CS 159 – Spring 2021 – 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 April 2, 2021 at 11:00pm (LOCAL WEST LAFAYETTE, IN TIME). **No late work will be accepted.**

Weekly Quiz #8:

The weekly quiz will be available (Week 11 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 6 material, the written problems in this document (including redirection and other UNIX tools), 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?**
  - Please establish a plan within your group to complete this assignment at least 24 hours before it is due. This will go a long way to ensure that a successful submission is made and allow for time to seek assistance with any unexpected errors.
  - The decimal to octal conversion and calculating the difference of odd and even digits between the two numbers were already solved in the previous lab. How do you take those existing functions and now repeatedly use them for a range of values? It is likely that a `for` loop finds an application here.
  - Review as a group the use of selection and repetition inside of the `main` function as described in the course standards.

(Task #1) - Solve the following problems related to material found in Chapter 6, redirection, and 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**
According to the course standards a `for` loop should only be used with counter-controlled processes.	
According to the course standards if all three expressions are not needed in a `for` loop then you should instead make use of a `while` loop for your pretest looping needs.	
A reasonable effort should be made to convert most `while` loops into `for` loops.	
The number of times the update action is executed is equal to the number of times the loop control expression is evaluated in a `for` loop.	
It is possible to update/change/alter the loop control variable of a `for` loop inside of its body.	
You can make use of `x++`, `++x`, `x += 1`, and `x = x + 1` interchangeably as the update (third) expression of a `for` loop to increment the loop control variable.	
The `gcc` compiler as used on the `guru.itap.purdue.edu` server this semester will permit a variable to be declared and initialized in the first expression of a `for` loop. See examples on pages 318-319 of your C programming text.	
This `for` loop will iterate 10 times: `for(i = 0; i < 10; i++)`	
This `for` loop will iterate 5 times: `for(i = 12345; i != 0; i /= 10)`	
This `for` loop will iterate 6 times: `for(i = 1; i <= 32; i * 2)`	
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.	

Redirection Problems:

Given an executable file (`a.out`) that accepts input from the keyboard and produces output to the monitor.

1. How would you redirect input from a file called `labData` into the executable file?

2. How would you redirect both input to the executable file from a file called `labData` and redirect output to another text file called `results`?

3. What makes it difficult to redirect output only while permitting input to come from the keyboard?

4. What takes place when you attempt to redirect output to an existing file?

5. How can output be redirected to append to the end of an existing file?

More useful UNIX commands for working with files and file management:

Move to your `lab00` directory where you should still have a file named `lab00.c`. If you do not have this file then refer to any previous `.c` file assignment which includes on top the course assignment header.

Heads or Tails?

Enter the following command: `tail lab00.c`

And next: `head lab00.c`

And another command: `more lab00.c`

- What do these commands do?

Try the following variations of the `tail` and `head` commands.

- `tail -3 lab00.c`
- `tail -13 lab00.c`
- `head -5 lab00.c`
- `head -15 lab00.c`

- What does the number in each of the above commands control?

Redirection of the Heads and Tails Commands – Output

Enter the following command: `head -5 lab00.c > out_test`

- What is contained in the newly created `out_test` file?

Now try: `tail -5 lab00.c >> out_test`

- How did the contents of the `out_test` file change?

Lastly, try: `ls >> out_test`

- How did the contents of the `out_test` file change?

```
 1 #include<stdio.h>
 2
 3 int calcSum(int);
 4
 5 int main()
 6 {
 7   int x = 35564;
 8   int sum;
 9
10   sum = calcSum(x);
11
12   return(0);
13 }
14
15 int calcSum(int x)
16 {
17   int sum = 0;
18
19   printf("Received value of x: %d\n", x);
20
21   if(x > 0)
22   {
23     sum = x % 10;
24     sum += calcSum(x / 10);
25   }
26
27   printf("Returned values of sum: %d\n", sum);
28
29   return(sum);
30 }
```

What is the output of the recursive program above?

How many total times if the `calcSum` function called in the program above?

What is the condition of the base case of the recursive function in the program above?

Lab #8 – Programming Assignment
Due: Friday April 2, 2021 at 11:00pm (time local to West Lafayette, IN)
10 Points Possible

Problem: Given a range of non-negative decimal (base 10) integer values as input display the smallest value in the range that represent **the largest change in odd and even digits when comparing the decimal value to its octal** (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:

```
Enter starting decimal value -> 200
Enter ending decimal value -> 700

-=-==========-==========-Change-Decimal-=-Octal-
Largest odd digit change:      -3     600    1130
Largest even digit change:      2     200     310
```

Example Execution #2:

```
Enter starting decimal value -> 700
Enter ending decimal value -> 900

-=-==========-==========-Change-Decimal-=-Octal-
Largest odd digit change:      -3     840    1510
Largest even digit change:     -2     770    1402
```

Example Execution #3:

```
Enter starting decimal value -> 11111
Enter ending decimal value -> 33333

-=-==========-==========-Change-Decimal-=-Octal-
Largest odd digit change:       4   11312   26060
Largest even digit change:     -4   11312   26060
```

Example Execution #4 (input validation expectations demonstrated):

```
Enter starting decimal value -> -1

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

Enter starting decimal value -> 50
Enter ending decimal value -> 50

Error! The ending value provided should be greater than 50.

Enter ending decimal value -> 51

-=-==========-==========-Change-Decimal-=-Octal-
Largest odd digit change:       1      50      62
Largest even digit change:     -1      50      62
```

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: `lab08.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.