CS 159 – Spring 2021 – Lab #4

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 February 26, 2021 at 11:00pm (LOCAL WEST LAFAYETTE, IN TIME). **No late work will be accepted.**

Weekly Quiz #4:

The weekly quiz will be available (Week 6 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?**
    - This will be the first programming assignment (it's due before the third homework assignment) in which user-defined functions are required. With so much new terminology, syntax, and ideas associated with user-defined functions you should consider this opportunity to collaborate with others as important to improving your understanding of this central programming concept.
    - How will your group factor the tasks found within this problem? Are you making good design choices that take advantage of the benefits associated with function use?
    - Communicate early and often to get off to a good start and to maximize the potential contribution of each team member!

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

Statement	**True / False**
The coding portion of lab #4 requires the use of the specific user-defined functions described in the assignment.	**TRUE**
The function call is an executable statement.	
The function declaration requires the data types and identifiers for each parameter.	
The function call requires the data types and identifiers for each parameter.	
The first line of the function definition requires the data types and identifiers for each parameter.	
The function definition contains executable statements that perform the task of the function.	
The first line of the function definition terminates with a semicolon (`;`).	
The value of a local variable may be returned through a `return` statement of a user-defined function.	
Parameters are defined as local variables in the function header (the first line of the function definition) and should not be re-defined within the local declaration section of the function.	
Additional local variables can be defined in the local declaration section of a function.	
A local variable cannot be referenced through its identifier outside of the function in which it is defined.	
A variable declared in the local declaration section of a function can have the same identifier as one of the parameters of the same function.	
Individual tasks in a program must be factored into individual user-defined functions.	
One benefit of user-defined functions is the potential reduction or elimination of duplicate code.	
A function assignment header for every user-defined function must be inserted immediately above the definition of the function it is documenting.	
Parameters being received by a function will be commented to the right of where they are defined.	
The `return` statement cannot contain an expression.	
Data sent from the calling function to the function being called will be received in the same order in which it was passed.	
The individual task represented by a function should be testable apart from the rest of the program.	
The sequence in which the statements are executed in a program can be altered through the use of functions and function calls.	
A user-defined function may be called more than once in a program.	
The control of the program always returns from the called function to the `main` function.	
A function may return at most one value.	
The role of the `main` function is to coordinate the function calls and to establish the data needs for a program.	

What is the output generated by problem #26 (page 222-223 of your C programming text**)?**

Given the `main` function below, write the function declaration for each user-defined function called by `main`:

```
int main()
{
  float income;
  float rate;

  income = getIncome();
  rate = calcRate(income);

  displayResults(income, rate);

  return(0);
}
```

Given the `main` function below, write the function declaration for each user-defined function called by `main`:

```
int main()
{
  float annualRate;
  int term;
  float principal;
  float moPay;

  annualRate = getAnnualRate();
  term = getTerm();
  principal = getPrincipal();

  moPay = calcPayment(annualRate, principal, term);

  printAnalysis(term * 12, annualRate, principal, moPay);

  printAlternatives(term, annualRate, principal);

  return(0);
}
```

Lab #4 – Programming Assignment
Due: Friday February 26, 2021 at 11:00pm (time local to West Lafayette, IN)
10 Points Possible

Problem: Given three points that will always create a triangle with a positive area, calculate the distance between the points and the coordinate of the incenter found within the triangle created. For this assignment you will be **required** to implement the user-defined functions (from chapter 4).

Example Execution #1:

```
Enter X coordinate #1 -> 13
Enter Y coordinate #1 -> 20
Enter X coordinate #2 -> 25
Enter Y coordinate #2 -> 40
Enter X coordinate #3 -> 30
Enter Y coordinate #3 -> 15

Distance from point 1 to point 2: 23.32
Distance from point 2 to point 3: 25.50
Distance from point 3 to point 1: 17.72
Location of incenter: 22.15, 23.57
```

Example Execution #2:

```
Enter X coordinate #1 -> -5
Enter Y coordinate #1 -> 25
Enter X coordinate #2 -> 15
Enter Y coordinate #2 -> 0
Enter X coordinate #3 -> 10
Enter Y coordinate #3 -> 30

Distance from point 1 to point 2: 32.02
Distance from point 2 to point 3: 30.41
Distance from point 3 to point 1: 15.81
Location of incenter: 5.18, 21.99
```

Example Execution #3:

```
Enter X coordinate #1 -> 0
Enter Y coordinate #1 -> -5
Enter X coordinate #2 -> 35
Enter Y coordinate #2 -> 25
Enter X coordinate #3 -> 30
Enter Y coordinate #3 -> -10

Distance from point 1 to point 2: 46.10
Distance from point 2 to point 3: 35.36
Distance from point 3 to point 1: 30.41
Location of incenter: 21.88, 1.10
```

Example Execution #4:

```
Enter X coordinate #1 -> 55
Enter Y coordinate #1 -> -5
Enter X coordinate #2 -> 4
Enter Y coordinate #2 -> -2
Enter X coordinate #3 -> 30
Enter Y coordinate #3 -> -10

Distance from point 1 to point 2: 51.09
Distance from point 2 to point 3: 27.20
Distance from point 3 to point 1: 25.50
Location of incenter: 30.17, -6.72
```

Example Execution #5:

```
Enter X coordinate #1 -> 5
Enter Y coordinate #1 -> 5
Enter X coordinate #2 -> -5
Enter Y coordinate #2 -> 0
Enter X coordinate #3 -> -2
Enter Y coordinate #3 -> -8

Distance from point 1 to point 2: 11.18
Distance from point 2 to point 3: 8.54
Distance from point 3 to point 1: 14.76
Location of incenter: -1.55, -1.35
```

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, 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.**
   - All floating-point variables must be of the `double` type.
   - All input will be of the integer (`int`) 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: `lab04.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.