CS 159 – Spring 2021 – Lab #5

What will you submit? A single C-file will be submitted electronically via the guru server. An example submission was conducted as part of the Account Configuration Activity. If you have a concern regarding how to submit work, please contact course staff prior to the deadline for this, and all, assignments. The programming assignment is due on Friday March 5, 2021 at 11:00pm (LOCAL WEST LAFAYETTE, IN TIME). **No late work will be accepted.**

Weekly Quiz #5:

The weekly quiz will be available (Week 7 module on Brightspace) until the same date and time that the programming assignment is due. It is strongly recommended that you complete the attached problems, the programming assignment, and watch all relevant lectures before attempting the quiz.

The quiz will emphasize chapter 4 material, the written problems in this document, the lab programming assignment, and the course programming and documentation standards as used in this lab. 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. Each quiz has a 15-minute time limit.

Collaborative Teaming:

- **How do I know who is on my lab team?**
  - **On-campus students:** TAs have contacted their students via e-mail.
  - **On-line students:** Visit the Start Here module on Brightspace and locate the Distance Learning Team Assignment spreadsheet.
- **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.
- **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. Other on-line tools may come in handy when trying to collaborate on specific segments of code, just make sure they protect your code from being posted publicly! One popular service from previous semesters is codeshare.io.
- **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. **Only one person per group will make submissions for the entire group**. The grader for your section cannot be expected to grade submissions from multiple members of the same group to determine which submission you actually want graded.
  - In order for each member of the group to get credit for the programming problem associated with this lab, their **Purdue University e-mail address must appear in the assignment header**.
- **How might collaboration be useful on this particular programming assignment?**
  - Be the member of your team that actually reads the lab document and can share that the degree symbol can be generated on `guru` with the following code: `printf("%c%c", 0xC2, 0xB0);`
  - **Start with a structure chart.** Identify those tasks in the larger problem and how functions can be implemented to complete those tasks.
  - Determine which tasks for lab #4 can be moved directly into lab #5. Delete those that are not relevant to the current lab, don't leave any functions in your program that are never called.
  - The final step before writing code is to work out the logic of each individual function. While some details are left when it comes time to code, discuss as a group the potential logic concerns within each task.

(Task #1) - Solve the following problems related to material found in Chapter 4 and the course standards.

Statement	True or False
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.	
Each rectangle on a structure chart represents the user-defined and standard library functions used in a program.	
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.	

(Task #1 Continued) Solve the following problems related to material from Chapter 4:

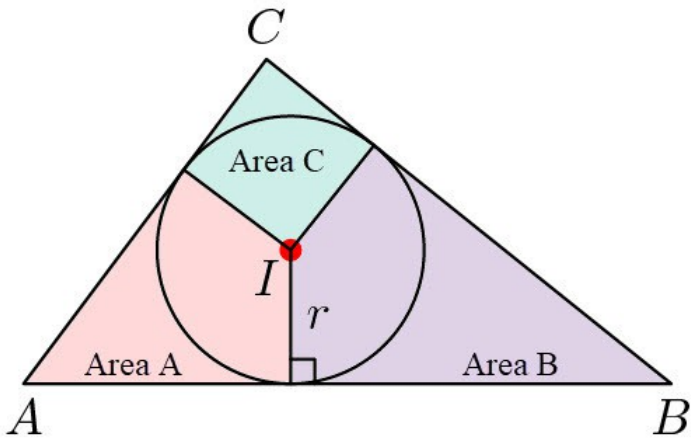
Statement	True / False
It is possible to determine if any parameters are passed to a function by address from the declaration statement of the function.	
It is never possible to determine if any parameters are passed to a function by address from an example call to the function.	
It is possible to determine 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.	
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:

Lab #5 – Programming Assignment
Due: Friday March 5, 2021 at 11:00pm (time local to West Lafayette, IN)
10 Points Possible

Problem: Given three points (A, B, C) that will always create a triangle with a positive area, calculate the measures of its three internal angles and the areas of the three quadrilaterals (Area A, Area B, Area C) created when a perpendicular lines are connected from the incenter to each side of the triangle.

For this assignment you will be **required** to implement the user-defined functions (from chapter 4).



Example Execution #1:

```
Enter X coordinate #1 -> 1.5
Enter Y coordinate #1 -> 1.5
Enter X coordinate #2 -> 2.8
Enter Y coordinate #2 -> 3.8
Enter X coordinate #3 -> 1
Enter Y coordinate #3 -> 6

Internal triangle angles: 35° 48' 57.89", 111° 14' 4.93", 32° 56' 57.17"
Area of quadrilaterals: 1.51, 0.33, 1.65
```

Example Execution #2:

```
Enter X coordinate #1 -> -1
Enter Y coordinate #1 -> 3
Enter X coordinate #2 -> -2
Enter Y coordinate #2 -> -4
Enter X coordinate #3 -> 2
Enter Y coordinate #3 -> -1

Internal triangle angles: 45° 0' 0.00", 45° 0' 0.00", 90° 0' 0.00"
Area of quadrilaterals: 5.18, 5.18, 2.14
```

All course programming and documentation standards are in effect for this and each assignment this semester. Please review this document!

Additional Requirements:

1. Add the **lab assignment header** (`vi` shortcut `hlb` while in command mode) to the top of your program. An appropriate description of your program must be included in the assignment header. Include the Purdue University e-mail addresses of **each contributing group member** 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. 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.** All floating-point variables must be of the `double` type.

Additional Requirements (continued):

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 calls functions.
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 the `guru` server prior to the posted due date to be considered for partial credit. The C-file you submit must be named exactly: `lab05.c`

Course Programming and Documentation Standards Reminders:

- Use the course function header (`head_fx 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.**
- Maximize your use of symbolic/defined constants and minimize your use of literal constants.
- Indent all code found within the `main` and all user-defined functions **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**.
- There is no need to include example output with your submission.

Auto-Grade Tool

- We have implemented what is being referred to as the auto-grade tool. At the time of a successful assignment submission you may receive some feedback on your program in regards to course programming and documentation standards. This feedback may include a potential deduction that you will receive once your assignment is reviewed by your grader.
- It is expected that graders verify those notes identified by this tool to ensure that they are indeed applicable and reasonable to the submission. Graders may make additional deductions for those standards not identified by the new tool.
- We hope that this feedback helps with the enforcement of course standards, consistency in grading across sections, and to encourage students to revise their work when problems are identified before the assignment deadline passes. It is possible to resubmit an assignment for grading up to the advertised deadline. Only the final successful submission is retained and evaluated.