ENGG1410: Introductory Programming for Engineers

Lab 1: “Introduction to C Programming”: Implementing a pseudocode Writing, Running, Executing a C Program Error free

09/17/2024

Table Of Contents

Part 1 3
Problem Statement: 3
Assumptions and Constraints: 3
How you solved the Problem: 3
Error Analysis: 4
errors encountered during the development and how they were resolved: 4
Part 2 5
Problem Statement: 5
Assumptions and Constraints: 5
How you solved the problem: 5
Error analysis: 6
errors encountered during the development and how they were resolved: 6
Part 3 7
Problem Statement: 7
Assumptions and Constraints: 7
How you solved the problem: 7
Error analysis: 9
errors encountered during the development and how they were resolved: 9
Conclusion and self-assessment: 10
What you learned in the lab. 10

Part 1

Problem Statement:

Inform user of the methods to circumvent issues with back slash, percentage and new lines in the C programming language.

Assumptions and Constraints:

Output must exactly as shown below:

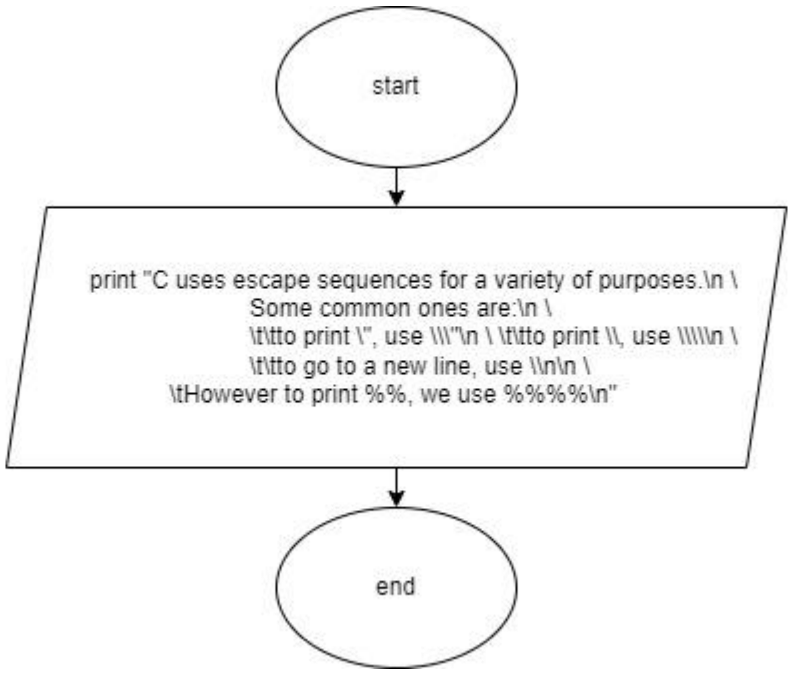
```
C USES ESCAPE SEQUENCES FOR A VARIETY OF PURPOSES.
            SOME COMMON ONES ARE:
                        TO PRINT ", USE \"
                        TO PRINT \, USE \\
                        TO GO TO A NEW LINE, USE \N
                  HOWEVER TO PRINT %, WE USE %%
```

How you solved the Problem:

I) **Pseudo Code.**

```
Print   "C uses escape sequences for a variety of purposes.\n \
         Some common ones are:\n \
                \t\tto print \", use \\\"\n \
                \t\tto print \\, use \\\\\n \
                \t\tto go to a new line, use \\n\n \
            \tHowever to print %%, we use %%%%\n"
```

II) **Flowchart.**


start
print "C uses escape sequences for a variety of purposes.\n \
Some common ones are:\n \
\t\tto print \", use \\\"\n \ \t\tto print \\, use \\\\\n \
\t\tto go to a new line, use \\n\n \
\tHowever to print %%, we use %%%%\n"
end

Error Analysis:

errors encountered during the development and how they were resolved:

- **Program did not compile due to it detecting a back slash; this was resolved by adding another back slash**
- **Program did not compile due to it detecting a percentage symbol; this was fixed by adding another percentage symbol**
- **Program was printed all on one line; this was resolved by adding \n**
- **Program did not print \n; this was resolved by adding a back slash**
- **Program was not properly indented; this was resolved by adding \t**

Part 2

Problem Statement:

Write a program that obtains the total cost of a bill, the tip percentage and the number of people. Then, inform the user the amount to be paid by each person when the bill is split evenly.

Assumptions and Constraints:

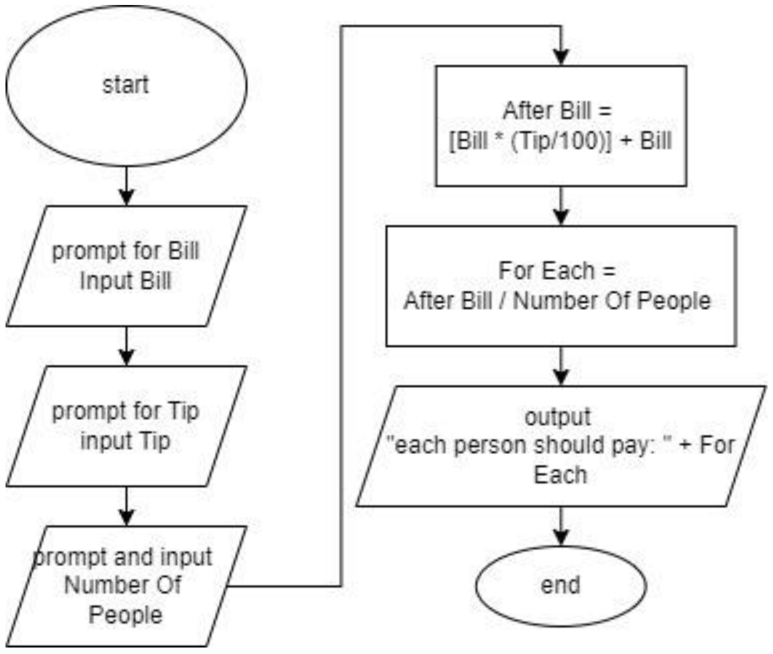
- Assume all input is positive
- Cost of bill is up to 2 decimals places
- Tip percentage is not given in decimal
    - Ex. 15% would be given as 15 not .15
- Number of people is integer
- Output is up to 2 decimal places

How you solved the problem:

I) **Pseudo Code**

```
double bill
double percentage
int numOfPeople

Print Enter the original bill amount:
Input -> Bill

Print Enter the tip percentage:
Input -> percentage

Print Enter the number of people splitting the bill:
Input -> numOfPeople

double afterTip = (bill * (percentage / 100)) + bill
double forEach = afterTip / numOfPeople

Print Each person should pay: + forEach //2 decimal places
```

II) **Flow chart**



Error analysis:

errors encountered during the development and how they were resolved:

- **Incorrect format specifier** - **this was fixed with the correct format specifier**

Part 3

Problem Statement:

Debug the given C program such that it achieves its intended purpose which is to convert kilograms to pounds, ounces, and fraction of ounces up to 2 decimal places.

Assumptions and Constraints:

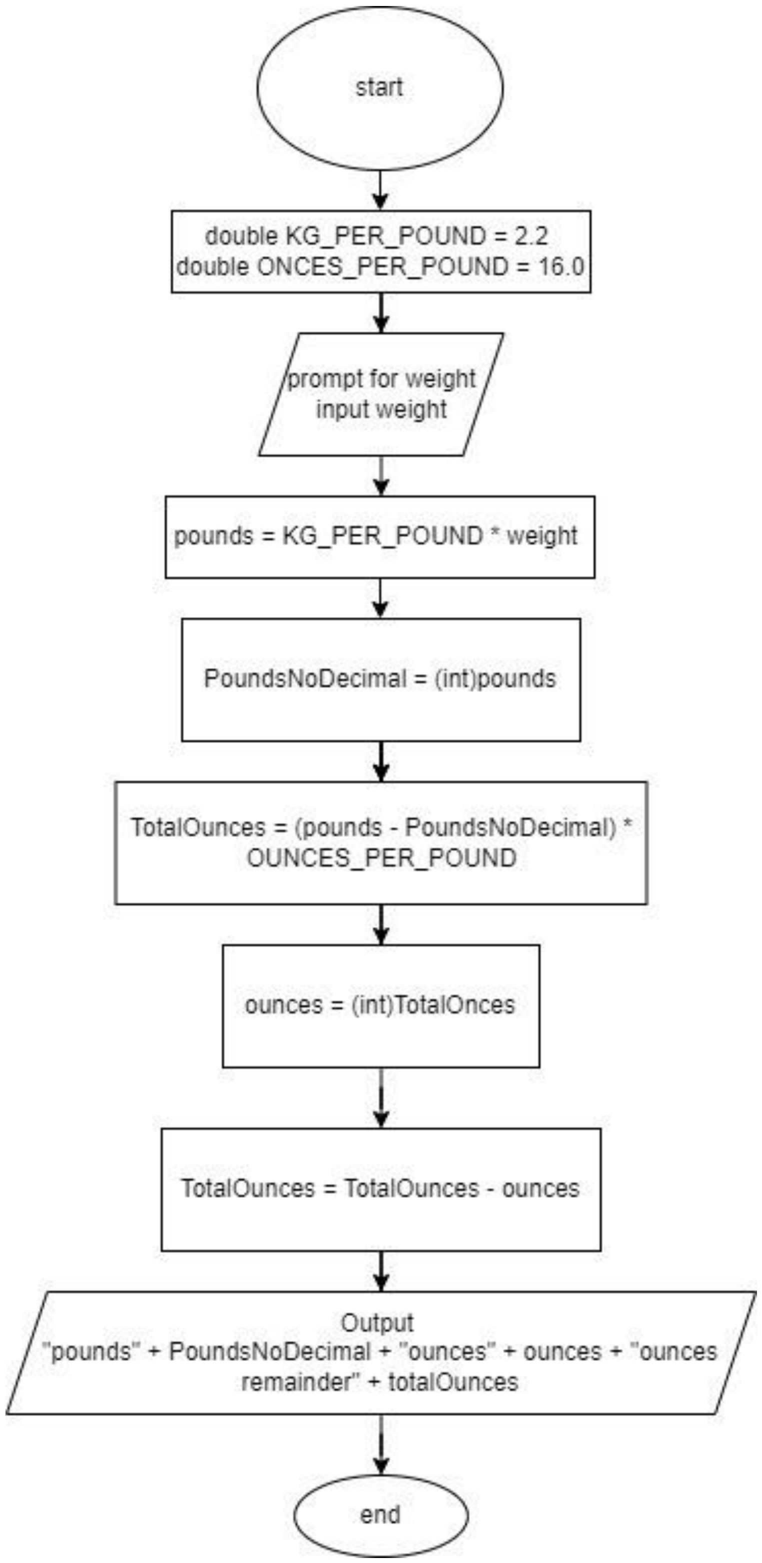
- Kilogram can have decimals
- Output must separate decimals to smaller unit
    - Ex. 4.78 kg = 10.5381 pounds however the output must be 10 pounds, 8 ounces and .26 ounces remainder

How you solved the problem:

I) Pseudo Code

```
  const double KG_PER_POUND = 2.2
  const double OUNCES_PER_POUND = 16.0
  double weight

  Print Please enter a weight in kilograms:
  Input -> weight

  double weightInPounds = weight * KG_PER_POUND

  int poundsNoDecimal = weightInPounds           // int removes decimal

  // Convert remaining pounds to ounces
  double totalOunces = (weightInPounds – poundsNoDecimal) * OUNCES_PER_POUN
  int ounces = totalOunces                    // int removes decimal
  totalOunces = totalOunces – ounces          // decimal by itself

  Print pounds + poundsNoDecimal + ounces + ounces + ounces remainder +
totalOunces
```

II) **Flow Chart**


start
double KG_PER_POUND = 2.2
double ONCES_PER_POUND = 16.0
prompt for weight
input weight
pounds = KG_PER_POUND * weight
PoundsNoDecimal = (int)pounds
TotalOunces = (pounds - PoundsNoDecimal) * OUNCES_PER_POUND
ounces = (int)TotalOnces
TotalOunces = TotalOunces - ounces
Output
"pounds" + PoundsNoDecimal + "ounces" + ounces + "ounces remainder" + totalOunces
end

Error analysis:

errors encountered during the development and how they were resolved:

- Remove line numbers (should not copy with the code)
- Remove space and semicolon in the **INCLUDE STATEMENT**
- Remove extra equal sign in **KGPERPOUND** assignment
  - Rename **KGPERPOUND** to **KG_PER_POUND**
    - (Replace all instances of variable with new name)
- Add semicolon to **OUNCESPERPOUND** variable declaration
  - Rename **OUNCESPERPOUND** to **OUNCES_PER_POUND**
    - (Replace all instances of variable with new name)
- Change weight from type int to type double
  - Change scanf format identifier to **%LF**
- Add semicolon to weightInPounds assignment statement
- Remove line 12 [weightInPounds = weightInPounds - pounds;]
  - Rename **POUNDS** to **POUNDSNODECIMAL**
    - (Replace all instances of variable with new name)
- Change totalOunces assignment statement to use remainder of pounds instead of total pounds
- Remove space in scanf format identifier and newline characters
- Reformat to be more readable

Conclusion and self-assessment:

What you learned in the lab.

In this lab, I learned how to use integer truncation to achieve the desired output and the potential risks of unwanted integer truncation from integer variables.

This lab reinforced my understanding of integer truncation, I/O operations, pseudo code, flow charts and debugging.

The final part of this lab improved my ability to debug programs. It also gave me a deeper understanding of the C programming language.