Introduction

In this week's lab, you will develop utilities to perform basic string modifications.

Lab Objectives

By successfully completing today's lab, you will be able to:

- Conduct a brainstorming session with one or two individuals in the class.
- Create a planning document outlining your solution to the code.
- Implement a programming project that will allow a user to specify modifications to strings.

Prior to Lab

- You should be familiar with input and output, conditionals such as if, if/else, for loops, and arrays.
- This lab corresponds with zyBooks Chapters 2.15, 3.12, 5.10-5.12 and 6 (6.10 is especially helpful).

Deadline and Collaboration Policy

- Part 2 of this assignment is due by 11:00 PM on Tuesday (10/16/2018) via Canvas.
- Part 3 of this assignment is due by 11:00 PM on Friday (10/19/2018) via Canvas.
    - More instructions on what and how to submit are included at the end of this document.
- You should write your solutions to Part 2 and Part 3 of this lab by yourself. In this lab, you will talk at a high level with others in your lab about a solution. However, you will create all code by yourself and **you should not talk about specific code to anyone but a course instructor or lab teaching assistant.**
- Your zyBooks chapters and lecture slides are available resources you can use to assist you with this lab.
- When seeking additional assistance on this lab from a TA, you must have your planning document with you. You should be able to use your planning document to help explain your code and what you're working on and having trouble with. This should hopefully make the process easier, and give you better feedback.

Lab Instructions

Good programmers:

- Think before they write code.
- Structure their code so that it's modular. That is, a program is broken up into well-defined groups of statements called a function. In a well-defined and well-structured program, the main is often considered just a **driver** that calls functions that do the actual work. The best functions are those that are well-named (that is, the name clearly reflects what the function does) and typically only perform one process.
- Write their code to gracefully deal with errors.

In this week's lab, you will create a tool in which a user can enter a string (up to 25 characters) and choose how they wish to manipulate the string from a menu your program will display (see the run-through). The program will continue to ask for commands until the user enters the "quit" command. The commands are:

- **"Replace All"** If a user types "replace all" using ANY sort of capitalization, your program will prompt the user to enter the character to change and the new character. Afterwards, it will replace all instances of

the first character with the new character and will printout the new string. This method does change the original sentence entered.

- **"Quit" –** If the user types "quit" with any sort of capitalization, the program will stop running.

Note: Examples of capitalizations to consider are ALL CAPS, all lowercase, or mIxED CapITAlIzAtIon.

For this lab, you need to make sure your program can gracefully handle user errors. That is, simply stating "An error occurred" does not help a user recover from a mistake. Specific errors your program should be capable of fixing (not a complete list) are:

- If the user enters a character that does not exist within the string, your program should display an error and prompt them for the next command.
- Case matters within the run of each command. If a user requests that the letter 'q' get replaced, your program should not replace any capital 'Q' letters.
- If the user enters an invalid command, you should display an error message and ask for a valid command.

Example Run:

```
% prompt: ./a.out

Enter a string (up to 25 characters) to be transformed: Go Tigers

Enter your command (quit, replace all): REPLACE ALL
Enter the character to replace: e
Enter the new character: 3
Your new sentence is: Go Tig3rs

Enter your command (quit, replace all): replace some aaaaaaaaaaaa12341234
Sorry, that command is invalid. Please type one of the options.

Enter your command (quit, replace all): replace all
Enter the character to replace: i
Enter the new character: 1
Your new sentence is: Go T1g3rs

Enter your command (quit, replace all): rePLace ALL
Enter the character to replace: q
Error, q is not in the string.

Enter your command (quit, replace all): QUIT

% prompt:
```

Assignment

Lab 08 Structure	
Tuesday	Thursday
30 minutes: work with 1-2 other individuals on Part 1 20 minutes: work individually on Part 2 Submit results of Part 2 to Canvas by 11:00PM.	50 minutes: work individually on Part 3 (You may start working on this prior to Thursday after you leave lab on Tuesday) Submit to Canvas by 11:00PM on Friday.

Part 1: Planning with a Group (First 30 minutes)
***** Do not use a computer or calculator in this part *****

This lab may look simple on the surface, but there are lots of subtle requirements that you need to consider while making your programs. Do not rush through this part. To encourage you to spend more time clearly thinking about the problem, the planning document that you will submit in Part 2 will be graded for sufficiency and depth.

Working with the people in your row, brainstorm what will be required for this program to accomplish its goal. Make sure everyone in your row has a good understanding of the requirements and the structure of your solution before moving on, and be sure to read the specifications in Part 3 to understand how your program should be constructed. Remember, you have to do all string manipulation in the `replaceAll` function, and your main should only be used to prompt the user for input, and to call the `replaceAll` function.

Specific items that you should consider with your group during your planning session:

- What's proper syntax to declare a function in C?
  - What are the required arguments, if any?
  - What should each function return?
- How can you ignore the case of a command?
- What is the general process to undertake for replacing a single or all characters?
- Are there any additional errors that someone might encounter? How can you deal with these?

Part 2: Planning on Your Own (20 minutes of Tuesday Lab)
***** It's okay to use a computer and/or calculator for this portion *****
Note: This should not be when you're coding your solution.

Make sure that you follow the steps below, using the guidelines and sample code above, to write and then submit your document before you begin.

Taking the information that was created during your brainstorm, write your own solution *in plain English*. How you choose to structure the planning document is of your choice (e.g., writing in a narrative form, list of steps, an outline), but you need to do the following:

- Thoroughly consider all the issues you need to fully create a program as specified above (such as how to make the formatting look correct, how many variables do you need, what type, etc.).
- Avoid "hand-waving" (that is, skipping steps or avoiding details) or simply restating the problem. "Replace all the characters with what the user entered" will not be helpful in coding your solution. Instead, break down all the steps required to do so, making note of what you'll need to do it.
- Include the following mandatory header:
  - Your First (optionally, their Preferred name) and Last Name

- Your Lab and Lecture Section (e.g., CPSC 1010-001 and CPSC 1011-003)
- Your CID number
- Lab 08 – Part 2
- Today's Date
- Collaboration Statement: In this statement you indicate who you worked with, and which lecture sections they are enrolled in.

Submit this part of the assignment to Canvas by 11:00PM (as specified in the submission Guidelines). Late submissions will not be accepted. This part is to help you think before you code.

Part 3: Implementation (50 minutes on Thursday).

Lab Exercise

Write a program called `string_manipulator.c` that accomplishes the requirements listed earlier in this assignment. Additional program specifications include:

- **Your main should not manipulate any strings or do any tasks aside from gathering user input, and calling functions as needed.**
- Your program should include a function called `replaceAll` that replaces all characters in the input string with the desired new character. **This is the only place where any sort of string manipulation should occur. You are responsible for writing the complete declaration and implementation of this function.** You can choose to implement this function above your main, or afterwards with a declaration above the main. This function can take any parameters you deem appropriate.
- Once the program starts, all error messages should display helpful messages and then re-prompt the user with the choices (as shown in the sample run through). Your handling of all possible user input cases is being graded more heavily on this assignment than previously. (See the rubric).
- Your output formatting should mimic the sample output provided in this document. Do not add extra spacing or change the wording.

You should always break up your programs into small parts, doing each one incrementally, compiling in between each step before continuing to the next step. Working in smaller sections to accomplish the larger goal will help you, especially if you utilize the thinking you developed in previous labs for splitting tasks apart.

Lab Extra Credit

In programming, an **easter egg** is an unexpected or undocumented feature. Google often incorporates easter eggs in their products: https://en.wikipedia.org/wiki/List_of_Google_Easter_eggs#Search_engine

For this lab, you can create an undocumented function `replaceSingle` that gives the user control over the exact character they want to replace. This command will not be presented to the user as one of the valid choices at the prompt, but if they type `replace single` (in any case), your program will replace the desired character. This feature should be implemented in your existing `string_manipulator.c` file. See the output below for an example, and remember that this should handle errors gracefully as with the primary command.

```
% prompt: ./a.out

Enter a string (up to 25 characters) to be transformed: Go Tigers

Enter your command (quit, replace all): REPLACE single
Enter the character to replace: e
```

```
Enter the new character: 3
Which e would you like to replace? 2
Error, there are not 2 e characters.

Enter your command (quit, replace all): replace sinGle
Enter the character to replace: i
Enter the new character: 1
Which i would you like to replace? 1
Your new sentence is: Go T1gers

Enter your command (quit, replace all): QUIT

% prompt:
```

Code Formatting Guidelines

While you are learning the basics of programming, it's important to keep in mind that programming style matters for code readability. Here are some basic practices that we will look for when grading your labs from here on out:

1. **Consistent Indentation**. By now, you have probably seen first-hand that consistent indentation helps you spot errors quicker. There are several ways to ident code---just be sure to do so consistently.

2. **Commenting & Documentation:** A good rule of thumb for commenting is: "Would this help me remember why I'm doing what I'm doing if I look at this code 2 years down the road?" Comments should help make clear what your thought process is, what assumptions are being made, etc. However, you do not want to be redundant by over-commenting.

3. **Code Grouping:** If a task within your code requires a few lines of code, it's generally a good idea to keep these tasks within separate blocks of code. Declare variables close to where you use them so you don't lose track of anything.

4. **Consistent Naming Conventions:** Word boundaries are typically via either camelCase or by using underscores. Temporary variables, such as counters, use the same common naming scheme. Variables should typically have descriptive names to help you remember what is happening.

5. **Avoid long horizontal lines of code:** Humans generally prefer reading tall and narrow columns of text. Writing shorter lines of code makes it easier to spot errors (because some of the code may be off the screen to the right!). A line of code is generally preferred to be 80 characters or fewer.

If you'd like to see an example of good versus bad code, check out the Code Style section at the bottom of Lab 5.

Submission Guidelines

- Part 1: No submission.
- Part 2: Submit your writeup to the Canvas assignment associated for this lab by 11:00PM on Tuesday (10/16/2018). You should verify within Canvas to make sure your submission was uploaded correctly and in its entirety.

- Part 3: Submit your `string_manipulator.c` program to Canvas assignment associated for this lab by 11:00PM on Friday (10/19/2018). You should verify within Canvas to make sure your submission was uploaded correctly and in its entirety.

Grading Rubric

- If you are not present and attending lab during Part 1 and Part 2 of this lab on Tuesday, you will not receive credit for the planning portion of this lab.
- If your document for Part 2 is not submitted successfully to Canvas by the due date, you will not receive credit for the planning portion of this lab.
- Your assignment will be graded out of 100 points. The approximate grading distribution is:
    - Brainstorming document completeness/accurate — 40 points
    - Program `string_manipulator.c`
        - Test Cases – normal operation — 20 points
        - Test cases – error handling — 20 points
    - Program compiles without errors or warnings — 10 points
    - Programming style and coding formatting — 10 points
    - Optional bonus exercises (`replace single`) — up to + 10 bonus points
        - Should be implemented into `string_manipulator.c`