SUPA C++ Labs 1+2

Jonathan Jamieson

November 13, 2023

Hopefully you will all have done the preliminary exercises ahead of the lab to make sure everything is working. Don't worry if not as you have time to do them here. The preliminary exercises do not count towards your final mark.

This first assignment is divided into multiple parts, each exploring topics that we covered during the lessons on 15/11 and 22/11.

The assignment is due by the 25th of November

Try and get into the habit of writing your name and date of creation as comments at the top of files and try to be "user friendly" when you write code:

- Comments are your friend, add them as you go to keep track of what the code (should) be doing, both for yourself and for anyone who has to read your code in the future
- Try and stress-test your code to account for any possible errors due to edge cases or mistakes in the input given by the user

1 Submission

Submission of your final code will be handled via git following the instructions at the bottom of the README. To help during marking please avoid uploading too many extra files beyond what is needed to complete the exercises, or at the very least make it obvious which files are important.

2 Preliminary

(I) Create a program that when executed prints Hello World! on the terminal.

(II) Declare two variables, x and y, and assign them the values x=2.3 and y=4.5. Assuming these are x and y components of a 2-D vector, compute the magnitude of the vector and have your program print the answer to the screen.

(III) Now create a function to calculate the magnitude of the vector which takes as input from the user x and y components. Use this function to calculate the magnitude of the vector in (II) and make sure the answer which is printed out is the same as in point (II)

3 Main assignment

The aim of this assignment is to get used to reading and manipulating different types of data using functions and to go through the process of designing and building up a complex piece of code in steps. The exercise is split into several steps through which we will build up the required functions bit-by-bit. The final result should be a steering file, "AnalyseData.cxx" which the user will run and interact with via the command line in order to read data from a file and decide what actions should be applied to it. The steering file will then call out to a series of functions defined in an external file in order to modify the data and write out the results.

The design brief is to write some software that allows the user to read in (x,y) coordinate data in plain text format and be able to perform the following tasks:

1. Print n lines of the data in plain text to the terminal
2. Calculate the magnitude of each data point assuming the coordinates make a vector with (0,0)
3. Fit a straight line to the data using the least squares method and assess the goodness of this fit using a χ^2 test (it shouldn't be very good)
   - Least squares method:

     $y = px + q$ with $p = \frac{N\sum x_i y_i - \sum x_i \sum y_i}{N\sum x_i^2 - \sum x_i \sum x_i}$, and $q = \frac{\sum x_i^2 \sum y_i - \sum x_i y_i \sum x_i}{N\sum x_i^2 - \sum x_i \sum x_i}$
   - χ^2 test:

     $\chi^2 = \sum \frac{(O_i - E_i)^2}{\sigma_i^2}$ where O_i are observed values, E_i are the expected values, and σ_i is the expected error on the measurement.
4. Calculate x^y for each data point (without using a for loop or in-built power function)
5. Write out the results from any of the above actions to a file, using a different file name for each task

4 Instructions

1)

It is always good to start with the simplest task and then build up to more complex functionality, so lets first only worry about reading the plain text file and making sure we understand what the data looks like. Create a file named "AnalyseData.cxx" which first opens the file "input2D_float.txt", reads the contents line-by-line and prints the contents out to the terminal.
Additionally you can see a visualisation of the data by opening "Outputs/png/plot.png". This plot can also be re-generated by running "gnuplot plotdata.gp" in the command line

2)

Now we know what the data looks like, pick a sensible data structure to hold the x and y values, and modify your code to first write the (x, y) values into this structure and then print to the terminal from this rather than from the file stream. (We do not know how many data points there are going to be apriori so ideally we want a dynamic data structure)

3)

Move the reading and printing functionality into two separate functions and modify the printing function to only print n lines. If n is larger than the total number of data points then the code should give a warning and then print the first 5 lines.

(For all functions beyond this point you can assume we always want to analyse the full dataset)

4)

Now let's add another function which calculates the magnitude of each data point assuming they are the (x, y) coordinates of a vector. Recall that in the final code we want to be able to write out the results of each action to a file so let's prepare for this now and save the magnitudes in another sensible data structure. (Hint: It's ok to use a dynamic structure again here even though we know exactly how many values we need to store, the alternatives are more complicated and go out of the scope of this lab)

5)

The user will also want to know the magnitude values as they are calculated, so let's print them out as well. Rather than writing new functionality, let's have the calculate_magnitude function just call the print function we have already written. (This is likely to require writing an overload for the print function so it can take in different objects as inputs)

6)

Now let's clean things up a little, we will move the read, print, and calculate_magnitude functions out of the main file by declaring them in a file called CustomFunctions.h, and putting the implementation in CustomFunctions.cxx. Now modify "AnalyseData.cxx" to ask the user which function they would like to use (print or magnitude) and if printing also ask for how many lines to print. (Hint: Remember we will need to modify the command to compile our executable now that some needed functions are defined in other files, a simple makefile may make things easier)

7)

Add a new function into CustomFunctions which takes the (x, y) data and fits a straight line function ($y = mx + c$) using the least squares method. Save the final function as a string to a new file and also print it to the terminal. Again try and re-use the same print functionality you already have.

8)

We can assess how good this function is at modelling the data by performing a χ^2 fit. Modify the least squares code to also calculate the χ^2/NDF and include it in the saving and printing functionality. Here NDF is the number of degrees of freedom in your fit. For this you will need to additionally read in the expected error on each data point from the file "error2D_float.txt", thankfully we already have a function to read data inputs in this format.

9)

Add a new function to calculate x^y for each data point with y rounded to nearest whole number and without using the pow function or a loop. Hint: The exponent needs to be converted to an integer before you can use it

10)

Write a final function (and any necessary overloads) which takes the output calculated from whichever step was run and saves it to a sensibly named output file. If all of the different functions above are run then there should be 3 separate output files in the end (One for each function the code can perform, except for printing)

11)

Lastly to tie everything together update "AnalyseData.cxx" to extend the instructions that appear on the terminal for the user when the code is executed to include all of the functions we have written. The prompt should ask them which operation to execute (try to use switch!), then the code should perform this action and ask the user if they intend to perform another action or exit.