OOPS LAB ASSIGNMENT 07 Date – 03/10/2023

Roll No - 21BCS043

Title: Lessons from Derived Classes, Abstract Classes, and Virtual Functions

Notes

Introduction: These concepts form the foundation for building hierarchies of related classes and enable polymorphism, but they also present certain challenges.

Derived Classes:

Inheritance:

- Derived classes allow for the creation of new classes that inherit attributes and behaviors from existing (base) classes.
- This promotes code reuse and the establishment of a hierarchy, where common functionalities are shared among related classes.

Access Modifiers in Inheritance:

- Understanding the impact of access modifiers (public, private, protected) in derived classes is crucial. It affects how the members of the base class are inherited and accessed in the derived class.

Abstract Classes:

Pure Virtual Functions:

- Abstract classes cannot be instantiated and often contain pure virtual functions (functions with no implementation). These functions must be implemented by concrete (non-abstract) derived classes.
- Abstract classes serve as interfaces, defining a common set of functions that derived classes must implement.

Polymorphism:

- Abstract classes contribute to polymorphism, allowing different derived classes to be treated uniformly through a common interface. This enhances flexibility in the design of class hierarchies.

Challenges:

A problem occurs in languages with multiple inheritance when a class inherits from two classes that have a common ancestor. It can lead to ambiguity in the inheritance hierarchy and the potential for unpredictable behavior.

Crafting abstract classes with well-defined and meaningful pure virtual functions is a challenge. Striking the right balance between generality and specificity is crucial for creating effective interfaces.

Conclusion

In conclusion, understanding derived classes, abstract classes, and virtual functions is fundamental to effective object-oriented programming in C++. While these concepts provide powerful tools for building flexible and extensible systems, challenges such as the ambiguity problem and effective interface design must be carefully navigated to ensure the robustness of the code.