Lab 1: Writing, Compiling, Executing Programs and Implementing Pseudocode in C

# Part 1

## Problem – Objective

Create a C program in Visual Studio Code that, once compiled, can output alpha-numeric characters to the terminal that resembles a given block of text.

## Assumptions and Constraints

* The given block of text is constant and cannot be changed other than inside the .c file.
* The input and output are accessed through a terminal.
* Once the expected output appears in the terminal, the program is terminated.

## Pseudocode

“C uses escape sequences for a variety of purposes. (newline)

Some are: (newline)

(increase indent x2) to print (backslash)“, use (backslash)\” (newline)

(increase indent x2) to print (backslash)\, use (backslash)\(backslash)\ (newline)

(increase indent x2) to go to a new line, use (backslash)\n (newline)

(increase indent) However to print (percent)%, use (percent)%(percent)%”

# Part 2

## Problem – Objective

Create a C program that, once compiled, calculates and outputs to the terminal the total cost and the cost per person of a bill after prompting the user for the number of people, the tip percentage, and the cost of the bill before tip.

## Assumptions and Constraints

* The user can access and input their bill’s initial cost, tip percentage and the number of people to split inside the terminal.
* The tip must be inputted as a number that represents the percentage of the bill that is added to the initial cost.
* The bill is split equally between each person.
* The total cost of the bill and the cost per person is calculated and output by the program to two decimal places.

## Flowchart

Start

End

Prompt for bill

Input: bill

Prompt for tip

Input: tip

total = bill \* (1 + tip/100)

Output total

Prompt for # of people

Input: # of people

cost per person = total / # of people

Output: cost per person

## Pseudocode

Output -> statement prompting for bill

Bill amount <- received input

Output -> statement prompting for tip percentage

Tip percentage <- received input

Calculate bill total = bill amount \* (1 + tip percentage/100)

Output -> statement prompting for number of people to split with

Number of people to split <- received input

Calculate cost per person = bill total / number of people to split

Output -> bill total (round to 2 decimal places)

Output -> cost per person

# Part 3

## Problem – Objective

Revise and debug a C program that, once compiled, prompts the user for a mass in kilograms that is recalculated to pounds, ounces, and fractional ounces to two decimal places.

## Assumptions and Constraints

* The user inputs a mass in kilograms that can be integer or fractional value.
* The program is terminated once one entry is submitted and the conversion to pounds, ounces and fractional ounces is output to terminal.

## Flowchart

End

Prompt for mass (kg)

Input: mass (kg)

Output: mass (pounds, ounces, frac. ounces)

Start

hold = mass (kg) \* 2.20

Integer typecast:

(hold) -> pounds

hold -= pounds

hold \*= 16

Integer typecast:

(hold) -> ounces

hold -= ounces

## Pseudocode

Output -> statement prompting user for mass (kg)

Mass(kg) <- received input

Mass(lb) = mass(kg) \* 2.20

Pounds = typecast to integer: mass(lb)

Mass(lb) -= pounds

Mass(oz) = pounds \* 16

Ounces = typecast to integer: mass(oz)

Mass(oz) -= ounces

Output -> pounds + “ pounds,” + ounces + “ ounces,” + (round to 2 decimal places: mass(oz)) + “ ounces remainder”

Comment - typecast to integer does the equivalent of a mathematical floor() function.

# Conclusion

In this lab, I learned how to automate specific tasks to C programs based on my knowledge of variables, variable types, I/O operations and mathematical operations.

I learned to use special characters that change the text to make the output to terminal resemble a desired result. I also debugged a program that had multiple compile-time and runtime errors by modifying the source code to achieve the objective of converting a user input into a desired output.

This lab reinforces my understanding of I/O and mathematical operations in C, strengthening my ability to convert a described task into pseudocode. Such pseudocode is then properly implemented in C which results in an executable that automates such task.

After writing programs, I also strengthened my ability to debug my own and others’ programs to prevent compile-time, runtime and logic errors due to my mistakes.