ENGG1410: Introductory Programming for Engineers

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

School of Engineering, University of Guelph
Fall 2024

Start Date: Week #2 2024
Duration: 1 Week

1 Objectives:

The primary objective of this laboratory exercise is to enable students to effectively translate high-level pseudocode representations into functional C code. Emphasis is on basic input output and using processing operators. Furthermore, students will gain firsthand experience in

- Going through the steps of implementing a simple algorithm using C
- Writing the source code – using high-level programming language C to implement a pseudocode representing the solution to a given problem statement.
- Understanding programming logic and simple C syntax `printf( ), scanf( )`
- Compiling the source code – translating the source code to machine code, and then running the executable.
- Diagnosing and rectifying both compile-time and run-time errors when they arise, enhancing the problem-solving and debugging skills.

2 Introduction

Specifically in lab 1 you will be writing two simple C prgrams and debugging a third program. In the first program you will print some basic messages, with intention to test your knowledge and understanding of `printf`. In the second program you will write a program to perform simple calculations. The algorithm, pseudocode and all the work is already carried out, you need to implement it in C. In the third part, you will debug and fix a program. You will "read" the program and check for possible errors, you will analyze it with and without access to the computer for error checking and correction. This part assess your ability to find and correct syntax errors, logic errors, expression evaluation and data type conversion - implicit

and explicit - casting issues among other possible errors.

The convention used in the sample output examples that will be provided to you as part of the problem statement is that the text that would be entered by the program user is displayed using a **bold** font. The text <**enter**> stands for the user pressing the enter key on the keyboard. On some keyboards, the enter key may be labeled <`return`>. When you execute your programs, the user's text will not be displayed in bold and <`enter`> will not be shown.

In Visual Studio Code, place your solution for each part in a separate directory, so that Visual Studio Code will not attempt to compile multiple files and link them together. For example, have four different directories, `lab1part1, lab1part2,` and `lab1part3`. Each should have one .c file corresponding to your solution for that part. Please ensure you are selecting the folder with the proper **.c** file you want to compile.

2.1 Pseudocode and Flowcharts

Pseudocode is an informal, high-level description of a computer program or algorithm, written in a human-readable format that outlines the logic and steps of the solution without adhering to specific syntax of any programming language. *flowchart* is a visual representation of the sequence of steps and decision points in a process or algorithm, using standardized symbols to depict the flow of control from one step to the next

When presented with a problem statement, begin by formulating a solution and then use pseudocode and/or flowcharts to conceptualize and represent the step-by-step flow of your solution which will be your program. Pseudocode and/or flowchats are tools that offer a structured approach to illustrating the logical steps, decisions and looping probably involved in solving the problem at hand.

This lab exercise is devised to bridge the gap between such abstract representations and actual code implementation using the C programming language. Implementing programs from pseudocode or flowcharts helps in refining logical reasoning and translating abstract thoughts into tangible code. Once this is done, there comes the role of debugging.

Debugging is a methodical process of finding and reducing the number of bugs (defects or errors) in a computer program, thus making it behave correctly as originally intended and expected.

2.2 Debugging and Types of Errors

There are two main types of errors that need debugging:

1. **Compile-time Errors**: These errors occur during the compilation phase of the program. They usually arise due to issues with syntax, such as missing semicolons, mismatched brackets, undeclared variables, or the misuse of language reserved keywords. A program with compile-time errors will not produce an executable until these errors are rectified. Students will need to closely inspect error messages from the compiler, which usually indicate the file and line number where the error occurred, aiding in the debugging process.These misuse of language constructs, such as syntax errors are normally fairly easy to find by using compiler tools and warnings to fix reported problems.

2. **Run-time Errors**: Unlike compile-time errors, *run-time errors* occur only when the program is executed. They result from logical errors or unforeseen conditions in the code. Examples include division by zero, unexpected inputs, declaring variables with incorrect types, accessing out-of-bounds array indices, or dereferencing null pointers. Although the code may compile successfully, these errors can cause unexpected behavior, crashes, or incorrect outputs during execution. Students will learn to use debugging tools and techniques, such as adding watchpoints, print statements for

"intermidiate" variables of interest, and debuggers, to identify and resolve these issues. These errors can be challenging to diagnose, as they lead to incorrect outputs or cause the program to "crash" during execution.

Part 2 and 3 will examine your ability to go from a problem statement and a solution algorithm *pseudocode* to an executable code while methodically debugging both compile and run-time errors in your C code.

3 Part 1 – Printing

Write a C program that produces the following output. The output must be **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 %%
```

4 Part 2 – Bill Splitting Program

This part will present to you the different steps a programmer encouter starting from getting a problem statement to thinking of an algorithm for solving the problem to writing a pseudocode and finally implementing the solution into sourcecode in a programming language. Specifically, you will carry the implementation part.

4.1 problem statement and sample input output sequence

When dining at a restaurant, it is common to add a tip to the bill based on the quality of service. You will write a program that calculates the total amount of the bill after adding a tip based on the percentage the user chooses. The program will also split the total bill evenly among a group of people.

The inputs to your program consist of: (i) The original bill amount (in dollars). (ii) The tip percentage (as a percentage of the bill). (iii) The number of people splitting the bill. The program should:

- Calculate the total amount after adding the tip.
- Calculate the amount each person should pay when the bill is split equally among the group.
- Both outputs should be rounded to two decimal places.

Here is a sample output from the execution of the program:
`Enter the original bill amount:` **100.00** <**enter**>
`Enter the tip percentage:` **15** <**enter**>
`Enter the number of people splitting the bill:` **4** <**enter**>
`The total bill including tip is:` 115.00
`Each person should pay:` 28.75

4.2 Algorithm

The algorithm is trivial, first consider what are your input and output variables and decide on the appropriate data type to associate with them. The processing is straight forward as you have a closed form calculations to calculate the output and get the desired result and hence the needed solution. We only require the multiplication and division of the inputs to get the output.

4.3 Pseudocode

START

1. Display the message: `Enter the original bill amount:`
2. Read the bill amount from the user and store it in 'billAmount'
3. Display the message: `Enter the tip percentage:`
4. Read the tip percentage from the user and store it in 'tipPercentage'
5. Display the message: `Enter the number of people splitting the bill:`
6. Read the number of people from the user and store it in 'numPeople'
7. Calculate the tip amount: Set 'tipAmount' to (billAmount * tipPercentage) / 100
8. Calculate the total bill amount: Set 'totalBill' to billAmount + tipAmount
9. Calculate the amount each person should pay: Set 'amountPerPerson' to totalBill / numPeople
10. Display the message: `The total bill including tip is:` followed by 'totalBill', rounded to two decimal places
11. Display the message: `Each person should pay:` followed by 'amountPerPerson', rounded to two decimal places

    END

From the above it is also clear we will need three input variables: billAmount, tipPercentage and numPeople. We will also need two output variables: totalBill and amountPerPerson, and an intermediate variable tipAmount. In your code you will need to **Declare** these variable before scanning them.

4.4 Code Implementation, compiling and running

Use VS Code to implement the pseudocode in C, then compile and run your code, and present your solution to your TA. For now, we will assume that the user will input valid and expected values, without any errors or invalid data.

5 Part 3 - Debugging

This problem involves debugging a C program that asks the user to input a weight (in kilograms) and converts this weight into pounds, ounces, and the remaining fractional portion in ounces, rounded to two decimal places. The conversion factors used are: 1 kilogram = 2.20 pounds and 1 pound = 16 ounces. For example, if the user enters 4.78 kilograms, the output should be:
`10 pounds, 8 ounces, 0.26 ounces remainder.` The prompt to the user should appear as:
`Please enter a weight in kilograms:` on a single line.
Note: If the output includes something like: `1 pounds, 2 ounces, 0.00 ounces remainder`, there is no need to modify the singular forms to match (i.e., leave it as "pounds" and "ounces"). Similarly, if pounds or ounces have a value of 0, they should still be printed in the output.

A beginner programmer has written a C program to solve this problem, but the program contains both compilation and runtime errors. The code is provided below.

Since debugging is an important skill, ENGG1410 students are encouraged to fix this program. You may face similar issues during exams, so take this opportunity to practice. Use the compiler's error messages to fix compilation issues and employ the debugger to catch runtime problems. Inspect the values of variables at each step to ensure the program behaves as expected and solves the problem correctly.

```c
#include <stdio.h>;
  int main(void) {
   const double KgPerPound == 2.20;
    const double OuncesPerPound = 16.0
        int weight;
   printf("Please enter a weight in kilograms: ");
   scanf("%d", &weight);
   double weightInPounds = weight * KgPerPound;

   // Truncate fractional part to get the whole number of pounds
   int pounds = weightInPounds;
   weightInPounds = weightInPounds - pounds;

   // Convert remaining pounds to ounces
   double totalOunces = weightInPounds * OuncesPerPound;

   // Truncate fractional part to get whole ounces
   int ounces = totalOunces;
   totalOunces =  totalOunces - ounces;
   printf("%d pounds, %d ounces, %.3f ounces remainder\n",
           pounds, ounces, totalOunces);
   return;
}
```

6 Requirements

1. Write, compile and **execute** the program in Part 1 and Demo it to your TA.
2. Implement, compile and **execute** the program in Part 2 and Demo it to your TA.
3. Fix the program in part 3 to work correctly. (consider different test cases).

7 Deliverables

7.1 DEMO:

- You will need to demonstrate to the Teaching Assistant that your programs fulfill the requirements and are correct. TAs might run test cases to check correctness.
- The DEMO will take place during the LAB hours and not outside these hours.

7.2 REPORT

- You will provide a report with your solutions to the different parts of the lab and your source code and executables. All these should be uploaded to the coure DropBox in due date to get your full grade and avoid penalties.

7.2.1 Lab Report File:

Below is the general format of a final report (when required).
Note: Not all sections mentioned below are needed, they are given as a guidance.

1. Title Page: *Required*
   - Course #, and Date
   - Lab # and name of experiment
   - Your Group #, and Names (if lab is carried as a group)
2. **Start a new page** and explain how you implemented your code by providing the following:
   (a) **Problem Statement**,
      i. Briefly describe the problem(s) solved in the lab.
   (b) **Assumptions and Constraints** - if there were any.
      i. Assumptions could be for example specific number used e.g. taxes 7%.
      ii. Constraints could be for example no optimization with compilation.
   (c) **How you solved the Problem**,
      i. Pseudo Code.
      ii. Flowchart.
      iii. Block Diagram.
      iv. Another method ...
   (d) **System Overview & Justification of Design** *optional*

i. Give an overview of the system to be designed.

ii. Briefly explain how the system works and reasons behind the design.

3. **Error Analysis** *optional*

   (a) Confirm no syntax errors are present.

   (b) Confirm no semantic and logical errors are present.

   (c) Describe any problems with your program.

   (d) If no problems in the final system, describe problems/errors encountered during the development and how they were resolved.

4. **Conclusion and self assessment** *Required*

   (a) What you learned in the lab.

   (b) What level of understanding or experteese you think you attained.

   (c) Any suggestions *optional*

7.2.2 Source Files and Executables:

Inlcude your source code to the different lab parts.

The Report as well as your sourcecode files and executables should be uploaded to the course Dropbox in due date to get your grade and avoid penalties. You are responsible to provide correct files.