Fondamenti di Informatica

Prof. Luigi Maria Ingrosso
Luigi.Maria.Ingrosso@uniroma2.it
Agenda

- Cell array
Objectives

Cell Arrays

- Creating Cell Arrays
- Accessing Cell Arrays
- Using Cell Arrays
- Processing Cell Arrays
Heterogeneous collections permit objects of different data types to be grouped in a collection.

- They allow data abstraction to apply to a much broader range of content.
- However, the fact that the contents of these collections may be of any data type severely restricts the operations that can be performed on the collections as a whole.
- Whereas a significant number of arithmetic and logical operations can be performed on whole number arrays, algorithms that process heterogeneous collections must deal with the data contents one item at a time.
Cell arrays, as the name suggests, have the general form of arrays and can be indexed numerically as arrays.

However, each element of a cell array should be considered as a container in which one data object of any class can be stored.

They can be treated as arrays of containers for the purpose of concatenation and slicing.

However, if you wish to access or modify the contents of the containers, the cells must be accessed individually.
Creating Cell Arrays

- By assigning values individually to a variable indexed with braces:
  ```
  >> A{1} = 42
  A = [42]
  ```

- By assigning containers individually to a variable indexed with brackets:
  ```
  >> B[1] = {[4 6]};
  B = [1x2 double]
  ```

- By concatenating cell contents using braces `{ . . . }`:
  ```
  >> C = {3, [1,2,3], 'abcde'}
  C = [3] [1x3 double] 'abcde'
  ```

- By concatenating cell containers:
  ```
  >> D = [A B C {'xyz'}]
  D = [42] [1x2 double] [3] [1x3 double] 'abcde' 'xyz'
  ```
Accessing Cell Arrays

Continuing the previous examples, we have the following:

```matlab
>> E = D(2) % parentheses - a container
E = [4 6]
```

However, braces are used to access the contents of the containers as follows:

```matlab
>> D{2} % braces - the contents
ans = 4 6
```

If the right-hand side of an assignment statement results in multiple cell arrays, the assignment must be to the same number of variables.

The built-in MATLAB function `deal(...)` is used to make these allocations.
Using Cell Arrays

- Containing lists of possible values for switch/case statements
- Substituting for parameter lists in function calls

  For example, suppose you have a function `largest(a, b, c)` that consumes three variables and produces the largest of the three values provided. It can be used in the following styles:

  ```
  A = 4;
  B = 6;
  C = 5;
  N = largest(A, B, C)
  params = { 4, 6, 5 };
  N = largest(params{1:3})
  ```
Processing Cell Arrays

- The template for processing cell arrays is:
  
  ```
  <initialize result>
  for <index specification>
    <extract an element>
    <check the element accordingly>
    <process the element accordingly>
  end
  <finalize result>
  ```

- Checking the class of the element can be achieved in one of two ways:
  - The function `class(item)` returns a string specifying the item type that can be used in a switch statement
  - Individual test functions can be used in an if... elseif construct;
    - examples of the individual test functions are `isa(item, 'class')`,
    - `iscell(...)`, `ischar(...)`, `islogical(...)`, `isnumeric(...)`, and
    - `isstruct(...)"
% Listing 7-2 - Cell array processing example

function ans = totalNums(ca)
% count the numbers in a cell array
    ans = 0 ;
    for in = 1 :length(ca)
        item = ca{in} ;    % extract the item
        if isnumeric(item) % check if a vector
            ans = ans + prod(size(item));
        end
    end
Cell arrays are vectors of containers; their elements can be manipulated either as vectors of containers, or individually by inserting or extracting the contents of the container using braces in place of parentheses.