AUTUMN 2016 CSE 2421 LAB 1

DUE: Sunday, September 11, by 11:30:00 p.m.

Objectives:

– Standard I/O

– Arithmetic statements

– If/Switch statements

– While, for, do-while statements

REMINDERS:

– You are allowed to work in pairs for this assignment, but you are not required to do so.

– Every lab requires a Readme file. For this lab, it should be named lab1Readme (***exactly*** this name (case matters), with no “extensions”; 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:

> - Student name(s) - to avoid the 10% deduction (as explained in the course syllabus) if working with a partner (although you should put your name in the file even if you are working on the lab individually;
>
> - Approximate effort distribution for each contributor (assumed to be 100% if you are not working with a partner);
>
> -Total amount of time in hours (approximate) to complete the entire lab;
>
> - Short description of any concerns, interesting problems or discoveries encountered, or comments in general about the contents of the lab;

– You should aim to always hand assignments in on time. If you are late (even by a minute – or heaven forbid, less than a minute late), 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 and execute without crashing or freezing. I**t 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 lab1p1.c -o lab1p1`

For Part 2: % gcc -ansi -pedantic lab1p2.c -o lab1p2

– 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:

https://piazza.com/osu/autumn2016/cse2421/resources

in the Resources section, under Labs. Be sure to follow the coding style information in that document, as the grader can, *and usually will*, deduct points for code that fails to follow the style requirements there.

GRADING CRITERIA (approximate percentages listed)

– (20%) 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 in the program including the programmer name(s) 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.

– (20%) 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.

– (60%) The results are correct, verifiable, and well-formatted. The program correctly performs as assigned.

LAB DESCRIPTION

PART 1. (50%). Mandatory file name: lab1p1.c

Write a program that reads in a series of lines of input character by character (using 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 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 three digits before the decimal point, and up to three 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

3.125

20.25
```

0.875

921.50

31

The required output is:

– Each of the input floating point values, printed on a separate line with three 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 3 digits of precision. For example, the total of the sample input given above is 976.750, so the required output for this input would be:

3.125

20.250

0.875

921.500

31.000

Total: 976.750

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

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.

CONSTRAINTS:

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

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

-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

Write a program to determine whether a given string with no white space characters is a valid C identifier. The input to the program will be a number of lines, each of which contains a string, which you can assume has no white space characters before the new line at the end of the line. Your program should scan the string, character by character, using getchar(), and determine if it is a valid C identifier, based on the rules discussed in class, and covered in the class slides. Keep in mind that, although identifiers which begin with an underscore are not used by user application programs in C by convention, they are valid C identifiers, so your program should accept them.

There will be 5 strings in the input. For each of these strings, after determining whether it is a valid C identifier or not, your program should print out either “Valid” or “Invalid”, followed by a new line (that is, each “Valid” or “Invalid” result will be printed on a separate line in the output).

Suppose the following input:

```
_Number1

1_2_3

total

Num1+Num2

big_number!
```

The output should be:

```
Valid

Invalid

Valid

Invalid

Invalid
```

Your program can use either if statements or switch statements for much of the algorithm. You CANNOT use C library functions such as isalpha() or isdigit(), which we will see later

Also keep in mind that you may need to "consume" remaining characters on the line after determining that a given line of input contains an invalid identifier.

CONSTRAINTS:

- You cannot use arrays of any kind (including strings).

- You CANNOT use C library functions other than getchar() and printf().

LAB SUBMISSION

You should submit all your lab assignments electronically using the submit command. The format of the submit command for this lab is as follows:

```
    9:10 section:     %submit c2421ab lab1 files-to-submit

    10:20 section:    %submit c2421ad lab1 files-to-submit

    1:50 section:     %submit c2421aa lab1 files-to-submit
```

where files-to-submit is a list of the file(s) that make up the lab. The names of these files should be separated only by spaces; do not use commas, semi-colons, etc. For more information about the submit command, review this section in your prelab practice problem.

Be sure to submit the following files: lab1p1.c lab1p2.c lab1Readme

NOTE:

- All of the files in a lab MUST be submitted using one command. If you use two submit commands, the files submitted by the second one will ***erase*** the files from the first submission. Therefore, if you make changes to your files and decide to re-submit the lab assignment before the deadline, be sure to submit all of the required files with a single submit command. Do not press the enter key in the middle of your submit command line; let it wrap to the following line if necessary.

- 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 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.