# SYDE 121 – Digital Computation - Lab #2

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

## OVERVIEW

1. Log on to your Nexus account and complete Exercise 1.
2. Complete Exercise 2.
3. Submit your lab to your LEARN dropbox.

**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

1. *Create a lab0201.txt text file*
   Open Windows NotePad (or the Mac text editor equivalent). Save the file as *lab0201.txt* to your "Lab02" directory.

2. *Download "EnterData.cpp" file from LEARN*
   Download the program "EnterData.cpp" from the LEARN Lab 2 dropbox, and save it in your Lab02 directory.

3. *Build and Run "EnterData.cpp"*
   Open the EnterData.cpp in Dev-C++ (double clicking it from the Lab02 directly should open it in Dev-C++ automatically. Build the executable (Execute→Compile & Run) and run it a few times, typing in appropriate input when the program prompts you.

4. *Experiment with Inappropriate Input*
   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, use your lab0201.txt file to record the final values for the variables the_int and the_double. Label each observation 4a, 4b, and so on. (Please list the information in a readable format.)

   a) Enter **-400** in response to the prompt for the_int, then enter **@#$!** in response to the prompt for the_double.

   b) Enter **.995** in response to the prompt for the_int.

   c) Enter **987654321987654321** in response to the prompt for the_int, and then **12345** in response to the prompt for the_double.

   d) Enter **876 543** (both numbers on the same line) in response to the prompt for the_int.

   e) Enter **x 123.4** in response to the prompt for the_int.

   f) Enter **12.3456** in response to the prompt for the_int.

   g) Enter **-12 degrees** in response to the prompt for the_int.

   h) Enter **100,000** (including the comma) in response to the prompt for the_int.

5. *Answer the following question in lab0201.txt*
   *Question:*
   The experiments you just 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. Write a short description of two separate observations in your lab0201.txt file, label your response as 5.

## 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 (e.g. Please enter a number: ) and the user enters 6, you should redisplay what they entered along with a short message (e.g. 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.
- Make sure you use variable names that convey some sort of meaning about their use. This makes the program easier to read and understand.
- You should have already read the section on "Naming Constants" in your Savitch textbook. You are expected to use constants 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 erasers. 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 | Erasers |
|---|---|---|---|
| Junior engineer | 10 | 5 | 3 |
| Administrative | 2 | 10 | 1 |
| Senior engineer | 7 | 5 | 2 |

He supplies pencils packaged in boxes of 25, pens packaged in boxes of 10, and erasers in boxes of 10.

### 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, pens, and erasers 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. *Create a lab0202.txt text file*
   Create a *lab0202.txt* in a text editor, and save it to your "Lab02" directory.

2. *Complete a problem analysis*
   Prepare a problem analysis (include the problem statement and define the inputs and outputs). Record the problem analysis in the lab0202.txt file. It is highly recommended that each student in the pair create a draft analysis separately and then bring them together to discuss and combine into a final analysis version.

   **Hint:** The attached file, *lab0202_supplement.pdf*, in the Lab 2 dropbox shows a simple example of a complete problem analysis and top-down decomposition.

3. *Complete a top-down decomposition*
   Design your system using a top-down decomposition. Use the example data above for the by-hand calculation portion. Record your completed decomposition in lab0202.txt in a clearly understandable format. For this portion you may find it easier to use a layout program such as PowerPoint. If you chose to use such a layout program, save the solution as an image file (e.g. lab0202.jpg format). Make sure all the text in the final image is clearly legible.

4. *Code and debug your solution.*
   Code your solution in a source file called *lab0202.cpp*.

   Test your code by ensuring the computer solution matches your by-hand calculated solution. You should also test using additional data sets to ensure the program is working properly. The TAs will provide you with test cases that specify same input and expected output for which your code **must** work. These cases, along with similar data will be used to test your program during marking.

   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, see **bonus feature** below if you already know how to do this).

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

Remember to add your commented header to the top of the file, and include appropriate comments throughout your program (see Lab 1 for details).

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

**BONUS FEATURE: Error Checking (Optional)**

As in any program that expects the user to input data, the user may enter the wrong data. The **clearer your instructions** to the user are, the less likely this is to happen; however, the user may simply make a mistake and enter unintended (and inappropriate) input. A good program should always check for such inappropriate data, and if found, should always provide the user an opportunity to re-enter the data. Again, providing clear instructions, for instance, that specify what appropriate input looks like, is important.

For bonus marks, include code that checks any input provided by the user for correctness. If correct, continue with the program. If incorrect, prompt the user to re-enter the data. You should clearly specify that incorrect data has been entered when asking the user to re-enter the data. Make these changes directly to your *lab0202.cpp* program.

5. *Create a README.txt text file*
   Create *README_Lab0202.txt* to explain how to run your program (see Lab 1 for instructions on creating README files.

   If you have completed the Bonus Feature (Error Checking), clearly indicate this in your README file.

**3. SUBMIT your lab to your LEARN dropbox.**

1. In your SYDE 121 webpage, under the "Dropbox", find the current Lab Assignment dropbox (for this lab it would be "Laboratory Assignment 2".
2. 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.
   a) Your *lab0201.txt* file from Exercise 1.
   b) Your *lab0202.txt* file from Exercise 2.
   c) Your *lab0202.jpg* (or other image format), if you used a layout program for your problem decomposition in Exercise 2.
   d) Your *lab0202.cpp* C++ source code for Exercise 2.
   e) Your *README_Lab0202.txt* README file for Exercise 2.

**Due Date**

All material must be submitted to your LEARN dropbox by Friday, Sept. 26 by 1:30pm.

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