**Lab Report**

**Object-Oriented Programming: Basketball Game Simulation**

By: Ishaa Jain, Nipun Chandra, Shivam Kharangate

**Introduction:**

The intention of this lab was to design, build, and refine a Measurement Converter class by learning and applying various core object-oriented programming concepts. The major topic of the class was handling and converting between different units of measurements, showing the importance of such software development as encapsulation, operator overloading, and normalization. Therefore, these concepts not only find a critical place in academic settings but have important use in professional careers in CS and engineering since they provide the base on which to construct solid, scalable, and maintainable code bases.

**Concepts Explained:**

Encapsulation: Displayed through the use of private member variables and public getter/setters, hence asserting that the modification of the class's internal state can only be done through the interfaces defined.

Operator Overloading: Shown by implementing arithmetic and comparison operators naturally working with objects of type MeasurementConverter—adding to the usability of the code.

Normalization: Applied using an encapsulated private method to ensure the integrity of data for the class, maintaining measurement values within bounds specified.

For a grounding in CS and engineering, it is critical to appreciate and be able to employ these because it will make it possible to design and implement software that is efficient and is not prone to errors, and it will be easy to debug and maintain or add more in the future.

**Design Choices - First Version:**

The subsequent were key parts in the first design of the MeasurementConverter class:

Private Member Variables: "littles", "lots", and "heaps" to store different units of measurements.

Public Constructors: Such that instances of this object could be created in different states.

Method Normalization: A private method that must be called following any operation bringing the object in a state not conforming to internal standards.

We do these choices so that it would capture the measurement data completely, with controlled access and modification and at the same time be easy for the class to adapt representing different measurement systems.

**For Task-Based, What Has Been Done:**

Added Arithmetic Operations: Overloaded the addition (+) operators to allow for natural combination between MeasurementConverter objects. This meant incorporating a normalize call within each operation so that the results will indeed conform to the class invariant.

String Representation: The `std::string` class can now be added with a conversion operator to allow easy output of objects of class `MeasurementConverter`. This will be invaluable if wanting human-readable forms of the measurements in both debug and user interaction.

Comparison: Overloading the equality operator == allowed to compare two MeasurementConverter objects directly, which improved the usability of the class within conditional statements and algorithms.

This has been guided by the need for the class to be intuitive in use, support consistency of data across operations, and offer a broad range of functionality to serve general programming situations.

**The major considerations that came into play in doing the design of the class included:**

Data Integrity: In view of the fact that the measurement values should not change in their consistency and validity following whatever operation, normalization plays a very critical role.

Usability: Making the class more intuitive through overloading the operators and writing clear and concise ways to handle the data makes it easier to use and probably integrate in more extended projects. Extensibility: Its general design has been set up in a manner to allow developers to extend the various units of measurement or operations that may apply without major refactoring.

**Conclusion:**

The task of developing the MeasurementConverter class gave me practical experience with key OOP notions, with emphasis on how to make well-structured, efficient, and maintainable software. This project finally displayed encapsulation, operator overloading, and normalization, as best practices to add to what makes the software development process more complex and more flexible; skills which are very important for anyone wishing to pursue a career in the field of computer science or engineering. In conclusion, this assignment has taught very insightful information regarding class design and implementation strategies that would be very helpful for our future academic and professional undertaking in Computer Science and Engineering.