Lab 5: Arrays of Objects, Strings, and Object Interactions

Course Code: CSE 4302
Week: 5
Topics: Arrays of Objects and String Manipulation
Instructors:
Md. Bakhtiar Hasan, Assistant Professor, CSE
Ishmam Tashdeed, Lecturer, CSE
Submission Deadline: 12:00 A.M., December 10, 2025

Upload and Turn In your `.cpp` files to **Lab 5** on **Google Classroom**. If you are unable to finish all the tasks, **submit the tasks you have completed within the given time** (You have the chance to complete the remaining ones individually later on and demonstrate them during the next week's lab evaluation). No late submissions are allowed.
DO NOT forget to rename the files like `XXX_T1.cpp`, `XXX_T2.cpp`, ... (Replace `XXX` with the last three digits of your student ID).
After the submission deadline, you will undergo a one-on-one evaluation by one of the course instructors. The instructor will ask you to show your submissions on Google Classroom. They may ask you to run the code locally and ask you questions about it.

Lab Tasks

1. Meme Museum

You are building a digital archive for "Dank Vault," a museum of internet memes. Each meme has a humor score, age, and viral status. The curator wants to filter memes that are both vintage and highly rated.
Class Specification: Create a class named `Meme`. The class should have private member variables to store the `memeName (String), originYear (int), humorScore (int, 1-100), and isStillViral (bool)`. Include two public constructors: one default and one parameterized.
Implement public getter and setter functions for all variables. Inside the setters, apply the following validation rules:

- Origin year must be between 2000 and 2025. If invalid, set to 2010.
- Humor score must be between 1 and 100. If invalid, set to 50.

Driver Program Requirements: In `main()`, create an array of 40 Meme objects. Write a function `void archiveMemes(Meme collection[], int size)` that lets the user enter data for the first 5 memes. For the rest, assign the name "`Lost Meme`", origin year random between

2005 and 2015, humor score random 1–100, and set `isStillViral` to false. Display all memes from before 2010 with a humor score above 70.

2. Cyber-Secure

In a near-future cyberpunk city, you are coding the firmware for a “Ripperdoc” clinic that installs cybernetic enhancements. The system tracks the power consumption and firmware versions of implants. If an implant is buggy, a technician needs to be able to “patch” it directly without creating a copy of the hardware.

Class Specification: Create a class named `CyberImpant`. The class requires private member variables for `modelName` (String), `bodyPart` (String), `powerConsumption` (in watts, integer), and `firmwareVersion` (double). Include a default constructor (setting version to `1.0`) and a parameterized constructor.

Implement public getter and setter functions for all variables. Inside the setters, apply the following validation rules:

- Body part must be one of “`Arm`”, “`Leg`”, “`Eye`”, or “`Heart`”. If not, set to “`Auxiliary`”.
- Power consumption must be positive. If it is greater than 1000 watts, clamp it to 1000 as safety limit.

Driver Program Requirements: Create a non-member function `void upgradeFirmware(CyberImplant &implant)` that takes an object by reference. This function should increment the firmware version by `0.1` and decrease power consumption by `5` watts. In your `main()` function, create an array of `5` `CyberImplant` objects. Manually set their details. Display their stats, pass each object to `upgradeFirmware`, and then display the new stats to prove the original objects were modified.

3. To Hodl or Not to Hodl

A stock market simulator needs to compare investment portfolios to see which one is performing better. Investors want to compare their portfolio against a friend’s and see which portfolio has the lower risk score.

Class Specification: Create a class named `StockPortfolio`. The class should have private member variables to store the `ownerName` (String), `totalValue` (double), `riskVolatility` (int, 1-10 scale), and `stockCount` (int). Include two public constructors: one default constructor and one parameterized constructor accepting arguments for all fields. Also, implement public getter and setter functions for all variables.

Implement a public member function `StockPortfolio compareSafety(...)`. This function compares the risk score of the calling object against the passed `other` object’s risk score. The risk score is calculated using the following formula:

$$R = \left(\frac{Volatility^{2} \times Total\ Value}{ln(Stock\ Count + 1)}\right)^{0.6} + \left(\frac{Volatity \times \sqrt{Total\ Value}}{(Stock\ Count)^{0.3}}\right) + 0.4\left(\frac{Volatility}{1 + e^{(-Total\ Value/10^{5})}}\right)$$

The function should return the `Portfolio` object that has the lower risk score. If they are equal, return the calling object.

Driver Program Requirements: In your `main()` function, create an array of `5` `StockPortfolio` objects with distinct risk levels, values, and counts. Write a loop (tournament style) where Portfolio `0` is compared with Portfolio `1`, the safer is compared with Portfolio `2`, and so on. Capture the returned object from each comparison in a separate `StockPortfolio` variable named `safer`. Display the details of the safest portfolio found.