SY DE 121 – Digital Computation - Lab #2

Recommendation: Organize your labs into different folders i.e., create a folder "Lab02" to contain all the work in this lab. Within this directory, you can create folders "Lab02 Ex1", and so forth. Once you have submitted the lab, *you can delete all the project files* - all you generally need to keep is your *.cpp and *.h files as well as any data files. This will help to reduce your disk space usage.

Note: you will be required to prepare a design plan prior to the start of the lab for Exercise 2 (marks are assigned to performing this task properly).

Exercise 1: Experimenting with Inappropriate Input

Learning Objective: To get a general idea of what can happen when a C++ program receives invalid input.

Read This First

When you are just beginning to write programs, getting programs to work with correct input is a challenge, so it is reasonable to pretend for now that users will never type in incorrect input. However, programs that do not have code designed to detect and deal with invalid input may generate nonsense output.

Invalid input usually does not cause a C++ program to crash. (A crash occurs when a program unexpectedly stops running; a crash is often accompanied by error messages in hard-to-understand computer jargon.) Instead, the program keeps running after it gets invalid input, but some of the program's variables will not have sensible values.

What To Do

Download the program "EnterData.cpp" from LEARN. Build the executable and run it a few times, typing in appropriate input when the program prompts you.

Now run the program several more times with various types of inappropriate input, **as listed below**. Sometimes you are not told what to enter in response to the prompt for the_double; that's because sometimes you won't get a chance to respond to that prompt. After each experiment, write down the final values for the variables the_int and the_double.

1. Enter **-42** in response to the prompt for the_int, then enter **!@#$** in response to the prompt for the_double.
2. Enter **35 degrees** in response to the prompt for the_int.
3. Enter **100,000** (including the comma) in response to the prompt for the_int.
4. Enter **876 543** (both numbers on the same line) in response to the prompt for the_int.
5. Enter **x 123.4** in response to the prompt for the_int.
6. Enter **12.3456** in response to the prompt for the_int.
7. Enter **.995** in response to the prompt for the_int.
8. Enter 987654321987654321 in response to the prompt for the_int, and then 12345 in response to the prompt for the_double.

Question

The experiments you have performed will not let you draw very precise conclusions about how characters in the input stream are processed. However, there are a couple of somewhat imprecise observations you should have made with regards to how the inputs interact/interfere with each other. Indicate two separate observations.

What to Submit

Hand in the following. Instead of creating and submitting a *.cpp file, just create a *.txt (or MSWord) file with the same filename procedure and upload to LEARN.

1) A list of observations for the eight experiments (please list the information in a readable format).
2) A short description of the two observations.

Exercise 2: Office Stationery Supplies

Learning objectives: To gain experience with input, computation with assignment statements and arithmetic expressions, and output.

Hints:

- Each time the user enters some input, you should get into the habit of displaying back what they typed. This is called "echoing the inputs". If you ask the user for a number (eg. Please enter a number:) and the user enters 6, you should redisplay what they entered along with a short message (eg. You entered 6). This allows the user to check to see if what they entered was correct, and will help you detect errors in your program.

- Be sure to try and use variable names that convey some sort of meaning about their use. This makes the program easier to read and understand.
- You are expected to use constants ('const') in this exercise.

Read This First

An industrious friend of yours has recently started up a small office stationery supply company, The Write Stuff, to service engineering companies. Since he is just starting out, he decides to only sell pencils, pens, and correction fluid. When placing an order, customers are required to tell him how many employees there are, what percentage of those employees are junior engineers and what percentage are senior engineers. There are also employees that are neither junior nor senior engineers; these employees are assumed to be administrative staff.

Your friend does some market research, and decides that each type of employee uses (on average) the following quantities of supplies in a typical month:

Type of employee	Pencils	Pens	Correction fluid
Junior engineer	8	7	60 ml
Administrative	2	6	40 ml
Senior engineer	5	3	12 ml

He supplies pencils packaged in boxes of 25, pens packaged in boxes of 10, and correction fluid in bottles of 200 ml.

What to do

Suppose your friend wins a major contract, and needs to supply stationery to a company of 480 employees, of which 55 per cent are junior engineers, and 20 per cent are senior engineers. He wants to be able to quickly determine the total number of boxes of pencils and pens as well as the number of correction fluid bottles required. Your friend asks you to provide a program that he can use on an ongoing basis, not just as a solution for this particular case. Your friend feels quite confident that the table data above will remain constant for some time.

1) Prepare a problem analysis (include the problem statement and define the inputs and outputs).
2) Design your system using a top-down decomposition. Use the example data above for the hand calculation portion.
3) You will have to upload a file containing the problem analysis and top-down decomposition to LEARN. You can use software (Powerpoint, MSWord) or write by hand and scan in (you need to find a scanner). An example problem analysis and top-down decomposition is found on LEARN as *lab0202_supplement.pdf*.
4) Code and debug your solution (reminder – use source file *lab0202.cpp*)
5) Test your code by ensuring the computer solution matches your hand calculated solution. You should also test using additional data sets to ensure the program is working properly.
6) What happens when you enter illogical data? For example, what happens if you enter negative values for the number of employees? What happens is the sum of the junior and senior engineer percentages exceeds 100%?? We will soon learn how to take care of such problems (do not be concerned about fixing them just yet).

Use type `double` for all numbers. Do not worry about the fact that your program will print answers including fractions of boxes and bottles. You will learn how to take care of this later in the course.

Hint: You might have to use a `system("pause");` statement to prevent the window from closing quickly. If the return is reached, your console window is typically terminated. See Lab 1 for an example of how to use this.

What to Submit

1. Create a zip file called "Lab2.zip". In this file, place your source code for Exercise #1
2. In your SYDE 121 webpage, under the "Dropbox", find the current Lab Assignment dropbox (for this lab it would be "Laboratory Assignment 2".
3. Upload the following files. Remember to follow the instructions given in Lab 1 on naming conventions to use, and what type of files to submit.

   Exercise 1: Your file containing the necessary responses.

   Exercise 2: Your file containing the problem analysis and top-down decomposition as well as your source *lab0202.cpp* file.

*** Due Date ***

All material must be submitted to LEARN by Friday, September 22 by 5:00pm.

Note that lab assignments are typically due the Friday afternoon after the lab session.