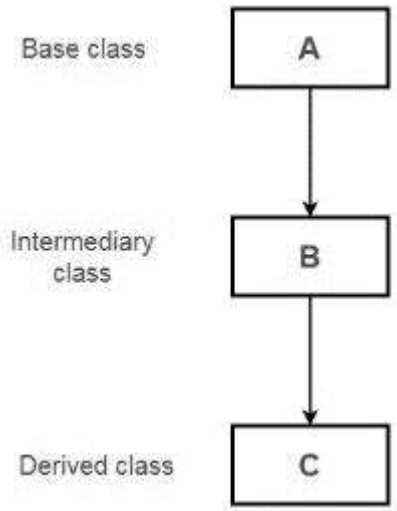
Unit 3:Extending classes using Inheritance **Marks - 16**

Sr.No	Questions	Marks	Year
1.	**List different types of inheritance**	**2**	**S-24**
	Types of inheritance: 1.Single inheritance 2. Multiple inheritance 3. Multilevel inheritance 4. Hierarchical inheritance 5. Hybrid inheritance		
2.	**What is multilevel inheritance? Develop a C++ program for multilevel inheritance.**	**4**	**S-24**
	Multilevel Inheritance: The inheritance in which a class can be derived from another derived class is known as Multilevel Inheritance. Suppose there are three classes A, B, and C. A is the base class. B is the derived class of A. and C is the class that is derived from class B.  **Fig: Multilevel Inheritance**		

	Example:
	#include<iostream>
	using namespace std;
	class electronicDevice
	{
	public:
	electronicDevice()
	{
	cout << "I am an electronic device.\n\n";
	}
	};
	class Computer: public electronicDevice
	{
	public:
	Computer()
	{
	cout << "I am a computer.\n\n";
	}
	};
	class Linux_based : public Computer
	{
	public:
	Linux_based()
	{
	cout << "I run on Linux.\n\n";;
	}
	};
	int main()
	{
	Linux_based obj;
	}		

3.	**Describe the concept of virtual base class with suitable example.**	**4**	**S-24**
	Explain virtual base class with an example.	**6**	**W-23**
	Illustrate the concept of virtual base class with suitable example.	**4**	**S-23**
	Develop a c++ program to implement virtual Base class.	**6**	**W-22**
	Describe the concept of virtual base class with example.	**4**	**S-22**
	Explain virtual base class with suitable example.	**2**	**W-19**
	Describe the concept of virtual base class with suitable example.		

Virtual Base Class:
An ancestor class is declared as virtual base class which is used to avoid duplication of inherited members inside child class due to multiple path of inheritance.

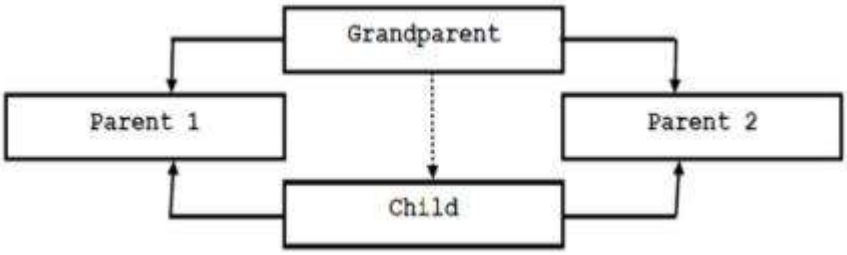


Fig.:Virtual Base Class (Multipath inheritance)

Consider a hybrid inheritance as shown in the above diagram. The child class has two direct base classes, Parent1 and Parent2 which themselves have a common base class as Grandparent. The child inherits the members of Grandparent via two separate paths. All the public and protected members of Grandparent are inherited into Child twice, first via Parent1 and again via Parent2. This leads to duplicate sets of the inherited members of Grandparent inside Child class. The duplication of inherited members can be avoided by making the common base class as virtual base class while declaring the direct or intermediate base classes as shown below.

```
class Grandparent
{
};
class Parent1: virtual public Grandparent
{
};
class Parent2: public virtual Grandparent
{
};
class Child: public Parent1, public Parent2
{
};
```

Example:

```
#include<iostream>
using namespace std;
class student
{
int rno;
public:
void getnumber()
{
cout<<"Enter Roll No:";
cin>>rno;
```

```
}
void putnumber()
{
cout<<"\n\n\t Roll No:"<<rno<<"\n";
}
};
class test: virtual public student
{
public:
int part1,part2;
void getmarks()
{
cout<<"Enter Marks\n";
cout<<"Part1:";
cin>>part1; cout<<"Part2:";
cin>>part2;
}
void putmarks()
{
cout<<"\t Marks Obtained\n";
cout<<"\n\t Part1:"<<part1;
cout<<"\n\tPart2:"<<part2;
}
};
class sports: public virtual student
{
public:
int score;
void getscore()
{
cout<<"Enter Sports Score:";
cin>>score;
}
void putscore()
{
cout<<"\n\t Sports Score is:"<<score;
}
};
class result: public test, public sports
{
int total;
public:
void display()
{
total=part1+part2+score;
putnumber();
putmarks();
```

	putscore();
	cout<<"\n\t Total Score:"<<total;
	}
	};
	int main()
	{
	result obj;
	obj.getnumber();
	obj.getmarks();
	obj.getscore();
	obj.display();
	}		

4.	**Describe a C++ program to declare a class college with name and code.Derive new class a student with member as name.Accept and display details of one students along with college data**	**4**	**S-24**

	#include<iostream>
	using namespace std;
	class college
	{
	char name[10];
	int code;
	public:
	void accept_college()
	{
	cout<<"Enter College Name:";
	cin>>name;
	cout<<"Enter Code:";
	cin>>code;
	}
	void display_college()
	{
	cout<<endl<<"College Name:"<<name;
	cout<<endl<<"College Code:"<<code;
	}
	};
	class student:public college
	{
	char sname[10];
	public:
	void accept_student()
	{
	cout<<"Enter student Name:";
	cin>>sname;
	}
	void display_student()
	{
	cout<<endl<<"Student Name:"<<sname;		

```
}
};
int main()
{
student s;
s.accept_college();
s.accept_student();
s.display_college();
s.display_student();
}
```

5.	**Describe all visibility modes and effects with example.**	**4**	**S-24**
	State different types of visibility mode in inheritance.	**2**	**S-23**
	Describe all visibility modes and effects with example.	**4**	**W-22**
	Describe visibility modes and their effects used in inheritance.	**4**	**S-22**
	State and explain the visibility modes used in inheritance.	**6**	**W-19**
	State and describe visibility modes and its effects used in inheritance.	**4**	**S-19**

Visibility modes:

private

protected

public

Base class visibility	**Derived class visibility**		
	Private	Protected	Public
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Private	Protected	Protected
Public	Private	Protected	Public

Private:

o When a base class is privately inherited by a derived class, „public members" and „protected members" of the base class become „private members" of the derived class.

o Therefore, the public and protected members of the base class can only be accessed by the member functions of derived class but, cannot be accessed by the objects of the derived class.

Syntax:

class derived: private base

{

//Members of derived class;

};

Public:

o When a base class is publicly inherited by a derived class then protected members‟ of base class becomes „protected members‟ and ‟public members‟ of the base class become public members‟ of the derived class.

o Therefore the public members of the base class can be accessed by both the member functions of derived class as well as the objects of the derived class.

Syntax:

class derived: public base

{

//Members of derived class;

};

Protected:

o When a base class is protectedly inherited by a derived class, „public and protected members‟ of the base class become protected members‟ of the derived class.

o Therefore the public and protected members of the base class can be accessed by the member functions of derived class as well as the member functions of immediate derived class of it but they cannot be accessed by the objects of derived class

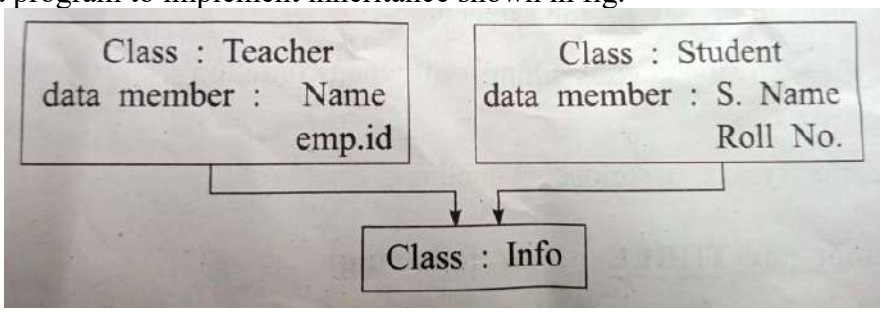
Syntax:

class derived: protected base

{

//Members of derived class;

};

6.	Write a program to implement inheritance shown in fig. 	6	S-24

```
#include <iostream>
using namespace std;
// Base class Teacher
class Teacher {
public:
 string Name;
 int emp_id;

 void setTeacherDetails()
 {
 cout<<"Enter Employee Name:";
 cin>>Name;
 cout<<"Enter Employee ID:";
 cin>>emp_id;
 }

 void displayTeacherDetails()
 cout << "Employee Name: " << Name << endl;
 cout << "Employee ID: " << emp_id << endl;
 }
};
// Base class Student
class Student {
public:
 string S_Name;
 int Roll_No;

 void setStudentDetails()
 {
 cout<<"Enter Student Name:";
 cin>>S_Name;
 cout<<"Enter Roll Number ID:";
 cin>>Roll_No;
 }
 void displayStudentDetails() {
 cout << "Student Name: " << S_Name << endl;
 cout << "Roll Number: " << Roll_No << endl;
 }
```

```
};
// Derived class Info inheriting from both Teacher and Student
class Info : public Teacher, public Student {
public:
 void displayInfo() {
 displayTeacherDetails();
 displayStudentDetails();
 }
};
int main() {
 Info info;
 info.setTeacherDetails();
 info.setStudentDetails();
 cout<<endl;
 info.displayInfo();
}
```

OR

```
#include <iostream>
using namespace std;
// Base class Teacher
class Teacher {
public:
string Name;
 int emp_id;

 void setTeacherDetails(string name, int id) {
 Name = name;
 emp_id = id;
 }

 void displayTeacherDetails() {
 cout << "Teacher Name: " << Name << endl;
 cout << "Employee ID: " << emp_id << endl;
 }
};
// Base class Student
class Student {
public:
 string S_Name;
 int Roll_No;

 void setStudentDetails(string name, int rollNo) {
 S_Name = name;
 Roll_No = rollNo;
 }

 void displayStudentDetails() {
 cout << "Student Name: " << S_Name << endl;
 cout << "Roll Number: " << Roll_No << endl;
 }
```

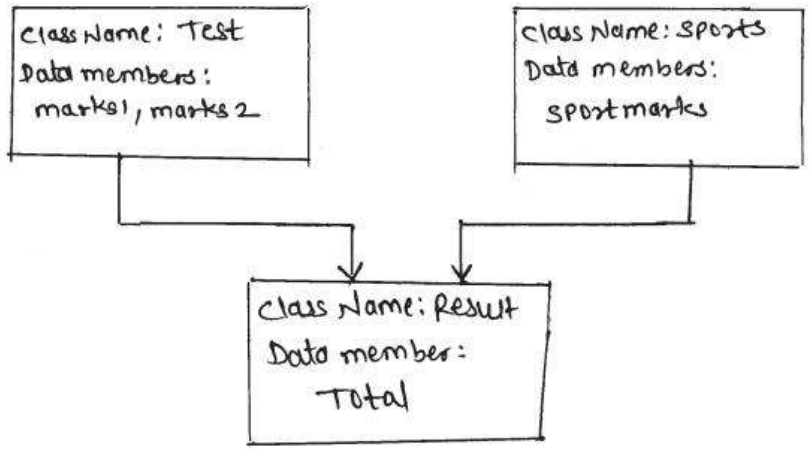
	<pre>};
	// Derived class Info inheriting from both Teacher and Student
	class Info : public Teacher, public Student {
	public:
	 void displayInfo() {
	 displayTeacherDetails();
	 displayStudentDetails();
	 }
	};
	int main() {
	 Info info;
	
	 info.setTeacherDetails("Alice", 123);
	 info.setStudentDetails("Bob", 456);
	
	 info.displayInfo();
	 }</pre>		

7.	**Explain the access specifier in C++.**	**4**	**W-23**
	1. Private 2. Public 3. protected		
8.	**What is inheritance? Give different types of inheritance.**	**4**	**W-23**
	What is inheritance? Give different types of inheritance.	**4**	**W-19**
	Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a new class (derived class or subclass) to inherit the properties and behaviors (methods) of an existing class (base class or parent class). This promotes code reusability and hierarchical relationships between classes. **Types of Inheritance** There are primarily five types of inheritance: **Single Inheritance:** A derived class inherits from only one base class. It's the simplest form of inheritance. Example: A Car class inheriting from a Vehicle class. **Multiple Inheritance:** A derived class inherits from multiple base classes. This type of inheritance can lead to the "diamond problem" (ambiguity when a		

	class inherits from two base classes with the same member). Example: A HybridCar class inheriting from both ElectricCar and PetrolCar classes. **Multilevel Inheritance:** A derived class inherits from a base class, which itself is derived from another base class.It forms a chain of inheritance. Example: SportsCar inheriting from Car, which inherits from Vehicle. **Hierarchical Inheritance:** Multiple derived classes inherit from a single base class.It forms a tree-like structure. Example: Sedan, SUV, and Hatchback inheriting from Car. **Hybrid Inheritance:** A combination of two or more types of inheritance.It's complex and often avoided due to potential ambiguities. Example: A class inheriting from a base class and also from a class that itself uses multiple inheritance.		
9.	**Explain multilevel inheritance with an example**	**4**	**W-23**
	Multilevel inheritance is a type of inheritance where a derived class inherits properties from another derived class, which in turn inherits from a base class. Example: **Vehicle** is the base class with properties like number of wheels, color, etc. **Car** is a derived class of Vehicle, inheriting its properties and adding specific car features like model, engine type, etc. **Sedan** is a derived class of Car, inheriting properties from both Vehicle and Car, and adding specific sedan features like number of doors, boot space, etc. #include <iostream> using namespace std; class Vehicle { public:		

```
    int num_wheels;
    string color;
    void displayVehicleDetails() {
        cout << "Number of wheels: " << num_wheels << endl;
        cout << "Color: " << color << endl;
    }
};
class Car : public Vehicle {
public:
    string model;
    string engine_type;
    void displayCarDetails() {
        cout << "Model: " << model << endl;
        cout << "Engine type: " << engine_type << endl;
    }
};
class Sedan : public Car {
public:
    int num_doors;
    int boot_space;
    void displaySedanDetails() {
        cout << "Number of doors: " << num_doors << endl;
        cout << "Boot space: " << boot_space << endl;
    }
};
int main() {
```

```
    Sedan sedan;
    sedan.num_wheels = 4;
    sedan.color = "Red";
    sedan.model = "Corolla";
    sedan.engine_type = "Petrol";
    sedan.num_doors = 4;
    sedan.boot_space = 500;
    sedan.displayVehicleDetails();
    sedan.displayCarDetails();
    sedan.displaySedanDetails();
    return 0;
}
```

10.	**Write a C++ program to implement multiple inheritance as shown in Figure No. 1. Accept and display data of test marks and sport's marks using object of class 'result'**  **Fig. No. 1**	**6**	**W-23**

```
#include <iostream>
using namespace std;
class Test {
public:
```

```
    int marks1, marks2;
    void getTestMarks() {
        cout << "Enter marks1: ";
        cin >> marks1;
        cout << "Enter marks2: ";
        cin >> marks2;
    }
    void displayTestMarks() {
        cout << "Test marks1: " << marks1 << endl;
        cout << "Test marks2: " << marks2 << endl;
    }
};
class Sports {
public:
    int sportMarks;
    void getSportMarks() {
        cout << "Enter sport marks: ";
        cin >> sportMarks;
    }
    void displaySportMarks() {
        cout << "Sport marks: " << sportMarks << endl;
    }
};
class Result : public Test, public Sports {
public:
    int total;
```

	void calculateTotal() {
	total = marks1 + marks2 + sportMarks;
	}
	void displayResult() {
	displayTestMarks();
	displaySportMarks();
	cout << "Total marks: " << total << endl;
	}
	};
	int main() {
	Result result;
	result.getTestMarks();
	result.getSportMarks();
	result.calculateTotal();
	result.displayResult();
	return 0;
	}		

11.	**Write a program on single inheritance.**	**4**	**S-23**

	#include <iostream>
	using namespace std;
	class Animal {
	public:
	void eat() {
	cout << "Animal eats." << endl;
	}
	};		

	class Dog : public Animal {
	public:
	void bark() {
	cout << "Dog barks." << endl;
	}
	};
	int main() {
	Dog d;
	d.eat(); // Inherited from Animal
	d.bark();
	return 0;
	}		

12.	**Write a program on hybrid inheritance.**	**6**	**S-23**

	#include <iostream>
	using namespace std;
	class A {
	public:
	void displayA() {
	cout << "Class A" << endl;
	}
	};
	class B {
	public:
	void displayB() {
	cout << "Class B" << endl;
	}		

	<pre>};
	class C : public A {
	public:
	void displayC() {
	cout << "Class C" << endl;
	}
	};
	class D : public B, public C {
	public:
	void displayD() {
	cout << "Class D" << endl;
	}
	};
	int main() {
	D obj;
	obj.displayA();
	obj.displayB();
	obj.displayC();
	obj.displayD();
	return 0;
	}</pre>		

13.	**Explain abstract class with suitable example.**	**6**	**S-23**
	An abstract class is a class that cannot be instantiated directly. It serves as a blueprint for other classes. Abstract classes contain at least one abstract method, which is a method declared without an implementation. **characteristics of abstract classes**: Cannot be instantiated.		

Can contain both abstract and concrete methods.

Used as a base class for other classes.

Provides a common interface for derived classes.

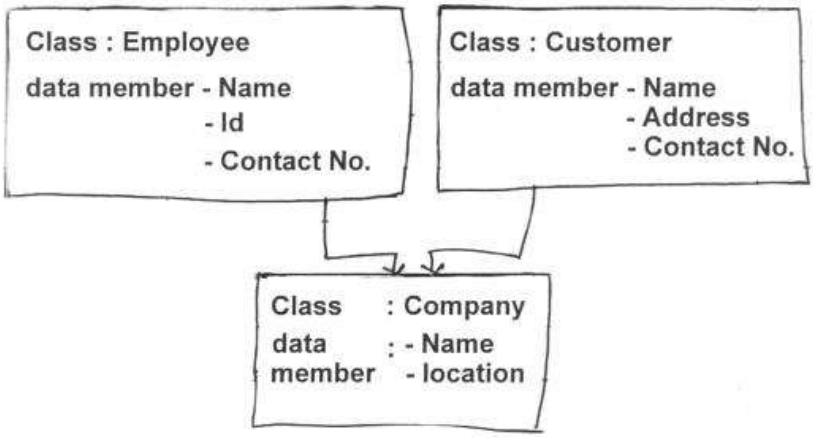
```
#include <iostream>
using namespace std;
class Shape {
public:
    virtual double getArea() = 0; // Pure virtual function
};
class Circle : public Shape {
public:
    double radius;
    Circle(double r) : radius(r) {}
    double getArea() override {
        return 3.14159 * radius * radius;
    }
};
class Rectangle : public Shape {
public:
     double width, height;
     Rectangle(double w, double h) : width(w), height(h) {}
    double getArea() override {
        return width * height;
    }
};
int main() {
```

```
    // Shape s; // Error: Cannot create an object of an abstract class

    Circle c(5);

    Rectangle r(4, 3);

    cout << "Circle area: " << c.getArea() << endl;

    cout << "Rectangle area: " << r.getArea() << endl;

    return 0;

}
```

14.	**Develop a c++ program for multilevel inheritance.**	**4**	**W-22**

```
#include <iostream>
using namespace std;
class Grandfather {
public:
    void displayGrandfather() {
        cout << "Grandfather" << endl;
    }
};
class Father : public Grandfather {
public:
    void displayFather() {
        cout << "Father" << endl;
    }
};
class Son : public Father {
public:
    void displaySon() {
        cout << "Son" << endl;
    }
};
int main() {
    Son son;
    son.displayGrandfather();
    son.displayFather();
    son.displaySon();
    return 0;
}
```

15.	**Develop a c++ program to implement inheritance shown in following fig.**	**4**	**W-22**




```
#include <iostream>
#include <string>
using namespace std;
class Company {
public:
   string name;
   string location;
   void setCompanyDetails(string name, string location) {
      this->name = name;
      this->location = location;
   }
   void displayCompanyDetails() {
      cout << "Company Name: " << name << endl;
      cout << "Company Location: " << location << endl;
   }
};
class Employee : public Company {
public:
   string name;
   int id;
   long contactNo;
   void setEmployeeDetails(string name, int id, long contactNo) {
      this->name = name;
      this->id = id;
      this->contactNo = contactNo;
   }
   void displayEmployeeDetails() {
      cout << "Employee Name: " << name << endl;
      cout << "Employee ID: " << id << endl;
      cout << "Employee Contact No.: " << contactNo << endl;
   }
   void displayAllDetails() {
      displayCompanyDetails();
```

```
        displayEmployeeDetails();
    }
};
class Customer : public Company {
public:
    string name;
    string address;
    long contactNo;
    void setCustomerDetails(string name, string address, long contactNo) {
        this->name = name;
        this->address = address;
        this->contactNo = contactNo;
    }
    void displayCustomerDetails() {
        cout << "Customer Name: " << name << endl;
        cout << "Customer Address: " << address << endl;
        cout << "Customer Contact No.: " << contactNo << endl;
    }
    void displayAllDetails() {
        displayCompanyDetails();
        displayCustomerDetails();
    }
};
int main() {
    Employee employee;
    Customer customer;
    // Set company details (common to both)
    employee.setCompanyDetails("XYZ Corporation", "Mumbai");
    customer.setCompanyDetails("XYZ Corporation", "Mumbai");
    // Set employee details
    employee.setEmployeeDetails("John Doe", 12345, 9876543210);
    // Set customer details
    customer.setCustomerDetails("Jane Smith", "Pune", 9876543211);
    cout << "\nEmployee Details:\n";
    employee.displayAllDetails();
    cout << "\nCustomer Details:\n";
    customer.displayAllDetails();
    return 0;
}
```

16.	**Write a C++ program to declare a class college with name and code. Derive a new class as student with members as name. Accept and display details of one student along with college data.**	**4**	**S-22**

```
#include <iostream>
```

```
#include <string>
using namespace std;
class College {
protected:
    string name;
    string code;
public:
    void setCollegeDetails(string name, string code) {
        this->name = name;
        this->code = code;
    }
    void displayCollegeDetails() {
        cout << "College Name: " << name << endl;
        cout << "College Code: " << code << endl;
    }
};
class Student : public College {
private:
    string name;
public:
    void setStudentDetails(string name) {
        this->name = name;
    }

    void displayStudentDetails() {
        cout << "Student Name: " << name << endl;
    }
};
int main() {
    Student student;
    string collegeName, collegeCode, studentName;
    cout << "Enter College Name: ";
    getline(cin, collegeName);
    cout << "Enter College Code: ";
    getline(cin, collegeCode);
    cout << "Enter Student Name: ";
    getline(cin, studentName);
    student.setCollegeDetails(collegeName, collegeCode);
    student.setStudentDetails(studentName);
    cout << endl << "College Details:" << endl;
    student.displayCollegeDetails();
    cout << endl << "Student Details:" << endl;
    student.displayStudentDetails();
```

```
    return 0;
}
```

17. **Write a C++ program to implement following inheritance: Refer Fig. No. 1.** **6** **S-22**

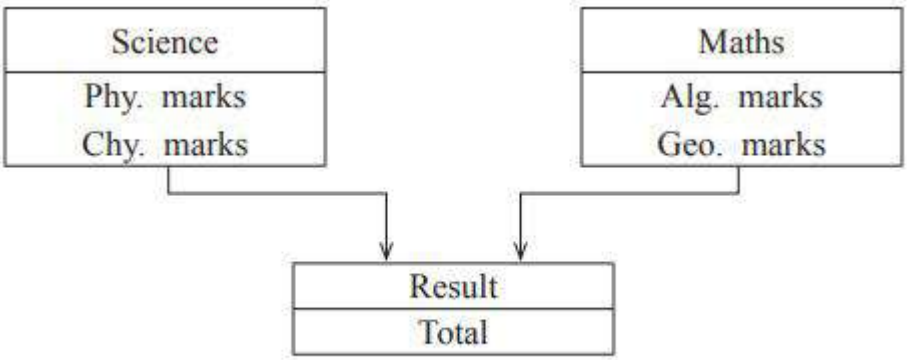


Fig. No. 1.

Accept and display total of one object of result.

```
#include <iostream>

using namespace std;

class Science {

protected:

    int phy_marks, chy_marks;

public:

    void setScienceMarks(int p, int c) {

        phy_marks = p;

        chy_marks = c;

    }

};

class Maths {

protected:

    int alg_marks, geo_marks;

public:

    void setMathsMarks(int a, int g) {
```

```
        alg_marks = a;
        geo_marks = g;
    }
};
class Result : public Science, public Maths {
private:
    int total;
public:
    void calculateTotal() {
        total = phy_marks + chy_marks + alg_marks + geo_marks;
    }
    void displayTotal() {
        cout << "Total Marks: " << total << endl;
    }
};
int main() {
    Result result;
    int phy, chy, alg, geo;
    cout << "Enter Physics marks: ";
    cin >> phy;
    cout << "Enter Chemistry marks: ";
    cin >> chy;
    cout << "Enter Algebra marks: ";
    cin >> alg;
    cout << "Enter Geometry marks: ";
    cin >> geo;
```

```
    result.setScienceMarks(phy, chy);
    result.setMathsMarks(alg, geo);
    result.calculateTotal();
    result.displayTotal();
    return 0;
}
```

18. **Write a program to implement inheritance as shown in figure No. 2. Assume suitable member function** — **6** — **S-22**

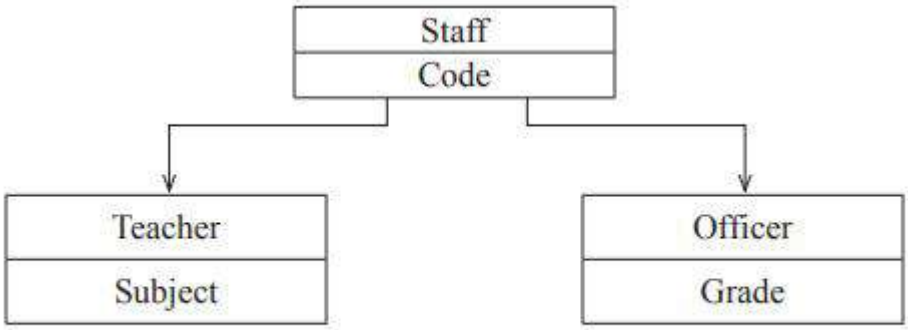


Fig. No. 2.

Accept and display data of one Teacher and one Officer.

```
#include <iostream>
#include <string>
using namespace std;
class Staff {
protected:
    string code;
public:
    void setCode(const string& c) {
        code = c;
    }
    void displayCode() const {
```

```
        cout << "Code: " << code << endl;
    }
};
class Teacher : public Staff {
protected:
    string subject;
public:
    void setSubject(const string& s) {
        subject = s;
    }
    void displaySubject() const {
        cout << "Subject: " << subject << endl;
    }
};
class Officer : public Staff {
protected:
    string grade;
public:
    void setGrade(const string& g) {
        grade = g;
    }
    void displayGrade() const {
        cout << "Grade: " << grade << endl;
    }
};
int main() {
```

```
Teacher teacher;
Officer officer;
// Accept data for Teacher
string teacherCode, teacherSubject;
cout << "Enter Teacher Code: ";
cin >> teacherCode;
cout << "Enter Teacher Subject: ";
cin >> teacherSubject;
teacher.setCode(teacherCode);
teacher.setSubject(teacherSubject);
// Accept data for Officer
string officerCode, officerGrade;
cout << "Enter Officer Code: ";
cin >> officerCode;
cout << "Enter Officer Grade: ";
cin >> officerGrade;
officer.setCode(officerCode);
officer.setGrade(officerGrade);
// Display data for Teacher
cout << "\nTeacher Details:\n";
teacher.displayCode();
teacher.displaySubject();
// Display data for Officer
cout << "\nOfficer Details:\n";
officer.displayCode();
officer.displayGrade();
```

```
    return 0;

}
```

19.	**What is multilevel inheritance? Draw the diagram to show multilevel inheritance. using classes with data member and member function.**	**2**	**W-19**

Multilevel inheritance is a type of inheritance in which a derived class inherits properties from a base class, and another derived class inherits from the first derived class. This creates a chain of inheritance.

```
Base Class (Grandparent)
    |
    v
Derived Class 1 (Parent)
    |
    v
Derived Class 2 (Child)
```

```
#include <iostream>

using namespace std;

class Grandfather {

protected:

    string name;

public:

    void setGrandfatherName(string n) {

        name = n;

    }

    void displayGrandfatherName() {

        cout << "Grandfather's name: " << name << endl;

    }

};

class Father : public Grandfather {

protected:
```

	string occupation;
	public:
	void setFatherOccupation(string o) {
	occupation = o;
	}
	void displayFatherOccupation() {
	cout << "Father's occupation: " << occupation << endl;
	}
	};
	class Son : public Father {
	public:
	void displayInfo() {
	displayGrandfatherName();
	displayFatherOccupation();
	cout << "Son is studying." << endl;
	}
	};
	int main() {
	Son son;
	son.setGrandfatherName("John Doe");
	son.setFatherOccupation("Engineer");
	son.displayInfo();
	return 0;
	}		

20.	**Write a program to implement single inheritance from the following Refer Figure No. 1.**	**4**	**W-19**

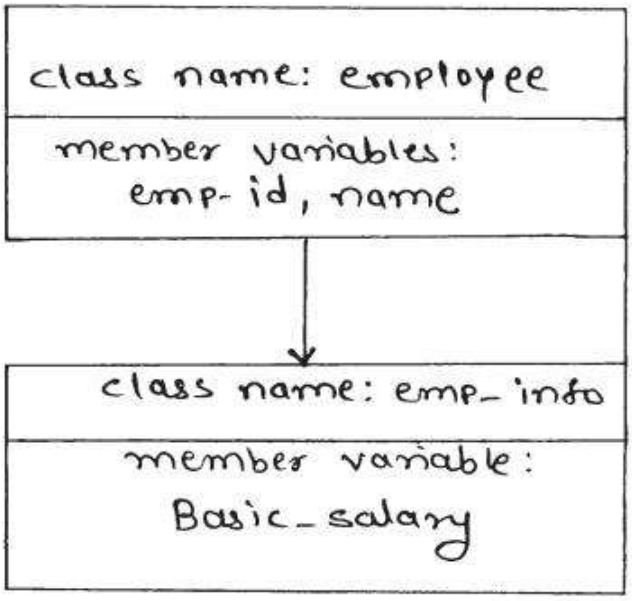


Fig. No. 1

```
#include <iostream>
#include <string>
using namespace std;
class Employee {
public:
    int emp_id;
    string name;
    void setEmployeeDetails(int id, const string& n) {
        emp_id = id;
        name = n;
    }
    void displayEmployeeDetails() {
        cout << "Employee ID: " << emp_id << endl;
        cout << "Employee Name: " << name << endl;
    }
};
```

	class emp_info : public Employee {
	public:
	float basic_salary;
	void setBasicSalary(float salary) {
	basic_salary = salary;
	}
	void displayEmployeeInfo() {
	displayEmployeeDetails(); // Inherit display from base class
	cout << "Basic Salary: " << basic_salary << endl;
	}
	};
	int main() {
	emp_info employee;
	employee.setEmployeeDetails(12345, "John Doe");
	employee.setBasicSalary(50000.0);
	employee.displayEmployeeInfo();
	return 0;
	}		

21.	**Write a program to implement the following hierarchy using suitable member functions. Refer Figure No. 2.**	**6**	**W-19**

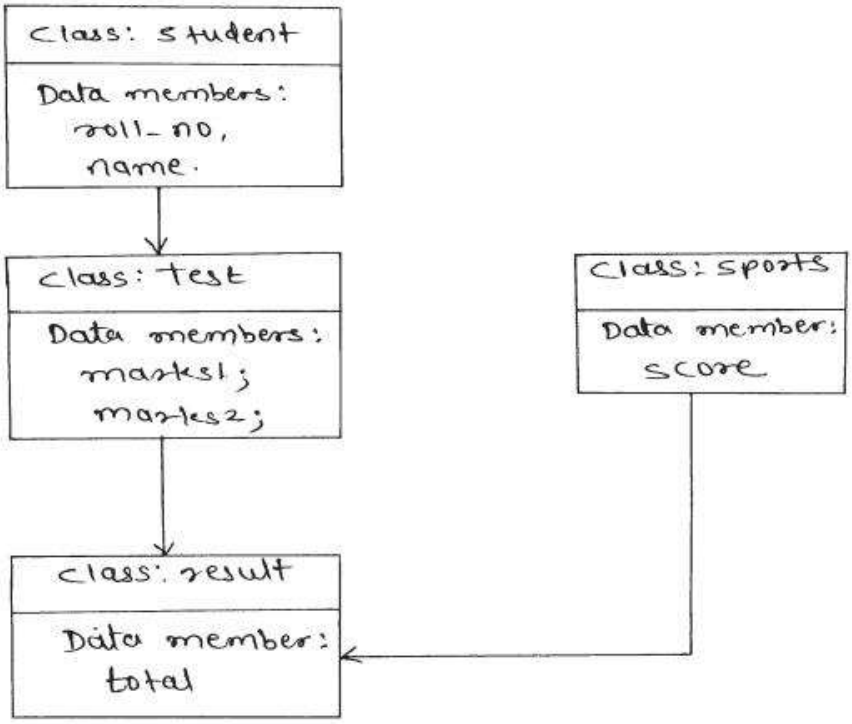


Fig. No. 2

```
#include <iostream>

using namespace std;

class Student {

public:

    int roll_no;

    string name;

    void setStudentDetails(int r, string n) {

        roll_no = r;

        name = n;

    }

    void displayStudentDetails() {

        cout << "Roll No: " << roll_no << endl;

        cout << "Name: " << name << endl;

    }
```

```
};
class Test : public Student {
public:
    int marks1, marks2;
    void setTestMarks(int m1, int m2) {
        marks1 = m1;
        marks2 = m2;
    }
    void displayTestMarks() {
        cout << "Marks 1: " << marks1 << endl;
        cout << "Marks 2: " << marks2 << endl;
    }
};
class Sports : public Student {
public:
    int score;
    void setSportsScore(int s) {
        score = s;
    }
    void displaySportsScore() {
        cout << "Sports Score: " << score << endl;
    }
};
class Result : public Test {
public:
    int total;
```

```
    void calculateTotal() {
        total = marks1 + marks2;
    }
    void displayResult() {
        displayStudentDetails();
        displayTestMarks();
        cout << "Total Marks: " << total << endl;
    }
};
int main() {
    Result result;
    result.setStudentDetails(123, "Alice");
    result.setTestMarks(85, 90);
    result.calculateTotal();
    result.displayResult();
    return 0;
}
```

22.	**Describe derived class with example.**	**2**	**S-19**

A derived class (also known as a subclass or child class) is a class that inherits properties and methods from another class called the base class (or parent class or superclass). It's a fundamental concept in object-oriented programming that promotes code reusability and hierarchical relationships between classes.

```
#include <iostream>
using namespace std;
class Shape {
protected:
    string color;
public:
```

```
    void setColor(string c) {
        color = c;
    }
    string getColor() {
        return color;
    }
};
class Circle : public Shape {
private:
    double radius;
public:
    void setRadius(double r) {
        radius = r;
    }
    double getArea() {
        return 3.14159 * radius * radius;
    }
};
class Rectangle : public Shape {
private:
    double width, height;
public:
    void setDimensions(double w, double h) {
        width = w;
        height = h;
    }
```

	double getArea() {
	return width * height;
	}
	};
	int main() {
	Circle circle;
	circle.setColor("Red");
	circle.setRadius(5);
	Rectangle rectangle;
	rectangle.setColor("Blue");
	rectangle.setDimensions(4, 6);
	cout << "Circle color: " << circle.getColor() << ", Area: " << circle.getArea() << endl;
	cout << "Rectangle color: " << rectangle.getColor() << ", Area: " << rectangle.getArea() << endl;
	return 0;
	}		

23.	**Write a C++ program to declare a class COLLEGE with members as college code. Derive a new class as STUDENT with members as studid. Accept and display details of student along with college for one object of student.**	**4**	**S-19**

	#include <iostream>
	#include <string>
	using namespace std;
	class COLLEGE {
	protected:
	string college_code;
	public:
	void setCollegeCode(string code) {		

```
        college_code = code;
    }
    void displayCollegeCode() {
        cout << "College Code: " << college_code << endl;
    }
};
class STUDENT : public COLLEGE {
private:
    int studid;
public:
    void setStudentId(int id) {
        studid = id;
    }
    void displayStudentId() {
        cout << "Student ID: " << studid << endl;
    }
    void displayDetails() {
        displayCollegeCode();
        displayStudentId();
    }
};
int main() {
    STUDENT student;
    string college_code;
    int studid;
    cout << "Enter College Code: ";
```

```
    cin >> college_code;

    cout << "Enter Student ID: ";

    cin >> studid;

    student.setCollegeCode(college_code);

    student.setStudentId(studid);

    cout << "\nStudent Details:\n";

    student.displayDetails();

    return 0;

}
```

24.	**Describe with examples, passing parameters to base class constructor and derived class constructor by creating object of derived class.**	**4**	**S-19**

Passing Parameters to Base and Derived Class Constructors

When creating an object of a derived class, the constructor of the base class is called before the constructor of the derived class. This allows you to pass parameters to both constructors to initialize members of both classes.

Passing Parameters to Base Class Constructor

To pass parameters to the base class constructor, you use the member initialization list in the derived class constructor.

```
#include <iostream>

using namespace std;

class Base {

public:

    int x;

    Base(int val) : x(val) {

        cout << "Base constructor called with value " << x << endl;

    }

};

class Derived : public Base {
```

```
public:
    int y;

    Derived(int val1, int val2) : Base(val1), y(val2) {
        cout << "Derived constructor called with value " << y << endl;
    }
};
int main() {
    Derived obj(10, 20);
    return 0;
}
```

25. **Write a program to implement multiple inheritance as shown in following Figure No. 1:** **4** **S-19**

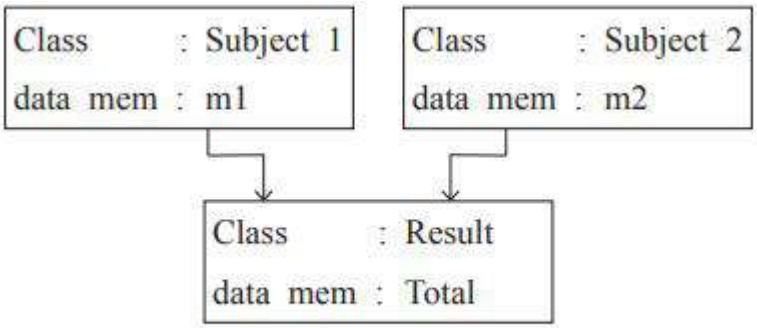


Fig. No. 1

Accept and display data for one object of class result.

```
#include <iostream>
using namespace std;
class Subject1 {
public:
    int m1;
    void setM1(int m) {
        m1 = m;
```

```
    }
    void displayM1() {
        cout << "Subject 1 marks: " << m1 << endl;
    }
};
class Subject2 {
public:
    int m2;
    void setM2(int m) {
        m2 = m;
    }
    void displayM2() {
        cout << "Subject 2 marks: " << m2 << endl;
    }
};
class Result : public Subject1, public Subject2 {
public:
    int total;
    void calculateTotal() {
        total = m1 + m2;
    }
    void displayResult() {
        displayM1();
        displayM2();
        cout << "Total marks: " << total << endl;
    }
```

```
};
int main() {
    Result result;
    int m1, m2;
    cout << "Enter marks for Subject 1: ";
    cin >> m1;
    cout << "Enter marks for Subject 2: ";
    cin >> m2;
    result.setM1(m1);
    result.setM2(m2);
    result.calculateTotal();
    result.displayResult();
    return 0;
}
```

26. **Write a C++ program to implement following inheritance. Refer Figure No. 2.** **6** **S-19**

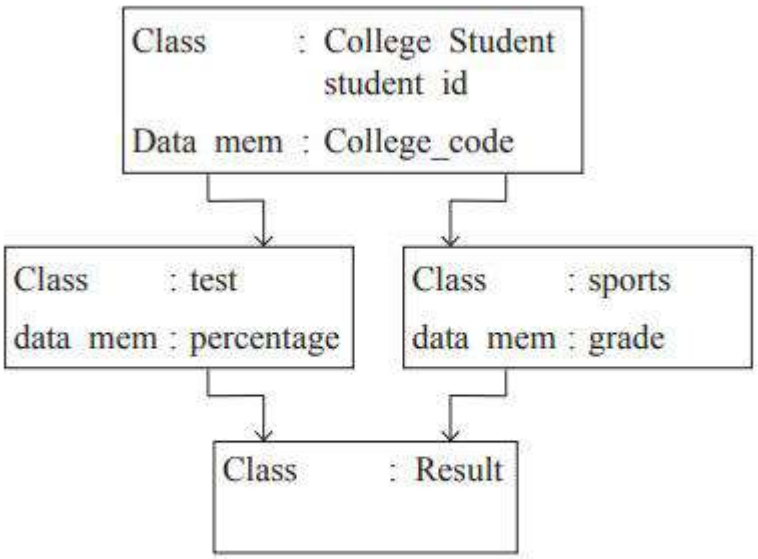


Fig. No. 2

Accept and display data for one object of class result (Hint : use virtual base

	class).		

	# include <iostream.h>
	#include<conio.h>
	class College_Student
	{
	int student_id;
	char College_code[5];
	public:
	void read_collegeStud_Data()
	{
	cout<<"Enter college code and student id\n";
	cin>>college_code>>student_id;
	}
	void display_collegeStud_Data()
	{
	cout<<"\ncollege code\tstudent id\n";
	cout<<college_code<<"\t"<<student_id<<"\n";
	}
	};
	class test: virtual public College_Student
	{
	float percentage;
	public:
	void read_test()
	{
	cout<<"\n Enter test percentage\n";		

```
cin>> percentage;
}
void display_test()
{
cout<<"\n test percentage:"<<percentage;
}
};
class sports: virtual public College_Student
{
char grade[5];
public:
void read_sportsData()
{
cout<<"\n Enter sport grade\n";
cin>> grade;
}
void display_sportsData()
{
Cout<<"\n sport grade:"<<grade;
}
};
class result: public test, public sports
{
public:
void read_result()
{
```

	read_collegeStud_Data() ;
	read_test()
	read_sportsData();
	}
	void display_result()
	{
	display_collegeStud_Data() ;
	display_test()
	display_sportsData();
	}
	};
	void main()
	{
	result r;
	clrscr();
	r.read_result();
	r.display_result();
	}		

27.	**Describe use of protected access specifier used in the class.**	**2**	**W-18**
	The protected access specifier in object-oriented programming languages like C++, Java, and C# provides a level of access control between a class and its derived classes.		
28.	**Write syntax to define a derived class.**	**2**	**W-18**

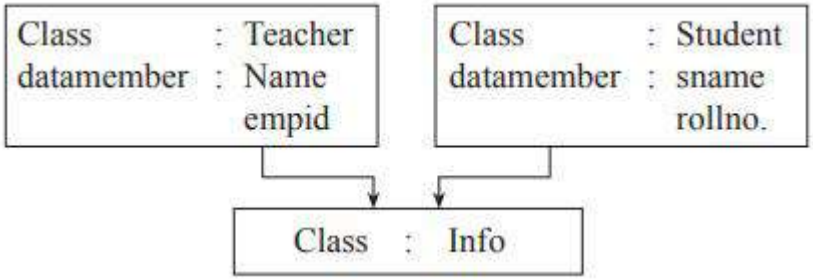
	class DerivedClassName : visibility_mode BaseClassName {
	// Members of the derived class
	};		

29.	**Write a C++ program to declare a class 'College' with data members as name and college code. Derive a new class 'student' from the class college with data**	**4**	**W-18**

	members as sname and roll no. Accept and display details of one student with college data.		

	#include <iostream>
	#include <string>
	using namespace std;
	class College {
	public:
	  string name;
	  string code;
	  void setCollegeDetails(string n, string c) {
	  name = n;
	  code = c;
	  }
	  void displayCollegeDetails() {
	  cout << "College Name: " << name << endl;
	  cout << "College Code: " << code << endl;
	  }
	};
	class Student : public College {
	public:
	  string sname;
	  int roll_no;
	  void setStudentDetails(string sn, int rno) {
	  sname = sn;
	  roll_no = rno;
	  }
	  void displayStudentDetails() {		

```
        cout << "Student Name: " << sname << endl;
        cout << "Roll No: " << roll_no << endl;
    }
    void displayAllDetails() {
        displayCollegeDetails();
        displayStudentDetails();
    }
};
int main() {
    Student student;
    string college_name, college_code, student_name;
    int roll_no;
    cout << "Enter College Name: ";
    getline(cin, college_name);
    cout << "Enter College Code: ";
    cin >> college_code;
    cout << "Enter Student Name: ";
    getline(cin, student_name);
    cout << "Enter Roll No: ";
    cin >> roll_no;
    student.setCollegeDetails(college_name, college_code);
    student.setStudentDetails(student_name, roll_no);
    cout << "\nCollege and Student Details:\n";
    student.displayAllDetails();
    return 0;
}
```

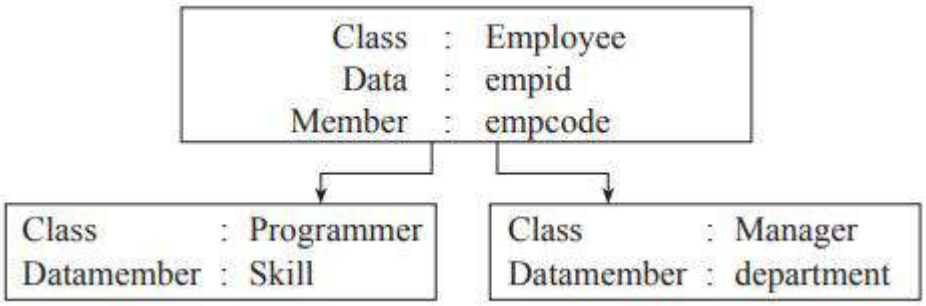
30.	**Write a C++ program to implement inheritance shown in following figure:**  **Accept and display data of one teacher and one student using object of class 'Info'.**	**4**	**W-18**

	#include <iostream>
	#include <string>
	using namespace std;
	class Info {
	public:
	  string name;
	  void setName(string n) {
	  name = n;
	  }
	  void displayInfo() {
	  cout << "Name: " << name << endl;
	  }
	};
	class Teacher : public Info {
	public:
	  int empid;
	  void setEmpId(int id) {
	  empid = id;
	  }		

```
    void displayTeacherDetails() {
        displayInfo();
        cout << "Emp ID: " << empid << endl;
    }
};
class Student : public Info {
public:
    int rollno;
    void setRollNo(int rno) {
        rollno = rno;
    }
    void displayStudentDetails() {
        displayInfo();
        cout << "Roll No: " << rollno << endl;
    }
};
int main() {
    Teacher teacher;
     Student student;
    // Teacher details
    teacher.setName("Mr. Smith");
    teacher.setEmpId(12345);
    // Student details
    student.setName("Alice Johnson");
    student.setRollNo(23456);
    // Display details
```

```
    cout << "Teacher Details:\n";
    teacher.displayTeacherDetails();
    cout << "\nStudent Details:\n";
    student.displayStudentDetails();
    return 0;
}
```

31. **Write a C++ program to implement following inheritance.** **6** **W-18**



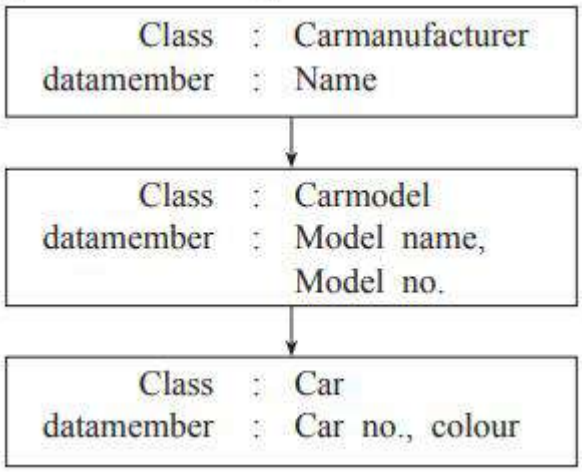
Accept and display data for one programmer and one manager. Make display function virtual.

```
#include <iostream>
#include <string>
using namespace std;
class Employee {
protected:
    int empid;
    string empcode;
public:
    void setEmpid(int id) {
        empid = id;
    }
    void setEmpcode(string code) {
```

```
        empcode = code;
    }
    virtual void display() = 0; // Pure virtual function
};
class Programmer : public Employee {
private:
    string skill;
public:
    void setSkill(string s) {
        skill = s;
    }
    void display() override {
        cout << "Programmer Details:" << endl;
        cout << "Emp ID: " << empid << endl;
        cout << "Emp Code: " << empcode << endl;
        cout << "Skill: " << skill << endl;
    }
};
class Manager : public Employee {
private:
    string department;
public:
    void setDepartment(string dept) {
        department = dept;
    }
    void display() override {
```

```
        cout << "Manager Details:" << endl;
        cout << "Emp ID: " << empid << endl;
        cout << "Emp Code: " << empcode << endl;
        cout << "Department: " << department << endl;
    }
};
int main() {
    Programmer programmer;
    Manager manager;
    programmer.setEmpid(101);
    programmer.setEmpcode("P001");
    programmer.setSkill("C++");
    manager.setEmpid(201);
    manager.setEmpcode("M001");
    manager.setDepartment("HR");
    programmer.display();
    cout << endl;
    manager.display();
    return 0;
}
```

32.	**Write C++ program for following multilevel inheritance.**	6	W-18



Accept and display data for one car with all details.

```
#include <iostream>
#include <string>
using namespace std;
class CarManufacturer {
public:
    string name;
    void setManufacturerName(string n) {
        name = n;
    }
    void displayManufacturerName() {
        cout << "Manufacturer Name: " << name << endl;
    }
};
class CarModel : public CarManufacturer {
public:
    string modelName;
    int modelNo;
    void setModelDetails(string mn, int mno) {
```

```
        modelName = mn;
        modelNo = mno;
    }
    void displayModelDetails() {
        cout << "Model Name: " << modelName << endl;
        cout << "Model No: " << modelNo << endl;
    }
};
class Car : public CarModel {
public:
    int carNo;
    string color;
    void setCarDetails(int cn, string c) {
        carNo = cn;
        color = c;
    }
    void displayCarDetails() {
        cout << "Car No: " << carNo << endl;
        cout << "Color: " << color << endl;
    }
    void displayAllDetails() {
        displayManufacturerName();
        displayModelDetails();
        displayCarDetails();
    }
};
```

```
int main() {
    Car car;
    string manufacturerName, modelName;
    int modelNo, carNo;
    cout << "Enter Manufacturer Name: ";
    getline(cin, manufacturerName);
    cout << "Enter Model Name: ";
    getline(cin, modelName);
    cout << "Enter Model No: ";
    cin >> modelNo;
    cout << "Enter Car No: ";
    cin >> carNo;
    cout << "Enter Car Color: ";
    cin.ignore(); // Ignore newline character
    getline(cin, car.color);
    car.setManufacturerName(manufacturerName);
    car.setModelDetails(modelName, modelNo);
    car.setCarDetails(carNo, car.color);
    cout << "\nCar Details:\n";
    car.displayAllDetails();
    return 0;
}
```