CS 159 – Fall 2020 – Lab #8

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 November 6, 2020 at 11:00pm (LOCAL WEST LAFAYETTE, IN TIME). **No late work will be accepted.**

Weekly Quiz #8:

The weekly quiz will be available 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 6 and chapter 8 material, the written problems in this document, the lab programming assignment, quiz questions found in the In-Person lecture recordings, 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 team?**
  - **On-campus students:** Your lab instructor should have e-mailed you and included the contact information of your lab partners.
  - **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. A partner that was given an opportunity to participate but fails to do so should have their e-mail excluded from the assignment header.
- **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 was 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 **career account login must appear in the assignment header**. The assignment header can be added to your file while you are editing it in `vi`. While in command mode you need only to enter `hlb` followed by the enter key to insert the header at the current location of the cursor.
- **How might collaboration be useful on this particular programming assignment?** In addition to the usual planning of user-defined functions this lab is the first programming assignment in which arrays are permitted and their use also requires planning, consider the following questions; What is the desired data to store? How will that data get into the array? How will the data be analyzed? Which functions will use an array as a parameter?

(Task #1) - Solve the following problems related to material found in Chapter 6 and the course standards.

Statement	**True or False**
The condition in a recursive function when which the recursive function calls stop is known as the base case.	
Recursion should not be used with event-controlled processes as the result may be more function calls than the memory of the computer can accommodate.	
Recursion is a repetitive process in which a function calls itself.	
An iterative solution involves the use of a loop to solve a repetition problem.	
Iterative solutions are always better than recursive ones.	

Use the table below to trace the execution of the recursive program below:

```
int calcSum(int);

int main()
{
  int x = 35564;
  int sum;

  sum = calcSum(x);

  return(0);
}

int calcSum(int x)
{
  int sum = 0;

  if(x > 0)
  {
    sum = x % 10;
    sum += calcSum(x / 10);
  }

  return(sum);
}
```

Function Call #	**Received value of X**	**Returned value of sum**
1		
2		
3		
4		
5		
6		
7		
8		

Identify the base and recursive cases in the code segment above:

Solve the following problems related to material found in Chapter 8 and the course standards.

Statement	**True / False**
Section 8.2	
In a fixed-length array the size of the array is known when the program is written.	
Arrays must be declared and defined before they can be used.	
Array declarations will determine the type, name, and size of the array.	
For a value to be potentially used as an index it must be an integral value or an expression that evaluates to such.	
The name of an array is a reference to the address of where it begins inside the memory of the computer.	
The index value represents an offset from the beginning of the array to the element being referenced.	
Declaration and definition of an array will include a default initialization of all elements.	
If the number of values provided for initialization of an array is fewer than the size of the array then the remaining elements have no known value.	
The address operator is not necessary in a `scanf` to accept input for an individual array element when using the indexing technique.	
When accessing an array element the C language does not check whether the index is within the boundary of an array.	
Section 8.3	
Arrays can be passed in two ways; by individual elements or the whole array.	
Elements of an array, themselves individual values of a given data type, are passed by value from calling to called function.	
The called function cannot identify whether the value it receives comes from an array, an individual variable, or an expression that evaluates to the expected type.	
Individual elements of an array can be passed by address through the use of the address operator.	
The reason that the C language does not pass whole arrays by value is the extra stress it would put on the memory of the computer to make a copy of an array.	
The name of an array is a primary expression whose value is the address of the first element in the array.	
Indexed references to individual elements of an array are simply calculated addresses where the index value is added to the address represented by the name of the array.	
Passing the array name to a function allows changes in the called function to be available back in the calling function after it terminates.	
When passing a whole array to the function the total size of the array is necessary in the function call.	
When passing a whole array to the function the total size of the array is necessary in the definition of the called function.	
It is only the starting point of the array in memory that is represented by the name of an array and not the ending point.	

What are the values in the array after the code below has been executed?

```
int x[5] = {1, 3, 7};
int i;

for(i = 0; i < 4; i++)
{
  x[i + 1] += x[i];
}
```
	```
	int x[5];
	int i;

	for(i = 0; i < 4; i++)
	{
	  x[i] = 2 * i - 1;
	}
	```

Additional Problems

Statement	**True / False**
It is a course standard to make use of a symbolic/defined constant to represent the size of a statically declared array.	
The conversion code to use for input or output of an array element depends on the data type of the array.	
Variables and loops are commonly used together to generate index values to access the elements of an array.	
Arrays in the C programming language use a one-based index.	
To pass the whole array to a function you need to use the name of the array followed by empty square braces `[]` in the function call statement.	
All elements of one array can be assigned to another through the use of the assignment operator and the name of each array (example: `x = y`).	
While the default technique of passing array elements is by value it is possible to pass elements by address using the `&` operator (and the `*` operator in the function being called).	
If more than one element of an array is passed to a function in a single function call then those elements are passed by address.	
Using the name of an array in the data list of a single `printf` function will result in the output of all elements of the array.	
All arrays sent to a given user-defined function must be of the same defined size.	

Lab #8 – Programming Assignment
Due: Friday November 6, 2020 at 11:00pm (time local to West Lafayette, IN)
10 Points Possible

Details on Random Number Generation: The problem which follows requires the use of the random number generator as found in chapter 4 of your C programming text (see page 191). For it to be possible to replicate our examples below your program must first accept a "seed" integer value from the user. More information on how seeding works can found in example 4-13 of your C programming text.

Problem: In this lab you are working with a set of integers that will range from 1 to 100 (inclusive of the end points). The integers will be created by the random number generator. Given a seed value for the random number generator and the size of the data set; display those values from 1 to 100 (inclusive of the end points) not present in the data set and those values that are present at least 2.5% of the data set size (take only the whole number of this value).

Example Execution #1 (2.5% of 200 is 5):

```
Enter seed value -> 1047
Enter data set size to generate -> 200

Values not present in data set: 17 18 23 28 31 33 36 57 63 73 99
Values present at least 5 times in data set: 20 56 85
```

Example Execution #2 (2.5% of 199 is 4.975, take only the integer 4 and truncate the remainder):

```
Enter seed value -> 1047
Enter data set size to generate -> 199

Values not present in data set: 16 17 18 23 28 31 33 36 57 63 73 99
Values present at least 4 times in data set: 20 25 34 38 56 69 80 85 93 94
```

Example Execution #3:

```
Enter seed value -> 17566
Enter data set size to generate -> 160

Values not present in data set: 8 10 45 50 53 54 60 65 76 79 97 100
Values present at least 4 times in data set: None
```

Example Execution #4:

```
Enter seed value -> 4547
Enter data set size to generate -> 279

Values not present in data set: 5 72
Values present at least 6 times in data set: None
```

Example Execution #5:

```
Enter seed value -> 8292
Enter data set size to generate -> 250

Values not present in data set: None
Values present at least 6 times in data set: 23 57
```

Example Execution #6:

```
Enter seed value -> 5305
Enter data set size to generate -> 300

Values not present in data set: None
Values present at least 7 times in data set: None
```

Example Execution #7 (input validation expectations demonstrated):

```
Enter seed value -> 0

Error! Seed must be a positive value!

Enter seed value -> 70430
Enter data set size to generate -> 0

Error! Minimum data set size is one!

Enter data set size to generate -> 275

Values not present in data set: 1 100
Values present at least 6 times in data set: 34 38 43 44 64
```

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.
   - Only integer (`int`) input will be used to test your program.
   - Input validation requirements can be seen in the seventh example execution. The smallest acceptable seed and data set size is one.
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 function calls. Selection and repetition constructs may be present in `main` to assist with 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 eight chapters of the book, notes, and lectures to be acceptable for use.
   - Each of the available selection and repetitions constructs, logical operators, and relational operators are available for use in this assignment. **Do not** make use of the `bool` data type due to the extra include required.
   - **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.
6. A program **MUST** compile 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 name of the source code file you attempt to submit must be `lab08.c`, no variation is permitted.

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` function **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.

Course Programming and Documentation Standards Reminders (continued):

- 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.