Structures; More on File Input / Output

Pedro Barahona
DI/FCT/UNL
Métodos Computacionais
1st Semestre 2018/2019

Structures

- Arrays (vectors, matrices, or multi-arrays) are very convenient structures to organize numerical information, since each "cell" should contain a number.
- In many cases, information is not only numeric, e.g. it includes text (we do not consider other types of information, such as visual or sound or video).
- Moreover, the data is organized in a mixed way, combining text and numerical information in "records".
- Take for example the information about the employees of a certain company. For each employee we may consider:
  - **id** – integer, representing a unique identification number in the company
  - **date** – text, in format YYYY-MM-DD, representing the date of employment
  - **name** – text, with the name of the employee
  - **salary** – real number, representing the monthly salary of the employee

Structures

- Although complex information is better maintained in a database, in simple applications, this heterogeneous information may be organized in a record, or in the MATLAB notation (borrowed from C) in a **structure**.

- A structure is similar to a vector with two main differences.
  - Different positions may contains different types of data; and
  - Positions are identified by field names, following the name after a ".".

Example:

- An employee, **emp**, may be represented by a structure, where fields have values:
  - id – 98
  - name – Rui Silva
  - date – 2011-10-23
  - salary – 1654.3

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3

Structures

- A structure may be initialised by simply assigning values of its fields.
  - Different positions may contains different types of data; and
  - Positions are identified by field names, following the name after a dot (“.”).

```
>> emp.name = "Rui Silva";
>> emp.id = 98;
>> emp.date = "2011-10-23";
>> emp.salary = 1654.30;
>> emp
emp =
  scalar structure containing the fields:
    id =  35
    name = Rui Silva
    date = 2011-10-23
    salary =  1654.3
```

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3

Structures

- Alternatively, a structure may be initialised by a simple instruction, identifying the fields and their values, as shown below;

```
>> emp2 = struct("id", 98, name, "Rui Silva",
          date, "2011-10-23", salary, 1654.30)
emp2 =
  scalar structure containing the fields:
    id =  35
    name = Rui Silva
    date = 2011-10-23
    salary =  1654.3
>>
```

- Note: In this notation, the names of the fields should be given as strings, i.e. delimited by quotation marks (") or by apostrophes (').

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3

Structures

- Once defined a structure, the data of the fields may be accessed “individually”, by using the “dot notation”.

```
>> emp = struct("id", 35, name, "Rui Silva",
          date, "2011-10-23", salary, 1654.30);
>> emp.name
ans = Rui Silva
>> double = emp.salary * 2
double = 3308.6
>> next_emp_id = emp.id + 1
next_emp_id = 36
>> next_day = emp.date + 1
Next_day = 51   49   50   50   46   50   49   46   51   52
```

- **Note**: the types of the fields are different (numbers and strings and are dealt with differently.

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3

Structure Arrays

- Most applications require to maintain several records in a “table”, i.e. a set of records of the same type, organised as a “vector”, i.e. each with its index.
- Complex applications, requiring several tables, should of course be supported in databases. Nevertheless, for simple applications, tables can be directly modelled by moe general programming languages.
- In MATLAB, this table data structure is available as a **structure array**. A table of employees may contain several records, each with the information of an employee.

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

- MATLAB is very “permissive” regarding these structure arrays. It is good practice that all records have the same fields, and that these are defined before the structure array is filled with information.

Structure Arrays

- Structure arrays share many properties of “usual” arrays, and in particular
  - Their size is available through function **length**;
    - In fact a structured array has 2 dimensions, but is similar to a row vector, whose elements are structures
  - Their ranges always start in index 1;
  - Structure sub-arrays may be obtained by a projection operation, and are composed of the structures whose indices are selected
    - Projection on the fields is not directly available, and must be programmed if needed.
  - As any vector, elements may be deleted by assigning them the empty ([]) value.
    - The remaining elements are shifted “downwards” so that no “holes” are created in the structured array

Structure Arrays

- Example: **Creating** the structure array

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

```
>> emp = struct("id", 98, "name", "Rui Silva",
          "date", "2011-10-23", "salary", 1654.30);
>> emps(1) = emp;
>> emps(2) = struct("id", 56, "name", "Maria Santos",
          "date", "2008-12-18", "salary", 1742.4);
>> emps(3) = struct("id", 43, "name", "Carlos Dias",
          "date", "2003-04-12", "salary", 2017.6);
>> emps(4) = struct("id", 12, "name", "Isabel Rio",
          "date", "1987-09-05", "salary", 2916.8) ;
```

Structure Arrays

- Example: **Displaying** the structure array

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

- Notice that the values are usually not displayed in the terminal, only the composition of the structure array is displayed.

```
>> emps
emps = 1x4 struct array containing the fields:
    id    name    date    salary
>> emps(3)
ans = scalar structure containing the fields:
   id =  43
   name = Carlos Dias
   date = 2003-04-12
   salary =  2017.6
```

Structure Arrays

- Example: Selecting from the structure array

id	name	date	salary (€)
98	Rui Silva	2015-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

```
>> emps(2:3)
ans =  1x2 struct array containing the fields:
    id    name    date    salary
>> emps(2).salary
ans = 1724
>> emps.salary
ans =  1654.3
ans =  1742.4
ans =  2017.6
ans =  2916.8
```

- **Note:** Beware this last interaction is *similar* to a loop, but does not yield an array!

Structure Arrays

- Example: **Deleting** an element from the array

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
~~43~~	~~Carlos Dias~~	~~2003-04-12~~	~~2017.6~~
12	Isabel Rio	1987-09-05	2916.8

```
>> emps(3) = []
emps =  1x3 struct array containing the fields:
    id    name    date    salary
>> emps(3)
ans =  scalar structure containing the fields:
   id =  12
   name = Isabel Rio
   date = 1987-09-05
   salary =  2916.8
```

Structure Arrays

- Example: **Clearing** the structure array

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

```
>> clear emps
>> emps

>> emps(1)
error: invalid use of script
    /Users/pedrobarahona/Desktop/octave-mc/emps.m
    in index expression
```

Stored Structure Arrays

- Typically, the information contained in a structure array is stored in a file, given the large volume of data it handles.
- Of course, processing this data directly from the file is very inneficient since
  - Access to files is sequential.
    - Once read say element I, reading element i-1 require to read the file again.
  - Access to the file is slow:
    - Although disks nowadays are much faster than some years ago, namely the SSD disks that are fully electronic and have no mechanical components, its access is typically at least one order of magnitude slower than that to RAM memory, that have better channels to the CPU.
- Hence, processing data in a table, is done in 3 steps:
  1. Reading the table from a file to a structure array
  2. Process the data, including adding or deleting elements of the structure array
  3. Write the new table into a file

Structure Arrays – Writing to Text Files

- Writing a structure array is straightforward. Typically, every structure is written in a single line.
- Moreover, since the data is stored as text the **fprintf** command can be used, taking into account the type of the data.
- It is important to take care of the characters that are used as delimiters between the different fields of the structure so that, when the structures are read the bounds of the different fields are known.
- Typically, a character is chosen that do not appear in the data strings of the different (text) fields of the structure.
  - For example, in CSV files, data is separated by commas, since commas do not appear in numbers …
    - Except if one uses a system like the Portuguese, where commas are used as decimal separators.
- Good candidates for field separators are the semi-colon (;) or, if the text fields are more general the horizontal tab(\t), or the vertical bar (|).

Structure Arrays – Writing to Text Files

- We can exemplify this writing with the previous structure array.

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

```
function print_employees(emps, filename)
% this function prints a structure array with fields ...
% into a text file with the given name
  fid = fopen(filename, "w")
  for i = 1:length(emps)
     fprintf(fid,"%i|%s|%s|%f\n", emps(i).id, emps(i).name,
                c = emps(i).date, d = emps(i).salary)
  end
 fclose(fid);
end
```

- Note the **newline** char (\n) given in the print template.

Structure Arrays – Writing to Text Files

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

- For the above structure, this function should produce a file with the text: exemplify this writing with the previous structure array.

```
>> print_employees(emps, "employees.txt")
>>
```

employees.txt

```
98|Rui Silva|2015-10-23|1654.3\n
56|Maria Santos|2008-12-18|1742.4\n
43|Carlos Dias|2003-04-12|2017.6\n
12|sabel Rio|1987-09-05|2916.8\n
```

Structure Arrays – Reading from Text Files

- There are many different ways/instruction to read text files taking into account its format (e.g. csv files).
- Here we will exemplify a line based method:
  - The file is opened;
  - A counter is initialised at 0;
  - While there are lines to read
    - A line is read
    - The counter is updated to take into account a new structure
    - The positions of the separator characters are identified
    - The fields between identifiers are assigned to the new structure
  - Finally, the file is closed

Reading a Structure Array

- The algorithm discussed is now implemented as follows.

```
function emps = read_employees(filename)
% this function reads a structure with fields ...
% from a text file previously opened  with channel fid
  fid = fopen(filename, "r");
  i = 0;
  while  !feof(fid)                % While there are lines to read
     i = i + 1;                    % The counter is updated
     line = fgetl(fid);            % A line is read
     seps = strfind(line, "|"); % The separators are found
     emps(i).id = str2num(line(1:seps(1)-1));
     emps(i).name = line(seps(1)+1:seps(2)-1);
     emps(i).date = line(seps(2)+1:seps(3)-1);
     emps(i).salary = str2num(line(seps(3)+1:end-1));
  end
  fclose(fid);
end
```

Reading a Structure Array

- The following interaction tests the functioning of this function

employees.txt

```
98|Rui Silva|2015-10-23|1654.3\n
56|Maria Santos|2008-12-18|1742.4\n
43|Carlos Dias|2003-04-12|2017.6\n
12|sabel Rio|1987-09-05|2916.8\n
```

```
>> emps = read_employees("employees.txt")
emps = 1x4 struct array containing the fields:
    id    name    date    salary
>> emps(3)
ans = scalar structure containing the fields:
   id =  43
   name = Carlos Dias
   date = 2003-04-12
   salary =  2017.6
```

Structure Arrays – Processing

- Once the structure array is read to memory, we can process it, namely finding information contained in it.

Three examples:

1. Find the average of the salaries of the employees;
2. Find the oldest employee (according to the dates)
3. Find the name of an employee with a given id

- In the first two cases, the structure array must be completely sweeped, whereas in the last case, the sweeping can stop once the employee is found.

Structure Arrays – Processing

Example 1: Find the average of the salaries of the employees

```
function av = average_salary(emps)
```

- This is very similar to obtain the average of an array,
- The average is obtained by summing all the salaries and dividing it by its number

```
function av = average_salary(emps)
% this function returns the average of the field salary
% from all structures of the array emps
  s = 0;
  n = length(emps);
  for i = 1:n
     s = s + emps(i).salary;
  end
  av = s / n;
end
```

id	name	date	salary (€)
98	Rui Silva	2015-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

```
>> avs = average_salary(emps)
avs = 2082.8
```

Structure Arrays – Processing

Example 2: Find the oldest employee (according to the dates);

```
function [id, name] = oldest(emps)
```

- Again this function is very similar to obtain the minimum of an array
- Once the index of the structure identified, the corresponding name and id are returned
- **Note 1:** Finding the minimum of a date in format “yyyy-mm-dd” can be done using an auxiliary function that compares two dates;
- **Note 2:** The result of this function is composed of two distinct fields, one is a number and the other a string.
  - To pack the two items in the returned result, they are shown between sqaure brackets, although the result is not an array!
  - In fact to obtain both results they must be declared in the function call.

Structure Arrays – Processing

Example 2: Find the oldest employee (according to the dates);

```
function [id, name] = oldest_emp(emps)
% this function returns the id and name of the oldest employee
% from all structures of the array emps
% the structures have fields date, id and name
  oldest_date = "3000-01-01";
  n = length(emps);
  for i = 1:n
     if before(emps(i).date, oldest_date)
        oldest_date = emps(i).date;
        oldest_idx = i;
     end
  end
  id = emps(oldest_idx).id;
  name = emps(oldest_idx).name;
end
```

Structure Arrays – Processing

Example 2a: Compare two dates in format yyyy-mm-dd

- The dates are compared by year, month and day, after turning these into numbers.

```
function bool = before(date1, date2)
% returns true if date1 is before date2, false otherwise
% both dates are in the format yyyy-mm-dd
   bool = false;
   y1 = str2num(date1(1:4)); y2 = str2num(date2(1:4));
   if y1 < y2
      bool = true; return;
   else y1 == y2
      m1 = str2num(date1(6:7)); m2 = str2num(date2(6:7));
      if m1 < m2
         bool = true; return;
      else m1 == m2
         d1 = str2num(date1(9:12)); d2 = str2num(date2(9:12))
         if d1 < d2
            bool = true;
         end
      end
   end
end
```

Structure Arrays – Processing

Example 2a: The auxiliary function should be tested before used (unitary tests)

```
>> before("2001-12-04", "2001-12-04")
ans = 0
>> before("2001-12-04", "2001-12-05")
ans = 1
>> before("2001-12-04", "2002-12-04")
ans = 1
>> before("2000-11-04", "2002-01-04")
ans = 1
>> before("2001-12-04", "2001-01-03")
ans = 0
>> before("2001-12-04", "2001-11-04")
ans = 0
>> before("2001-12-04", "2000-12-04")
ans = 0
```

- Note: In general, all possible conditional paths in a function should be tested

Structure Arrays – Processing

Example 2: The main function should also be tested

id	name	date	salary (€)
98	Rui Silva	2015-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

- Note the way in which the results are obtained (or ommited)

```
>> oldest_emp(emps)
ans =  12
>> ii = oldest_emp(emps)
ii =  12
>> [ii,nn] = oldest_emp(emps)
ii =  12
nn = Isabel Rio
>>
```

Structure Arrays – Processing

Example 3: Find the name of an employee with a given id

```
function [found, name] = emp_name(id, emps)
```

- This is very similar to finding an element in an array, but with a few differences
  - Once the element is found, its index is used to obtain the name;
- Again, the expected result is a tuple, composed of two elements:
  - The first is a Boolean indicating whether the employee exist
  - The second is a string with the name of the employee (or empty)

Structure Arrays – Processing

Example 3: Find the name of an employee with a given id

```
function [found, name] = emp_name(id, emps)
% Given structure array emps with fields id and name
% this function returns the name of the structure
% with the given id, if any
   i = 0;
   n = length(emps);
   found = false;
   name = "";
   while i < n && !found
      i = i+1;
      if emps(i).id == id
         found = true;
         name = emps(i).name;
         return;
      end
   end
end
```

Structure Arrays – Processing

Example 3: This function should also be tested

id	name	date	salary (€)
98	Rui Silva	2015-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

```
>> emp_name(100, emps)
ans =  0
>> [bb,nn] = emp_name(100, emps)
bb =  0
bb =
>> [bb,nn] = emp_name(98, emps)
bb =  1
nn = Rui Silva
>> [bb,nn] = emp_name(12, emps)
bb =  1
nn = Isabel Rio
>>
```