Autumn 2018 CSE 2421 LAB 1

DUE: Wednesday, September 12th, 2018, 11:30:00 p.m.

Objectives:

- Standard character-based I/O
- Arithmetic statements
- if/switch statements
- while, for, do-while statements
- passing parameters to C functions by value, return values from C functions

REMINDERS:

- This lab is an individual assignment. There is not an option to work with a partner.

- Every lab requires a README file. For this lab, it should be named LAB1README (***exactly*** this name (case matters), with no "extensions", such as .txt or any other; Linux can identify the file as a text file without any "extension," such as txt, doc, etc., so ***do not use these***). This file should include the following:

    1. Student name
    2. Total amount of time in hours (approximate) to complete the entire lab;
    3. Short description of any concerns, interesting problems or discoveries encountered, or comments in general about the contents of the lab;
    4. Results from gdb work detailed below.

- You should aim to always hand assignments in on time. If you are late (even by less than a minute), you will receive 75% of your earned points for the designated grade as long as the assignment is submitted by 11:30:00 pm the following day (NOTE: This is not necessarily the same as the following class day!), based on the due date listed at the top of this document. If you are more than 24 hours late, you will receive a zero for the assignment and your assignment will not be graded at all (so you will get no feedback on your work). Start early, and work steadily on the lab, and you should be able to hand the assignment in on time without a problem. If you are unable to finish on time, you will need to consider whether it is better to submit on time and get full credit for what you have done, or if it would be better to try to finish and submit one day late with the 25% penalty.

- Any lab submitted that does not build to an executable using the required gcc command (see below) and run without crashing or freezing WILL RECEIVE AN AUTOMATIC GRADE OF ZERO. No exceptions will be made for this rule - to achieve even a single point on a lab your code must minimally compile (without errors or warnings) and execute without crashing or freezing. **It is your responsibility to build your source code with the required gcc command shown below, and test it to make sure that it runs on the input as described without crashing or freezing.**

- For each part of the lab, you must build your source file to an executable file on stdlinux using the -ansi and -pedantic options with gcc on the Linux command line, as follows:

  For Part 1: % gcc -ansi -pedantic –g –o lab1p1 lab1p1.c
  For Part 2: % gcc -ansi -pedantic –g -o lab1p2 lab1p2.c

– You are welcome to do more than what is required by the assignment as long as it is clear what you are doing and it does not interfere with the mandatory requirements.

– You are responsible for making sure that your lab submits correctly.

– Make yourself aware of the CODING_STYLE_IN_C file posted on Carmen in Files, in the Labs folder. Be sure to follow the coding style information in that document, since the grader can, *and usually will*, deduct points for code that fails to follow the style requirements there.

GRADING CRITERIA (approximate percentages listed)

– (10 - 15%) The code and algorithm are well documented, including an explanatory comment for each function, and comments in the code.

- A comment should be included in the main function of the program including the programmer name as well as explaining the nature of the problem and an overall method of the solution (what you are doing, not how).
- A comment should be included before each function documenting what the function does (but not details on how it does it).
- A short comment should be included for each logical or syntactic block of statements.

– (10%) The program should be appropriate to the assignment, well-structured and easy to understand without complicated and confusing flow of control. We will not usually deduct points for the efficiency of your code, but if you do something in a way which is clearly significantly less efficient, we may deduct some points.

– (20%) There is a description in LAB1README of the gdb results asked for below for lab1p1.

– (55 - 60%) The results are correct, verifiable, and well-formatted. The program correctly performs as assigned with both the given input and one other (unknown) input file designed to test boundary conditions within your program.

– If the grader cannot compile your code using gcc –ansi –pedantic with no error messages, you will receive no points. If there are no error messages, but there are warning messages, points will be deducted also.

LAB DESCRIPTION

PART 1. (50%). Mandatory file name: lab1p1.c [NO CREDIT if you change this!]

Background: When input is read by C programs, using scanf or getchar, the input is read as

text (char) data. When your program code is written to process numeric data (int or floating point types), you need to call scanf in such a way that it converts the char input to a numeric type. scanf contains low-level input conversion code that performs the transformation of the char data in the input to a numeric form. In Part 1 of this lab, you will write some code to perform this kind of conversion from char input data to float. Your code will therefore do the same type of transformation that the low-level conversion code in scanf does.

Write a program that reads in a series of lines of input character by character (using the library function getchar()). The first line of the input contains an integer which specifies the number of remaining lines of input, each of which contains a floating point number. The integer value on the first line can be read with (the library function) scanf(), but all of the following lines can only be read with getchar(). Each line, after the first, contains a single floating point value, with up to four digits before the decimal point, and up to four digits following the decimal point, but there is not necessarily a decimal point in each number; i.e., it may appear to be an integer, but the digits should be read by your program, and the number should be converted to a corresponding floating point number. For instance, suppose the following input:

```
5
3.1255
20.25
0.875
1921.50
31
```

The required output is:

– Each of the input floating point values, printed on a separate line with five digits of precision, using printf();

– On the last line of the output, the string "Total:" followed by the sum of the input values, printed with printf() to 4 digits of precision. For example, the total of the sample input given above is 1976.7505, so the required output for this input would be:

1921.5000

3.12550

20.25000

0.87500

31.00000

Total: 1976.75050

Do not concern yourself with small differences in the total due to rounding, as the grader will not deduct points for this.

NOTE!! The input above is not the only input that can be entered in to the program that will be expected to work correctly. It is only one example of input.

Be sure to include a descriptive message for the user to input the values correctly. You can assume that the user will follow your directions (which should be consistent with the description above), so **no exception handling for incorrect input is required**. You cannot change the input specifications given above, however.

Run gdb on your **lap1p1** executable. Enter a description in to your LAB1README file that describes the iterative result (i.e. each of the interim results) of getchar loop that calculates one of your floating point numbers (just the first number in the sample input above).

CONSTRAINTS:

– You are not allowed to use arrays on this portion of the lab assignment.

– There is no maximum number of lines allowable. It all depends upon the first value of input. Since you aren't saving anything, it doesn't matter.

– You can assume that input will not contain more than 5 digits before or after the decimal point. You do not need to error check for this condition.

- You must use getchar() to read in the floating point values one character at a time (i.e. DO NOT USE scanf()).

- You must declare and use your floating point values as a **double** to minimize rounding errors.

-Only use printf() to output the floating point numbers and the total (Do not use putchar()).

– Be sure your directions to the user are clear so they are sure to enter the input data correctly.

PART 2. (50%). Mandatory filename: lab1p2.c [NO CREDIT if you change this!]

Write a program that simulates the game "Rock Paper Scissors". This is a two player game with the user of your program being one player and the computer being the other. The idea is for each player to choose one of the three options: rock, paper, or scissors. The winner for the pairings of these three options is defined to be:

- Rock wins for rock and scissors (rock pounds scissors)
- Paper wins for rock and paper (paper covers rock)
- Scissors wins for scissors and paper (scissors cut paper)
- Tie if the options chosen by both players are identical

Your program should:

1. Print an introductory message

This message should give a brief explanation of the game, and the rules for determining a winner, which are given above.

2. Prompt the user for one of the 4 options, input as a single upper or lower case R or r (rock), P or p (paper), S or s (scissors) or Q or q (quit, if the user does not wish to continue playing).

Prompt the user for an option any way you want, but be sure to give the user a non-ambiguous message about how to enter the input information. Your program should accept upper case letters for input, and should accept lower case letters for options (r, p, s, or q) as well. If the user's input does not conform to your instructions, you should print a message rejecting the user's input, and asking the user to try again, with a reminder about the acceptable form of the input. Your program should be able to catch bad input, and print a message to reject it, as well as prompting the user to try again.

3. Randomly determine the computer option (R, P, or S).

How do you obtain a random number for just 3 options? There is a random number generator function in the C library (which you will need to look up). The number returned from the function is rather large, though. How do you use this large random number to generate one of three options? HINT: Think about what the modulus operator does. **Be sure to output the computer choice to the screen.**

4. Declare a winner.

Based on the pairings defined above, determine a winner and output a message which shows what the user entered as their choice, what the computer chose as its choice, and a congratulatory message to the monitor announcing the winner (computer or user). The output should be similar to the following:

User chose: R

Computer chose: P

Congratulations, computer, you win!

5. Be sure to repeat the game until the user declines to play again, that is, until the user enters q or Q when prompted, at which point the program should terminate. Be very sure you do not commit the error of writing code which only plays the game once, and then terminates. The grader will play the game a number of times before choosing Q or q to quit, so your program should continue to play the game until the user chooses q or Q.

CONSTRAINTS:

- You must use **a switch statement** in your solution.
- You must use **an enumerated type** in your solution.

LAB SUBMISSION

You should submit the files for your lab assignment on Carmen, using the same method that was used to submit the prelab assignment:

- Run firefox from stdlinux;

- Browse to carmen.osu.edu in firefox;

- Log in to Carmen using your name.number and password, and select this course in Carmen;

- Choose the Lab1 assignment, and submit all required files.

Be sure to submit the following files: lab1p1.c lab1p2.c LAB1README

NOTE:

- Your programs MUST be submitted in source code form. Submit only .c files (and .h files when necessary, but that does not apply to this lab). **Do not submit .o files and/or executables!**

It is YOUR responsibility to make sure your code can compile (without errors or warnings)

and run on the CSE department server stdlinux.cse.ohio-state.edu. The way to do this is to compile your code on stdlinux with the commands shown earlier in this lab description, and to test your code on stdlinux also. You can make two input files for testing, called inputp1 and inputp2, and use redirection, as described in the class slides on I/O in C.