CS 159 – Spring 2021 – Lab #7

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 March 26, 2021 at 11:00pm (LOCAL WEST LAFAYETTE, IN TIME). **No late work will be accepted.**

Weekly Quiz #7:

The weekly quiz will be available (Week 10 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 5 and 6 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 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 **Purdue University e-mail address must appear in the assignment header**.
- **How might collaboration be useful on this particular programming assignment?**
  - The logic and length of this programming assignment are not nearly what you've been experiencing on recent and current homework assignments. There are a lot of little, but very important, programming concepts and course standards within this problem. We will build upon these concepts, not necessarily the numbering systems problem, for the remainder of the term. Take the time to collaborate with your lab partners on the written problems and reviewing the idea of numbering systems before you begin to write any code.
  - Did your first midterm score not meet your expectations? Share that with your lab partners and insist on a collaborative experience that will help all learn the course material and ensure a complete effort for lab #7.

(Task #1) - Solve the following problems related to material found in Chapter 5, Chapter 6, and the course standards.

Statement	**True or False**
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.	**TRUE**
Section 5.3	
Multiway selection chooses among several alternatives.	
When different variables are being evaluated as expressions it is better to use a nested `if...else` rather than the multiway `else-if`.	
The `switch` construct can only be used when the selection condition reduces to an integral expression.	
The control expression that follows the keyword `switch` may be a character expression.	
When the selection is based on a range of values, or the condition is not integral, we use the `else-if` for our multiway selection needs.	
The `case` label represents an integral type value that is a possible result of the control expression.	
Each `switch case` label is the keyword `case` followed by a constant expression inside of single quotes.	
Associated with a `case` label is zero or more executable statements.	
When the statements associated with one `case` have been executed the program flow continues with the statements for the next `case`.	
The course standards limit the use of the `break` statement to only `switch` statements.	
No two `switch case` labels can represent the same constant expression value.	
It is always a logical error to associate two `switch case` labels with a common set of actions.	
The maximum number of actions that can be associated with a `switch case` label is one.	
The `break` statement results in the control of the program exiting the `switch` statement.	
In an `else-if` the `if` condition is evaluated first and additional `else-if` conditions are evaluated until a condition is found to be true.	
Only the statements associated with the first true condition are executed in a multiway `else-if` construct.	
The `else` is executed only when all previously evaluated conditions are false.	
The `else` does not have a condition associated with it.	
Section 5.6	
Negative logic refers to any expression that begins with a NOT operator or that contains multiple NOT operators within.	
Complementing a condition is one way to potentially remove negative logic from an expression.	
The only way to complement a NOT operator is with another NOT operator.	
When writing a selection construct the most probable conditions should come before those that occur less frequently.	
It is possible to indicate on a structure chart when a user-defined is called from within a section construct.	

Solve the following problems on pages 290-293 of your C programming text: 30, 31, and 32.

Solve the following problems related to material found in Chapter 5, Chapter 6, and the course standards.

Statement	**T/F**
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.	
Course standards prohibit the use of `break` to terminate any repetition construct.	
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		

Lab #7 – Programming Assignment
Due: Friday March 26, 2021 at 11:00pm (time local to West Lafayette, IN)
10 Points Possible

Problem: Given a non-negative decimal (base 10) integer value as input display the octal (base 8) equivalent. More information on numbering systems can be found on page 1033 of your C programming text. The program is also expected to display the change in odd and even digits between the two numbers (base 10 and base 8 equivalent).

- At no point in the process of solving this problem will a value greater than that which can be represented by an `int` type will be input, generated, or output.

Example Execution #1 (one more odd digit and one less even digit in the octal equivalent):

```
Enter decimal value -> 123123

Octal equivalent: 360363
Odd digit change: +1
Even digit change: -1
```

Example Execution #2 (one more odd digit in the octal equivalent):

```
Enter decimal value -> 9

Octal equivalent: 11
Odd digit change: -1
Even digit change: 0
```

Example Execution #3:

```
Enter decimal value -> 123456

Octal equivalent: 361100
Odd digit change: 0
Even digit change: 0
```

Example Execution #4 (input validation expectations demonstrated):

```
Enter decimal value -> -3

Error! The decimal value provided should be non-negative.

Enter decimal value -> 24680

Octal equivalent: 60150
Odd digit change: -2
Even digit change: +2
```

Example Execution #5:

```
Enter decimal value -> 9999

Octal equivalent: 23417
Odd digit change: +1
Even digit change: -2
```

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 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 requirements can be seen in the fourth example execution. Review the course notes packet for course expectations of input validation.
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 calls 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 SIX chapters of the book, notes, and lectures to be acceptable for use.
   - 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 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: `lab07.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.
- 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.
- 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.