# Exercise #1 (Data read and manipulation)

Assume a data set in a file in CSV format, with semicolons (“;”) as field separators.

Each entry is made up of fields (from left to right)

* ID (sequence of at most 32 characters)
* Name (sequence of at most 64 characters)
* Age (natural number)
* Data of at most 256 bytes, represented in hexadecimal notation (i.e. at most 512 characters)

The ID is unique in the data set, the Name not.

Solve/answer the following tasks/questions:

1. Task 1: Read the data set into memory, representing the data in an array.  
    Your implementation must not use pointers to data items other than C strings and the   
    array(s) itself.  
    Based on command input to the program perform the following actions as efficient as   
    possible:
   1. Output the data in ascending order sorted by the ID (ID compare is string compare)
   2. Output the data in descending order sorted by ID
   3. Output the data in ascending order sorted by Name
   4. Allow to search for an ID (including output of matched data)
   5. Allow to search for a Name (including output of matched data)
2. Question 1: How can the sorting be utilized to implement the search efficiently?
3. Question 2: How can re-sorting the whole data array be avoided if 1.a, 1.b and 1.c are issued repeatedly?

## Question 1: How can the sorting be utilized to implement the search efficiently?

If we want to use a sorted array as basis for looking up and element in an array, we can use the binary search method. The search starts with comparing the target element with the middle element of the array.

If value matches then the position of the element is returned. In case the target element is less than the middle element (considering the array follows an ascending order) of the array then the second half of the array is discarded and the search continues by dividing the first half.

The process is the same when the target element is greater than the middle element, only, in this case, the first half of the array is discarded before continuing with the search. The iteration repeats until a match for the target element is found. For instance, a loop can be the following:

 first = 0;

**last** = n - 1;

   middle = (first+**last**)/2;

**while** (first <= **last**) {

**if** (array[middle] < search)

         first = middle + 1;

**else** **if** (array[middle] == search) {

**break**;

      }

**else**

**last** = middle - 1;

      middle = (first + **last**)/2;

   }

## Question 2: How can re-sorting the whole data array be avoided if 1.a, 1.b and 1.c are issued repeatedly?

Option 1:

Instead of practically reshuffling the data after a sorting command, the output of a sorting algorithm can be a list of element indexes corresponding to the original data array indexes (one index for each row for all database fields). This means that the original entries position are not modified and remain constant and that the sorting output returns the list as sorted but in reality only its indexes are. For example:

|  |  |  |
| --- | --- | --- |
| Original List[index] | Sorting Output (Ascending) | Final Result |
| 5[0] | [1] | List[1]=1 |
| 1[1] | [3] | List[3]=2 |
| 7[2] | [0] | List[0]=5 |
| 2[3] | [2] | List[2]=7 |

Then after each unique first time sorting command we can create an array list of indexes corresponding to the sorted elements by type. For instance, a structure called Sort\_List\_Indexes where for each row are stored the indexes coming from the ascending or descending sorting algorithm for each fields. This list should be updated only when new data have been loaded and initialized.

In this way, if no new data are present, when applying a repeated sorting command, no actually sorting will be done since the index values are already stored somewhere and they must be only accessed or printed out.

Option 2:

To avoid useless repetition of the same command we could create a flags system (eg. SortFlag) that acts when the item is sorted in a particular order, for example SortFlag=0 when not sorted as default, SortFlag=1 when sorted ascending, SortFlag=2 when sorted descending. In this way, when we issue a sorting command there shall be first a check of the current state of the list. If the command will be repeated twice, nothing shall be done. If the command goes from 2 to 1 or to 1 to 2 then instead of issuing the loop sort command again we can just flip order of the list. In effect, the sorting will only acting when transitioning from 0 to 1 or to 0 to 2. In this way, the actually sorting algorithm will be only issued once.

If the sorting field will change, ie First sorting command by ID and Second sorting command by ID, the ID flag should be reset. To avoid the sorting being recalled, it will be better to use the flag system in combination with Option1.

Option 3:

# Exercise #3 (DO-178C development)

Imagine the box from Exercise #2 being used on an aircraft to implement a temperature monitoring and warning function. The safety assessment of the function is DAL C for the warning and DAL B for the error signaling. Reliability of temperature measurement data is influenced by EMI effects, which may invalidate data on a single channel from time to time. Reaction time limit from measurement to signaling imposed by system requirements is 10ms. Discrete signal outputs are wired independently from each other in the system. Fault on a single line may occur with a probability of 10-5/FH.

Answer/solve the following questions/tasks:

1. Question 1: Which artefacts do you need for a DO-178C compliant development for the function? What is input to the SW development, what is created by the SW development?
2. Task 1: Write a set of high-level SW requirements covering the function according to DO-178C.
3. Question 2: What effects have the DAL levels of the system on your SW design/ requirements?
4. Task 2: Provide a SW architecture that represents your SW implementation to the function.
5. Question 3: Functions with different DALs are mixed in the box. Does partitioning make sense here to separate the functions by DAL?

## Question 1: Which artefacts do you need for a DO-178C compliant development for the function? What is input to the SW development, what is created by the SW development?

System to Software Inputs:

1. The system description and hardware definition.
2. List of System Requirements Allocated to Software, e.g. functional requirements, performance requirements, safety-related requirements.
3. Design constraints and design methods, for example partitioning, redundancy or safety monitoring.
4. Software Levels, SW levels determination, failure conditions categories.
5. Safety Risk Assessment, a document that evaluates critical and non safety critical functions and the required level of testing for the system.

Software Inputs:

1. Software Development Plan
2. Software Development Environment
3. Software Test Environment
4. Software Development Standards

SW development process expected output:

1. High-level requirements
2. Derived high-level requirements
3. Software architecture (SA)
4. Low-level requirements
5. Source Code
6. Executable Object Code

## Question 2: What effects have the DAL levels of the system on your SW design/ requirements?

The system DAL level can have an indirect impact on SW design and requirements.

In facts, for the SW life cycle the major consequence is at SW Verification level where for:

* LEVEL C, it is required to demonstrate structural coverage, data coupling and statement coverage
* LEVEL B, must be added also the decision coverage.

This mean that to be verifiable and to give enough proofs of LEVEL B compliance (for example), the Software should be designed in a way that:

* Every statement in the requirements is present at design level.
* Every statement in the requirements must been invoked at least once
* Every point of entry and exit in the program must been invoked at least once
* Every decision in the program must reach all possible outcomes at least once

System safety level may also require that partitioning may be required for a particular item, as consequence particular partitioning requirements may be added, and SW implemented as consequence.

## Question 3: Functions with different DALs are mixed in the box. Does partitioning make sense here to separate the functions by DAL?

Partitioning is a technique for providing isolation between functionally independent software components to contain or isolate faults and potentially reduce the effort of the software verification process. Basically, there is a need to ensure that fault in one component does not propagate to the other component. DO-178B partitioning is something that is dictated by Systems Requirements and then It is not something that software design can determine on its own arbitrarily.

In our case, functions partitioning is not necessary since the two signals are coming into the Temperature Box from two separate ADCs through two separate connections. As consequence, the faulty behavior of one sensor or signals doesn’t impact on the other. Also the probability of a contemporary faults on both is extremely low, 10-10.

In general, if there is no function partitioning, DAL B + DAL C = DAL B, which mean that the SW takes the most critical safety level as objective.

## Task 1: Write a set of high-level SW requirements covering the function according to DO-178C.

## Inputs

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Source** | **Data Type** | **Units** | **Min Value** | **Max Value** |
| **PT100T\_1** | PT100 Temperature from sensor 1 | Hardwired Analogue Signals | Real | Degree C° | -50.0 | 150.0 |
| **PT100T\_2** | PT100 Temperature from sensor 2 | Hardwired Analogue Signals | Real | Degree C° | -50.0 | 150.0 |

## Outputs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Meaning of True** | **Meaning of False** |
| **PT100T\_WARN\_1** | Temperature Warning Flag for sensor 1 | Boolean | PT100 measurement higher than warning limit | PT100 measurement lower than warning limit |
| **PT100T\_WARN\_2** | Temperature Warning Flag for sensor 2 | Boolean | PT100 measurement higher than warning limit | PT100 measurement lower than warning limit |
| **PT100T\_ERR\_1** | Temperature Error Flag for sensor 1 | Boolean | PT100 measurement higher than error limit | PT100 measurement lower than error limit |
| **PT100T\_ERR\_2** | Temperature Error Flag for sensor 2 | Boolean | PT100 measurement higher than error limit | PT100 measurement lower than error limit |

*Note: could be also used an enumerated type of 2 dimensions (1,2) and three states (ERROR,WARNING,InRange)*

## Development Variables (DV) and Graphical Data

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Units** | **Initial Value** | **Min Value** | **Max Value** |
| **PT100T\_Warn** | Constant value defining temperature threshold for Warning flag for PT100T | Real | Degree C° | 40.0 | n/a | n/a |
| **PT100T\_Error** | Constant value defining temperature threshold for Error flag for PT100T | Real | Degree C° | 60.0 | n/a | n/a |

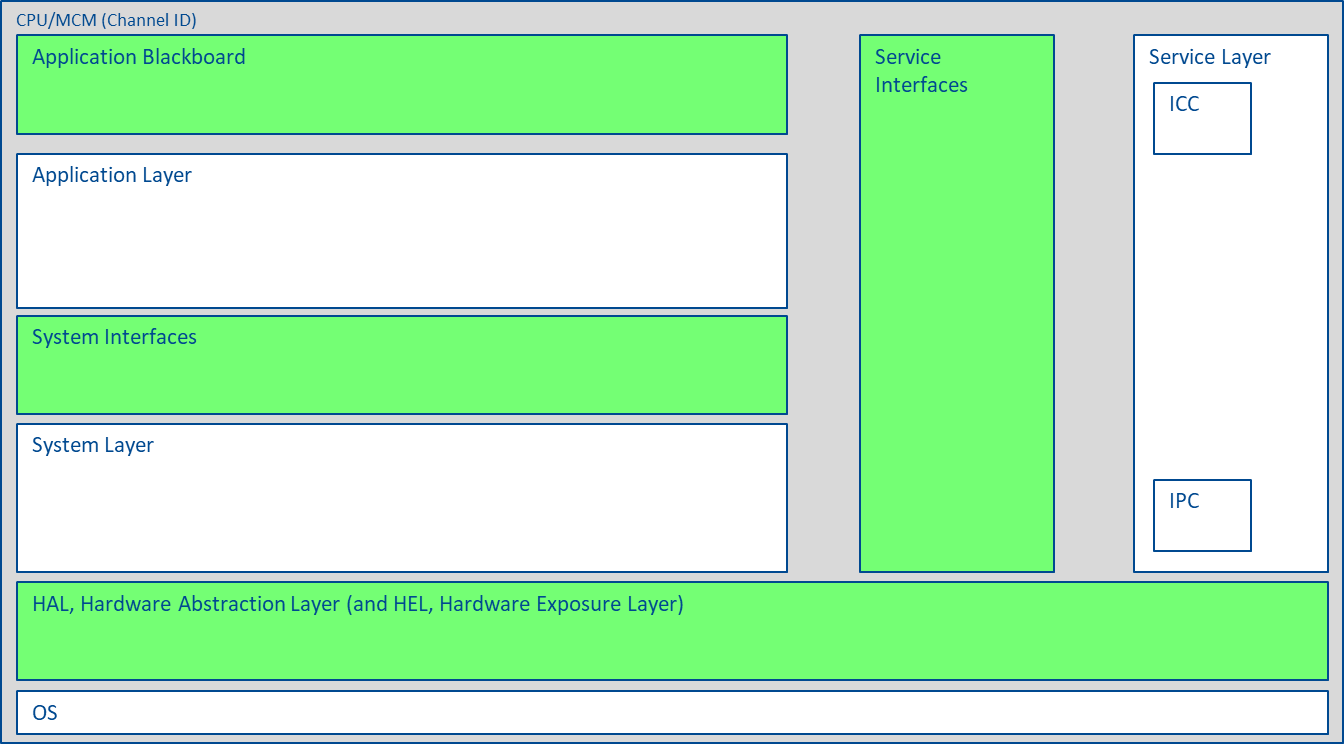
## Requirements

|  |  |
| --- | --- |
| **Req ID:** SRS\_TempBox\_01/1 | **Type:** Design Constraint |
| The Temperature Box shall run at a 10ms rate. | |
| **Rationale:** Reaction time limit from measurement to signalling imposed by system requirements. | |

|  |  |
| --- | --- |
| **Req ID:** SRS\_TempBox\_02/1 | **Type:** Functional |
| The Temperature Warning Flags for sensor 1 (**PT100T\_WARN\_1)** and sensor 2 (**PT100T\_WARN\_2**)shall be calculated as per table below:   |  |  |  |  | | --- | --- | --- | --- | | PT100T\_1 | PT100T\_2 | PT100T\_WARN\_1 | PT100T\_WARN\_2 | | > PT100T\_Warn | X | TRUE | X | | < PT100T\_Warn | X | FALSE | X | | X | > PT100T\_Warn | X | TRUE | | X | < PT100T\_Warn | X | FALSE | | |
| **Rationale:** The function of the box shall be to constantly measure the temperature and activate a WARN output when the temperature rises above 40°C. If temperature falls below the thresholds, warning flags shall be deactivated again. | |

|  |  |
| --- | --- |
| **Req ID:** SRS\_TempBox\_03/1 | **Type:** Functional |
| The Temperature Error Flags for sensor 1 (**PT100T\_ERR\_1)** and sensor 2 (**PT100T\_ERR\_2**)shall be calculated as per table below:   |  |  |  |  | | --- | --- | --- | --- | | PT100T\_1 | PT100T\_2 | PT100T\_ERR\_1 | PT100T\_ERR\_2 | | > PT100T\_Error | X | TRUE | X | | < PT100T\_Error | X | FALSE | X | | X | > PT100T\_Error | X | TRUE | | X | < PT100T\_Error | X | FALSE | | |
| **Rationale:** The function of the box shall be to constantly measure the temperature and activate a ERROR output when the temperature rises above 60°C. If temperature falls below the thresholds, error flags shall be deactivated again. | |

## Task 2: Provide a SW architecture that represents your SW implementation to the function



**PT100T\_1**

**PT100T\_2**

Temperature Box

**PT100T\_1**

**PT100T\_2**

**PT100T\_Error**

**PT100T\_Warning**

I\_Temperatures

IOS\_Temperatures