C++ 2 Week 3

Khaula Molapo

Year 2 Semester 2

1.

Imagine a game system where players inherit special powers from their ancestors. Each player begins as a basic character class with simple abilities like running, jumping, and defending. As they progress, they "inherit" advanced traits from parent classes, such as Warrior, Mage, or Archer. For example, if a player inherits from the Warrior class, they gain combat strength and weapon mastery. If they inherit from the Mage class, they unlock magical spells and mana control. Later, through deeper inheritance, hybrid classes like Battle Mage or Arcane Archer emerge, combining features of both parents. This system ensures variety while reducing repetitive coding, since shared features like health points or stamina bars exist only once in the base class. In real-world programming terms, this scenario reflects inheritance in object-oriented programming (OOP)—a way for child classes to reuse, extend, and customize properties and behaviours from parent classes.

2.

Multiple inheritance is an object-oriented programming concept where a class can inherit attributes and behaviours from more than one parent class. Unlike single inheritance, where a child class inherits from only one base, multiple inheritance allows the child to combine functionalities from several classes at once. For example, in C++, a class FlyingCar might inherit from both Car and Airplane, giving it the ability to both drive and fly. While powerful, multiple inheritance can introduce complexity, especially the diamond problem, where two parent classes inherit from the same ancestor, creating ambiguity. Despite this challenge, multiple inheritance is useful for building flexible and feature-rich systems.

3.

Writing this inheritance program in Visual Studio Code helped me understand the basic structure of C++ inheritance. The Person class acts as the parent, while the Student class inherits its functionality. By creating an object of Student, I was able to call both the parent method and the child method, which clearly demonstrated how inheritance avoids code

duplication. One major advantage I noticed is the simplicity in extending features without rewriting existing code. Using Visual Studio Code also made the process smooth because of the auto-complete suggestions and debugging tools. Overall, this practice reinforced my grasp of class relationships in C++.

4.

This UML diagram represents a simple inheritance hierarchy. The base class, Person, contains common attributes like Name and Age, along with a method to display information. Two child classes, Student and Teacher, inherit from the Person class. Each child class adds its own unique attributes and methods. The Student class introduces StudentID and the ability to show their role, while the Teacher class introduces Subject and a teaching method. The diagram uses arrows pointing from Student and Teacher back to Person, showing the direction of inheritance. This design demonstrates code reuse, since both students and teachers share basic features while extending their specific roles.