Workshop #2

Worth: 1% of final grade

Breakdown

- Part-1 Coding: 10%
- Part-2 Coding: 40%
- Part-2 Reflection: 50%

Submission Policy

- Part-1 is due on Thursday by the **end of day 23:59** EST (UTC – 5)
- Part-2 is due on Sunday by the **end of day 23:59** EST (UTC – 5)
- Source (.c) and text (.txt) files that are provided with the workshop MUST be used or your work will not be accepted. Resubmission will be required attracting a **15% deduction**
- **Late submissions will NOT be accepted**
- **All work must be submitted by the matrix submitter – no exceptions**
- **All files** you create or modify MUST contain the following declaration at the top of all documents:

```
/*****************************************************************************
               <assessment name example: Workshop - #2 (Part-1)>
Full Name  :
Student ID#:
Email      :
Section    :

Authenticity Declaration:
I declare this submission is the result of my own work and has not been
shared with any other student or 3rd party content provider. This submitted
piece of work is entirely of my own creation.
*****************************************************************************/
```

Notes

- Due dates are in effect **even during a holiday**
- You are responsible for **backing up your work regularly**
- It is expected and assumed that for each workshop, you will plan your coding solution by using the computational thinking approach to problem solving and that **you will code your solution based on your defined pseudo code algorithm**.

Late Submission/Incomplete Penalties

If any Part-1, Part-2, or Reflection portions are missing, the mark will be **ZERO**.

Introduction

In this workshop, you will code and execute a C language program that stores values in variables of the appropriate data type, performs calculations on the stored variables, and apply relational and logical expressions to determine the true or false result of various test case scenarios.

Topic(s)

- Types, Calculations, Expressions

Learning Outcomes

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

- Select appropriate types for storing program variables and constants
- Create meaningful variable identifiers contributing to easy to read and maintain code
- Code a simple calculation to solve a basic programming task
- Code relational and logical expressions to evaluate the true or false nature of various test statements
- Store user input values to program variables
- Describe to your instructor what you have learned in completing this workshop

Part-1 (10%)

Instructions

Download or clone workshop 2 (**WS02**) from https://github.com/Seneca-144100/BTP-Workshops
Note: If you use the download option, make sure you **EXTRACT** the files from the .zip archive file

1. Carefully review the "Part-1 Output Example" (next section) to see how this program is expected to work
2. Code your solution to Part-1 in the provided "**w2p1.c**" source code file.
3. This program can be broken down into a few major logical code sections:
   a) Variable declarations: All variables should be declared together into meaningful groups at the top of the main function
   b) Display product data: Summarize the product data in a tabular format to help make the data easy to read
   c) Display summary of test scenarios: Applying relational and logical expressions to produce **true** (non-zero) / **false** (0) representation of the results to each test scenario
4. Don't delete or modify the provided "*testValue*" variable declaration (you will need this for test case #3 in the data analysis section).
5. Create **nine (9) variables** to represent the following data. There are three (3) products, each with three (3) related pieces of information:

	Product-1	Product-2	Product-3
Product ID	111	222	111
Price	111.49	222.99	334.49
Taxed	Y	N	N

Note

- All variables are UNMODIFIABLE so be sure to prefix the variable declarations accordingly
- You must select the **appropriate data type** for each variable based on the type of data that needs to be stored
- You must use **self-describing variable names** to maximize readability and maintainability of the code

6. You will require one (1) extra variable that will be needed to store the average price of all the three products. Calculate the average price for all three (3) products and initialize this variable with the result.
7. You should have eleven (11) variables declared and initialized. **NO MORE VARIABLES** are to be declared.
8. Display each product information as illustrated in the sample output.
   - **YELLOW** highlighted parts represent where you should be using the variables declared in step #4 in the printf statements (do NOT hardcode the ID, Taxed, or Price values)!
   - Display the prices to 4 decimal precision points
9. Display the calculated average price (to 4 decimal precision points) of the three (3) products (again, do NOT hardcode this value, use the variable you declared in step #5).
10. Display the five (5) lines of text about "Relational and Logical Expressions". Pay close attention to the formatting.
11. Display the nine (9) data analysis parts according to the sample output.
    - All prices in this section should be displayed to 2 decimal precision points
    - YELLOW highlighted parts represent where you should be using variables in the printf statement
    - GREEN highlighted parts represent the result of a relational or logical expression. Use the appropriate variables in the expression to derive the necessary result.
    - **RED** highlighted part indicates where you will need to apply an in-line calculation (not stored to a variable).

Part-1 Output Example

```
Product Information
===================
Product-1 (ID:111)
  Taxed: Y
  Price: $111.4900

Product-2 (ID:222)
  Taxed: N
  Price: $222.9900

Product-3 (ID:111)
  Taxed: N
  Price: $334.4900

The average of all prices is: $222.9900

About Relational and Logical Expressions!
=========================================
1. These expressions evaluate to TRUE or FALSE
2. FALSE: is always represented by integer value 0
3. TRUE : is represented by any integer value other than 0

Some Data Analysis...
=====================
1. Is product 1 taxable? -> 1

2. Are products 2 and 3 both NOT taxable (N)? -> 1

3. Is product 3 less than testValue ($330.99)? -> 0

4. Is the price of product 3 more than both product 1 and 2 combined? -> 1

5. Is the price of product 1 equal to or more than the price difference
   of product 3 LESS product 2? ->  0 (price difference: $111.50)
```

```
6. Is the price of product 2 equal to or more than the average price? -> 1
7. Based on product ID, product 1 is unique -> 0
8. Based on product ID, product 2 is unique -> 1
9. Based on product ID, product 3 is unique -> 0
```

Part-1 Submission

1. Upload (file transfer) your source file "**w2p1.c**" to your matrix account
2. Login to matrix in an SSH terminal and change directory to where you placed your workshop source code.
3. Manually compile and run your program to make sure everything works properly:
   gcc -Wall w2p1.c -o w2 *<ENTER>*
   If there are no errors/warnings generated, execute it: ***w2*** *<ENTER>*
4. Run the submission command below (replace **profname.proflastname** with **your professors** Seneca userid and replace **NAA** with your section):
   ~profName.proflastname/submit 100w2/NAA_p1 *<ENTER>*
5. Follow the on-screen submission instructions

Part-2 (40%)

Instructions

1. Carefully review the "Part-2 Output Example" (next section) to see how this program is expected to work
2. Code your solution to Part-2 in the provided "**w2p2.c**" source code file
3. This program can be broken down into a few major logical code sections:
   a) Variable declarations: All variables should be declared together into meaningful groups at the top of the main function
   b) Product data input: Prompt for data describing three (3) products and store user-input to appropriate variables
   c) Display product data: Summarize the product data in a tabular format to help make the data easy to read
   d) User preference input (scenario 1 of 2): Prompt for the user's coffee preferences and store user-inputs to appropriate variables
   e) Display summary of results: Applying the user-input preferences, determine how each product matches the needs of the user defined preferences
   f) Repeat: Repeat step (d) and (e) for another preference scenario
4. Don't delete or modify the provided "***GRAMS_IN_LBS***" variable declaration (you will need this in the conversion from grams to pounds when appropriate).

5. Using the example output as a guide, declare the necessary **fifteen (15)** variables used to represent the three (3) product data "records".

   Note

   - You must select the **appropriate data type** for each variable based on the type of data that needs to be stored
   - You must use **self-describing variable names** to maximize readability and maintainability of the code

6. Prompting user-input for a **single-character** value can cause unexpected behaviour which you will learn about later in the semester, however, for now use the following ***scanf*** formatting specifier (between the double-quotes) to avoid strange behaviour (notably the **single-space before the percent sign**):

```
scanf(" %c", ...
```

7. Displaying the product data in a tabular format requires the application of some slightly more advanced formatting features (you will learn more about this later in the semester). For now, you can use the first product data line below to get you going (copy/paste into your code):

```
printf(" 1 |   %d   |   %d   |    %d   |   %d   | %4d | %6.3lf |   %d   | %5.1lf | %5.1lf\n", ...
```

8. The below table provides the mapping rules you must apply in matching the user input preferences to each product (Example: if the user prefers "Mild" coffee, and the product type is "Light" this would show as true (non-zero) in the summary table result):

Preference	**Product**
Coffee Strength	**Coffee Type**
Mild	Light
Rich	Blend
Coffee Making Equipment	**Coffee Grind Size**
Residential	Course
Commercial	Fine
Daily Servings *(inclusive range)*	**Coffee Package Weight** *(inclusive range)*
1 to 4	0 to 250 g
1 to 9	500 g
10 or more	1000 g
Like Cream with Coffee	**Suggest Serving with Cream**
Yes	Yes
No	No
Coffee Making Equipment	**Serving Temperature** *(inclusive range)*
Residential	60.0 to 69.9 (°C)
Commercial	70.0 or more (°C)

9. You should not need to declare more than **four (4)** additional variables to store the user-input values that describe a scenario of the user's preferences and equipment used in making coffee. The variables will be used in determining how each product "matches" the needs based on the input scenario (see above table).

   Reminder All variable declarations MUST be grouped together at the beginning of the "main" function to maximize the management of your program variables (will be easy to find being all in one place)

10. Displaying the preference to product summary results in a tabular format requires the application of some slightly more advanced formatting features (you will learn more about this later in the semester). For now, you can use the first product data line below to get you going (copy/paste into your code):

```
printf(" 1|       %d         |         %d          |      %d      |   %d   |      %d\n", ...
```

Note
You must code the required ***relational and/or logical expression(s)*** using the appropriate variables as required for each mapped field which will provide the true/false result using the matching rules table described earlier.

Part-2 Output Example (Note: Use the YELLOW highlighted user-input data for submission)

```
Take a Break - Coffee Shop
==========================

Enter the coffee product information being sold today...

COFFEE-1...
Type ([L]ight,[B]lend): l
Grind size ([C]ourse,[F]ine): c
Bag weight (g): 250
Best served with cream ([Y]es,[N]o): y
Ideal serving temperature (Celsius): 65.7

COFFEE-2...
Type ([L]ight,[B]lend): B
Grind size ([C]ourse,[F]ine): F
Bag weight (g): 500
Best served with cream ([Y]es,[N]o): N
Ideal serving temperature (Celsius): 70.0

COFFEE-3...
Type ([L]ight,[B]lend): L
Grind size ([C]ourse,[F]ine): f
Bag weight (g): 1000
Best served with cream ([Y]es,[N]o): n
Ideal serving temperature (Celsius): 80.5

---+---------------+---------------+---------------+-------+-------------
   |    Coffee     |    Coffee     |   Packaged    | Best  |   Serving
   |     Type      |  Grind Size   |  Bag Weight   | Served| Temperature
   +---------------+---------------+---------------+ With  +-------------
ID | Light | Blend | Course | Fine |  (G)  | Lbs   | Cream |   (C) |   (F)
---+---------------+---------------+---------------+-------+-------------
 1 |   1   |   0   |    1   |   0  |  250  |  0.551|   1   |  65.7 | 150.3
 2 |   0   |   1   |    0   |   1  |  500  |  1.102|   0   |  70.0 | 158.0
 3 |   1   |   0   |    0   |   1  | 1000  |  2.205|   0   |  80.5 | 176.9

Enter how you like your coffee and the coffee maker equipment you use...

Coffee strength ([M]ild,[R]ich): m
Do you like your coffee with cream ([Y]es,[N]o): y
```

```
Typical number of daily servings: 6
Maker ([R]esidential,[C]ommercial): c

The below table shows how your preferences align to the available products:

--------------------+--------------------+-------------+-------+--------------
  |     Coffee       |       Coffee       |  Packaged   | With  |   Serving
ID|      Type        |     Grind Size     | Bag Weight  | Cream | Temperature
--+------------------+--------------------+-------------+-------+--------------
 1|       1          |         0          |      0      |   1   |      0
 2|       0          |         1          |      1      |   0   |      1
 3|       1          |         1          |      0      |   0   |      1

Enter how you like your coffee and the coffee maker equipment you use...

Coffee strength ([M]ild,[R]ich): R
Do you like your coffee with cream ([Y]es,[N]o): N
Typical number of daily servings: 10
Maker ([R]esidential,[C]ommercial): R

The below table shows how your preferences align to the available products:

--------------------+--------------------+-------------+-------+--------------
  |     Coffee       |       Coffee       |  Packaged   | With  |   Serving
ID|      Type        |     Grind Size     | Bag Weight  | Cream | Temperature
--+------------------+--------------------+-------------+-------+--------------
 1|       0          |         1          |      0      |   0   |      1
 2|       1          |         0          |      0      |   1   |      0
 3|       0          |         0          |      1      |   1   |      0

Hope you found a product that suits your likes!
```

Reflection (50%)

Instructions

Record your answer(s) to the reflection question(s) in the provided "**reflect.txt**" text file.

1. What did you find to be the most difficult expression you had to code in this workshop? Copy the line of code that you are referring to and explain your answer in detail?

2. Find one of your coded **logical** expressions from Part-2 and show how this can be evaluated in two different ways but will produce the same desired result. One version of the statement should apply "deMorgan's Law" and the other will not.

3. In Part-1,you were instructed to create a variable to store the calculated average product prices, while in Part-2, the displaying of the converted Celsius to Fahrenheit temperature unit was NOT to be stored to a variable. Why do you think this was the preferred approach for both cases?

Academic Integrity

It is a violation of academic policy to copy content from the course notes or any other published source (including websites, work from another student, or sharing your work with others).

Failure to adhere to this policy will result in the filing of a violation report to the Academic Integrity Committee.

Part-2 Submission

1. Upload your source file "**w2p2.c**" to your matrix account
2. Upload your reflection file "**reflect.txt**" to your matrix account (to the same directory)
3. Login to matrix in an SSH terminal and change directory to where you placed your workshop source code.
4. Manually compile and run your program to make sure everything works properly:

   ```
   gcc -Wall w2p2.c -o w2 <ENTER>
   ```

   If there are no errors/warnings generated, execute it: ***w2*** *<ENTER>*
5. Run the submission command below (replace **profname.proflastname** with **your professors** Seneca userid and replace **NAA** with your section):

   ```
   ~profName.proflastname/submit 100w2/NAA_p2 <ENTER>
   ```
6. Follow the on-screen submission instructions