Introduction

This week's lab is another example of a split lab to give you more experience with planning a project before you sit down to write code. We will be combining several concepts this week.

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 uses random numbers and loops.
- Get more experience working and exchanging ideas with people in your 1011 lab. (An important skill!)

Prior to Lab

- You should be familiar with input and output, conditionals such as if, if/else, and switch statements, as well as for and while loops.
- This lab corresponds with zyBooks Chapters 4.1 – 4.4.
- Random numbers are covered in zyBooks Chapter 2.19.

Deadline and Collaboration Policy

- Part 2 of this assignment is due by 11:00 PM on Tuesday (9/25/2018) via Canvas.
- Part 3 of this assignment is due by 11:00 PM on Friday (9/28/2018) 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.

Lab Instructions

The Problem:

This week's lab involves creating an electronic version of a timeless game: Rock, Paper, Scissors! In this game, two players choose either to be a "rock", "paper", or a pair of "scissors" and shows their choice to the other player at the same time. The winner of each match is chosen by the following rubric:

- Rock wins against scissors
- Scissors wins against paper
- Paper wins against rock
- If both players select the same entity, the result is a tie.

Again, we structured this week's lab to have substantial planning on the first day, and you coding your solution on the second day:

Lab 05 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 II to Canvas by 11:00 PM	50 minutes: work individually on Part 3. 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 *****

Working with the people in your row, brainstorm ways that you can implement a game of rock, paper, scissors between a user and the computer. You probably already know them a bit by now, but ask them what their favorite part of campus is (I love the Martin courtyard). Once you've discussed if the library is better than the McAdams main lab for working, take a look at the specifications for how your program needs to operate:

- The user must be prompted for the number of games the user wants to play.
- The program should keep track of the score of both the computer and the human.
- The program should keep running until the number of games specified are completed.
- During each turn, the computer randomly selects its choice.
- After each turn, the program should display who won the previous round and the score after that turn.
- After all matches are played, a summary of the final score is displayed.

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

- What are the best ways to keep track of scores?
- What are all the tasks that need to be accomplished and where do they occur
  - (e.g., inside a loop? At the beginning of the program? At the end?)
- Is a loop needed? If so, which one best accomplishes the goal? What would the conditional be?
- How do we check to see if input is correct?
- What is the best way for the computer to make its choice?

```
%prompt: ./a.out

Starting the CPSC 1011 Rock, Paper, Scissors Game!

Enter the number of matches to play: 3

      Match 1: Enter R for rock, P for paper, or S for scissors: R
      The computer chose scissors. You win!
      Scores:  You-1

      Match 2: Enter R for rock, P for paper, or S for scissors: R
      The computer chose paper. You lose.
      Scores:  You-1  Computer-1

      Match 3: Enter R for rock, P for paper, or S for scissors: S
      The computer chose scissors. You tied.
      Scores:  You-1  Computer-1  Ties-1
```

```
The game of 3 matches is complete. The final scores are:
You:       1
Computer:  1
Ties:      1
```

Part 2: Planning on Your Own (20 minutes of Tuesday Lab)
***** It's okay to use a computer and/or calculator for this portion *****

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:

- Avoid using overly technical jargon
  - Do not say something like "Input a value into a variable". Rather say something like, "Ask the user for the unit price for a cup of coffee. User types a value into the keyboard, which is then saved for reference."
- 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.).
- 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)
  - Lab 05 – Part 2
  - Today's Date
  - Collaboration Statement: In this statement you indicate who you worked with, and which lecture sections they are enrolled in.

A note on writing your documents:

- Each bullet / sentence should cover ONE action.
- Don't write a single paragraph to explain the entire functionality.
  - If your thinking is cluttered, your code will be too, and that makes it harder to write, because you might not account for something that's needed.

Submit this part of the assignment to Canvas by 11:00PM (as specified in the submission Guidelines)

Part 3: Implementation (50 minutes on Thursday).

Lab Exercise 2.

Using your planning document from Part II, you will create a C program called `rockpaperscissors.c` that creates the game. When playing, the game should match the example given above, visually and in functionality.

Make sure you include the standard lab comment block introduced in Lab 03, but **make sure your comment block displays properly no matter how you resize the screen**. (Hint: if you copy and pasted, there are probably new-lines that cause things to not display properly in older versions, because the size of this document is different than your code environment). You can find a nicer-looking version in Lab 04.

Make sure the computer selects its choice randomly and doesn't factor in the input from the user in making its

decision. While technically this could be done, cheating isn't allowed in Rock Paper Scissors!

Lab Exercise Bonus

Create a copy of your `rockpaperscissors.c` file called `rockpaperscissors-enhanced.c` that extends your program by the following:

- At the start of the game, the user is prompted to decide between two different versions of the games as shown in the output below.

```
%prompt: ./a.out

Starting the CPSC 1011 Rock, Paper, Scissors Game!

Would you like to play until:
(Option S) a set number of matches are played?
(Option M) either player wins a certain number of matches?

Enter S or M:  M
Enter the number of matches a player must win to end the game: 3

      Match 1: Enter R for rock, P for paper, or S for scissors: P
      The computer chose rock. You win!
      Scores:   You-1  Computer-0

      Match 2: Enter R for rock, P for paper, or S for scissors: R
      The computer chose scissors. You win!
      Scores:   You-2  Computer-0

      Match 3: Enter R for rock, P for paper, or S for scissors: R
      The computer chose paper. You lose.
      Scores:  You-2  Computer-1

      Match 4: Enter R for rock, P for paper, or S for scissors: P
      The computer chose scissors. You lose.
      Scores:  You-2  Computer-2

      Match 5: Enter R for rock, P for paper, or S for scissors: S
      The computer chose rock. You lose.
      Scores:  You-2  Computer-3

The game is complete, because a player reached 3 wins. Final scores:
You:       2
Computer:  3
```

Submission Guidelines

- Part 1: No submission, but don't skip it. Thinking aloud and collaborating at a high level is incredibly important in computer science. It'll help you ask better questions at office hours, write cleaner code, do great group work, and even do better in interviews.
- Part 2: Submit your writeup to the Canvas assignment associated for this lab by 11:00PM on Tuesday (9/25/18). You should check within Canvas to make sure your submission was uploaded correctly and in its entirety.
- Part 3: Submit your `rockpaperscissors.c` program to Canvas assignment associated for this lab by 11:00PM on Friday (9/28/18). You should verify within Canvas to make sure your submission was uploaded correctly and in its entirety.
    - If you opt to complete `rockpaperscissors-enhanced.c`, submit the program to the associated Canvas assignment by 11:00PM on Friday (9/28/18). Please submit both `rockpaperscissors.c` and `rockpaperscissors-enhanced.c` to the assignment if you choose to complete the optional bonus.

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.
- If you do not check in your code with a TA for Part 3 of this lab on Thursday, you will not receive credit for the coding portion of this lab!
- If your code for Part 3 is not submitted successfully to Canvas by the due date, you will not receive credit for the coding portion of this lab.
- Your assignment will be graded out of 100 points. The approximate grading distribution is:
    - Brainstorming document completeness/accurate
        - Logic covers all requirements specified — 10 points
        - Document is simple and easy to follow — 10 points
    - Program `rockpaperscissors.c`
        - Program operates correctly — 20 points
        - Output is formatted correctly — 20 points
    - Program compiles without errors or warnings — 10 points
    - Proper code formatting and commenting (See code style section below) — 30 points
    - Optional bonus exercise `rockpaperscissors-enhanced.c` — up to + 10 bonus points
        - Requires `rockpaperscissors.c` is submitted

Code Style

Forming good habits in your coding style can have significant benefits. The word style implies that maybe it is something less important than functionality, but that's not true. Coding style is incredibly important for a wide

variety of reasons. Need to reference an earlier lab for you latest class project? It'd be a lot easier to do so if you can immediately understand what you wrote over a month ago! First time working with code for a job? Your work will be a lot better if it can be understood by others[1]. Good coding style is important, and readability is the biggest factor in this. That's why good code is:

- Consistently indented. Loops should always have their contents at a different, and consistent level.
- Not too long. A line of code should be 80 characters or less, to fit small terminals. A longer line should probably split up anyway (see simplicity).
- Grouped. Declaring a bunch of variables for a similar purpose? Keep them together, and near where they're used. This will make it easier to understand how parts of your program interact.
- Consistent and detailed in naming. Not even you will remember what `_temp_variable_new45` is 3 months from now when you need to refresh yourself on how to do something that was covered in lab.
    - Permanent: Use names so that it's easy to understand what a variable does. Also, `camelCaseVariables` and `snake_case_variables` should not be mixed
    - Temporary: Don't switch things around too much. These variables aren't too important, so don't overthink them. For example, when making a loop, use I and J for iterating, or X, and Y, but don't mix them. It could become confusing. Also, try not to use too many.
- Simple, avoiding complicated nesting of multiple loops upon loops and repetition of similar functionality. It makes it hard to keep track of what is happening, when, and why.
- Commented appropriately, for whenever the functions get a bit complicated, so you can explain the more intricate parts.

The code below is very hard to read, and isn't easy to figure out what is going on. It doesn't follow any of the guidelines, so it's not going to do very well on points.

```
#include <stdio.h>
int main (){
FILE * temp;
int temp;
int n = 0;
temp=fopen ("myfile.txt","r");
if (temp==NULL) return -1;
else {
while (temp2 != EOF) {
Temp2 = getc (pFile);
if (temp2 == '$') n++;
}
fclose (temp);
printf ("File contains %d$.\n",n);
}
return 0;
}
```

[1] Imagine you work for a big company that has hundreds of software programs each with hundreds of thousands of lines of code. Code style is incredibly important in maintaining this software for future work and maintenance that needs to be done, and ensures that new employees can understand what their coworkers have built!

On the other hand, here's some good looking code. It has comments, uses clear variable names, and groups things appropriately based on purpose and function. It even has good indenting, and doesn't have long lines. This would do really well on coding style points!

```
/*
Taken from http://www.cplusplus.com/reference/cstdio/getc/
getc example: money sign counter

This program reads an existing file called myfile.txt character by character and
uses the n variable to count how many dollar characters ($) does the file contain.
*/

#include <stdio.h>

int main ()
{
  FILE * input_file;
  input_file = fopen ("myfile.txt","r"); // try to open the file
  if (input_file == NULL)
  {
    return -1; // error occurred, stop program
  }
  else
  {

    int read = getc (input_file);
    int number = 0;

    while (read != EOF){ // go through the entire file
      read = getc (input_file);
      if (read == '$') number++; // count the dollar signs
    }

    fclose (file);
    printf ("File contains %d$.\n",n);
  }
  return 0;
}
```

Additional Resources

- If you need some help with random numbers, they are covered in zyBooks Chapter 2.19.