Assign-01b

Overloaded Functions – Test Harness

Assign-01a - Review

In Part (a) of the assignment, you created a standalone program that prompted the user for an input (representing a student's mark(s) in a course), you parsed the input to determine which of the 3 different input types was actually entered by the user and then you called one of three overloaded functions. The overloaded functions were to have the following interfaces:

- The *string version* of the function took a single parameter (char *) containing a letter grade or special situation
- The *double version* of the function took a single parameter (double) containing the final mark
- The *assignment version* of the function took five integer parameters containing assignment marks (or an array of five integers)

You were also allowed to have 4 different potential outputs. An example of each of the 4 different styles of output are given here:

- `Student achieved  77.25 % which is a PASS condition.`
- `Student achieved  34.23 % which is a FAIL condition.`
- `Student has Special Situation :  AU (Audit Condition)`
- `Sorry, there was an error in your input ...`

The wording of your error statement was totally under your control within your program – but there was a definite need to have an error statement for the input case where the user entered invalid data.

Notice how the assignment's requirements never mentioned or guided you towards the need to have a return value from these functions – or even what data-type might possibly be returned? That is because that decision now becomes part of this assignment. Here is a list of the files that you need to submit for this assignment - submit **only the following files**:

- unitTest.cpp — contains the functions needed to perform your UNIT TESTS on the various assessGrade() functions
- assign1.cpp — this is a source module created for Assign-01a (it contains your `main()`)
- assessGrade.cpp — this is a source module created for Assign-01a (containing the overloaded functions)
- assessGrade.h — this is a source module created for Assign-01a

Please ZIP up and submit these files to the appropriate eConestoga drop-box by the due date and time.

Assign-01b

Overloaded Functions – Test Harness

Unit Testing

You were introduced to the necessity of testing (various levels of testing (unit, integration, system, etc.) as well as various types of tests (normal/functional, exception, boundary, etc.)) in SEF last semester. This semester, we continue in this learning by having you create and run your own unit *test harness* program.

A unit test harness is nothing more than another source module (with its own `main()` function) that is used to run certain and specific tests on the various functions and methods that you create. In this case, you need to run the following types of tests on the different overloaded `assessGrade()` functions.

Function Name	Type of Test
assessGrade() – string version	Normal / Functional Tests • you will need to create and run 5 different normal (also called functional) tests • e.g. assessGrade("A+") or assessGrade("DNA") Exception Tests • you will need to create and run 5 different exception (also called fault) tests • e.g. assessGrade("A-") or assessGrade("Hello")
assessGrade() – double version	Normal / Functional Tests • you will need to create and run 5 different normal tests • e.g. assessGrade("42.37") or assessGrade("80.0") Exception Tests • you will need to create and run 5 different exception tests • e.g. assessGrade(-23.5) or assessGrade(234.1) Boundary Tests • you will also need to create and run 3 different boundary tests • a boundary test is meant to "check out" specific values at the edges of acceptance • e.g. assessGrade (0.00), assessGrade(100.00)
assessGrade() – assignment version	Normal / Functional Tests • you will need to create and run 5 different normal tests • e.g. assessGrade(75,90,80,70,60) or assessGrade(80,20) Exception Tests • you will need to create and run 5 different exception tests • e.g. assessGrade(20,-10,90,80)

Since the source module called unitTest.cpp (your test harness) and the module called assign1.cpp (your program) both have a `main()` function, you cannot include them both at the same time in your Visual Studio project when compiling. You will need to include or exclude each of these source modules depending on whether you are running the unit tests or your program.

NOTE: In coding up your test-harness, you need to make sure that all tests for the ***string version*** are carried out before you test the ***double version*** of the `assessGrade()` function. Similarly all ***double version*** tests are completed before starting the testing of the ***assignment version*** of the function.

Assign-01b

Overloaded Functions – Test Harness

Unit Testing - Output

In SEF, we also discussed it is always nice when you tell the person running your unit test program which test you are currently about to run (using a unique test identifier as well as indicating which *type* of test it is), what function you are testing, what you expect the result to be, what the actual result is and as well whether the test passed or failed (based on the actual versus expected result) For this reason, your unit-testing output needs to look like this :

```
Test #1 : Functional test of assessGrade(char *)
   >> Submitting "A+" as the student's mark
     >> Expect Result : Student achieved  95.00 % which is a PASS condition.
     >> Actual Result : Student achieved  95.00 % which is a PASS condition.
   ** TEST PASSED **
. . .
Test #11 : Functional test for assessGrade(double)
   >> Submitting "42.37" as the student's mark
     >> Expect Result : Student achieved  42.37 % which is a FAIL condition.
     >> Actual Result : Student achieved  42.37 % which is a FAIL condition.
   ** TEST PASSED **
. . .
Test #24 : Functional test for assessGrade(int[])
   >> Submitting "[75,90,80,70,60]" as the student's mark
     >> Expect Result : Student achieved  75.00 % which is a PASS condition.
     >> Actual Result : Student achieved  75.00 % which is a PASS condition.
   ** TEST PASSED **
```

We do this so that the user can capture the output from your test-harness and use it as a test report or in their test log document.

Test-Harness – Code Structure

In this assignment, you are required to use the concept of function pointers in the design and implementation of your unit test-harness. You need to create 3 testing functions called `functionalTest(…)`, `exceptionTest(…)` and `boundaryTest(…)`. These functions .will be called in order to perform that type of test for each of your 3 overloaded functions. These functions will call the various `assessGrade()` functions for the different types of tests.

The return data-type and value as well as the parameter lists for these 3 testing functions are totally under your control. You need to determine what you need to pass into the functions when running a test case. In choosing your parameter lists – you will effectively be choosing which style of function pointer implementation you will be using.

Another important factor which may help you make this decision is discussed in the following section.

Assign-01b

Overloaded Functions – Test Harness

Visualizing Your Test Data Structure

In this test harness, you are allowed to hard-code your test scenario / test case data within your test-harness. In a real test-harness implementation, we would script each test case's data and read the test data in from a file to be used. But this Assign-01b I want you to simply hard-code your test scenario data within the test-harness source code.

It is not necessarily as simple as it sounds. Often times you will find that the way you represent the structure of your data within a program actually helps streamline your code design! That is, you can help solve part of the program's design and structure by carefully choosing the way that you structure and represent your data entities.

In this test harness, ask yourself – what pieces of data do you really need in each test case? What data will be changing as you run each test scenario and print the required test output? How can you represent the pieces of data that you need as input to each test scenario? To help you visualize what you need in each test case and how to represent it – think about and remember sometimes the most obvious solution is not always the best. Sometimes you may need to think a little more *abstractly* …

As well, think about the 33 test cases you will need data for – you will be cycling through the functional test for the string version of `assessGrade()` first, followed by the exception tests for the same function … Could you possibly house all of the test data for the string version of the function in one place?

Other Concepts to Consider

1. As mentioned at the start of these requirements, Assign-01a never told you or guided you as to what (if any) the return value should be from your overloaded functions. You now need to think hard about what needs to be returned (what values and what data-type) from the 3 overloaded functions…

   Take a look at the required Unit Testing - Output requirements above. Your unit testing output needs to print the actual overloaded function result (i.e. one of the allowed 4 styles of print messages). As well, your unit testing functions need to dynamically determine if the **test** PASSED or FAILED (not the student). This determination needs to be done at runtime in your harness by comparing the expected result against the actual result.

   The question is how will you accomplish this? How will you actually get the printed result from your overloaded functions as well as some value/status/indicator as to what the actual result is so you can use it in a PASS/FAIL comparison within your testing function?

   You may feel like you are overwhelmed with this requirement – how can you do this? What possibly could the return data-type and value be from your overloaded `assessGrade()`

function be in order to accomplish this? You can easily figure this out by examining your Assign-01a code and knowing this assignment's requirements and thinking about these questions:

- Where do the "student passed/failed" print statements come from in my Assign-01a? When I'm testing my `assessGrade()` functions – will these print statements execute?
- How can I get a piece of information back from the `assessGrade()` function to be used in a comparison between "expected" answer and "actual" answer (i.e. the value in the print statement) for the dynamic determination of the test PASS/FAIL statement?
- What about the "special situation" print statements? Where are they being printed in my Assign-01a? And how can I get a piece of information back from the `assessGrade()` function as to which special situation it is – to be used in the test PASS/FAIL comparison?
- Where does my "invalid input" error statement come from in my Assign-01a? How can I get a piece of information back from the `assessGrade()` function indicating this error so I can use it in my test-harness?

2. If you need to, <u>you are allowed to change the logic within the body of you're A-01a `assessGrade()` functions</u> in order to be able to properly test the functions as required in this assignment. The types of changes you might want to make include the changing of the `assessGrade()`'s return data-type, the addition some range checking, some other `assessGrade()` requirements which may have been missed or overlooked in A-01a. You are not allowed to change the function calling mechanism (prototypes) of the `assessGrade()` functions as required in A-01a.