CS 241L – Data Organization
Spring 2022

Project 3

Total points: 30

Due on Monday, 3/7/22

In this project you will create a modular program that converts units of temperature, weight, length, currency, and volume.

You will write a C program called `convert.c` that will perform the following conversions:

Temperature: Celsius to Fahrenheit, and Fahrenheit to Celsius

Weight: Kilograms to Pounds, and Pounds to Kilograms

Length: Centimeters to Inches, and Inches to Centimeters

Currency: Euros to Dollars, and Dollars to Euros (use €1 = $1.14)

Volume: Liters to Ounces, and Ounces to Liters

Your program will interact with the user via the command line. However, for testing and grading, we will use redirection from `stdin` and `stdout` to read and create files.

After compiling your program with

```
gcc –Wall –pedantic –ansi convert.c –o convert
```

Run it with:

```
./convert
```

On the command line, you should see the following menu:

```
Main menu
What would you like to convert?
Enter:
1 – Temperature
2 – Weight
3 – Length
4 – Currency
5 – Volume
```

Once the user chooses their option, a second menu appears, depending on the user's choice. If they choose temperature, they should see:

```
You entered Temperature.
Choose units:
1 – Celsius to Fahrenheit
2 – Fahrenheit to Celsius
```

If the user chooses option 2, they see:

```
You entered Fahrenheit to Celsius.
Enter Temperature in Fahrenheit
```

At this point, the user should enter the value they want to convert. Suppose they've entered 100:

```
100.00 Fahrenheit = 37.78 Celsius
```

Once the conversion is complete, the program will ask if the user would like to continue:

```
Would you like to convert another value? (y/n)
```

Every time this question appears, if the user enters `y`, the program returns to the main menu. If they enter `n`, the program prints `Bye!` and quits. If the answer is neither `y` nor `n`, the output should be:

```
Invalid option. Exiting.
```

And the program quits.

You are given a file `unitconversion.out` that shows an example of what the user should see when performing some conversions. Values entered by the user are not shown in the file (only values and sentences that the program outputs are shown).

The file `unitconversion.in` contains the input values that generate the file `unitconversion.out`. Run it with:

```
./convert < unitconversion.in > myunitconversion.out
```

And do a diff test:

```
diff myunitconversion.out unitconversion.out
```

To make sure the two output files are identical.

Your program should be able to handle errors. If the user enters a numeric option not available in the menu, your program should output:

```
Invalid Option
Would you like to convert another value? (y/n)
```

If the answer to this question is also invalid, the output should be:

```
Invalid option. Exiting.
```

The only conversions that can be negative are conversions of temperature. All other options should convert only positive real numbers. If the user enters a negative number for a variable other than temperature, they should see the message:

```
Error: Value should be a positive float.
Would you like to convert another value? (y/n)
```

Keep in mind that you should make your program modular by using functions. Notice that many actions are repetitive. Remember our course coding standards and do not write repetitive code.

Hint: When reading a char c, use two consecutive `c=getchar()`. If data has been entered before and the user has hit `<return>`, `getchar()` reads this return instead, and not the char that you intended it to read. Writing two consecutive `c=getchar()` will take care of this issue.

Hint: Find conversion factors in books or on the internet. Add at least 5 significant digits for conversion factors of volume, weight, and length, but notice that your program should round the results to two decimal places.

What to submit:

The files:

- `convert.c`
- `myunitconversion.out`

Note: Submit each file separately on Learn, do ***not*** combine them in a single zip file.

Grading Rubric:

+ 2 pts: Your C program starts with a comment on top of the file with your name and description of the program
+ 2 pts: Your C program compiles with the `–ansi –pedantic –Wall` options without errors or warnings.
+ 10 pts: Your C program follows the class coding standards. **Points will be taken off if your code is too repetitive.** Points will also be taken off if indentation is not done correctly (go to our lab video on Emacs customization to fix indentations).
+ 6 pts: Your output file `myunitconversion.out` passes a `diff` test when compared with the `unitconversion.out` file.

+ 10 pts: Your program correctly handles an input file that is similar to `unitconversion.in` (but with different values and possibly error cases) that only the graders have access to.