

Workshop #3

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 - #3 (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 accepts numerical values from the user, stores the values in variables of the appropriate data type, performs monetary calculations on the stored variables (including the modulus operator) and casts one data type to another.

Topic(s)

- [Types](#), [Calculations](#), [Expressions](#)

Learning Outcomes

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

- Code a simple calculation using C operators and expressions
- Accept a numerical value from the user using scanf
- Cast a value from one data type to another
- Use integral data types to manage 2-digit precision points for a monetary application and manually manage rounding to avoid misrepresented values that can be stored when using floating-point types
- Describe to your instructor what you have learned in completing this workshop

Part-1 (10%)

Instructions

Download or clone workshop 3 (**WS03**) from <https://github.com/Seneca-144100/IPC-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 “**w3p1.c**” source code file.
3. Create the necessary variables that will be used to store the prices of **three (3)** shirt sizes. Use meaningful self-described names and of the appropriate data type
4. Display the shirt price list using the standard **two-decimal precision** representation for the prices
5. Monetary systems represent currency in dollars and cents and commonly to two (2) decimal precision points (representing the cents portion of the value: \$17.96). This would lead you wanting to use floating-point data types throughout your solution, but this CAN lead to unexpected results depending on the operations you need to perform (presents itself as an unexplained case of "lost" pennies/cents).

To address this problem (and for this workshop) you must restrict your operations to using only INTEGRAL data types and work in the lowest coin denomination "pennies" (cents). Therefore, all remaining variables you declare must be INTEGRAL types only.

Hints

- You will need to **manually apply rounding** as required to bump up a cent in cases when half a cent or more is encountered (ex: if a calculated value results in **77.34578**, then the 4.578 cents should be **stored as 7735** total cents).
- To **display** currency stored as total cents (ex: **7735**) in the friendly dollars and cents format (ex: **77.35**) you will need to divide the total cents variable by 100 and apply casting to promote the value to a float-point type

6. You will need to calculate and store the **sub-total** (before taxes are applied) to a variable based on the number of shirts being purchased at the appropriate shirt price
7. You will need to calculate and store the **taxes** to a variable based on the sub-total
8. You will need to calculate the **total** by adding the calculated sub-total and taxes values
9. Display the shopping cart details including a breakdown of charges into the sub-total, taxes, and total components.

Note

Display the currency components to **four (4) decimal precision** points. Use the following **printf** format specifier (replace the ... parts as required to complete the statement accordingly):

```
printf("... $%8.4lf"...
```

Part-1 Output Example (Note: Use the **YELLOW** highlighted user-input data for submission)

Set Shirt Prices

=====

Enter the price for a SMALL shirt: \$**17.96**

Enter the price for a MEDIUM shirt: \$**26.96**

Enter the price for a LARGE shirt: \$**35.97**

Shirt Store Price List

=====

SMALL : \$17.96

MEDIUM : \$26.96

LARGE : \$35.97

Patty's shirt size is 'S'

Number of shirts Patty is buying: **8**

Patty's shopping cart...

Contains : 8 shirts

Sub-total: \$143.6800

Taxes : \$ 18.6800

Total : \$162.3600

Part-1 Submission

1. Upload (file transfer) your source file "**w3p1.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 w3p1.c -o w3 <ENTER>
```

If there are no errors/warnings generated, execute it: w3 <ENTER>

4. Run the submission command below (replace **profname.proflastname** with **your professors** Seneca userid and replace **NAA** with your section):

```
~profName.proflastname/submit 144w3/NAA_p1 <ENTER>
```

5. Follow the on-screen submission instructions
-

Part-2 (40%)

Instructions

1. Copy the main function code from your Part-1 solution and paste it into the provided "**w3p2.c**" source code file. Be careful not to overwrite the starter code provided in the "**w3p2.c**" file.
2. Carefully review the "Part-2 Output Example" (next section) to see how the program is expected to work
3. You will need to modify the code as required to produce a solution to work as demonstrated in the sample output.
4. Displaying the sales 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 data line below to get you going (copy/paste into your code), and complete the **printf** statement accordingly:

```
printf("Patty    %-4c %5.2lf %3d %9.4lf %9.4lf %9.4lf\n",...
```

Similarly, the totals row for the above table also requires some more advanced formatting. Use the following **printf** statement and complete it accordingly:

```
printf("%33.4lf %9.4lf %9.4lf\n\n",...
```

5. In a tabular format, show how the daily total retail sales would be broken down by coin denominations if it were to be converted to only coins (start from the largest denomination working down to the smallest). To accomplish this, you will need to apply integer **division (/)** to obtain the number of coins for a given denomination, followed by an application of the modulus **operator (%)** to obtain the new remaining amount (to be used in the next coin calculation).

Note

- The first table is based on the **sub-total and excludes taxes**
 - The second table is based on the **total and includes taxes**
6. The first data row shows only the starting balance, followed by the coin denominations in the subsequent rows. Use the partially formed **printf** statements below which give you a hint at how the first two (2) rows can be formatted (you will need to complete the statements accordingly):

```
printf("%22.4lf\n",...  
printf("Toonies  %3d %9.4lf\n", ...
```

7. After each table, display the calculated average cost per shirt accordingly

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

Set Shirt Prices

=====

```
Enter the price for a SMALL shirt: $17.96  
Enter the price for a MEDIUM shirt: $26.96  
Enter the price for a LARGE shirt: $35.97
```

Shirt Store Price List

=====

SMALL : \$17.96

MEDIUM : \$26.96

LARGE : \$35.97

Patty's shirt size is 'S'

Number of shirts Patty is buying: 6

Tommy's shirt size is 'L'

Number of shirts Tommy is buying: 3

Sally's shirt size is 'M'

Number of shirts Sally is buying: 4

Customer	Size	Price	Qty	Sub-Total	Tax	Total
-----	----	-----	----	-----	-----	-----
Patty	S	17.96	6	107.7600	14.0100	121.7700
Sally	M	26.96	4	107.8400	14.0200	121.8600
Tommy	L	35.97	3	107.9100	14.0300	121.9400
-----	----	-----	----	-----	-----	-----
				323.5100	42.0600	365.5700

Daily retail sales represented by coins

=====

Sales EXCLUDING tax

Coin	Qty	Balance
-----	----	-----
		323.5100
Toonies	161	1.5100
Loonies	1	0.5100
Quarters	2	0.0100
Dimes	0	0.0100
Nickels	0	0.0100
Pennies	1	0.0000

Average cost/shirt: \$24.8854

Sales INCLUDING tax

Coin	Qty	Balance
-----	----	-----
		365.5700
Toonies	182	1.5700
Loonies	1	0.5700
Quarters	2	0.0700
Dimes	0	0.0700
Nickels	1	0.0200
Pennies	2	0.0000

Average cost/shirt: \$28.1208

Reflection (50%)

Instructions

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

1. Given the following C statements:

```
double price1 = 10.10, price2 = 10.12, price3 = 10.15;
int cents1 = (int)(price1 * 100);
int cents2 = (int)(price2 * 100);
int cents3 = (int)(price3 * 100);
printf("%d 10.10 = %.15lf\n", cents1, price1);
printf("%d 10.12 = %.15lf\n", cents2, price2);
printf("%d 10.15 = %.15lf\n", cents3, price3);
```

Briefly explain why the cents2 variable doesn't properly represent the total cents of price2.

2. You had to use integer division (/) to get the number of coins and the modulus (%) operator to easily obtain the remainder amount after each coin denomination calculation. Let's assume this program needed to be modified to only determine the breakdown of paper bills (\$50, \$20, \$10, \$5). Provide the few lines of code that would be required to demonstrate how the \$20 denomination would be calculated (use your solution variables in your example).
3. Displaying total cents in the format of dollars and cents (ex: 7245 to \$72.45) required converting (casting) an integer to a floating-point type (along with a simple calculation). Without changing any variable data types, show **three (3) different ways** to accomplish the same outcome (use your solution variables in your example). If you are having problems thinking of how to approach this, think about how an expression can be promoted or narrowed in several ways based on explicit casting, coercion, and how each operand plays a role in this determination.

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 “**w3p2.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 w3p2.c -o w3 <ENTER>
```

If there are no errors/warnings generated, execute it: w3 <ENTER>

5. Run the submission command below (replace **profname.proflastname** with your professors Seneca userid and replace **NAA** with your section):

```
~profName.proflastname/submit 144w3/NAA_p2 <ENTER>
```

6. Follow the on-screen submission instructions