CS 159 – Spring 2022 – Lab #12

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 #12 – 5 points possible: The lab quiz will be made available on Brightspace (Week #14 module) during the final 10 minutes of your lab today. The quiz will emphasize material from chapters 9-10 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 #12 – 5 points possible

- **How might collaboration be useful on this particular programming assignment?** Open up your course notes packet to page 198 and 199. The program found on these two pages can serve as an example of how you will use dynamic memory allocation in lab #12.

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?			
When is this lab assignment due? Provide month, day, and time.			

Solve the following problems related to material found in Chapters 9-10 of the C programming text:

Statement	True or False
Section 9.1	
The value of a pointer variable can change during the execution of a program.	
The proper initialization of a pointer variable will include an address on the right side of the assignment operator with the pointer variable on the left.	
Working with an uninitialized pointer variable is a logical error.	
Section 9.2	
Every time we want a called function to access a variable in the calling function we pass the address of that variable to the called function.	
A user-defined function may be declared to return a pointer value.	
Section 10.1	
The name of an array is a pointer constant to its first element.	
The name of an integer array can be assigned to an integer pointer variable.	
Section 10.3	
When a whole array is passed to a function the called function can declare the array using the traditional indexing notation `[ ]` or as a simple pointer variable.	
Section 10.4	
The C programming language provides two options for requesting memory, static allocation and dynamic allocation.	
Static memory allocation requires that the declaration and definition of the memory be fully specified in the source program.	
Dynamic memory allocation uses predefined functions to allocate memory for data while the program is running.	
All of the memory management functions are found in the standard library (`stdlib.h`).	
The `malloc` function allocates a block of memory that contains the number of bytes specified in its parameter.	
The memory allocated as a result of the `malloc` function is not initialized and we should assume that it will contain unknown values.	
If we need to know the size of any data type the `sizeof` operator will give us the exact size in bytes.	
The `malloc` function returns the starting address of the memory allocated.	
The result of the `malloc` function is assigned to a pointer variable.	

Solve the following problems related to material found in Chapters 9-10 of the course notes packet:

Statement	True or False
A pointer is a variable that stores a memory address as its value.	
The value stored by a pointer variable may be the location of another variable in the program.	
The data stored in the location to which a pointer is referring can be accessed and manipulated by the pointer.	
An asterisk character (`*`) is used in the declaration of a pointer variable. This asterisk can be attached to either the data type or the name of the variable.	

Solve the following problems related to material found in Chapters 9-10 of the C programming text (continued):

Statement	True or False
A pointer that has been declared on the first line of a function definition will receive its initial value from the calling function.	
Another use of the asterisk character (`*`) is as the indirection operator.	
The indirection operator will access the value stored at the location to which the pointer references.	

- Review and be prepared to answer questions about code similar to that found on page 194 of your course notes packet.

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 #12 - Programming Assignment
Due: At the usual time for your lab section April 26 – April 29. Lab will not meet during this period.
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 positive integer to serve as the seed for the random number generator along with the desired data set size, generate the specified number of integers ranging from 0 to 999 (inclusive).

For every value in the data set that is a multiple of nine, change it to the value in its ones place (example, 81 becomes 1, 135 becomes 5, 0 remains 0). All values that are not multiples of nine can be changed into negative one (-1).

The non-negative data represents locations along a street from which a report was made that the occupants of a home heard a very loud noise, perhaps generated by an unidentified object crashing into the ground. The number (0 through 9) is the distance from the home (in potentially both directions) that the sound was believed to have originated. A number zero would indicate that the sound originated on that property, a number one would indicate the origin is in one of the immediate neighboring properties, and a number five would indicate the sound originated at a property five away from the current.

Display the possible locations on the street from which the loud sound may have originated.

Example Execution #1:

```
Enter seed value -> 1000
Enter desired data set size -> 10

Possible locations: 4 5
```

Original data: 790 779 **513** 398 604 141 803 570 268 **855**
Values updated based on whether they are multiples of nine: -1 -1 3 -1 -1 -1 -1 -1 -1 5
Possible locations as the origin of the loud sound: -1 -1 3 -1 0 0 -1 -1 -1 5

- **Note in the first example that the two properties which reported hearing the loud noise can represent another property in one direction.**
- The "3" at index 2 identifies the property at index 5 (2 + 3) as the source of the noise.
- The "5" at index 9 identifies the property at index 4 (9 - 5) as the source of the noise.
  - Seeking the source in the opposite direction at index 2 (2 - 3) or index 9 (9 + 5) would generate a location that does not exist on the street.

Example Execution #2:

```
Enter seed value -> 7000
Enter desired data set size -> 10

Possible locations: 3 6 7
```

Original data:	412 **405** 104 146 953 **702** 476 132 159 280
Values updated based on whether they are multiples of nine:	-1 5 -1 -1 -1 2 -1 -1 -1 -1
Possible locations as the origin of the loud sound:	-1 5 -1 0 -1 2 0 0 -1 -1

- Note in this second example that the property at index 5 can identify the potential source of the noise at being either index 3 (5 - 2) or index 7 (5 + 2). The property at index 1 can only account for index 6 (1 + 5) as the alternative direction represents a property that does not exist.

Example Execution #3:

```
Enter seed value -> 6000
Enter desired data set size -> 10

Possible locations: None
```

Original data:	480 785 514 **477** 119 813 478 317 652 596
Values updated based on whether they are multiples of nine:	-1 -1 -1 7 -1 -1 -1 -1 -1 -1
Possible locations as the origin of the loud sound:	-1 -1 -1 7 -1 -1 -1 -1 -1 -1

Example Execution #4

```
Enter seed value -> 2685
Enter desired data set size -> 10
Possible locations: 0 2 9
```

Original data:	736 355 **450 846** 727 899 **738 837 198** 412
Values updated based on whether they are multiples of nine:	-1 -1 0 6 -1 -1 8 7 8 -1
Possible locations as the origin of the loud sound:	0 -1 0 6 -1 -1 8 7 8 0

- Note in this example that the property at index 2 identifies itself as the source of the noise.

Example Execution #5 (input validation expectations):

```
Enter seed value -> 0

Error! Positive seed values only!!

Enter seed value -> 1500
Enter desired data set size -> 0

Error! Positive values only!!

Enter desired data set size -> 25

Possible locations: 14 20 22
```

- Note in this example that the data set size is 25. See the additional requirements for more information.

All course programming and documentation standards are in effect for this and each assignment this semester.

Checkpoints During Lab #12:		**TA Verification**
1 – Planning	Begin with a visual representation of your data, a structure chart, and flowcharts for those functions that will make use of repetition. **Demonstrate for your lab instructor that you understand the problem you are attempting to solve.**	
2 – Getting Started in Vocareum	File `lb12.c` created, assignment header completed, `main` function inserted, variables declared and commented, relevant symbolic/defined constants created.	
3 – Accept and Validate Input	See example execution #5.	
4 – Use `malloc` and Generate Data Set	Demonstrate for your lab instructor that you are able to create the original data set seen in example execution #1.	
5 – Multiples of Nine	Demonstrate for your lab instructor that you are able to output the revised data set as -1 for those values that are not a multiple of nine and those that are should be changed to their value modulus ten.	
6 – Identify Possible Locations	Replace those -1 values with zero if they have been identified as a possible location for the noise.	

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 fifth example execution.
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 previously presented material in the first **TEN** chapters of the book, notes, and lectures to be acceptable for use.
    - Failure to use `malloc` will result in no credit for this assignment. No new arrays should be declared using the square braces [ ] in this assignment.
    - Refer to your course notes packet for examples that use the memory allocation function. While it is good to do in practice we will not verify successful allocation of requested memory or attempt to free the memory in your program.
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: `lb12.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.
- 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.