OOPS LAB ASSIGNMENT 08 Date – 10/10/2023

Roll No - 21BCS043

Title: Lessons from Runtime Polymorphism in C++

Performance Optimization

- In simple insertion sort we compared points using norm by invoking the norm() member function every time on comparison that took much more time.
- But we optimized this by declaring a public data member norm that stored the norm of that point and then we applied the sorting algorithm that did not invovle the member function call.
- This led to more than 10 times boost in performance

Notes

Introduction: Runtime polymorphism is a key concept in object-oriented programming that enables dynamic and flexible behavior in C++. This report outlines my recent learnings from exploring the principles and applications of runtime polymorphism.

Virtual Functions:

- At the core of runtime polymorphism is the use of virtual functions. A virtual function is declared in a base class and can be overridden by derived classes, allowing dynamic binding during runtime.

Dynamic Binding:

- Runtime polymorphism allows the correct function to be called at runtime based on the actual type of the object, rather than its declared type. This is achieved through the use of virtual functions and pointers/references to base class objects.

Polymorphic Behavior:

- With runtime polymorphism, a base class pointer or reference can refer to objects of its derived classes, allowing for polymorphic behavior. This enhances code flexibility and extensibility.

Abstract Classes and Interfaces:

- Abstract classes with pure virtual functions are often used to define interfaces. Derived classes then provide concrete implementations for these interfaces, ensuring a common contract for polymorphic usage.

Challenges

Performance Overhead:

- Dynamic binding and virtual function calls may introduce a slight performance overhead compared to static binding. However, the impact is often negligible in many applications.