CS 159 – Spring 2022 – Lab #5

Overview of Current Lab:

1. **Important** – submit your attendance at the start of every lab meeting!
2. Please read over the collaborative teaming expectations.
3. **Work with your team on the written problems found in this document.** These problems are the basis for the lab quiz and may include content not introduced in lecture.
4. Begin your work on the programming assignment.
5. **New lab partners, ensure you have their contact information.** Complete the Collaborative Group Communication form and submit to your lab instructor for review.
6. During the final 10 minutes of your lab today you will complete the lab quiz.

Collaborative Teaming:

- **Why utilize and rotate roles among the members of a team?**
  - The use of, and rotation through, the roles described will allow each member to gain experience with the entire process of developing a solution to a programming problem. As a team there should be a shared interest in strengthening each individual member such that they are able to continue to contribute as problems become longer and increase in complexity. Individuals should insist on a rotation of roles during the entire development process to better prepare for homework programming assignments which are individual efforts.
  - The roles are designed to provide the opportunity for individual members to learn about the problem-solving process and the implementation of a solution using the tools of the course. The roles do not emphasize efficiency as each lab programming assignment is due approximately one week after it becomes available. Do not allow your desire to complete every assignment quickly be at the expense of learning the material.

- **Groups are expected to communicate to share their ideas when it comes to solving the conceptual and programming problems associated with this lab.** You may find a collaborative document to be a helpful way to share thoughts on the written problems and to formulate the logic for the programming problem. Supporting documentation may be requested when seeking assistance from course staff.

- **As a group you must determine who will make the final submission for your group**, when that submission will be made, and how the concerns regarding submission will be communicated with the other members.

- **What if a partner does not respond to your communication?** Then the remaining active partners need to be prepared to proceed on the assignment to meet the deadline.

Lab Quiz #5 – 5 points possible: The lab quiz will be made available on Brightspace (Week #6 module) during the final 10 minutes of your lab today. The quiz will emphasize material from chapter 4 and the course programming and documentation standards relevant to the lab programming assignment. **Lab quizzes are individual efforts and you may not use any resources while completing the quiz.** Questions are presented one at a time and cannot be revisited. Be sure to save your answers to each question and to finish your quiz to ensure it is submitted for grading. Most problems on lab quizzes will be multiple-choice or true-false.

- Quizzes are password protected and will be provided by your lab instructor.
- The time at which you take your quiz is monitored and students completing their quiz at an unexpected time or in an unexpected location are subject to loss of points and a potential academic integrity inquiry.

Lab Programming Assignment #5 – 5 points possible: More user-defined functions! The written problems place an emphasis on passing parameters. There is nothing about this problem that would require passing by address but you are permitted to use this technique if you can defend its use and your understanding to the lab instructor.

- **How might collaboration be useful on this particular programming assignment?** Planning is once more the central activity of the group. Implementation is important as the topic of user-defined functions is still new and will be the final topic covered on the first midterm exam, invest time in understanding!

CS 159 – Collaborative Group Communication

Name	Purdue career account name	Who will make the final submission of this lab assignment?
		The first individual listed here is responsible for the final submission of this lab programming assignment.
		These partners must be responsible for submitting at least one lab assignment for this team.

Each member should initialize affirming the following:

Every member has the contact information of the other lab partners.			
The collaborative team has been created and finalized in Vocareum.			
How will members of the group be contacted should a concern arise with the assignment? Be specific (text message, e-mail, groupme, slack).			
When and where is the group meeting next?			
When will the final electronic submission be made?			
Who is responsible for bringing the C programming text and class notes to the next lab meeting?			

Solve the following problems related to material found in Chapter 4 and the course standards.

Statement	True or False
The coding portion of lab #5 requires the use of the specific user-defined functions described in the assignment.	**TRUE**
Lab quizzes are individual efforts. No resources may be referenced during the lab quiz. Only one browser tab may be open during the lab quiz and any open second tab, or second browser, will be considered an academic integrity violation.	**TRUE**
Section 4.4	
In downward communication (passing by value) it is only a copy of the data that is sent from the calling function to the called function.	
It is not possible to access a variable in the calling function by its identifier when inside the called function.	
Given the address of a variable the called function can access and manipulate the value of a variable in the calling function.	
The called function must declare a special type of variable known as a pointer to store a memory address that is sent from the calling function.	
To obtain the address of a variable use the address operator (`&`).	
The asterisk (`*`) when used in a variable declaration indicates that such variables are not data variables but address (pointer) variables which can store the addresses of other variables in the program.	
The asterisk has two different uses, declaring an address variable (pointer) and indirectly accessing the data (in the memory location to which the variable points).	
When only one data item needs to be returned to the calling function then we should use the standard `return` statement rather than passing a single parameter by address.	
Section 4.6	
The scope of an object determines the region of the program in which it is visible (and defined).	
A variable declared in the local declaration section of a function has a scope that extends until the end of that function.	
Objects with a global scope are visible (defined) everywhere in the program.	
It is poor programming style to reuse identifiers within the same scope.	
Section 4.8	
A structure chart should be created after your program has been written.	F
Each rectangle on a structure chart represents the user-defined and standard library functions used in a program.	F
No code is contained in a structure chart as it only demonstrates the function flow of the program.	
A structure chart may show the data that is exchanged between functions.	
Functional cohesion is a measure of how closely the processes in a function are related.	
A function that does one and only one process is functionally cohesive.	
It is a good design practice to limit user-defined functions to only a single task.	
It is a good design practice to not repeat the logic of one function in other functions of the program.	
It is a good design practice to design a user-defined function such that it is testable apart from the rest of the program.	

Solve the following problems related to material from Chapter 4 and the course standards:

Statement	True / False
It is possible to determine if any parameters are passed to a function by address from the declaration statement of the function.	
It is never possible to determine if any parameters are passed to a function by address from an example call to the function.	F
It is possible to determine if any parameters are passed to a function by address based on the first line of the definition of the function (also known as the function header).	
A function that passes at least one parameter by address must pass them all by address.	
All functions that utilize pass by address must be `void` functions.	F
One benefit of pass by address is that it allows multiple changes to be made in a function and to have those changes available in the calling function.	
With the use of pass by address it is now permissible for a function to be written to complete several sub-tasks of the program.	
When working with a parameter that has been passed by address it is unnecessary to use the `&` (address) operator in the `scanf` because the parameter already represents a memory location.	
The `*` and `&` operators are inverse operations of each other.	

Consider using this space below to design the structure chart of your program:

STOP!		**TA Verification:**
Complete Written Problems	Problems from pages 3-4 in this document are complete as the group prepares for programming assignment and quiz at the end of the lab session.	

Lab #5 - Programming Assignment
Due: 30 minutes prior to the start of your next lab meeting.
5 Points Possible

Collaborative Roles for the Lab Session

Collaborative Teaming. For this lab you will be working in your assigned teams. If you are unable to complete your assignment during the lab then it is expected that your team meet and collaborate outside of class to finish and submit the problem assigned. **Be sure to exchange contact information during lab today!**

Role:	**Description: Every member will rotate roles at every checkpoint.**
Driver	The driver is in charge of the computer which includes **entering code, saving, testing, and submitting**. This individual should be soliciting the other two members for advice.
Navigator	The role of the navigator is to look over the shoulder of the driver for **syntax errors, logical errors, and concerns related to course standards**. With the introduction of user-defined functions the role of Navigator should include tracking function names, return types, and parameters to help the driver as they enter code.
Manager	The manager may not be as close to the driver as the navigator but still plays an important role ensuring that the **algorithm to be implemented is correct,** can be tested using a variety of input to verify correctness, and **complies with the additional requirements of the assignment.**

Problem: The user will provide information on two lines that intersect. Locate that point of intersection to create a triangle. Output the lengths of each side and the interior angles of the triangle.

Input will be a point followed by the angle created with the x-axis by a line passing through that point. A second point and angle will follow. The lines will always intersect and neither of the lines input will be vertical. A triangle will always be created (points will be unique and all three will never be co-linear).

Checkpoints During Lab #5:		**TA Verification**
1 – Code Review	Members should have access to the submission made by their previous group for lab #4 as well as their individual efforts from the second homework assignment. You may reference both as you develop your solution for lab #5.	
2 – Getting Started in Vocareum	File `lb05.c` created, assignment header completed, `main` function inserted, variables declared and commented, relevant symbolic/defined constants created.	
3 – Function Declarations and Calls	Add the function declarations to the global declaration section of the program. Complete the `main` function with the necessary function calls.	
4 – Implementation of Input Functions	Demonstrate input operates as expected. Other functions called in `main` should be made inactive.	
5 – Implementation of Calculations	Demonstrate functions related to required calculations operate as expected. Output functions may be inactive and replaced with diagnostic print statements at this time.	
6 – Implementation of Output Functions	Final output matches expected, additional test cases created to further demonstrate correctness of logic.	

Example Execution #1:

```
Enter coordinates for point #1 -> 0 0
Enter angle at x-axis for line passing through point #1 -> 45
Enter coordinates for point #2 -> 4 0
Enter angle at x-axis for line passing through point #2 -> 108.4

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Line #1 Distance:   4.0 Angle:  63.4
Line #2 Distance:   3.2 Angle:  45.0
Line #3 Distance:   4.2 Angle:  71.6
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Example Execution #2:

```
Enter coordinates for point #1 -> 0 5.2
Enter angle at x-axis for line passing through point #1 -> 100.89
Enter coordinates for point #2 -> -1 0
Enter angle at x-axis for line passing through point #2 -> 0

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Line #1 Distance:   5.3 Angle:  79.1
Line #2 Distance:   2.0 Angle:  21.8
Line #3 Distance:   5.3 Angle:  79.1
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Example Execution #3:

```
Enter coordinates for point #1 -> 1 1
Enter angle at x-axis for line passing through point #1 -> 45
Enter coordinates for point #2 -> 2 5
Enter angle at x-axis for line passing through point #2 -> 120

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Line #1 Distance:   4.1 Angle: 105.0
Line #2 Distance:   2.2 Angle:  31.0
Line #3 Distance:   3.0 Angle:  44.0
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Example Execution #4:

```
Enter coordinates for point #1 -> -5 -1
Enter angle at x-axis for line passing through point #1 -> 60
Enter coordinates for point #2 -> 0 4.5
Enter angle at x-axis for line passing through point #2 -> 120

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Line #1 Distance:   7.4 Angle:  60.0
Line #2 Distance:   1.8 Angle:  12.3
Line #3 Distance:   8.2 Angle: 107.7
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Additional Requirements:

1. Add the **lab assignment header** (`vi` shortcut `:hlb` while in command mode) to the top of your program. An appropriate description the logic of your program must be included in the assignment header.
2. **Each of the example executions provided for your reference represents a single execution of the program.** Your program must accept input and produce output **exactly** as demonstrated in the example executions, do not add any "bonus" features not demonstrated in the example executions. Your program will be tested with the data seen in the example executions and an unknown number of additional tests making use of reasonable data.
3. For this assignment you will be **required** to implement the user-defined functions (from chapter 4). Failing to follow course standards as they relate to good user-defined function use will result in a **zero for this assignment.**
4. Revisit **course standards as it relates what makes for good use of user-defined functions, what is acceptable to retain in the** `main` **function, and when passing parameters by address is appropriate.**
    - In many cases user-defined function use should result in a `main` function that only declares variables and makes function calls.
5. Course standards **prohibit** the use of programming concepts not yet introduced in lecture. For this assignment you can consider all material in the first **four** chapters of the book, notes, and lectures to be acceptable for use.
6. A program **MUST** compile, be submitted through Vocareum as demonstrated during the lab #0 exercise, and submitted prior to the posted due date to be considered for credit. The C-file you submit must be named exactly: `lb05.c`

Course Programming and Documentation Standards Reminders:

- Use the course function header (`vi` shortcut `:hfx` while in command mode) for every user-defined function in your program.
    - List and comment **all parameters** to a function, one per line, in the course function header.
    - **All function declarations** will appear in the global declaration section of your program.
    - **The user-defined function definitions will appear in your program after the** `main` **function.**
- Indent all code found within the `main` function **exactly** two spaces.
- Place a **single space** between all operators and operands.
- Comment **all** variables to the right of each declaration. Declare only one variable per line.
- Notice that several programs (see program 2-9 on pages 74-75) in the programming text use a single line comment to indicate the start of the local declaration and executable statement sections of the `main` function.
    - At no point during the semester should these two sections ever overlap. You might consider adopting this habit of commenting the start of each section to help you avoid this mistake.
- Select **meaningful identifiers** (names) for all variables in your program.
- Do not single (or double) space the entire program, **use blank lines when appropriate**.
- Maximize your use of symbolic/defined constants and minimize your use of literal constants.