Unit-3

Extending classes using Inheritance

sub:- object oriented programming. (3133304)

* Introduction :-

Inheritance in OOP is a core mechanism that allows a new class to adopt the properties and behaviour of an existing class.

↳ It promotes feature of code reusability.

↳ It is used to establish 'IS-a' relationship.

* Some terms used in inheritance :-

i) Parent class :-

↳ It is also known as Base class or superclass.

↳ It is existing class whose features and behaviour are inherited.

ii) child class :-

↳ It is also known as subclass or Derived class.

↳ It is a new class that inherits from parent class. It can use parent's code and its own too.

iii) Resuability :-

It is ability to use existing code without rewriting it, reducing duplication and development time.

* Defining a derived class :-

A derived class is created from an existing class. It allows the derived class to reuse data members and behaviours of the base class while adding its own unique features.

Syntax:-

In c++, inheritance is defined using a colon (:) followed by access specifier (e.g. public, protected, private)

```
class derived : access modifier Base class.
{
  ---
  ---
  ---
};
```

Where, Derived is name of derived class
Base is name of Base class
Access modifier may be public, private OR protected.

e.g.
```
class Vehicle {          // Base class
public:
    void start() {
    --
    }
};
class car : public Vehicle {
public:
    void Engine()
    {
    --
    }.
};
```

* Visibility modes & its effects:-

Visibility modes are also known as access specifiers).
These modes define the scope & accessibility of class's members (data & functions).

There are three main type of visibility modes (access specifier).

① public — Access from anywhere.
② Protected — Accessible within the class & its derived classes.
③ Private. — Accessible only within the base class itself.

① Public :-

If we derive a subclass from a public base class. Then the public member of the base class will become public in derived class, and protected members of base class will become protected in derived class.

② Protected :-

If we derive a subclass from a protected base class, then both public member and protected members of the base class will become protected in the derived class.

③ Private :-

If we derive a subclass from a private base class, then both public member and protected members of base class will become private in derived class.

summary of visibility modes

Base class visibility	Derived class visibility.		
	public	private	protected.
Public	public	private	protected.
protected	protected	private	protected.
private	Not inherited	Not inherited	Not inherited.

* Types of inheritance in OOP:-

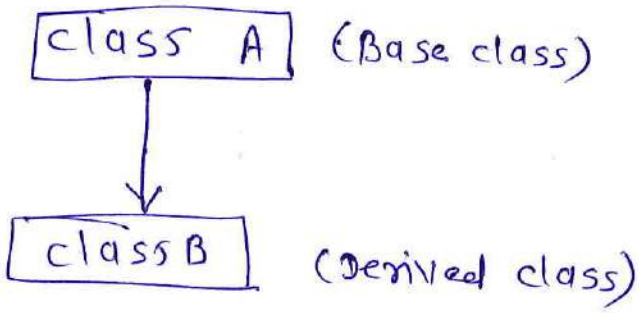
In object oriented programming, there are five primary types of inheritance that define how child class can reuse and extend the behaviour of a parent class.

1. single level Inheritance
2. Multilevel Inheritance.
3. Multiple Inheritance.
4. Hierarchical Inheritance.
5. Hybrid Inheritance.

1. single level Inheritance:-

It is most basic type of inheritance in object oriented programming.

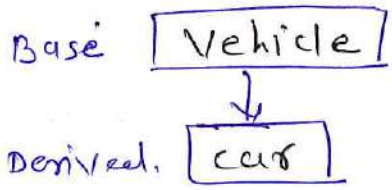
It occurs when exactly one child (derived) class inherits from exactly one parent (base) class.



Structure :→ A → B.

↳ This type of inheritance is supported by all object oriented languages.

e.g.

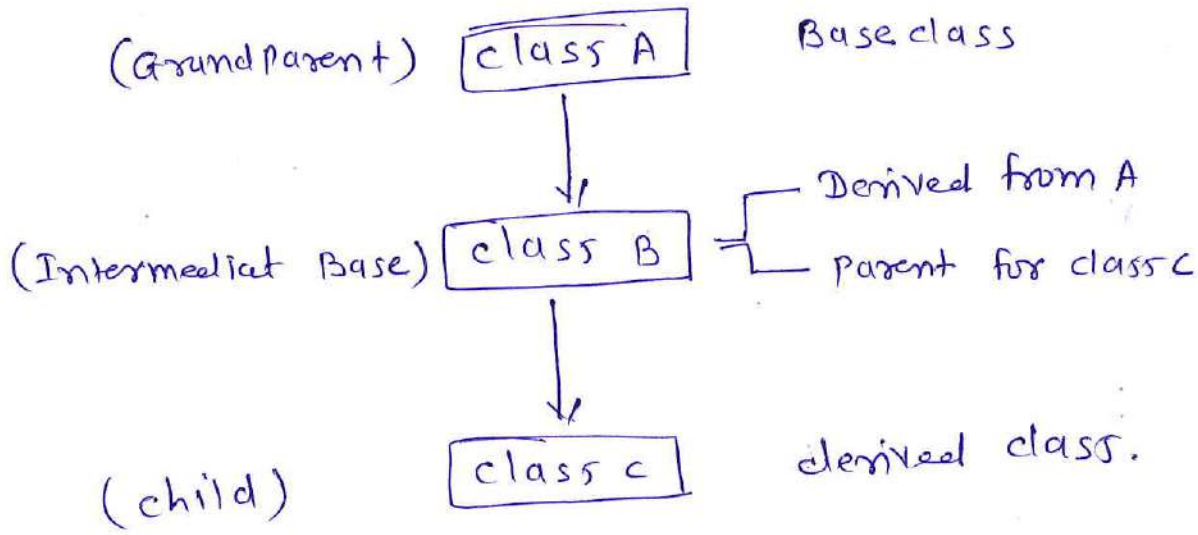



```
#include <iostream.h>
using namespace std;

class Vehicle {          // Base class.
 public:
    void startEngine() {
      cout << " Engine started" << endl;
    }
};

class car : public Vehicle {    // Derived class.
  public :
     void sound() {
      cout<< " Beep! Beep!" << endl;
    }
};
int main()
{
  car mycar;              // object of car class.
  mycar. startEngine();   // calling base class method
  mycar. sound();         // calling own method.
  return 0;
}
```

(2) Multilevel Inheritance:-

Multilevel inheritance is another core concept of object oriented programming.

↳ Here, derived class inherits from another derived class, forming chain of relationship.



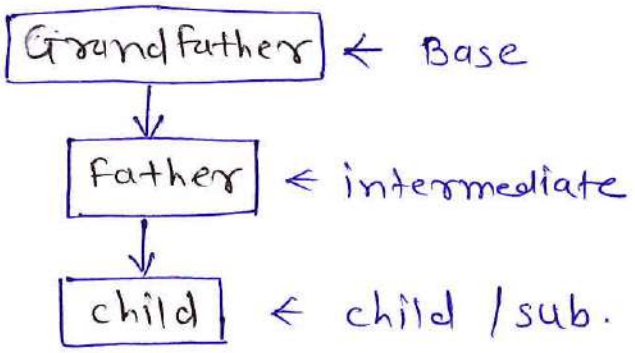
Syntax:-

```
class A {
    - -
};
class B : specifier A {
   —
   - -
};
class c : specifier B {
  —
  —
};
```

e.g.

```
#include<iostream.h>
using namespace std;
class Grundfather {
```

```
    public:
        void g(){
        cout << "This is Grandfather class" << endl;
        }
};
class Father : public Grandfather {
  public:
      void f(){
       cout<<" This is  futher class" << endl;
      }
};
class child : public Father {
public:
     void c(){
       cout<<" This  is  child class"<<endl;
     }
};
int main()
{
    child obj;  // object of child class.
    obj. c();  // calling method of own class.
    obj. f();  // calling method of parent class.
    obj. g();  // calling method of grandparent class.
    return 0;
}
```

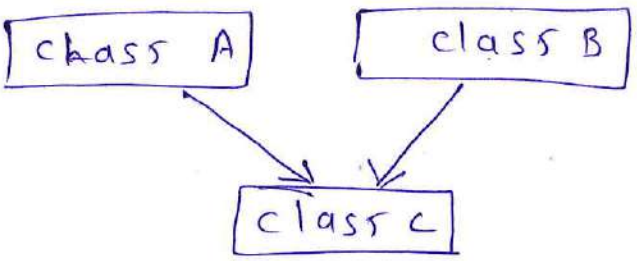


(3) Multiple Inheritance:-

Multiple inheritance is a feature of object oriented programming.

↳ Here a single subclass inherit attributes and methods from more than one parent class.

↳ one child inherits from more than one parent class.



Structure:- A + B → C.

Syntax:-

```
class A {
   // Base class A
};
class B {
   // Base class B
};
class C : Public A, Public B {
   // Derived class B.
};
```

e.g.

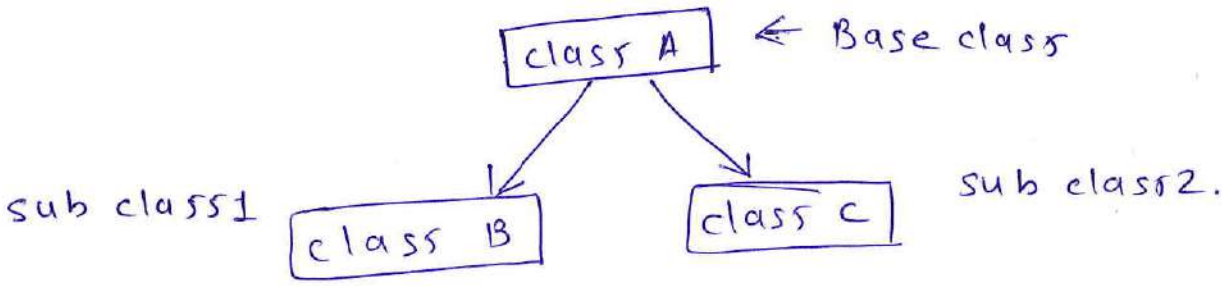
```
#include<iostream>
using namespace std;
class A{
public:
    A(){
    cout<<"method of classA"<<endl;
    }
};
class B{
public:
   B(){
   cout<<"method of class B"<<endl;
   }
};

class c: public B, public A {
public:
    c(){
    cout<<" method of classc"<<endl;
    }
};
int main()
{
    c obj;    // object of derived class c.
    c.c(); // calling own method
    c.B(); // calling Base class B method
    c.A(); // calling Base class A method
```

(4) Hierarchical Inheritance:-

Hierarchical inheritance is an object oriented programming feature.

↳ Here, multiple child classes inherit directly from ~~as~~ a single parent class.

↳ It is one to many relationship, creating tree like structure.



Syntax:-

```
class A
{
  ____
  ____
};
class B: access_specifier class A
{
  ____
  ____
};
class c: Access-specifier class A
{
  ____
  ____
};
```

e.g.

```
#include <iostream.h>
using namespace std;
```

```
class A          // Base class
{
  public:
     void show-A().
     {
       cout << " method of class A " << endl;
     }
};
class B : public A     // sub class 1
{
  public:
     void show-B()
     {
       cout << " method of class B" << endl;
     }
}
class C : public A     // subclass 2
{
  public:
     void show-c().
     {
       cout<< " method of class c" << endl;
     }
};
int main()
{
     B b;          // object of class B.
     b.show-B();  // calling own method of class B
     b.show-A();  // calling parent method of class A
     C c;          // object of class C
     c.show-c();  // calling own method of class C
     c.show-A();  // calling parent method of class A
}
```

Structure of above program is

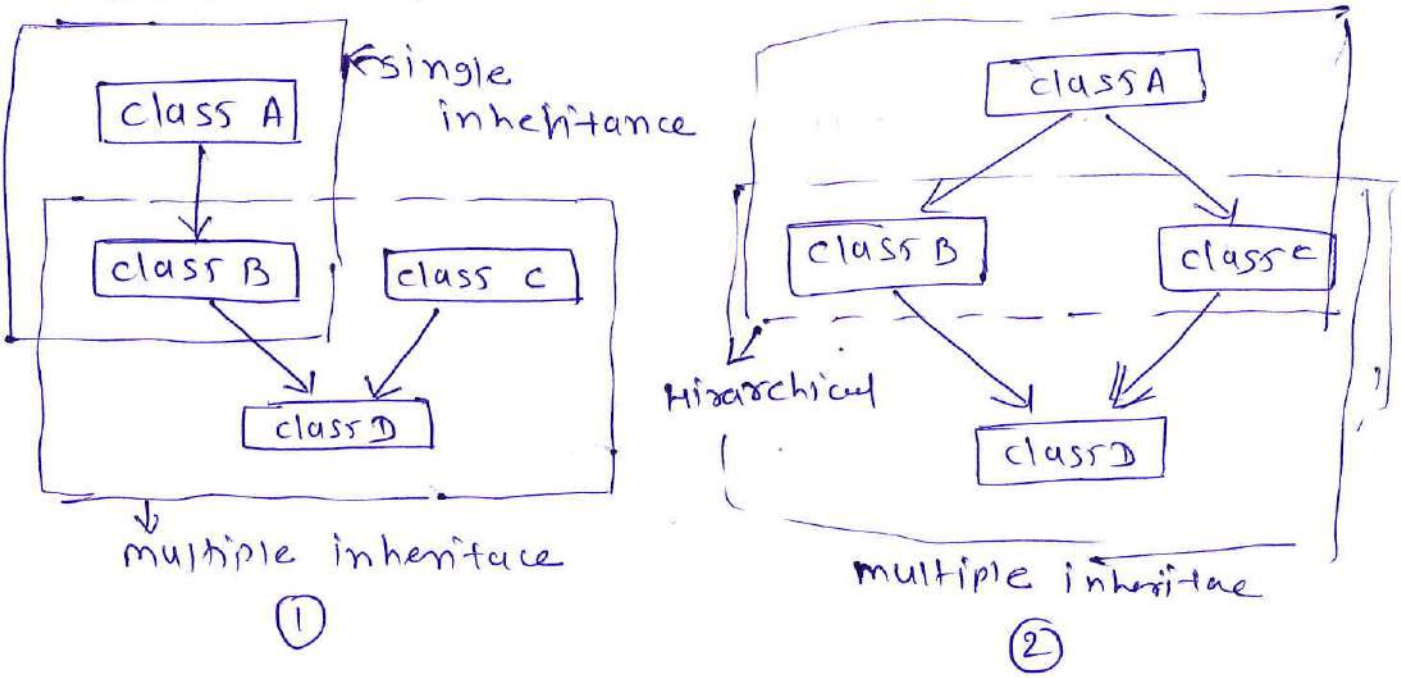
A → B and A → C where A is Parent class

and B and C are child classes inherited from class A.

(5) Hybrid Inheritance:-

Hybrid inheritance is a combination of two or more types of inheritance. such as single, multiple multilevel or hierarchical inheritance.

In hybrid inheritance, we can have elements of single inheritance, multiple inheritance, multilevel inheritance and hirarchicel inheritance.

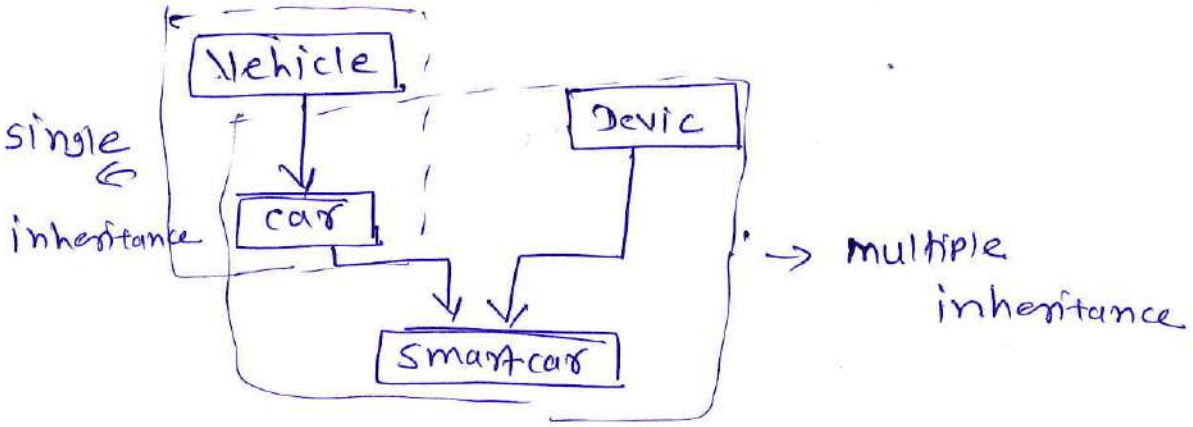


↳ In above two diagram is example of two types of hybrid inheritance.

↳ First is combination of single level and multiple inheritance

↳ second is combination of Hierarchical and multiple inheritance.

e.g. This example combines ~~Hierarchical~~ single and multiple inheritance.



Here, Vehicle is Base class.

~~Car~~ car inherits from vehicle class.

Device is second Base class.

smartcar inherits from both car and Device class.

```
e.g. #include <stdio.h>
     Using namespace std;
class Vehicle {          // Base class1
public:
     void vehicleType(){
      cout<<" Vehicle is first Base class"<<endl;
     }
};
class Device {           // Base class2
public:
     void
        devicefeature(){
        cout<<" Device is another Base class"<<endl;
     }
};
class car: public Vehicle{     //
public:
     void cartype(){
      cout<<" It is subclass of Vehicle class "<<endl;
};   }
```

(d) class smartcar : public car, public Device{

```
public:
    void smart() {
    cout << "It is created from car and Device
            class" << endl;
    }
};
int main()
{
    Smartcar sc;
    sc.devicefeature(); // calling parent class method.
    sc.cartype(); // calling method of car class (parent)
    sc.smart(); // calling own method
    sc.Vehicletype(); // calling grand parent class method.
    return 0;
}
```

Virtual Base class :-

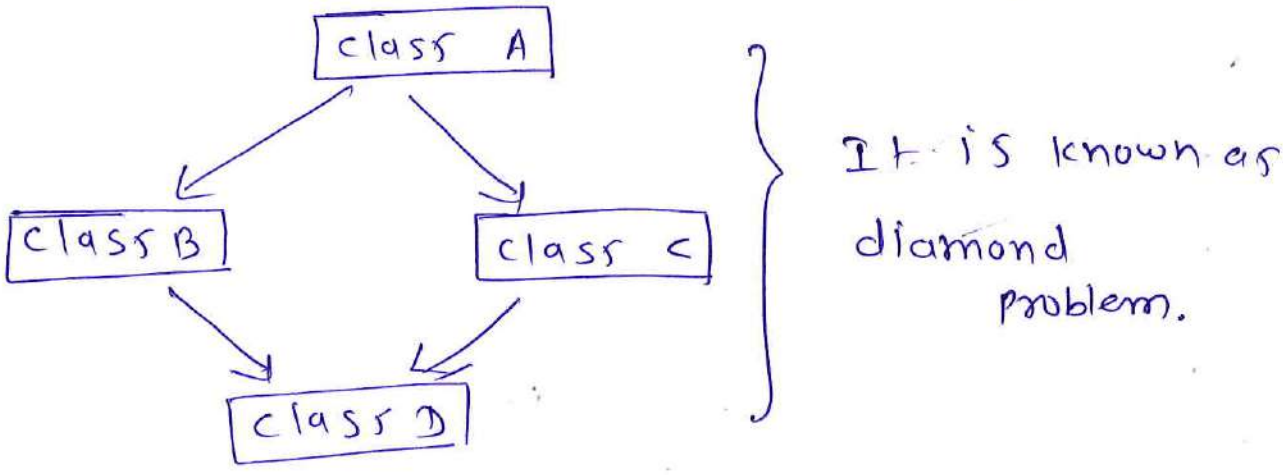
To understand virtual Base class first, it is necessary to understand its need.

* Need for virtual base class :-

↳ consider situation where we have one class A This class A has inherited two other classes B and C (A → B & A → C i.e. Hierarchical inheritance)

↳ Both these classes have one sub class D. (B → D, C → D i.e. multiple inheritance).

↳ This situation is illustrated in figure below:



↳ Above figure is combination of hierarchical and multiple inheritance i.e. hybrid inheritance.

↳ It form a diamond like structure, as shown in figure.

↳ In above figure data members of class A are inherited to class D through two paths, forming diamond shape.

↳ An ambiguity that occurs in OOP during multiple inheritance, when sub class inherits

from two parent classes, and both these Parent class (here class B and C) having common base class (here class A).

↳ The ambiguity is to use properties of grand parent class, child class is confused which copy to use either from parent class 1 ie. class A or from parent class 2 ie. class B.

↳ second problem is data duplication as both class i.e. class B and class C have same copies of properties of class A, result in waste of memory.

↳ In object oriented programming has solution for it.

↳ In OOP virtual base class is used for that.

↳ virtual base class is used in virtual inheritance to solve problem of ambiguity due to Diamond problem' discussed before.

↳ By declaring the grandparent class as virtual, it is ensured that only one copy of properties and methods of grandparent class exist in memory.

↳ To implement is virtual keyword is used before parent class.

Syntax:-

The virtual keyword is used before or after access specifier during inheritance. order of virtual keyword does not matter, we can place it either before or after the any access specifier.

```
class GrandParent{
public:
    int x;
};
class parent1 : virtual public GrandParent
    { };
class parent2 : public virtual GrandParent
    { };
class child : public parent1, public parent2
    { };
```

e.g.

```
#include <iostream.h>
class Animal {
public:
    int age = 5;
    Animal()
    {
        cout << " Animal method " << endl;
    }
};
```

```
class mammal : virtual public Animal{};
class Bird : virtual public Animal {};
class Bat : public mammal, public Bird {};
int main()
{
    Bat bject;
    cout << " Bat's age " << bject.age << "\n";
    return 0;
}
```

o/p.

Bat's age : 5

* Abstract class :-

A class with at least one pure virtual function is an abstract class that cannot be instantiated and serves as a blueprint for derived classes, which provide their own implementation.

- L> A class with at least one pure virtual function becomes an abstract class and objects of abstract class cannot be created directly.
- L> virtual function means, function with no implementation in a base class.

↳ Abstract classes used to define interfaces and for common structure among derived classes.

↳ These classes are useful in polymorphism where different classes share the same interface but diffent behaviour,

```
# include <iostream.h>
Using namespace std;
class shape {
public:
    virtual void draw() = 0;
};
class circle : public shape {
public:
    void draw() override {
        cout << "Drawing circle";
    }
};
int main() {
    shape * s = new circle();
    s -> draw();
    delele s;
}
```

* constructor in derived class :-

What is constructor ?

↳ A constructor is a special function in a class.
↳ It initializes objects of the class.
↳ It has same name as the class name.
↳ It is called automatically when object is created.

* constructors in derived class :-

↳ A derived class can have its own constructor.
↳ The derived class constructor can also call the constructor of Base class.

↳ It is important to initialize base class properties first before the derived class adds its own.

↳ The base class constructor is always executed before the derived class constructor to ensure that all inherited members are properly initialized.

How to call Base class constructor —

↳ In c++, base class constructor can be called:

```
BaseclassName (parameters).
```

↳ Derived class constructor is called after base class constructor.

↳ If the base class constructor requires arguments, derived class must call it explicitly.

↳ Using constructor ensures proper initialization of both base and derived class members.

e.g.

```
#include< iostream.h>
Using namespace std;
class Vehicle{
 protected :
        string brand;
 public:
    Vehicle (string b) {
      brand = b;
      cout<< " vehicle constructor is called"<< endl;
    }
};
class car : public vehicle {
   int doors;
 public:
    car (string b, int d) : vehicle(b) // call base
                                       class
 doors = d;                            constructor.
   cout << " car constructor is called " <<endl;
  ↳> void display () {
  }       cout << " Brand
};
int main()
{
   car mycar ("Maruti", 4);
   mycar. display();
   return 0;
}
```

O/P.

Vehicle constructor called

car constructor called

Brand: maruti, Doors: 4.

(19)

Explaination

↳ when mycar is created, first vehicle's constructor is called.

↳ Then car's constructor runs.

↳ brand is initialized by base class, doors by derived class.

9/6/26

Unit-3.

Asc (c)

Object oriented programming.

-: Question Bank :-

2 Marks.

(1) Explain in brief abstract base class with example (Winter-2024)

(2) State any two visibility modes. (Winter-2024)

(3) Define multilevel inheritance with diagram. (Sum-2025)

(4) Define inheritance. List its types (Win-2025)

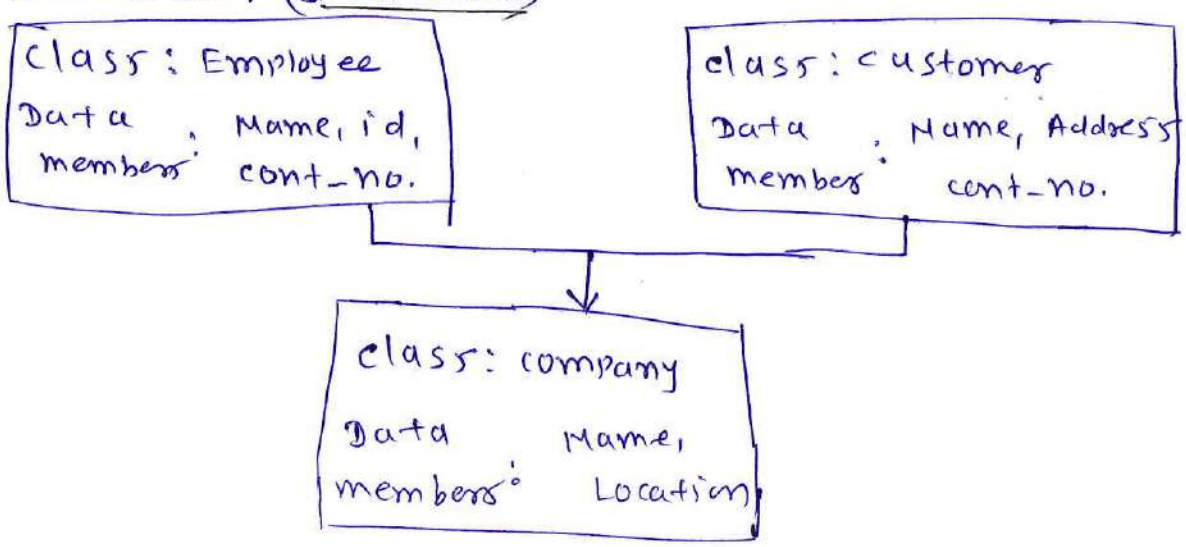
4 Marks

(5) Explain virtual base class with suitable example. Win-2024 (4 marks) and Sum-2025 (2 marks)

(6) Explain pointer to derived class with suitable example. (Win-2024)

(7) List types of inheritance with suitable example. Explain single inheritance (Win-2025)

(8) Write c++ program to implement inheritance shown in figure below: (Sum-2025)



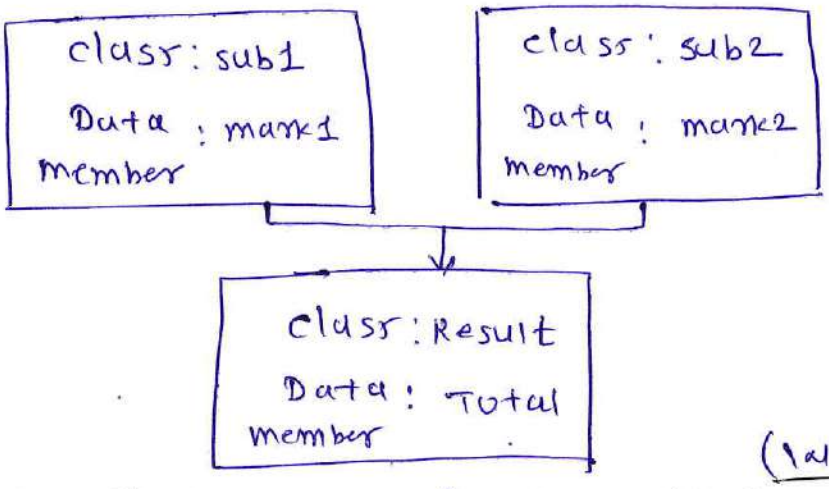
(9) Define hybrid inheritance. Explain with example (Sum-2025)

(10) Explain multiple inheritance with suitable example (Win-2025)

(11) Explain the concept of constructor in derived class with suitable example (Win-2025)

6 Marks

(12) Write a c++ program to implement following inheritance.



(Win-2024)

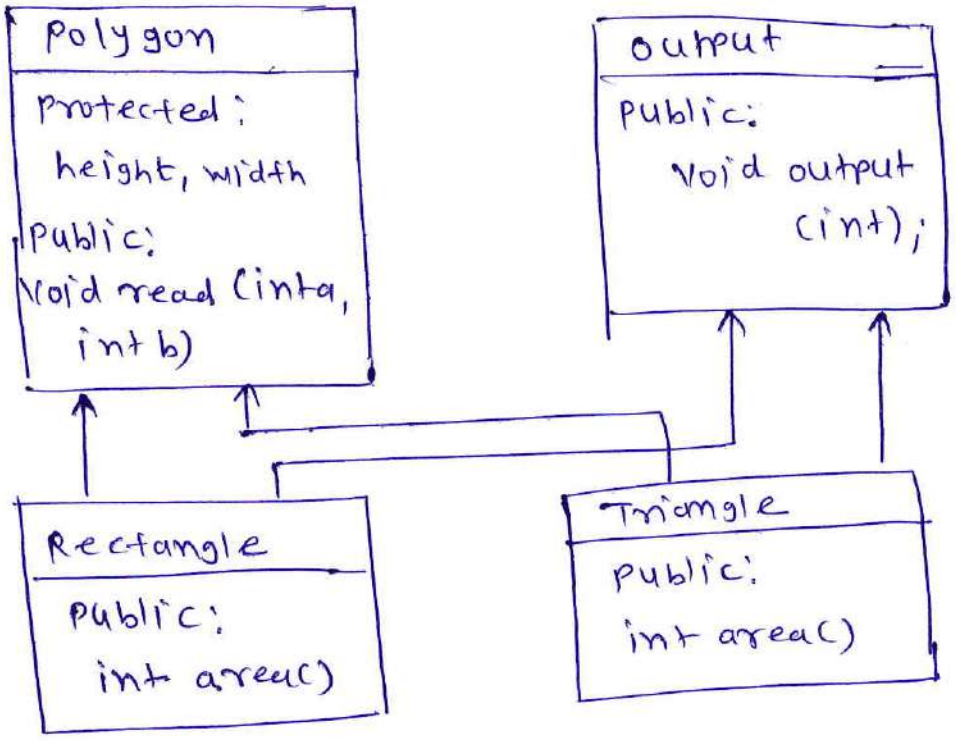
Accept & display data for two objects of class Result.

(13) Write a c++ program to show ~~for two objects of class Result~~. Use of multiple constructors in base class & single constructor in derived class. create one object (Win-2024)

(14) Write a program to demonstrate single and multilevel inheritance. (Win 2025)

(15) Explain the concept of virtual base class with suitable example. (Win 2025)

(16) Write a program to define the following relationship using multiple inheritance: (Sum-25)



8/6/16