Date. No.

Inheritance in C++

→ Inheritance in C++ allows us to define a new class in terms of another existing class.

→ Inheritance has "IS-A" relationship model.

(x): Importance:

i) reuseability of code
ii) maintainability of code.

→ Base class: The class from which the new class inherits data members and functions is called base class. / parent class / superclass.

→ Derived class: The new class that is created inheriting data members and functions from existing class is called derived class.

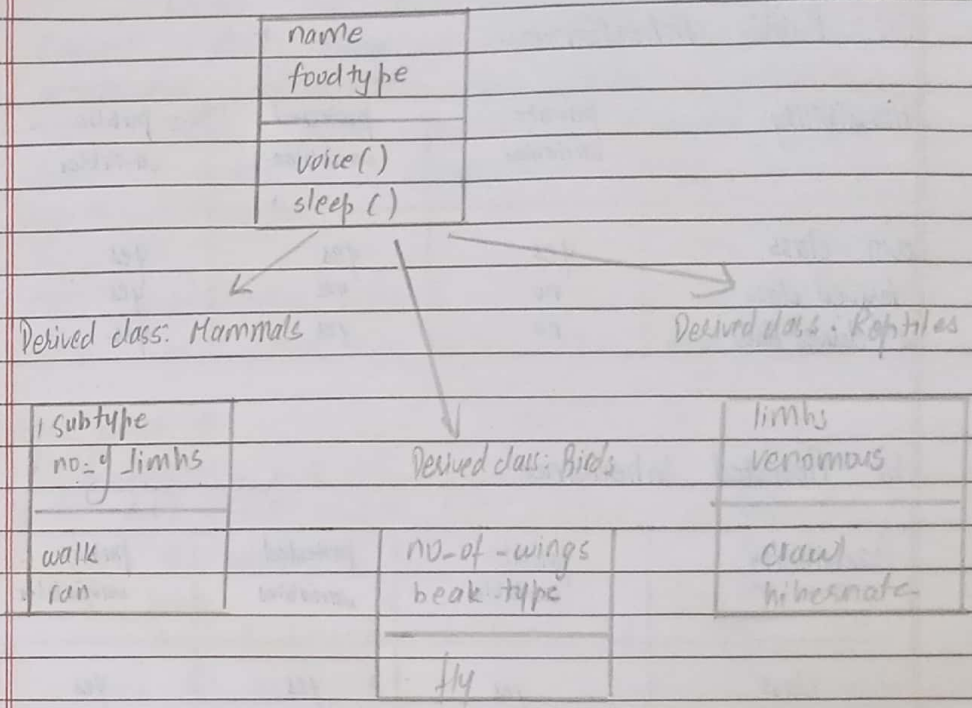
X) Syntax:

```
class derived_class_name : access specifier base_class.
{
    // data members and members functions
};
```

Date. No.

Eg:

Base class: Animals



Date. No.

(*) Accessibility in Inheritance:

(a): Public Inheritance:

Accessibility	private variables	protected variables	public variables.
own class	yes	yes	yes
derived class	no	yes	yes
2nd derived class	no	yes	yes.

(b) Protected Inheritance:

Accessibility	private variables	protected variables	public variables
own class	yes	yes	yes
derived class	no	yes	yes
2nd derived class	no	yes	yes

(c) Private Inheritance.

Accessibility	private variables	protected variables	public variables
own class	yes	yes	yes
derived class	no	yes	yes
2nd derived class	no	no	no.

Date. No.

(*) Order of Construction of Derived Class:

When we instantiate an instance of Derived, first Base portion of Derived is constructed using the Base default constructor.

Once base portion is finished, the derived portion is constructed using derived default constructor.

For inheritance chain, C++ always constructs the "first" or "most ~~eas~~ base class first.

Eg:
```
class A
{ public:
    A() { std:: cout<< "A\n"; }};
class B: public A
{ public:
    B() { std:: cout << "B\n"; }};
class C: public B
{ public:
    cC) {std:: cout << "C\n"; }}
int main()
{
std:: cout << "Constructing A: \n"; AcA;
std:: cout << "Constructing B: \n"; BcB;
std:: cout << "Constructing C: \n"; C cC; }
```

Output:

```
Constructing A
A
Constructing B
A
B
Constructing C
A
B
C
```

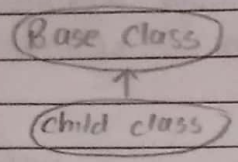
Output:

→ ~~desi~~ Default constructor is implicity available from Parent to Child class.

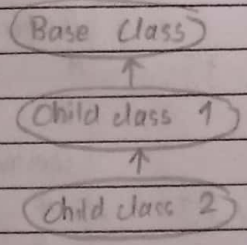
(x) Types of Inheritance:

i) Single Inheritance
ii) Multilevel Inheritance
iii) Multiple Inheritance
iv) Hierarchical Inheritance
v) Hybrid Inheritance.

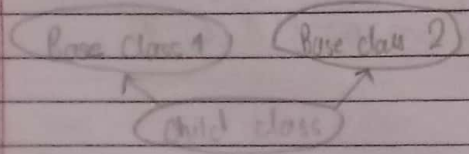
(i): Single Level Inheritance:



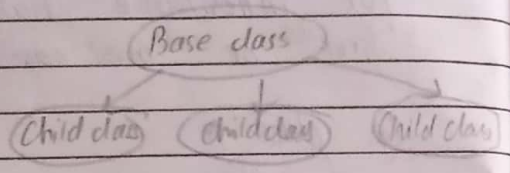
(ii) Multilevel Inheritance:



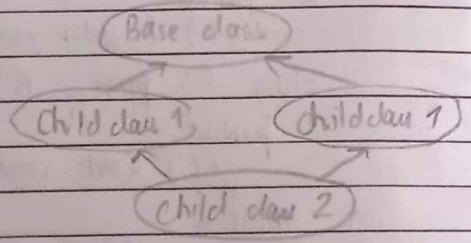
(iii) Multiple Inheritance:



(iv) Hierarchical Inheritance.

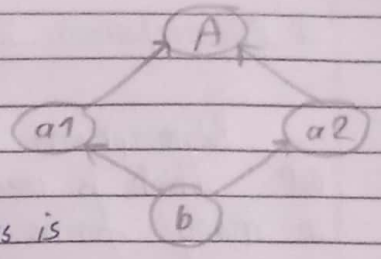


(v) Hybrid Inheritance.



(x) Diamond Problem:

When two superclasses of a class have a common base class.



In this issue, the common class is called twice.

To resolve the ambiguity, we use ~~virtual functions~~. the common base class as virtual base class using 'virtual' keyword.

Eg: class a1: virtual public A {};

Function Overriding

If derived class defines same function as defined in its base class, it is known as function overriding in C++.

→ If you create an object of the derived class and call the member function which exists in both classes (base and derived), the member function of the derived class is invoked and the function of the base class is ignored.

→ It enables you to provide specific implementation of the function which is already provided by its base class.