SIMPLE ARITHMETIC

Lab 3

SUBMITTED BY:

CODY BROOKS

SUBMISSION DATE:

2015-09-23

Problem

The purpose of Problem 1 was to learn and practice debugging skills by looking at code that someone else already wrote and looking for errors.

Problem 2 exercised our understanding of how the GCC compiler performs arithmetic functions, specifically numeric data types and order of operations.

Problem 3 showed us how multiple programs can be ran together to result in a more desirable outcome as well as the ability to look at another person's undocumented code and try to figure out what it's purpose and function is.

Analysis

Problem 1: The primary constraint was the code that was provided. We had to use the existing code and fix the errors to solve the problem. There was no input to the program itself, and the expected output was simple arithmetic. However, from a different perspective the code given to us, the engineers, was the input and the expected output was error-less code. We had to assume that nothing in the program worked and carefully pay attention to the compiler's error messages in order to properly identify what the actual problems were.

Problem 2: The constraints of this problem was that we had to use the given syntax to calculate simple arithmetic. Like Problem 1, there was no input to the program but the input to the engineers was a list of equations which may or may not have had correct syntax. The output of the program was the values calculated from each given equation.

Problem 3: The constraints of this problem lied in the code we were given. We were given both the code to interpret the data coming from the Esplora as well as the code that translates the data from `explore.exe` into data that is more readable and easier to understand. The input of `explore.exe` is data from the Esplora via the computer's COM port. The output is raw numeric data describing the Esplora's acceleration. The input for `lab3-3.c` is the raw data from `explore.exe`. The output is data that is formatted to be more readable, including text that describes what each set of numbers means.

Design

Problem 1: Compile the code as-is. Look at the top error message to see what line the error is on and a description of what it might be. Once that error is addressed, re-compile the code and look at the new top error message. Repeat this process until the code compiles successfully. Run the executable and evaluate the output. If the output seems correct, you are finished. If not, look in the code for minor errors such as integer division.

Problem 2: Look at each equation without any parentheses and determine the logical order of operations, and replace parentheses accordingly. Then determine if the result will be an integer or floating point number. If the result will be floating point, prevent integer division by adding '.0' to the end of any whole numbers in division (i.e. change '20 / 3' to '20.0 / 3.0'). Also make sure that the variable that the result is assigned to is the correct numeric representation.

Problem 3: Since no code was written for this problem, the design comes from how the purpose of the given code is deduced. First, run `explore.exe` without piping it through the given code. Look at the output, then compare it to running `explore.exe` while piped through the given code. See what information is the same and what information is added; pay attention to

the text that is also printed with the numerical data. Also look for any comments that may exist in the given source code.

Testing

Problem 1: Type each equation into a calculator. Is the result the same between the program and the calculator?

Problem 2: Type each equation into a calculator. Is the result the same between the program and the calculator?

Problem 3: Run `explore.exe` without piping it through the given program. Record what you think the outcome would be if you piped `explore.exe` through the given code, and check your results. Be sure to do any mathematical computations that the given code may perform on the data from `explore.exe`.

Comments

One thing that I learned in the lab was that it is often faster to solve one problem at a time and check to see if it was fixed or not before moving on to the next one rather than trying to solve multiple problems at the same time and missing a mistake that persists. There is nothing I would change about the way I did the lab.