OOP Short Note

0. Class and Object:

Class: A class is a user-defined data type in C++ that encapsulates data members and member functions and it works as an object constructor or a "blueprint" for creating objects.

Object: An object is a data structure that is an instance of a class

Constructor: A constructor is a special member function of a class that is called when an object is created. It initializes the object's data members to their initial values.

Two types of constructors:

a) Default Constructor: This takes no parameter and sets the value of object members to some fixed values

b) Parameterized Constructor: This takes parameters and sets the values of object members accordingly.

Destructor: A destructor is a special member function of a class that is called when an object is destroyed. It cleans up any resources that the object was using.

Virtual destructor: A virtual destructor is used to free up the memory space allocated by the derived class object or instance while deleting instances of the derived class using a base class pointer object. A base or parent class destructor use the virtual keyword that ensures both base class and the derived class destructor will be called at run time, but it called the derived class first and then base class to release the space occupied by both destructors.

1. Encapsulation:

Encapsulation: Encapsulation is a process of combining member functions and data members in a single unit called a class. The purpose is to prevent access to the data directly from outside.

2. Inheritance:

Inheritance: Inheritance is a mechanism that allows a class to inherit the properties and behaviors (methods) of another class.

In C++, there are five types of inheritance:

Single inheritance: A derived class inherits its property from only one base class.
Multiple inheritance: More than one base class is used to create a derived class.
Hierarchical inheritance: Here, One base class inherits its properties to more than one derived class.
Multilevel inheritance: When one derived class inherits its property from another derived class.
Hybrid inheritance: The combination of any of the above four types of inheritance.

How constructors work with Inheritance:

In case of creating a derived class object without passing any argument (just declaring a variable):

The fact is when you create an object of the derived class, one constructor of the base class must be called. So if your default constructor of the derived class doesn't inherit the base class default constructor, the default constructor of the base class will be automatically called. And if you define the derived class default constructor by inheriting the default constructor of the base class, then the default constructor of the base class will be called once. Not automatically but by the default constructor of the derived class.

In practice, you should define the default and parameterized constructors of the derived class by inheriting the same type of constructor of the base class.

Example of derived class constructors' definition:

```
class cube :  rect
{
   int height;
public:
   cube() : rect() { height = 0; }
   cube(int w,int l,int h) : rect(w,l) { height = h; }
};
```

In case of creating a derived class object by passing arguments:

You can do this,

```
class cube :  rect
{
   int height;
public:
   cube() : rect() { height = 0; }
   cube(int w,int l,int h) { height = h; }
};
```

In that case, the default constructor of the base class will be called automatically and obviously, the parameterized constructor of the derived class will be called for the parameterized request of creating the object.

Lets have a look on the logical way of definatioin

```
class cube :  rect
{
   int height;
public:
   cube() : rect() { height = 0; }
   cube(int w,int l,int h) : rect(w,l) { height = h; }
};
```

In that case, for the parameterized request of creating the object, the parameterized constructor of the derived class will be called, and by it, the parameterized constructor of the base class will also be called.

Ambiguity in Multiple Inheritance: In multiple inheritance, we derive a single class from two or more base or parent classes. So, it can be possible that both the parent classes have the same-named member functions. While calling via the object of the derived class (if the object of the derived class invokes one of the same-named member functions of the base class), it shows ambiguity. To avoid this we can call the function this way, derived_obj.base_class_name::function_name();

Diamond problem in Multiple Inheritance: It occurs when a class inherits from two or more classes that have a common base class, resulting in a diamond-shaped inheritance hierarchy.

-> Class A has a function named, introduce()
-> Class B and C derive from Class A.
-> Class D derives from both B and C

```
   A
  / \
 B   C
  \ /
   D
```

The problem is, in Class D the introduce() function of class A is copied two times which results in ambiguity. To avoid this we have to inherit Class A virtually for both class B and C.

3. Polymorphism:

Polymorphism: Polymorphism is a feature of object-oriented programming that let the same entity (function or operator) behaves differently in different scenarios.

Following are the two types of polymorphism in C++ :

1. Compile Time Polymorphism
   a) Function overloading b) Operator overloading
2. Runtime Polymorphism
   a) Function overriding b) Virtual function

Function Overloading: Function overloading in object-oriented programming (OOP) refers to the ability to define multiple functions with the same name but different parameters within a class. It allows a class to have multiple methods with the same name but different behaviors, based on the type, number, or order of the parameters.

Operator Overloading: As long as we are working with user-defined types like objects or structures, we are able to overload an operator in C++. We cannot use operator overloading for basic types such as int, double, etc.

In C++, the following operators cannot be overloaded:

1. Member selection operator: . (dot)
2. Pointer-to-member operator: .* (pointer-to-member-by-reference)
3. Scope resolution operator: :: (scope resolution)
4. Sizeof operator: sizeof
5. Conditional operator: ?: (ternary operator)

Function Overriding: We can have the same name and/or the same parameterized function in both base and derived classes. If the function is called using a derived class object, then the derived class function is executed. And, if the function is called using a base class object, then the base class function is executed. This feature is called function overriding.

Virtual function : We may not be able to override functions if we use a pointer of the base class to point to an object of the derived class. Using virtual functions in the base class ensures that the function can be overridden in these cases.

So, to call a derived class function using base class pointer you have to make the base class function virtual.