Functions, Arrays and Structures

Marks Analyzer V3.0

Workshop 8 (worth 3% of your final grade)
URL: https://github.com/Seneca-144100/IPC-WS8

In this workshop, you are going to code C functions and the main program to store student ids and marks in an array of structures. The program will do very simple marks analysis and display the result.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities to:

- use pointers to assign to variables (primitive data types) from inside functions.
- use a pointer to assign to a struct-type variable from inside a function
- use an array parameter to access an array from inside functions
- use an array of structures to store related data together
- decompose a problem into four or more modules
- code a C function for each module
- implement structured programming principles, including single-entry/single-exit logic.

SUBMISSION POLICY

Your workshops are divided into two sections; in_lab and at_home.

The "in_lab" section is to be completed **during your assigned lab section**. It is to be completed and submitted by the end of the workshop. If you do not attend the workshop, you can submit the "in_lab" section along with your "at_home" section (a 20% late deduction will be assessed). The "at_home" portion of the lab is **due the day before your next scheduled workshop**

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible for regularly backing up your work.

IN-LAB: ITEM CLASS (70%)

Download or clone workshop 8 from https://github.com/Seneca-144100/IPC-WS8

Note: Four function prototypes have been declared for you in **analyzer.c**.

1- In **analyzer.c** , define MAX to 100. (#define)
2- Create an array of MAX *studentMark variables*. (The struct type has been declared for you in **analyzer.c**)
3- Declare a *studentMark variable* that will be used to store a single student id and the associated mark.
4- Use a do-while loop to control the data entry process. The user will enter a student id and the associated mark on one line. (See the sample output below.) The data entry process will stop if the array is full or the user has entered -1 for either the student id or the associated mark or both. Inside the body of the loop, the main program calls the function **getData( )** to prompt the user to enter a student id and the associated mark. If none of the two input values is -1, the function stores them into a *studentMark* variable (that is declared in the main program) and returns 0. The main program then copies the *struct* variable into the array. If either of the input values is -1, the main program will stop the data entry process.
5- The main program calls the function **show( )** to display the student data that is stored in the array. Student ids and associated marks are displayed line by line. (See the sample output below.)

Output Example:

```
####### Marks Analyzer V3.0 #######
Enter student id and marks (e.g. 12345 89.5): 12345 55.55
Enter student id and marks (e.g. 12345 89.5): 23456 66.66
Enter student id and marks (e.g. 12345 89.5): 34567 77.77
Enter student id and marks (e.g. 12345 89.5): 45678 88.88
Enter student id and marks (e.g. 12345 89.5): 99999 -1
1. 12345, 55.55
2. 23456, 66.66
3. 34567, 77.77
4. 45678, 88.88
```

For submission instructions, see the SUBMISSION section below.

IN_LAB SUBMISSION:

To test and demonstrate execution of your program use the same data as the output example above or any information needed.

If not on matrix already, upload your `analyzer.c` to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit ipc_w8_in_lab <ENTER>
```

and follow the instructions.

AT_HOME: TITLE (20%)

After completing the in_lab section, upgrade analyzer.c such that it will do a simple analysis of student marks. The program will also display the result of marks analysis. (See the sample output below.)

Marks Analysis

The main program calls the **analyze( )** function to do the following analysis:

1- the mark average,
2- the highest mark,
3- the lowest mark.

The function uses two pointers to assign the highest and lowest marks to two variables declared in the main program. It then returns the average marks.

Finally, the main program calls the **printSummary( )** function to display the result of running the analyze function.

Output Example:

```
####### Marks Analyzer V3.0 #######
Enter student id and marks (e.g. 12345 89.5): 12345 55.55
Enter student id and marks (e.g. 12345 89.5): 23456 66.66
Enter student id and marks (e.g. 12345 89.5): 34567 77.77
Enter student id and marks (e.g. 12345 89.5): 45678 88.88
Enter student id and marks (e.g. 12345 89.5): 99999 -1
1. 12345, 55.55
2. 23456, 66.66
3. 34567, 77.77
4. 45678, 88.88
### result of marks analysis ###
average: 72.22, highest marks: 88.88, lowest marks: 55.55
```

AT-HOME REFLECTION (10%)

Please provide brief answers to the following questions in a text file named **`reflect.txt.`**

1) What is the advantage of using an array of struct variables versus two parallel arrays?
2) What is the purpose of declaring a function parameter/argument as a pointer?
3) When is the keyword const used when passing an array or pointer variable to a function?

AT_HOME SUBMISSION:

To test and demonstrate execution of your program, use the same data as the output example above.

If not on matrix already, upload your **`analyzer.c`** **`and`** **`reflect.txt`** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit ipc_w8_at_home <ENTER>
```

and follow the instructions.