CS 159 – Spring 2021 – Lab #10

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 April 23, 2021 at 11:00pm (LOCAL WEST LAFAYETTE, IN TIME). **No late work will be accepted.**

Weekly Quiz #10:

The weekly quiz will be available (Week 13 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 8 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 evaluate submissions from multiple members of the same group to determine which submission is the official submission of the group. 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?**
  - Sorting is emphasized in the written problems that follow. It is needed at several different points in this program. Does this need for sorting in multiple places require multiple different sorting functions? Or can a single function be created with the "differences" between the sorting needs being passed to the function?
  - Note that the course notes packet and the C programming text both have sorting logic available for use within this assignment. Please comment on your source in the user-defined function header and be sure to bring all code up to course standards. Do not make use of other algorithms or sources outside of the class.
  - Your understanding of each sorting algorithm will be tested on the second midterm exam. How can this assignment help with getting better acquainted with each of the three algorithms introduced this semester?

(Task #1) - Solve the following problems related to material found in Chapter 8 and the course standards.

Statement	True or False
Section 8.5	
A (sort) pass is the movement of one element from the unsorted to sorted sublist.	
The selection sort will identify one value in the unsorted sublist to move and become a part of the sorted sublist.	
The bubble sort operates faster when moving the larger values to the highest index than when moving the smaller values towards index zero.	
The number of exchanges that can potentially occur on a given pass of the bubble sort may be greater than 1.	
Exchanging the `'<'` for a `'>'` in the code on line 21 of page 495 of the C programming text will sort the data in the array from largest (index 0) to smallest (index of `last` minus one).	
The insertion sort takes a value from the unsorted sublist and inserts it into the proper location of the sorted sublist based on the values currently present in the sorted sublist.	
Values once placed in the sorted sublist of the insertion sort are subject to be moved again as the algorithm progresses.	
To sort an array of N elements a N – 1 sort passes are required to guarantee that data always ends in a sorted state.	
In all three sorting algorithms studied the list (array) is divided into two sublists (subsections), sorted and unsorted.	
The outer loop in each of the three sorting algorithms is responsible for ensuring the number of passes required are completed.	
More on Sorting Algorithms - With all problems related to specific sorting algorithms you may want to consider how the statements are true or false for the other sorting algorithms that are not specifically mentioned in the individual problems. See the next three statements as an example.	
The **selection sorting** algorithm will complete one exchange involving at most two elements per pass.	
The **bubble sorting** algorithm will complete one exchange involving at most two elements per pass.	
The **insertion sorting** algorithm will complete one exchange involving at most two elements per pass.	
The selection sorting algorithm can only be used to sort data in an ascending order (from smallest to largest).	
On the final pass through the selection sorting algorithm TWO values are brought over from the unsorted list into the sorted list.	
It is possible that during a single pass of the selection sorting algorithm that the order of the data in the array will be the same as it was after the previous pass.	
The bubble sorting algorithm compares neighboring elements in the unsorted list of the array and swaps their positions when they are not in the desired order.	
The bubble sorting algorithm is optimized to stop the sorting process when the array is detected as being in a sorted state.	
Once the selection sort places a value in the sorted list that value will never move again in the remainder of the passes.	
The insertion sorting algorithm begins with one value in the sorted list before the first pass.	

(More Task #1) - Solve the following problems related to material found in Chapter 8 and the course standards.

Statement	True / False
Section 8.6	
Searching assumptions for each statement unless specified otherwise: • The data in the array is unique. • The amount of data in the array is equal to its capacity. • The use of the binary search is always applied to a sorted array.	
The goal of a searching algorithm is to find the location of a target element inside of an array.	
In general, the use of the sequential search is limited to small data sets or those that are not searched often.	
To determine a target value is not found in an unsorted list while using the sequential searching algorithm every element must be examined.	
One motivation for making use of the binary search instead of the sequential search is the poor worst case performance of using the sequential search with a large data set.	
With each comparison made in the binary search approximately half of the remaining elements in the array are eliminated as possible locations of the target.	
The binary searching algorithm will always find a target in an array faster than the sequential searching algorithm.	
The binary searching algorithm will terminate when the `first` variable is greater than the `last`.	
More on Searching Algorithms	
If a target value is not present in a sorted list then every element of that list must be compared before that fact can be determined.	
The binary searching algorithm can be modified to work with an array that has been sorted from largest (at index zero) to smallest (at index SIZE of the array minus one).	
The binary searching algorithm is not applicable when the values found in the array are not unique.	
Sections 8.7, 8.8, and 8.9 [Arrays with more than one dimension]	
The declaration of a multidimensional array will include the extent, or size, of each dimension.	
The capacity of a multidimensional array is the sum of the extent, or size, of each dimension.	
In the declaration of a user-defined function with a two-dimensional array as a parameter the size of the second dimension is required.	

Solve problems 22-26 from page 544 of your C programming text:

Lab #10 – Programming Assignment
Due: Friday April 23, 2021 at 11:00pm (time local to West Lafayette, IN)
10 Points Possible

Problem: Given up to twenty non-negative decimal integer values as input; separate the data into even and odd numbers then sorting from smallest to largest within each classification. Once the data is sorted as described take each value of the array modulus the data set size and use this number to represent an index and display the total of these elements.

Example data input with set size of six: {11, 2, 9, 6, 3, 4}

Separated into even and odd: {2, 6, 4, 11, 9, 3}

Sorted as described: {2, 4, 6, 3, 9, 11}

Each element modulus data set size: {2, 4, 0, 3, 3, 5}

Data found at above index values: {6, 9, 2, 3, 3, 11} creates a total of 34.

Example Execution #1 (described in example above):

```
Enter up to 20 integer values -> 11 2 9 6 3 4 -1

Sorted array: 2 4 6 3 9 11
Calculated sum: 34
```

Example Execution #2:

```
Enter up to 20 integer values -> 3 5 7 9 1 5 7 -1

Sorted array: 1 3 5 5 7 7 9
Calculated sum: 29
```

Example Execution #3:

```
Enter up to 20 integer values -> 8 6 4 2 -1

Sorted array: 2 4 6 8
Calculated sum: 16
```

Example Execution #4 (the -1 is not needed when the data set size is equal to the maximum possible of 20):

```
Enter up to 20 integer values -> 9 8 7 6 5 4 3 2 1 9 8 7 6 5 4 3 2 1 5 4

Sorted array: 2 2 4 4 4 6 6 8 8 1 1 3 3 5 5 5 7 7 9 9
Calculated sum: 96
```

Example Execution #5:

```
Enter up to 20 integer values -> 20 40 60 80 30 50 70 90 21 11 61 41 81 31 91 71
51 10 0 -1

Sorted array: 0 10 20 30 40 50 60 70 80 90 11 21 31 41 51 61 71 81 91
Calculated sum: 600
```

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 may 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.** Good function design will limit each function to a single task, eliminate redundant logic in the program, and maximize re-use of functions within a program.
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. Selection and repetition constructs in a `main` function are to facilitate the calling of 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 EIGHT chapters of the book, notes, and lectures to be acceptable for use.
    - **The use of** any dynamic array structures (chapters 9 and 10) would violate this requirement and result in **no credit being awarded for your effort.** See course standards below for array declaration expectations.
    - Only a single integer array of size 20 is necessary to solve this problem.
6. A program **MUST** compile, meet assignment requirements, and be submitted through the `guru` server prior to the posted due date to be considered for partial credit. The submission script will reject the submission of any file that does not successfully compile on the `guru` server. The C-file you submit must be named exactly: `lab10.c`

Course Programming and Documentation Standards Reminders:

- It is common to make use of a symbolic/defined constant when the size of the array is known prior to the start of a program.
- The course standards expect all arrays to be of a fixed size. Variable-size arrays, even those demonstrated in chapter 8 of the text, would violate course standards.
- 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.
- Note the standard related to control forcing statements found on page 15 of the course notes packet.
- 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.
- At no point during the semester should the local declaration and executable statement sections of a function 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**.

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.