CS 159 – Spring 2021 – Lab #11

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 30, 2021 at 11:00pm (LOCAL WEST LAFAYETTE, IN TIME). **No late work will be accepted.**

Weekly Quiz #11:

The weekly quiz will be available (Week 15 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 material from chapters 7, 9/10, 11, 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?**
  - Work as a group on the written problems of this lab! The corresponding weekly quiz is the final one of the semester and from many students this is the time of the semester when every point is meaningful!
  - This assignment has a requirement to use dynamic memory allocation as introduced in lecture. Look to the resources of the course for guidance. In CS 159 only a small part of the topic that is dynamic memory allocation is introduced, there is so much more to this idea and external sources are likely to include many techniques never introduced in the course.
  - How many "arrays" are needed this problem? What are they?

(Task #1) - Solve the following problems related to material found in Chapters 9/10 and the course standards.

Statement	True / False
Section 9.1	
A pointer is a variable that stores an address as its value.	
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.	
The indirection operator (`*`) is a unary operator whose operand must be a pointer value.	
The act of dereferencing a pointer variable permits access to the value at the memory location to which the pointer variable points (or refers).	
The asterisk character (`*`) is used in two different contexts for pointer variables; for declaration and for dereferencing.	
The indirection and address operators are the inverse of each other and when combined in an expression they cancel each other.	
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 dereference of an array name is the value of its first element.	
The name of an integer array can be assigned to an integer pointer variable.	
Section 10.2 – only concerned with one-dimensional arrays	
When adding an integer to the name of an array the result is a value that corresponds to another index location.	
The size of an element is determined by the type of the pointer.	
The following expressions are identical for the array `a` and integer `n`: `*(a + n)` and `a[n]`	
The following expressions are identical for the array `a` and integer `n`: `(a + n)` and `&a[n]`	

The following code segment uses pointer arithmetic to access and display the contents of the array.
 	
`int i;`
`int a[5] = {1, 3, 5, 2, 4};`
 	
`for(i = 0; i < 5; i++)`
`{`
`  printf("%d ", *(a + i));`
`}`
 	
`printf("\n");`	

Solve the following problems related to material found in Chapters 9/10 and the course standards.

Statement	True or False
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 Chapter 7 and the course standards.

Statement	True or False
If using an external file (via a **file pointer**) for input to a program then all input for the program must come from the external source.	
The value returned from the `fopen` function is used to initialize a **file pointer**.	
The name of the external file being accessed is case insensitive when referenced in the `fopen` function call.	
The name of the external file is the **parameter** to the `fclose` function.	
The `fclose` function can only be used to a close a connection that has been successfully opened.	
The `fscanf` and `fprintf` functions reference the name of the **file pointer** but NOT the name of the external file (as one of its **parameters**).	
When the **file pointer** stores a value equal to zero it means the external file failed to open as expected.	
A **file pointer** used to read external data into a program should always be compared with an error value to determine the data file was opened as expected before allowing the program to continue.	
The value returned from the `fscanf` function is the data value that was read from the external file.	
The value of a **file pointer** must be returned when making use of a user-defined function to open an external file. [In other words, **file pointers** ARE NOT passed by address.]	
A value of the `FILE*` type cannot be used as a parameter to a user-defined function.	
The `EOF` constant needs to be defined by the programmer before it can be used in your program.	

Solve the following problems related to material found in Chapter 11 and the course standards.

Statement	True or False
Section 11.1	
A string is a series of characters treated as a unit.	
A character known as a delimiter can be stored within a string to mark the end of the string data.	
Section 11.2	
A string is stored in a character array.	
A delimiter (null) character is used by the standard string functions to detect the end of the string data.	
Section 11.5	
The `strlen` function from `string.h` will return the total number of characters in the array up to and including the delimiter (null) character.	
The `strcpy` function will copy the contents of the first character array parameter into the second.	
If the two strings being compared by the `strcmp` function store the same data then the value returned will be one.	

Lab #11 – Programming Assignment
Due: Friday April 30, 2021 at 11:00pm (time local to West Lafayette, IN)
10 Points Possible

Problem: Given a positive integer representing the size of an integer data set, accept input of that data set, and display the data sorted by odd and even with one additional requirement, an index that stores an odd value at the time of input must store an odd value after the data is sorted with the same requirement holding for even values.

Example data input with set size of ten: {10, 8, 9, 7, 6, 4, 5, 3, 2, 1}

Even values are found at indexes: 0, 1, 4, 5, 8

Odd values are found at indexes: 2, 3, 6, 7, 9

Final result after odd and data values are sorted: {2, 4, 1, 3, 6, 8, 5, 7, 10, 9}

Other Requirements:

- The size of the data set is unknown until the program is running. This will **necessitate** the use of pointers and the `malloc` function to create all of the arrays in your solution.
- Excessively large arrays or extra unnecessary arrays will be consider a deduction in the **technique** score when your program is evaluated.

Example Execution #1 (described in example above):

```
Enter data set size -> 10
Enter 10 integer values -> 10 8 9 7 6 4 5 3 2 1

Final data set order: 2 4 1 3 6 8 5 7 10 9
```

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

Example Execution #2:

```
Enter data set size -> 12
Enter 12 integer values -> 12 10 8 6 4 2 11 9 7 5 3 1

Final data set order: 2 4 6 8 10 12 1 3 5 7 9 11
```

Example Execution #3:

```
Enter data set size -> 9
Enter 9 integer values -> 17 15 13 11 9 7 5 3 1

Final data set order: 1 3 5 7 9 11 13 15 17
```

Example Execution #4:

```
Enter data set size -> 14
Enter 14 integer values -> 20 18 16 14 12 10 9 22 24 26 28 8 6 4

Final data set order: 4 6 8 10 12 14 9 16 18 20 22 24 26 28
```

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 ten chapters of the book, notes, and lectures to be acceptable for use.
    - It remains acceptable to use sorting logic and code from the course notes, lectures, and/or the official C programming text of the course. Be sure to reference your source in the header of the user-defined function and to bring all code up to course standards.
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: `lab11.c`

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.
- Note the standard related to control forcing statements found on page 15 of the course notes packet.

Course Programming and Documentation Standards (continued):

- 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.