CS 159 – Spring 2022 – Lab #8

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. 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 (structure charts, flowcharts) 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 #8 – 5 points possible: The lab quiz will be made available on Brightspace (Week #10 module) during the final 10 minutes of your lab today. The quiz will emphasize material from chapter 6 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 #8 – 5 points possible: This is the first lab assignment that makes use of repetition. Determine which tasks are pretest versus post-test and whether each repetitive process is event-controlled or counter-controlled. These determinations will guide the best choice for implementing repetition in your solution.

- **How might collaboration be useful on this particular programming assignment?** Paper and pencil can guide you through examples to develop the logic to solve this problem. From there the structure chart and specific steps of logic in a flowchart will help advance you further toward a solution.

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? • Lab teams will be re-assigned for the final four labs of the semester. Be prepared! New lecture seats begin March 31.			

Solve the following problems related to material found in Chapter 6 and the course standards.

Statement	**True or False**
The contents of a loop have been identified to be repeated in a program.	
An iteration is one execution of all statements found inside the body of a loop.	
The initialization of the loop control variable must take place outside the body of a pretest loop.	
The condition that determines whether the task to repeat is finished is known as the loop control expression.	
In a pretest loop the control expression is evaluated before each iteration, including the first iteration.	
In a post-test loop the minimum number of times that the statements found inside of the loop are executed is one.	
The action that is responsible for changing the result of the loop control expression from true to false is the loop update.	
The loop control variable is commonly a part of the loop control expression and the recipient of the loop update action.	
In an event-controlled loop we know the number of times that the actions found inside the body of the loop will be executed.	
A counter-controlled loop may execute a constant number of iterations.	
It is possible for the number of times a counter-controlled loop will iterate to depend on the value of a single variable or expression.	
The number of times that the loop control expression is evaluated is one more than the number of iterations in a pretest loop.	
The `while` loop requires the use of parentheses around the loop control expression.	
The `do-while` loop will terminate with a semicolon after its loop control expression.	
Similar to their required use in the `if` construct it is a course standard to always make use of `{` and `}` with all looping constructs.	
A limited amount of control structures are permissible in the `main` function to ensure that it is the `main` function which makes most of the function calls for a program.	
A nested loop is a repetitive process contained inside of another repetitive process.	
One approach to potentially make solving problems that require nested loops easier is to separate each repetitive process into its own function.	
In order for two, or more, repetitive processes to be considered nested they must appear in the same user-defined function.	
Input validation is an example of an event-controlled problem.	
Selection by itself is insufficient for input validation because it provides only a finite number of opportunities for the user to input valid data.	
In this course you will be expected to validate for the input of both the range of acceptable values and the correct data type.	
It is expected for many cases that the code for the validation of input be found in the same function that contains the prompt for input and `scanf` statement.	
The short-circuit method of evaluating logical expressions does not apply to loop control expressions.	
Control-forcing statements such as `break`, `continue`, `exit`, and the use of multiple `return` statements in a user-defined function are prohibited by course standards as mechanisms to terminate repetitive processes.	

Complete the following table from page 309 of the C programming text (Table 6-1):

Pretest Loop		**Post-test Loop**	
# of times loop control expression is evaluated		# of times loop control expression is evaluated	
# of times actions inside the body of the loop are executed		# of times actions inside the body of the loop are executed	
Minimum # of times loop is iterated		Minimum # of times loop is iterated	

Use the table below to trace the execution of the loop provided:

```
int x = 3415263;
int ct = 0;

while(x > 0)
{
 if(x % 2 == 0)
 {
  ct++;
 }

 x = x / 10;
}
```

End of Iteration #	**Value of X**	**Value of CT**
1		
2		
3		
4		
5		
6		
7		
8		

Use the table below to trace the execution of the loop provided:

```
int x = 3415263;
int ct = 0;

do
{
 x = x / 10;

 if(x % 2 == 0)
 {
  ct++;
 }

}while(x > 0);
```

End of Iteration #	**Value of X**	**Value of CT**
1		
2		
3		
4		
5		
6		
7		
8		

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 #8 - 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.

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: Given a number composed of only digits 1 and 2 determine the minimum number of changes you would have to make such that the ending number would be composed of alternating digits 1 and 2.

Example Execution #1:

```
Enter the initial configuration -> 1222121

Fewest # of required changes: 1
```

- Note: the change to `1212121` would necessitate one flip of a 2 digit to a 1 digit.

Example Execution #2:

```
Enter the initial configuration -> 11111

Fewest # of required changes: 2
```

Example Execution #3:

```
Enter the initial configuration -> 12221

Fewest # of required changes: 1
```

Example Execution #4:

```
Enter the initial configuration -> 21212

Fewest # of required changes: 0
```

Example Execution #5:

```
Enter the initial configuration -> 112121212121

Fewest # of required changes: 1
```

Example Execution #6:

```
Enter the initial configuration -> 221212111

Fewest # of required changes: 2
```

Example Execution #7:

```
Enter the initial configuration -> 1112222

Fewest # of required changes: 3
```

Example Execution #8 (validation expectations demonstrated):

```
Enter the initial configuration -> 112113112112

Error! Input must contain only digits 1 and 2!

Enter the initial configuration -> 112112112112

Fewest # of required changes: 6
```

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

Checkpoints During Lab #8:		**TA Verification**
1 – Planning	Begin with a structure chart and then flowcharts for those functions that will make use of repetition. Seek to maximize the benefits of user-defined functions.	
2 – Getting Started in Vocareum	File `lb08.c` created, assignment header completed, `main` function inserted, variables declared and commented, relevant symbolic/defined constants created.	
3 – Implementation of Functions	Use diagnostic print statements demonstrate functions related to required tasks operate as expected. Rotate between members operating the keyboard in between each successfully coded and tested function.	

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.
   - Input validation expectations are demonstrated in the final example execution.
   - The user will always provide a positive integer value and your solution must reject any number that contains a digit other than 1 or 2. No leading zero digits are of concern.
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 **SIX** chapters of the book, notes, and lectures to be acceptable for use.
   - Course standards **prohibit** the use of programming concepts beyond the material found in the first SIX chapters of the book, notes, and lectures. The use of arrays, including character pointers to represent string data would violate requirements of this assignment and result in no credit being awarded for your effort.
6. A program **MUST** compile, be submitted through Vocareum as demonstrated during the lab #0 exercise, and successfully submitted prior to the posted due date to be considered for credit. The C-file you submit must be named exactly: `lb08.c`, no variation is permitted.

Course Programming and Documentation Standards Reminders:

- Code found inside the body of relevant selection and repetition constructs must be indented two additional spaces.
- Make use of `{` and `}` with all relevant selection and repetition constructs.
- See page 258 of your C programming text regarding the proper indentation for a `switch` construct.

Course Programming and Documentation Standards Reminders (continued):

- 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` 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 a function.
  - At no point during the semester should these two sections ever overlap.
- 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.
- Maximize your use of symbolic/defined constants and minimize your use of literal constants.