CE4703 Computer Software 3
ED5071 Computer Engineering Fundamentals

Assignment #1

Dr Reiner Dojen [1]

Due 11:00h on Thursday, 09.11.2023

[1] reiner.dojen@ul.ie

1 Overview

Your task is to develop a program that can create and anlyse arrays of integers in various ways. While developing the program, you must follow the principle of modular programming (I also strongly encourage to re-use code as much as possible). Also, for any non-trivial function, you must follow the 7 Steps of Systematic Program Construction. Furthermore, you must comment **all** your code for Doxygen.

All code must be developed as a Microsoft Visual Studio (VS) project using standard C.

You also need to construct a report (in plain text format - just add a text file named after you student ID to your VS project) that contains the following:

- A list of modules that make up your program.
- For each module, list what functions it contains. Also, provide a function prototype (i.e. a function declaration) for each function.
- Specification for each function.
- Pseudocode representation for each function. For simple functions, a single iteration is sufficient - for any non-trivial function pseudo-code representation provide (at least) two iterations of refinement. As discussed in the lecture, I recommend to also include your pseudo-code as "in-code" comments in your source files.

2 Modular Structure

Your program must implement the functions listed below in Section 2.1 Required Functions. Before you start implementing these functions, you must design a modular structure - that is, define the modules that will make up your program. For each module, decide what functions it contains.

2.1 Required Functions

You must provide a function for each of the listed tasks below. Feel free to implement additional functions.

Note: For this assignment, arrays distinguish between "used" and "unused" elements. This means, that the size (or capacity) of an array indicates the maximum number of elements that can be stored in the array. However, not all elements may be "used" - in the extreme case, nothing is stored in an array: That is, while an array may have 20 elements, none of these are used to store a value. Thus, you somehow need to find a way to store values in the array in such a way that you can distinguish between "used" and "unused" array elements (various ways are possible, e.g. you can use a marker value that is stored in "unused" locations or you can use a secondary array to indicate which locations are used and which are not used (other methods do exist)).

Any function that takes in an array needs to be aware of this distinction - for example, the function to compute the average value should only consider "used" elements and ingore "unused" elements.

- Return a random positive integer number. Use the standard library function `rand` to generate these numbers - use the same range as `rand()`. Feel free to seed the random number generator.
- Return a random integer number with given limits (stated limits should be inclusive, that is if limits 10 and 20 are given number both 10 and 20 may be returned as the random number).
- Fill a given array of integers with a given size with value 0 - that is fill the array to its capacity (all elements are now "used").
- Fill a given array of integers with a given size with a user-defined value n - that is fill the array to its capacity.
- Fill a given array of integers with a given size with random values within a given range - that is fill the array to its capacity.
- Clear an array of integers with a given size - that is, mark all array elements as being "unused".
- "Defragment" an array of integers with a given size: move all "used" elements to the beginning of the arra and all "free" elements to the end of the array.
- Sort an array of integers with a given size in ascending order (you need to find a method yourself - any method that works is acceptable, it does not need to be particularly efficient).
- Randomize an array of integers with a given size - that is rearrange the elements of an arry in a random fashion.

- Print only "used" elements of an array of integers with a given size in form $\{n1, n2, n3, \ldots, n\}$. An empty array (array with only "unused" elements) is printed as {}.
- Print (all) elements of an array of integers with a given size in form $\{n1, n2, n3, \ldots, n\}$. This function prints both, used and unused elements.
- Return the minimum element of an array of integers with a given size.
- Return the maximum element of an array of integers with a given size.
- Compute and return the average value (as double) of and array of integers with a given size.
- Obtain and return the median value of and array of integers with given size.
- Compute and return the variance (as double) of and array of integers with a given size. Variance v of $\{n1, n2, n3, \ldots, n_N\}$ is given as:

$$\frac{\sum_{i=1}^{N}(n_i - avg)^2}{N},$$

  where avg is the average value and N are the number of elements in the array).
- Compute and return the standard deviation (as double) of and array of integers with a given size. Standard deviation is calculated as follows:

$$\sqrt{\frac{\sum_{i=1}^{N}(n_i - avg)^2}{N}},$$

  where avg is the average value and N are the number of elements in the array).
- Return the number of used elements in an array of integers of a given size (this is not neccessarily the same as the size).
- Return the number of unique used elements in an array of integers of a given size. For example, if your array holds elements {3, 1, 2, 3, 4, 3, 2, 2, 3, 4}, it holds 10 elements in total, but it holds only 4 unique elements (elements 1,2,3,4).
- Print (to the screen) a frequency distribution of the unique elements of an array of integers of a given size. That is, print to the screen a summary how often each (unique) element occurs in the array. For example, if your array holds elements {3, 1, 2, 3, 4, 3, 2, 2, 3, 4} then the following ouput should be obtained:

```
N     Count
3     4
1     1
2     3
4     2
```

Note: The output should be something like this. Minor differences in formatting (number of blanks etc.) will not impact on the marking. The order in which the elements occur in the two column display is not important.

- A test `main()` function - see comments in Section 3.

3 Module Implementation

Implement your application one module at a time (all modules should be placed within the same VS project). Each module consists of two files: a header file (with a .h extension - make sure it contains an inlude guard) that contains all declarations and a source file (with a .c extension) that contains the implementation for all functions of a given module. As these modules are quite small, there is no need to organize them in folders/directories. Also, please make sure to store the `main()` function in a separate C souce file.

Also, your program must use the following:

- Files need to #include your own header files as required.
- At least one simple Pre-Processor macro must be defined and used.
- At least one Pre-Processor macro that takes in two parameters must be defined and used.
- Conditional Inclusion in at least one location.
- Define the following symbolic constants:

Symbolic Constants Name	**Value**
MYSIZE1	10
MYSIZE2	50
MIN1	0
MAX1	10
MIN2	100
MAX2	120

3.1 The `main()` Function

The `main()` function performs the following (whenever an array is printed to the screen, make sure to also print the array's name):

- Create array `data1` with MYSIZE1 elements, clear the array and print the array.
- Fill `data1` with random values in range MIN1 to MAX1 and print the array.
- Sort `data1` and print it to the screen.
- Randomize `data1` and print it to the screen.
- Fill `data1` with values {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} and print it to screen. Remove values 1, 4, 5, and 9 from array (mark their locations being "unused") and print *all* of the array. Also, print the number of used elements in `data1`.
- Defragment the array and print again *all* of the array.
- Obtain and print minimum, maximum, average and median value of `data1`.
- Obtain and print variance and standard deviation of `data1`.
- Create array `data2` with MYSIZE2 elements, fill it with values {3, 1, 2, 3, 4, 3, 2, 2, 3, 4} and print it to the screen.
- Obtain and print the number of used elements in `data2`, the number of unique used elements and print the frequency distribution of `data2`.
- Fill `data2` with MYSIZE2 random values in range MIN2 to MAX2 (overwrite previous values).
- Obtain and print minimum, maximum, average and median value of `data2`.
- Obtain and print variance and standard deviation of `data2`.
- Obtain and print the number of used elements in `data2`, the number of unique used elements and print the frequency distribution of `data2`.
- Sort `data2` and print it.

4 Marking

This is an individual assignment - each student must develop his/her own solution. Any duplicate solutions will receive 0 marks.

The following items will impact on your marks:

- Does your solution perform the required actions correctly?
- Quality of Modular Structure.
- Overall quality of your code (including choice of names for variables and structure of your code).
- Do not use global variables - unless you provide a very good justification why global variables make sense, you will loose marks!
- Quality of your comments (cf. slide "Commenting Guidelines" in Unit 1). Lack of comments will result in very significant loss of marks!!! And yes, *you do need "in code" comments* in addition to the Doxygen comments
- Quality of your code format - follow K&R Coding Style as discussed in lecture (cf. slides "K & R Coding Style" in Unit 1).
- Presence of warnings will cause loss of marks! Please make sure to use standard C, enable warninga and use separate compilation.
- If your code **does not compile** you will **receive 0 marks!**
- Thus, if you are not able to finish any part of the exercise successfully, comment out the sections of code that cause the problem (don't delete it - I might find some merrit in it and you may gain some marks).

Marking Scheme hline Modular Structure & report	30
Correcly implemented functions ($1\frac{1}{2}$ marks each)	30
Complete & suitable Doxygen Comments in code, doxygen documentation generated & submitted	20
All Pre-Processor features implmented	10
Correct & complete `main()` function (2/bullet-point).	30
Penalties:	
Poor Modular Structure:	Up to -50%
Insufficient comments:	Up to -30%
Poor code format:	Up to -30%
Bad coding style (e.g. using `goto` or global variables)	Up to -50%
Compile Time Warning:	-10% each
Compile Time Error:	-100%
Total: (Note: Marks will be scaled down to 20%.)	**120**

5 Deadline & Submission

Deadline for this assignment is 11:00h on Thursday, 09.11.2023.

Please submit your solution as a single zip file via the module's Brightspace page. All solutions must be submitted as MV Studio projects - please put your entire solution into a zip archive (Remove the ".vs" folder in your solution and peform Build→Clean before you zip your solution).

A complete solution contains:

- All source & header files (suitable formatted & commented) as part of a VS project.
- Generated Doxygen documentation in HTML format (stored in a subfolder in the project's base folder).
- Report - named after your ID number - in text format, containing: List of modules, list of functions per module, specification for each functions, pseudo-code for each function.

6 Queries

Please post any queries regarding the assignment on the forum "Assignment #1 Q&A" (found in "Discussions" tab on the Brightspace page). This will ensure that the entire class gets the benefit of the answer.