Computations

Workshop 2 (out of 10 marks - 2% of your final grade)

In this workshop, you will code and execute a C-language program that accepts numerical values from the user, stores the values in variables of appropriate data type, performs calculations on the stored variables and casts from one data type to another.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities:

- to declare variables of integral and floating point types
- to code a simple calculation using C operators and expressions
- to accept a numerical value from the user using scanf
- to cast a value from one data type to another
- to describe to your instructor what you have learned in completing this workshop

SUBMISSION POLICY

Part 1 of the workshop is to be completed by 23:59 of the day after your workshop. Part 2 of the workshop is **due no later than four (4) days following the part 1 assigned date (even if that day is a holiday) by 23:59.**

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to back up your work regularly.

PART 1: DONE IN CLASS (30%)

Download or clone workshop 2 (**WS02**) from https://github.com/Seneca-144100/IPC-Workshops

In the in_lab directory of workshop 2, right-click on the file in_lab.vsxproj and select the "Open With" context menu item then select "Microsoft Visual Studio 2019" to open the workshop in Visual Studio.

In the Visual Studio solution explorer panel, open and write your code in **cashRegister.c** for workshop 2.

Start the program by asking the user to enter the amount due. Print the following message:

```
Please enter the amount to be paid: $
```

Assume the user enters 8.68, this is what the screen should look like:
(<ENTER> means hitting the enter key)

```
Please enter the amount to be paid: $8.68 <ENTER>
```

Read the amount due and store it as a double.

Calculate the number of loonies and quarters required to pay the amount due and display the following:

```
Loonies required: 8, balance owing $0.68
Quarters required: 2, balance owing $0.18
```

Execution and Output Example:

```
Please enter the amount to be paid: $8.68
Loonies required: 8, balance owing $0.68
Quarters required: 2, balance owing $0.18
```

For submission instructions, see the SUBMISSION section below.

PART 1 SUBMISSION:

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload your `cashRegister.c` to your matrix account. Compile and run your code and make sure everything works properly.

```
AtThePrompt> gcc -Wall cashRegister.c –o ws<ENTER>
```

-Wall activates the display of warnings in GCC compiler.
-o sets the executable name (ws)

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid and replace **NAA** with your section)

```
~profname.proflastname/submit 144w2/NAA_lab <ENTER>
```

and follow the instructions.

PART 2: (30%)

After completing part 1, edit and upgrade cashRegister.c to add the GST to the total entered by the user.

Display the amount of the GST and then the total due. Then display the number of quarters, dimes, nickels and pennies required to pay the total amount.

Problem: 8.68 * .13 is equal to 1.1284, which should be rounded up to 1.13, and the format specifier %.2lf will display as 1.13, but the number will be truncated to 1.12 on a C standard compiler, such as matrix, if we multiply by 100 and cast to an int, which we must do at some point in the program.

To find the correct value for GST do the following:

> GST = amountOwing * .13 + .005; (result is 1.1334 which will resolve to 1.13 when either rounded or truncated)

After Calculating the number of loonies required, subtract the number of loonies from the amountOwing, then cast the remaining decimal portion to an int. The subsequent operations must be done using **integer division** and **modulus**. Although the answer can be derived without using the modulus operator, failure to use modulus will result in a grade of 0 for the At_Home portion of the workshop.

Hints:

- *Read about casting, division and modulus in the notes*
- *0.35 * 100 = 35*
- *You can output the value 5, stored in an int variable as $0.05 like this:*
  `printf("$%1.2f", (float)intBalance/100);`

Execution and Output Example:

```
Please enter the amount to be paid: $8.68
GST: 1.13
Balance owing: $9.81
Loonies required: 9, balance owing $0.81
Quarters required: 3, balance owing $0.06
Dimes required: 0, balance owing $0.06
```

```
Nickels required: 1, balance owing $0.01
Pennies required: 1, balance owing $0.00
```

PART 2 REFLECTION (40%)

Please provide answers to the following in a text file named `reflect.txt.`

In three or more paragraphs, explain what you learned while doing this workshop. Tell us what was interesting and what you found difficult. Explain why it is often a best practice to convert floating-point values to integers when performing arithmetic operations. Why is it a best practice to use the modulus operator rather than division and subtraction to find a remainder?

Reflections will be graded on clarity of thought, correctness, grammar and spelling.

Note: when completing the workshop reflection it is a violation of academic policy to cut and paste content from the course notes or any other published source, or to copy the work of another student.

PART 2 SUBMISSION:

To test and demonstrate execution of your program using the same data as the output example above.

If not on matrix already, upload your `cashRegister.c` and `reflect.txt` to your matrix account. Compile and run your code and make sure everything works properly.

```
AtThePrompt> gcc -Wall cashRegister.c -o ws <ENTER>
```

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid and replace **NAA** with your section)

```
~profname.proflastname/submit 144w2/NAA_home <ENTER>
```

and follow the instructions.