CS 159 – Spring 2021 – Lab Exam #3

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 April 16, 2021 at 11:00pm (LOCAL WEST LAFAYETTE, IN TIME). **No late work will be accepted.**

What previous code can be referenced for this assignment? The problem is similar to that of previous labs this semester. You are permitted to access work previously developed by **yourself** for a homework assignment, work that was previously developed **by your lab team** for which you were credited as a contributing member, and any code found in the lectures, course notes packet, and C programming text of the course.

What help can be had from other individuals for this assignment? None. The course staff will respond to requests regarding understanding of the problem but questions regarding your solution and its development are not allowed. **Working with any other individual is prohibited.** All solutions will be compared for similarity. Any code identified on-line relevant to this assignment will be included in the similarity comparison. A violation of course policies as they relate to academic integrity will result in a **failing grade for the course** and a referral to the Office of the Dean of Students for additional disciplinary action.

Lab Exam #3 – Programming Assignment
Due: Friday April 16, 2021 at 11:00pm (time local to West Lafayette, IN)
20 Points Possible

Problem: Given as input a numbering system for odd integers and a second numbering system for even integers, accept as input a data set consisting of up to 20 non-negative integers and display the conversion of all odd numbers to the specified numbering system followed by the even numbers to the second specified numbering system. All data input and output are guaranteed to be within the limits of the `int` data type.

Example Execution #1 (data set terminates with -1 when limit of twenty is not reached):

```
Enter base for odd numbers -> 2
Enter base for even numbers -> 8
Enter up to 20 integer values -> 1 2 3 4 5 6 7 8 9 -1

Binary values: 1 11 101 111 1001
Octal values: 2 4 6 10
```

Example Execution #2:

```
Enter base for odd numbers -> 3
Enter base for even numbers -> 9
Enter up to 20 integer values -> 1 2 3 4 5 6 7 8 9 10 11 -1

Ternary values: 1 10 12 21 100 102
Nonary values: 2 4 6 8 11
```

Example Execution #3 (-1 value to terminate input is not needed when data set is filled to capacity):

```
Enter base for odd numbers -> 7
Enter base for even numbers -> 4
Enter up to 20 integer values -> 10 11 12 13 14 15 16 17 18 19 20 1 2 3 4 5 6 7 8 9

Septenary values: 14 16 21 23 25 1 3 5 10 12
Quaternary values: 22 30 32 100 102 110 2 10 12 20
```

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

Example Execution #4:

```
Enter base for odd numbers -> 6
Enter base for even numbers -> 5
Enter up to 20 integer values -> 1 2 3 4 5 6 7 8 9 19 18 17 16 15 14 13 12 11 10 -1

Senary values: 1 3 5 11 13 31 25 23 21 15
Quinary values: 2 4 11 13 33 31 24 22 20
```

Example Execution #5 (input validation expectations demonstrated):

```
Enter base for odd numbers -> 0

Error! Base must be in the range from 2 to 9.

Enter base for odd numbers -> 2
Enter base for even numbers -> 10

Error! Base must be in the range from 2 to 9.

Enter base for even numbers -> 3
Enter up to 20 integer values -> 200 300 400 -1

Binary values: None
Ternary values: 21102 102010 112211
```

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.
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.** Example execution #5 displays the expectation for input validation.
3. For this assignment you will be **required** to implement 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. Your solution must make good use of the limited resources of the computer. Avoid wasting memory through excessive variables, especially multiple arrays, and use logic that solves the problem quickly without time-consuming and unnecessary iterations. Matters of technique and design are considered during evaluation.
6. 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 **character arrays**, including character pointers (chapter 11) to represent string data, would violate requirements of this assignment and **result in no credit being awarded for your effort.**
   - **The use of** any dynamic array structures (chapters 9 and 10) would violate this requirement and results in **no credit being awarded for your effort.** See course standards below for array declaration expectations.
7. 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 any file that does not successfully compile on the `guru` server. The C-file you submit must be named exactly: `exam03.c`

Selected 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**.

When you submit... only the final successful submission is kept for grading. All other submissions are over-written and cannot be recovered. You may make multiple submissions but only the last attempt is retained and graded.

- Verify in the confirmation e-mail sent to you by the course that you have submitted the correct file (must be named `exam03.c`), to the correct assignment (`exam03`), and to the correct section.
- Leave time prior to the due date to seek assistance should you experience difficulties completing or submitting this assignment.
- All attempts to submit via a method other than through the `guru` server as set up in the Account Configuration Activity will be denied consideration.

Assignment deadlines... are firm and the electronic submission will disable promptly as advertised. We can only grade what you submit as expected prior to the assignment deadline.

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.