Project 2 - Statistical Data Processing

Due Apr 28 by 11:59pm **Points** 30 **Submitting** a file upload **File Types** h and cpp **Available** until May 1 at 12:01am

This assignment was locked May 1 at 12:01am.

ST.CHARLES
COMMUNITY COLLEGE

CPT-182 - Programming in C++

Programming Project - Statistical Data Processing (30 Points)

(Number in Question Bank: Project **4.1**)

Project Overview

In this project, you are going to write a C++ program that does some **statistical data processing**. You need to write a class of **`Dynamic_Array`** of floating-point values using pointers, and use it to store the input data which come from an input file. The calculated statistical results should be written to an output file.

In this project, you **must** define a **`Dynamic_Array`** class (using pointers). Using static arrays or vectors for whatever reason will result in **zero** points for the project.

The `Dynamic_Array` Class

In this project, you need to define a dynamic array of floating-point values (type `double`). Please note that in Module **8**, a dynamic array of integers was used as an example. In this project, you are doing almost the same thing except that you are defining a dynamic array of **floating-point values** this time.

It is your responsibility to determine the **`private`** data fields for the dynamic array by yourself.

In addition to the **`private`** data fields, your dynamic array should include the following class-member functions:

- **Default constructor**. You will use the default constructor to initialize a dynamic array in the **`main()`** program. The default capacity of the dynamic array should be **10**.
- **Copy constructor**. Since pointers are involved in the class of dynamic array, it is a **must** to overload the copy constructor.
- **Destructor**. Since pointers are involved in the class of dynamic array, it is a **must** to overload the destructor.
- **Deep-copy assignment operator**. Since pointers are involved in the class of dynamic array, it is a **must** to overload the deep-copy assignment operator. Please review the lectures to study how to overload the operator.
- **Subscript operator** ("[]"). When you are processing the data stored in the dynamic array, you need to use **index** to iterate through the array. Therefore, you **must** overload the subscript operator in the class of dynamic array. Do **not** forget that you need to overload the operator twice, one for **`lvalue`** and the other for **`rvalue`**.
- **`size()` function**. The function returns the size of the dynamic array.
- **`empty()` function**. The function returns **`true`** if the dynamic array is empty; **`false`** otherwise.
- **`resize()` function**. The function does **not** ~~take any argument~~, **nor** ~~return a value~~. This function should be in the **`private`** section of the class since it is **not** expected to be called outside the class. The function doubles the capacity of the dynamic array and keeps the current elements **unchanged**.
- **`push_back()` function**. The function takes a **`double`** as its only argument and returns **`void`**. This function inserts the value passed in to the rear end of the dynamic array. If the dynamic array is full, then you should call the **`resize()`** function to double the capacity first before inserting the value.

For all class-member functions, it is your responsibility to correctly determine the return type, number of arguments, types of arguments, whether the function is a **`const`** function, whether the function is a **`static`** function, and whether the function is a **`friend`** function. **Failing** to do these will result in **losing points**.

The Input File

- The input file is a **plain text file** (filename: `data.txt`).
- In the input file, each row contains exactly **2** floating-point values (except for those empty lines). The first value is the **x**-value; the second value is the **y**-value. In other words, each row in the input file stores an **x**-**y** pair.

- You **cannot** assume (or guess) the number of **x**-**y** pairs in the input file. In other words, no matter how many **x**-**y** pairs are stored in the input file, your program should correctly process all of them.
- There are may be **empty lines** at the beginning, in the middle, and/or at the end of the input file. Your program should smartly skip those empty lines and only process the rows that contain data.
- Please refer to the **sample input files** to better understand the input file format.

The Output File

- The output file is a **plain text file** (filename: `results.txt`).
- The following statistical results should be written to the output file:
  1) Number (count) of **x**-**y** pairs stored in the input file
  2) Average **x**-value and average **y**-value
  3) Standard deviation of **x**-values and standard deviation of **y**-values
  4) Correlation coefficient of **x**-values and **y**-values
  5) Slope (**m**) and intercept (**b**) in the best fitting line, **y** = **mx** + **b**, of **x**-values and **y**-values.
- Please refer to the **sample output files** to better understand the output file format.

The `main()` Program

- You need to create two dynamic arrays to store the **x**-values and **y**-values, respectively.
- Your **must** use appropriate **functional decomposition**. In other words, each of your statistical formulas should be in their own function.
- When you use the sample input files to test your program, your program should generate output files that are byte-to-byte the same as the sample output files.

Formulas for Statistical Data Processing

In the formulas below, the **sigma** $\left(\sum\right)$ indicates **addition**. The **bar over a variable** indicates the **average**.

- Standard deviation (σ)

$$\sigma_x = \sqrt{\frac{\sum_i (x_i - \bar{x})^2}{N}} \qquad \sigma_y = \sqrt{\frac{\sum_i (y_i - \bar{y})^2}{N}}$$

N is the number of **x**-**y** pairs.

- Correlation coefficient (r)

$$r = \frac{\sum_i ((x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}$$

Please note that the denominator terms refer to the **sum of the squared differences** (find the difference, square it, and add up the squares) and **not** ~~squared sum~~.

- Best fitting line

The **least-square best-fit straight line** associated with n points, $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$, has the form:

$y = mx + b$

where

$$m = \frac{n\left(\sum xy\right) - \left(\sum x\right)\left(\sum y\right)}{n\left(\sum x^2\right) - \left(\sum x\right)^2} \qquad b = \frac{\left(\sum y\right) - m\left(\sum x\right)}{n}$$

Here, $\sum$ means "the sum of". Thus,

$$\sum xy = x_1y_1 + x_2y_2 + x_3y_3 + \cdots + x_ny_n$$

$$\sum x = x_1 + x_2 + x_3 + \cdots + x_n$$

$$\sum y = y_1 + y_2 + y_3 + \cdots + y_n$$

$$\sum x^2 = x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2$$

Sample Input and Output Files (Click to Download)

Sample Input File 1 (https://drive.google.com/uc?export=download&id=1T5IGPVoah-MJQUT-NAJ4sUFb3_oHtQ29) **Sample Input File 2** (
Sample Output File 1 (https://drive.google.com/uc?export=download&id=10oST_wyUO7AmflG5W3LXw1RDkP34z4Vi)**Sample Output File 2**

Project Submission and Grading (Please Read)

- Please upload all your `.h` (if any) and `.cpp` files (**not** the ~~entire Microsoft Visual Studio project folder~~) on Canvas.
- Before the project deadline, you can submit your work unlimited times. However, only your latest submission will be graded.
- At least **20**% of your code should be **comments**. All variable, function (if any), and class (if any) names should "make good sense". You should let the grader put **least effort** to understand your code. Grader will **take off points**, even if your program passed all test cases, if he/she has to put extra **unnecessary** effort to understand your code.
- Please **save a backup copy** of all your work in your computer hard drive.
- Your program will be graded (tested) using another valid input file (still named **`data.txt`**) to check whether it can generate the expected (correct) output file (with correct format and correct output values in it). As long as the input file is valid, your program should generate a correct output file. In other words, your program should work for **any** valid input file, **not** just the sample input files provided in the project instructions.
- In this class, you can assume that the input file (input data) is always **valid** and **has correct format**. You do **not** need to deal with ~~invalid input~~ or ~~error handling~~.
- Your work will be graded after the project deadline. All students will receive their project grades at (almost) the same time.