File Name:** `updateInventory.m`
- **Inputs:**
  - `inventoryTable`: Table containing the current inventory.
  - `recipeCell`: Cell array representing a recipe.
- **Outputs:**
  - `inventoryTable`: Updated inventory table.

### Script: `recipeManager.m`

Write a MATLAB script to create, add, and “use” recipes.

2. **Create a Cell Array of Recipes:**
  - Store at least **two** recipes in a cell array.
3. **Create an Ingredient Inventory Table:**
  - Define a table with columns **Name**, **Quantity**, and **Unit** to represent the inventory.
  - Initialize the table with some ingredients and their quantities.
4. **Save and Load Inventory:**
  - Save the inventory table to a CSV file `ingredientInventory.csv` using `writetable`.
  - Provide functionality to load the inventory from the CSV file.

#### Requirements / Instructions:

- Create a script `recipeManager.m` that:
  - Initializes the recipes and inventory with at least **two** recipes.
    - \* Each recipe should be represented as a cell array containing:
      - Recipe name (string)
      - Cell array of ingredients (each ingredient is a structure with fields **Name**, **Quantity**, **Unit**)
  - Stores at least **two** recipes in a cell array.
  - Displays the recipes in a readable format.
  - Updates the inventory based on a selected recipe.
  - Saves and loads the inventory from `ingredientInventory.csv`.

### Example Output

Recipes:

1. Pancakes
  - Flour: 200 grams
  - Eggs: 2 each
  - Milk: 300 ml
2. Omelette
  - Eggs: 3 each
  - Cheese: 50 grams
  - Salt: 1 teaspoon

Updating inventory for Pancakes...  
Inventory updated successfully.

Current Inventory:

Name	Quantity	Unit
-----	-----	-----
Flour	800	grams
Eggs	5	each
Milk	700	ml
Cheese	100	grams
Salt	50	grams

## Testing

You can test your functionality running the following test files:

- `testUpdateInventory`
- `testRecipeManager`

## Deliverables

1. Submit the script `recipeManager.m`.
  2. Include comments explaining your code and any assumptions made.
  3. Submit the function `updateInventory.m`.
- 

## 3: Particle Simulation Data Analysis

### Task

Simulate data for particles moving in space and manage the data using **structures**, **cell arrays**, and **tables**. Analyze the data to calculate average velocities and plot particle trajectories.

### Function 1: `simulateParticleMotion`

#### Requirements:

- **File Name:** `simulateParticleMotion.m`
- **Inputs:**
  - `numParticles`: Number of particles to simulate (integer).
  - `numTimeSteps`: Number of time steps in the simulation (integer).
- **Outputs:**
  - `particles`: Structure array containing simulated particle data.

#### Instructions:

2. **Write the `simulateParticleMotion` Function:**
  - The function should generate random positions over time for a given number of particles.
  - Store the particles in a structure array.

### Function 2: `calculateAverageVelocity`

#### Requirements:

- **File Name:** `calculateAverageVelocity.m`
- **Inputs:**
  - `particle`: Structure containing particle data.

- **Outputs:**
  - `avgVelocity`: Average velocity of the particle over the simulation time (scalar).

#### Instructions:

##### 1. Write the `calculateAverageVelocity` Function:

- Calculate the average velocity for a particle using the distance formula and time differences.
- Use the positions and times stored in the particle structure.

#### Script: `particleSimulation.m`

Create a MATLAB script named `particleSimulation.m` that uses the `simulateParticleMotion`, and `calculateAverageVelocity` functions to product a set of random data to be displayed and plotted.

#### Requirements / Instructions:

##### 1. Define a Particle Structure:

- Each particle should be represented as a structure with the following fields:
  - `ID` (integer)
  - `Position` (Nx3 matrix for N time steps, columns for X, Y, Z coordinates)
  - `Time` (Nx1 vector of time points)

##### 2. Populate the data set

- Simulate particle data using `simulateParticleMotion`.
- Save the data to `particleData.mat`.
- Load the data from `particleData.mat`.
- Calculates the average velocity for each particle using `calculateAverageVelocity`.
- Store the results in a table.
- Plot the 3D trajectories of all particles on the same graph using `plot3`.

#### Example Output

After running your script, the command window displays:

```
Particle Average Velocities:
ID    AverageVelocity (units/s)
1      7.3204
2      6.368
3      6.2247
```

#### Example Plot

#### Testing

You can test your functionality running the following test files:

- `testSimulateParticleMotion`
- `testCalculateAverageVelocity`
- `testParticleSimulation`

#### Deliverables

1. Submit the script `particleSimulation.m`.
2. Include comments explaining each part of your code.
3. Submit the functions `simulateParticleMotion.m`, and `calculateAverageVelocity.m`.

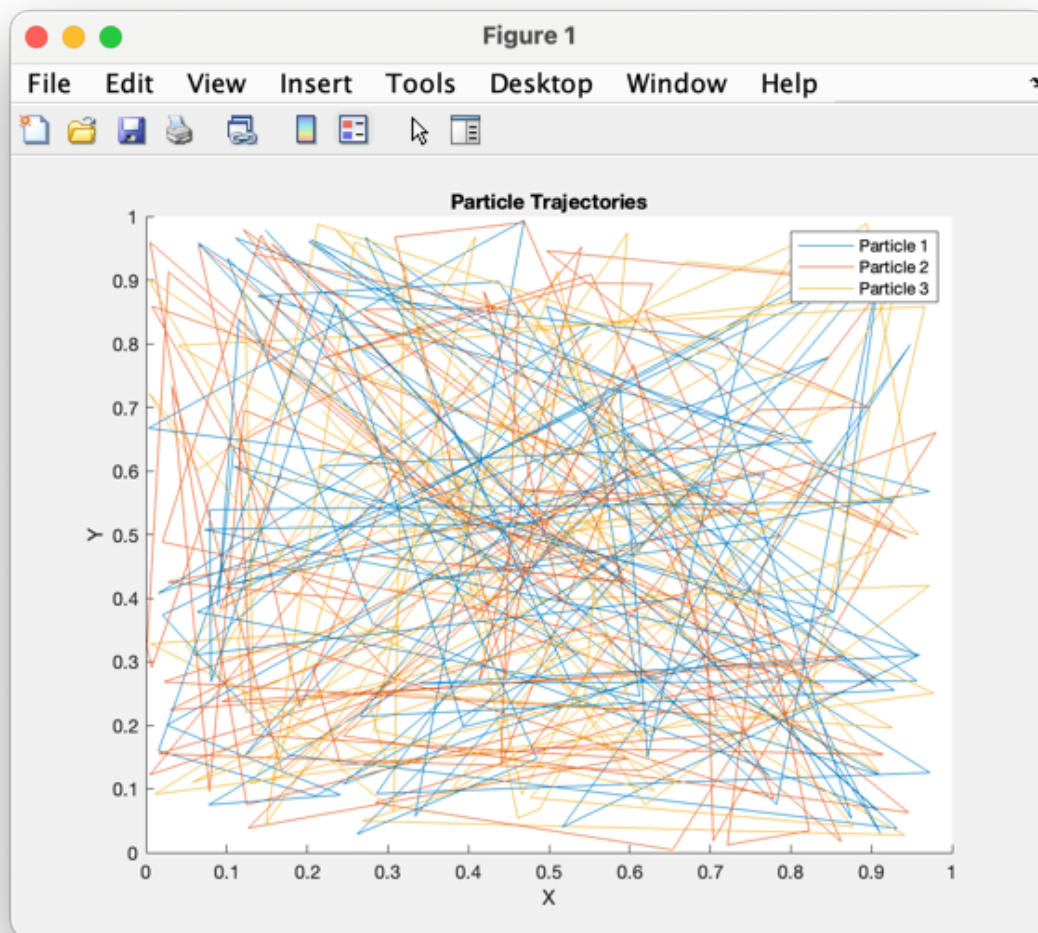


Figure 1: img.png

## Definition of Done

Your Week7 folder shall contain **at minimum** the following files:

- `addComponent.m`
- `componentDatabase.m`
- `struct2Table.m`
- `updateInventory.m`
- `recipeManager.m`
- `simulateParticleMotion.m`
- `calculateAverageVelocity.m`
- `particleSimulation.m`

Ensure that each script and function is well-documented and follows good coding practices.

## Additional Instructions

- **Testing:**
  - While explicit test scripts are not provided, you are encouraged to test each function individually.
  - Verify that your functions handle edge cases and invalid inputs gracefully.
  - Use `assert` statements or conditional checks where appropriate.
- **Data Files:**
  - If any data files are required for your scripts (e.g., `particleData.mat`), ensure they are properly generated and saved within your scripts.
- **Plotting:**
  - For assignments involving plots, make sure your plots are properly labeled with titles, axis labels, and legends where appropriate.
  - Use formatting to enhance readability (e.g., grid lines, markers, line styles).