Advanced Programming – PROG36859

Workshop # 1

Logic, Loops, If, Else If, Validations, Functions

In the process of doing your workshop 1, in the part 1 you are to create a `Test Mark Report` program coded in two source files. In the part 2 you will write a series of foolproof I/O functions in a file called `utils.c` that will be used in other coming workshops as well.

Learning Outcomes

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

- Use If, If else and If /else if statements to code decision making logic.
- Use functions which return a value.
- Perform foolproof numerical data entry.
- describe to your instructor what you have learned in completing this workshop.

Submission Policy

This workshop is divided into two coding parts and one non-coding part:

- Part 1: A step-by-step guided workshop, worth 20% of the workshop's total mark

- Part 2: A Do It Yourself type of workshop that is much more open-ended and is worth 50% of the workshop's total mark.
- *reflection*: non-coding part, to be submitted together with *DIY* part. The reflection is worth **30%** of the whole workshop's mark. If your professor deems it insufficient, you will lose 30% of your workshop mark.

Due Dates

The Due dates depend on your section. You can check the Slate for the due dates. You have approximately 1 week to complete all the parts of the workshop and submit your solution to the slate. Follow the instructions of the submission at end.

Late penalties

You are allowed to submit your work up to 3 days after the due date with 10% penalty for each day. After that, the submission will be closed, and the mark will be zero.

Citation

Every file that you submit must contain (as a comment) at the top:
your name, **your Sheridan email**, **Sheridan Student ID** and the **date** when you completed the work.

For work that is done entirely by you (ONLY YOU)

If the file contains only your work or the work provided to you by your professor, add the following message as a comment at the top of the file:

I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

For work that is done partially by you.

If the file contains work that is not yours (you found it online or somebody provided it to you), **write exactly which part of the assignment is given to you as help, who gave it to you, or which source you received it from.** By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrongdoing.

- Add the citation to the file in which you have the borrowed code.
- In the 'reflect.txt` submission, add exactly what is added to which file and from where (or whom).

: warning: *Workshops with no Citation will receive a mark of zero.*

: warning: *This Submission Policy only applies to the workshops. All other assessments in this subject have their own submission policies.*

If you have helped someone with your code

If you have helped someone with your code. Let them know of these regulations and in your 'reflect.txt', write exactly which part of your code was copied and who was the recipient of this code.
By doing this, you will be clear of any wrongdoing if the recipient of the code does not honour these regulations.

Compiling and Testing Your Program

All your code should be compiled and check the output, and closely compare it with the expected output.

Part – 1 (20%)

In this part of the workshop, you will start by writing a program implemented in two files:

`main.c` *(Code provided in the details below)*
`marks.c` *(Incomplete code provided in detail below)*

- main.c (Already implemented, containing the `main` function and the prototype of other functions)
- marks.c (Which will contain your function definitions for: `getNoOfStudents`, `getAverage`, `printReport` and `prnGrade`)

After completion, your program will calculate the average of all the marks received by the students in a test and prints a report stating what the average mark of the whole class is and what the letter grade equivalent for the mark is, using the following table:

Mark	Grade
Below 50	F
50 to 54	D
55 to 59	D+
60 to 64	C
65 to 69	C+
70 to 74	B
75 to 79	B+
80 to 89	A
90 to 100	A+

Main logic

- We will start with printing a title of our program.
- Then we will call the `getNoOfStudents` function, which starts by prompting the user to enter the number of the students in the class. This function will return the value read back to the main program.

- We will use the number returned by `getNoOfStudents` and pass it to the `getAverage` function.
- `getAverage` will print a message to enter students marks and then will proceed to obtain the marks of all the students (the function must ensure the user enters a number between 0 and 100 for each mark). After obtaining the user's values, calculate the average of the whole class. In the end, the function will return the average to the main program.
- Finally, the main program will pass the number of the students and their class's average to the printReport function. The print report function will then display a report, showing the number of students, and their average and will print the letter grade equivalent (see table above for number to letter grade conversion) of the mark by passing it to the `prnGrade` function.

Here is what the main looks like: (***Note:*** *do not change this code)*

```
/*************************************************************************
// AP Workshop 1 p1: tester program
//
// File main.c
// Version 1.0
// Date Fall 2024
// Author  Mahboob Ali
// Description
//
// This file will be replaced by another tester program during the
submission
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
//
/////////////////////////////////////////////////////////////////
*************************************************************************/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int getNoOfStudents(void);
int getAverage(int NumberOfStudents);
void printReport(int NumberOfStudents, int average);
int main(void) {
   int num;
   int average;
   printf("Class test marks report program...\n\n");
   num = getNoOfStudents();
   average = getAverage(num);
   printReport(num, average);
   return 0;
}
```

Preparing the workshop 1 project.

Create a Project

Visual studio

- Open Visual Studio
- Create a new project
- Select Empty Project: `C++ Windows Console`
- in project name type: `lab`
- in location select the `ws1` folder you just created
- Make sure `Place solution and project in the same directory is checked
- Click on `Create`

Xcode

- Open Xcode
- Select `Create a new Xcode project`
- From the template window select `macOS`
- In the application section of the same window select `Command Line Tool`
- Click on `Next`
- In `Product Name` type the name of the project. In our case, it is `lab`
- In `Organization Identifier` type whatever you want to appear in the comment of the sample file
- Click on `Next`
- Select the `ws1` you created earlier
- Click on `Create`
- rename the sample `main.c` file to `lab1.c`

Writing your first program

Create a file called main.c in the lab project: `Advanced-Programming-Works/workshops/ws1/lab/main.c`

Visual Studio

- Click on Solution Explorer
- Right-click on `Source Files`, select `Add`, and then `New Item`
- In the `Visual C++` section, select `C++ files` (this includes C files too)
- In the Name section, type `main.c`
- Click on `Add`

XCode Mac

- Edit the `main.c` file you created in the previous step.

Start coding:

Step 1

- Create another file in the project called `marks.c`, (your project should have two file, `marks.c` and `main.c`).
- write a main function to only test the `prnGrade` function that is not used in the main logic. Complete the `prnGrade` first and then continue coding the rest of the functions. A reminder that the end goal of prnGrade is to take a numerical grade and then display the letter grade equivalent of that numerical grade.

```
// tester for prnGrade function
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
void prnGrade(mark);
int main(void){
   int mark;
   printf("Mark: ");
   scanf("%d", &mark);
   prnGrade(mark);
   return 0;
}
```

coding `prnGrade`

Note: *Read the process of coding first below and if you understand the whole logic then you don't need to follow each step you can just write the code for the function prnGrade which will take input for marks and then print the letter grade.*

To try and code `prnGrade`, follow the step-by-step process outlined below:

- Start by coding the function definition of prnGrade in marks.c.

```
void prnGrade(int mark)
```

- Make the function only print the NUMERICAL mark first. Compile and run the program to make sure the function call, syntax and the main function are good to go for coding.
  (If you pass this step, then you can proceed with revising your code to display the LETTER grade).
- Depending on the mark passed to `prnGrade`, you will need to print different letter grades.
- Continue by creating an `if/else if` structure to print the letter grades based on the value of the mark
  - if the mark is below 50, print `F`
  - else if the mark is below 55, print `D`
  - else if the mark is below 60, print `D+`
  - else if the mark is below 65, print `C`

        - etc…
    - When printing each letter, do not go to a new line

|| *note that validation is not needed since we assume the mark passed to this function IS a value between 0 and 100* ||

    - recompile and rerun the function many times with different values making sure it works accurately
    - Save the tester program YOU made, `main.c` under a different name (for example `prnGradeTester.c`) for possible future use

Step 2

- Copy the FULL main program's logic into `main.c` (i.e., the first code snippet)
- We will focus on coding `getNoOfStudents` as it is the first function called in our main program.
- As such, temporarily comment out all of the function calls in the main program BELOW the getNoOfStudents(); function call
- For YOUR testing purposes (not for the end result of getNoOfStudents) add a print statement that displays the value that was returned from your function call.
  This will help you in debugging your code.

Coding `getNoOfStudents`

    - add the function to `marks.c` file: `int getNoOfStudents(void)`
    - create a local integer variable (to return it after reading from the console)
    - prompt the user for the number of students:
`"Please enter the number of students: "`
    - scan the number into the address for the local variable
    - return the value

> no validation is needed; we assume the user enters the value properly
> - test the function by recompiling and rerunning the program and make sure it works correctly
> - remove the added print statement (the one you added earlier to test the returned value)

Step 3

- Now that you've coded getNoOfStudents, un-comment the `getAverage` function call
- Like in the last step, add a print statement to print the return value of your function call
  In this function we are going to get marks for all students in a loop, then add the mark to a (sum) variable to calculate the average later, and add one to the loop counter to stop

when the number of students has fulfilled. But each time we get the mark, we will make sure the mark is between 0 and 100 and only then will we add the value to the sum and increase the counter, otherwise we'll just print an error; requesting for correction.

coding `getAverage`

Note: *this function asks to calculate the average of the marks.*

- add the function to `marks.c` file: `int getAverage(int NumberOfStudents)`
- create the following local integer variables: counter(for the loop), mark(to be scanned) and sum(for average calculation)
- prompt the user to enter the marks for the students:
`"Enter ## student marks...\n"` (replace ## with the number of students)
- while the counter is less than the number of students (*do not use a do-while and for-loop*)
  - print a prompt for mark entry with row number: `"##> "` (replace ## with row number)
  - scan the mark
  - if the mark is below 0 print `"Invalid Mark, values should be greater than or equal 0.\n"`
  - else if the mark is above 100 print `"Invalid Mark, values should be less than or equal to 100.\n"`
  - else add one to the loop counter and add the mark to the sum
- after the loop is done return `the division of sum by the number of students` as average
- recompile and rerun your function making sure it works correctly
- if all works as expected, remove the the print statement that displayed the value returned by your getAverage function call

Step 4

- un-comment all the code to test the last function `printReport`

coding `printReport`

- add the function to `marks.c` file: `void printReport(int NumberOfStudents, int average)`
- print a message with the number of students and their numerical class average
- print the letter grade (using `prnGrade` function) as follows:

```
Number of students: AA
Class average: BB% (G)
```

replace `AA` with the number of students
replace `BB` with the average and `G` with the letter grade

Compiling and running the program

Visual Studio:

To test and run your program using Visual Studio, you can do one of the following options:

- Hold `<Ctrl>` key and press `[F5]`
  OR
- From the `Debug` Menu select `Start Without Debugging

If there are no errors in your program, you should see that the program compiles and the output will be displayed on a `Console` window. You can hit `Enter` on your keyboard to close the console window after the program's execution.

Xcode (mac)

Click the "play button" :arrow_forward: at the top of the left panel to compile and run your program

Data Entry

```
3
67
-10
200
78
89
```

Sample output

```
Class test marks report program...

Please enter the number of students: 3
Enter 3 student marks...
1> 67
2> -10
Invalid Mark, values should be greater than or equal 0.
2> 200
Invalid Mark, values should be less than or equal to 100.
2> 78
3> 89
Number of students: 3
Class average: 78% (B+)
```

Part 2 DIY (50%)

In a file called `utils.c` create four foolproof number entry functions.

First add the `void line(char fill, int lenght)` function to the `utils.c`

Startup your code using your teacher's code developed in class (the getInt function) and complete and debug it then reuse it and its logic to develop these four functions:

```
int getInt(void);
int getIntMM(int min, int max);
double getDouble(void);
double getDoubleMM(double min, double max);
```

int getInt(void)

This function will receive an integer form the console, making sure only an integer is entered;

execution sample

```
Enter an integer value: abc
Invalid Integer, try again: 10abc
Enter only an integer, try again: 10
```

Data Entry

```
abc
```

```
10abc
10
```

int getIntMM(int min, int max)

This function will receive an integer form the console, making sure only an integer is entered and the value is within the min and max limits

execution sample for `getIntMM(10, 20)`

```
Enter an integer value between 10 and 20 inclusive: abc
Invalid Integer, try again: 10abc
Enter only an integer, try again: 9
[10<=Number<=20], try again: 21
[10<=Number<=20], try again: 15
```

Data Entry

```
abc
10abc
9
21
15
```

double getDouble(void)

This function will receive a double form the console, making sure only a double is entered;

execution sample

```
Enter a Double value: abc
Invalid Double, try again: 10.1abc
Enter only a Double, try again: 10.1
```

Data Entry

```
abc
10.1abc
10.1
```

double getDoubleMM(double min, double max)

This function will receive a double form the console, making sure only a double is entered and the value is within the min and max limits

execution sample `etDoubleMM(double min, double max)`

```
Enter a Double value between 10.1 and 20.1 inclusive: abc
Invalid Double, try again: 10.1abc
Enter only a Double, try again: 10
[10.100<=Number<=20.100], try again: 20.2
[10.100<=Number<=20.100], try again: 15.1
```

Data Entry

```
abc
10.1abc
10
20.2

15.1
```

DIY part 2 tester program

```
/*****************************************************************
// AP Workshop 1 p2: tester program
//
// File main.c
// Version 1.0
// Date Fall 2024
// Author  Mahboob Ali
// Description
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
//
/////////////////////////////////////////////////////////////////
*****************************************************************/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int getInt(void);
int getIntMM(int min, int max);
double getDouble(void);
double getDoubleMM(double min, double max);
void line(char fill, int len);

int main(void) {
   int value;
   double dvalue;
   printf("Enter an integer value: ");
   value = getInt();
   printf("You entered: %d\n", value);
   line('*', 50);
   //****************************************
   printf("Enter an integer value between 10 and 20 inclusive: ");
   value = getIntMM(10, 20);
   printf("You entered: %d\n", value);
   line('*', 50);
   //****************************************
   printf("Enter a double value: ");
   dvalue = getDouble();
   printf("You entered: %.2lf\n", dvalue);
   line('*', 50);
   //****************************************
   printf("Enter a double value between 10.1 and 20.9 inclusive: ");
   dvalue = getDoubleMM(10.1, 20.9);
   printf("You entered: %.2lf\n", dvalue);
   line('*', 50);
```

```
    //*****************************************
    return 0;
}
```

Reflection (30%)

Study your final solutions for each deliverable of the workshop, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time and the result is suggested to be at least 150 words in length.**

Create a file named `reflect.txt` that contains your detailed description of the topics that you have learned in completing this workshop and mention any issues that caused you difficulty.

You may be asked to talk about your reflection (as a presentation) in class.

Submission Process:

Data Entry

Follow the instructions during submission

Files to Submit

`utils.c`
`main.c`

Submission to Slate

Zip the whole visual studio project and upload it to slate.

- Your VS solution should be having two projects.
- For part 1 and part 2