**Submission of Your Work**

You need to prepare and submit ONE SINGLE MS Word document to Canvas (in your lab section) as LastName\_FirstName\_Labxy.doc. It must contain:

* Your NAME is the only item on page 1
* For ***each*** question:
  + Specify the question number, if more than one question.
  + Source code. Copy/Paste your final source code. You must include standard “comment header” even if code is provided. *Do Not* paste a snippet of your source code, it must be copy/pasted.
  + Initial test plan. After reading the requirements, but **before** beginning any coding, create the test case table, below, completed through column Expected Output. Include in your report.
  + Final test plan. Write your program then complete the **test table** with actual output results and include in your report *AFTER* your source code.
  + Output results. Paste in a snippet of output showing results for **every listed test case in your final test plan**, labeled with test case #

Test Table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test # | Valid / Invalid Data | Description of test | Input Value | Expected Output | Actual Output | Test Pass / Fail |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |

* Add / delete rows from Test Table as necessary
* Modify column widths as necessary
* Test both valid and invalid input
* Test for every output expected
* If failure is an expected output and it happens then that test Passes
* Any test that fails means the program must be fixed so that it passes the test
  + Failing tests need a new test row, ie 1a, 1b, etc, showing corrections from original
  + Unless you don’t have time to finish your testing, there should never be a Fail in final test plan

Program:

Use separate source files and header files to create a class to represent students.

* 1. A student object has attributes of major (string) and credit hours taken (integer).
  2. Member functions (methods) are as follows (no additional ones may be created):
     1. Using a constructor initializer list, create a constructor with default parameter values of “General Studies” and -1, and that displays “\n\tConstructor called for major <major> and hours <hours>\n”, replacing <major> and <hours> with attribute values.
     2. individual const get methods (in-line definition) for each attribute
     3. a const get method (NOT in-line) that uses pass-by-reference for both parameters and has return-type void
     4. one set method (NOT in-line) that sets both attributes
     5. PrintMe method (NOT in-line) that returns string “I’m a <major> major and have completed <x> credit hours.”, replacing <major> and <x> with the appropriate attribute values
     6. Destructor (in-line) that displays “\n\tDestructor called for <major> major.\n”, replacing <major> with the appropriate attribute value

Write a complete program that uses the student class you created and tests its functionality

* 1. Declare a single student object and during declaration initialize it to have major “Astrophysics”; do not provide a value for number of credit hours
  2. Call the PrintMe method of the single student object and display to the screen/console
  3. Ask the user for the name of a file to which output should be written
  4. Declare a vector of 5 objects of student class
     1. Ask the user for 5 majors and 5 credit hours and assign them to each student in the vector. The credit hours must be verified as non-negative or else get another number. This must be tested in your output.
     2. Open the file and write to it using the value returned by PrintMe for every student in your vector
  5. Explicitly close the output file
  6. Any other tests you deem appropriate to prove your program works perfectly

In order to properly capture the destructor messages, put a breakpoint on the return statement in main. When the program pauses at that line, press F10 once, capture the output screen as a snippet, then press F5 to finish execution.