Assignment: Formatted Console Output

Learning Outcomes

- Practice functional decomposition and algorithm design
- Develop skills in programming with different data types, operators, and expressions
- Develop familiarity with formatted console output

Task

Introduction

Devices that measure temperature are calibrated in various temperature scales with the most common scales being:

- Celsius scale [denoted $^{\circ}\mathrm{C}$],
- Fahrenheit scale [denoted $^{\circ}\mathrm{F}$],
- Kelvin scale [denoted K].

Your task is to define a function that converts temperatures from one scale to the other two scales. Conversion between these three different temperature scales are described here.

Problem Description

You are to declare in header file $\mathrm{q.h}$ a function `temp_converter` and define the function in source file $\mathrm{q.c}$. Function `temp_converter` will take a parameter specifying a temperature value in Fahrenheit scale and will display to the console the corresponding temperatures in Celsius and Kelvin scales.

The statement `temp_converter(50);` in function `main` [in source file $\mathrm{qdriver.c}$ that is written by me] will result in the following text printed to the console:

```
Fahrenheit     Celsius        Kelvin
---------------------------------------------
50             10.00          283.15

```

The output is formatted to consist of 3 left-justified columns with each column having a width of 15 characters. Notice that floating-point values are always printed with left justification and with 2 digits after the decimal point. Finally, notice the new line after the text in the output's third line.

The statement `temp_converter(0);` will result in the following text printed to the console:

```
Fahrenheit     Celsius        Kelvin
---------------------------------------------
0              -17.78         255.37

```

A third statement `temp_converter(100);` will result in the following text printed to the console:

```
Fahrenheit     Celsius        Kelvin
---------------------------------------------
100            37.78          310.93

```

Notice that in all cases the output is formatted exactly as described earlier.

> In this assignment, your definition of function `temp_converter` must only use `int`, `double`, and `void` data types.

Implementation Details

Open a Window command prompt, change your directory to C:\sandbox [create the directory if it doesn't exist], create a sub-directory ass03 , and launch the Linux shell.

Download driver source files qdriver.c and q.c, header file q.h, and output file good.txt containing the correct output written to the standard output by my version of function `temp_converter`.

Using Code, open header file q.h and fill in the file- and function-level documentation block and the declaration of function `temp_converter`.

Before editing source file q.c, devise a complete algorithm that practices all the steps of the algorithm formulation process to ensure your algorithm is precise and correct. Then, edit file q.c by providing the file-level and function-level [even though the function-level documentation is not graded] documentation blocks. Include necessary C standard library header file(s). Next, convert your previously constructed algorithm into a definition of function `temp_converter`.

Note that you will not be submitting qdriver.c. The submission page has its own copy of qdriver.c exactly similar to the one provided to you. The automatic grader will compile your q.c, then link with both qdriver.o and with C standard library to create an executable. The automatic grader will determine the submission grade by running the executable and comparing its output with the correct output in good.txt.

File-level and Function-level documentation

Every source and header file you submit *must* contain file-level documentation blocks whose purpose is to provide human readers [yourself and other programmers] useful information about the purpose of this source file at some later point of time

Every function that you declare in a header file [and define in a corresponding source file] must contain a function-level documentation block.

See specs from the Lab 2 for a template of the appropriate file-level documentation block and Assignment 2 for a template of the appropriate function-level documentation block.

> ***Don't copy and paste documentation blocks from previous assignments. Annoyed graders will definitely subtract grades to the full extent specified in the rubrics below when they detect such copy-and-paste scenarios.***

Compiling, linking, and testing

Compile and link your source file q.c along with driver source file qdriver.c using the following full suite of required gcc flags:

```
$ gcc -std=c11 -pedantic-errors -Wstrict-prototypes -Wall -Wextra -Werror -Wconversion qdriver.c q.c -o q.out
```

Notice the new flag `-Wconversion`. When this flag is present, the compiler will warn for implicit conversions that may alter a value. This includes conversions between floating-point and integer values. You can find more information about this flag in the GCC manual.

Run executable q.out like this to display the characters on `stdout`:

```
$ ./q.out
```

Use the shell's redirection operator `>` to redirect the program's output to a file:

```
$ ./q.out > your-output.txt
```

You're given output file good.txt containing the correct output. Your output which was redirected to file your-output.txt must exactly match the contents of good.txt. Use bash shell command diff to compare your implementation’s output with the correct output, like this:

```
$ diff -y --strip-trailing-cr --suppress-common-lines your-output.txt good.txt
```

Options `-y`, `--strip-trailing-cr`, and `--suppress-common-lines` are described here. If diff completes the comparison without generating any output, then the contents of the two files are an exact match - that is, your source file generates the exact required output. If diff reports differences, you must go back and amend q.c to remove the reported differences.

Submission and automatic evaluation

1. In the course web page, click on the submission page to submit the necessary files.
2. Read the following rubrics to maximize your grade. Your submission will receive:
    1. F grade if your submission doesn't compile with the full suite of gcc options [shown above].
    2. F grade if your submission doesn't link to create an executable.
    3. $A+$ grade if the submission's output matches the correct output. Otherwise, the auto grader will provide a proportional grade based on how many incorrect results were generated by your submission.
    4. A deduction of one letter grade for each missing documentation block. Every submitted file must have one file-level documentation block. Every function that you declare in a header file must provide a function-level documentation block. A teaching assistant will physically read submitted source files to ensure that these documentation blocks are authored correctly. Each missing block will result in a deduction of a letter grade. For example, if the automatic grader gave your submission an $A+$ grade and the three documentation blocks are missing, your grade will be later reduced from $A+$ to $B+$.

Another example: if the automatic grade gave your submission a C grade and the three documentation blocks are missing, your grade will be later reduced from C to F.