**Lab 6: INHERITANCE**

**Objective:**

* To understand the concept of inheritance in C++.
* To learn how to inherit the characters from base class to derived class.
* To demonstrate the syntax and rules of access specifiers and its types.
* To learn about the IS-A relationship and HAS-A relationship.

**Theory:**

**Introduction to inheritance**

Inheritance is a fundamental concept in OOP that facilitates the creation of new classes by acquiring the properties and behaviors of existing classes. It helps to promote code reusability, modularity, and hierarchical classification, which are essential for building scalable and maintainable software systems. In C++, inheritance allows a derived class to inherit the data members and member functions of a base class.

**Access specifier**

C++ provides access specifier to control the visibility and accessibility of the class members. There are 3 types of access specifier which are:

1. Private access specifier

Private members are the variables that can be accessed only inside the class where they are declared. They can’t be accessed directly by ferived classes. This is the strictiest level of access control and is used to hide data and protect it from unwanted changes.

Syntax:

Class Person{

Private:

int age;

public:

void setAge(int a){

age=a;

}

int getAge(){

return age;

}

};

1. Protected access specifier

Protected members are class variables or functions that can be used inside the class itself and also inside any class that inherits from it. But they can’t be directly accessed from outside the class.

Example:

class Parent {

protected:

int id;

};

class Child : public Parent {

public:

void setId(int x) {

id = x;

}

int getId() {

return id;

}};

1. Public access specifier

Public members are class variables or functions that can be accessed from anywhere in the program. These members are available inside the class, in derived classes, and also outside the class through objects. Public members form the interface of the class, allowing interaction with objects of that class.

Example:

class Bike {

public:

string brand;

void display() {

cout << "Brand: " << brand << endl;

}

};

int main() {

Bike myBike;

myBike.brand = "BMW";

member

myBike.display();

return 0;

**IS-A relationship**

The is-a relationship indicates that one class is specialized form of another class. It is established through inheritance. It is implemented using public inheritance. The derived class inherits all the public and protected members of the base class and can add its own members or modify existing ones. Example: a student IS-A Person.

Syntax:

class Base {

};

class Derived : public Base {

};

**HAS-A relationship**

The has-a relationship occurs when a class contains one or more objects of another class as its members. It represents a part-of or ownership connection, indicating that one object is composed of another object. It helps in promoting modular design by dividing systems into smaller parts.

Syntax:

class ClassA {

};

class ClassB {

private:

ClassA obj;

public:

};

**Public inheritance**

Public inheritance is a type of inheritance in C++ where a derived class inherits from a base class using public access specifier. It establishes an “IS-A” relationship between the derived and base class. Public members of the base class remain public, protected members remains protected and private members are inaccessible in the derived class.

**Syntax**

class Base {

};

class Derived : public Base {

};

**Protected inheritance**

Protected inheritance is a type of inheritance in C++ where a derived class inherits from a base class using protected access specifier. It hides the public interface of the base class from the outside world while still making it accessible within the derived class and its further derived classes. Public members of the base class become protected, protected members remains protected and private members are inaccessible in the derived class.

**Syntax**

class Base {

};

class Derived : protected Base {

};

**Private inheritance**

Private inheritance is a type of inheritance in C++ where a derived class inherits from a base class using private access specifier. It doesn’t establish an “IS-A” relationship but implies that the derived class is implemented-in-terms-of the base class. Public members of the base class become private, protected members become private and private members are inaccessible in the derived class.

**Syntax**

class Base {

};

class Derived : private Base {

};

**Member function overriding**

Function overriding is a fundamental concept in object-oriented programming that allows a subclass to provide its own version of a function already defined in its superclass. This is done by using the same function name, return type, and parameters, effectively replacing the base class’s implementation. Overriding plays a key role in runtime polymorphism, enabling the program to determine at run time which version of the function to execute, depending on the actual type of the object involved. This feature allows derived classes to modify or enhance inherited behavior, offering greater flexibility and enabling more specialized functionality while preserving a unified interface. While declaring, the function in the base class must be declared using the keyword “virtual”. “override” keyword can be used in the derived class.

Syntax

class Base {

public:

virtual ReturnType functionName(ParameterList) {

}};

class Derived : public Base {

public:

ReturnType functionName(ParameterList) override {

}};