**OOPs**

**(Object-Oriented Programming)**

**Introduction:**

Programming Paradigms: Procedure-oriented programming and Object-oriented programming.

The C++ programming language is categorized as a partial object-oriented programming language.

The programming paradigm where everything is represented as an object is known as truly object-oriented programming language. **Smalltalk** is considered as the first truly object-oriented programming language.

Object means a **real word entity** such as pen, chair, table etc. Object is a runtime entity, it is created at run-time.

Any entity that has **state/properties and behaviour** is known as an object. It can be physical and logical.

**Collection of objects** is called class. It is a logical entity which represents the template or blueprint of the objects

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance.

1. Super Class/Base class
2. Sub Class/Derived class
3. Reusability

**When one task is performed by different ways**i.e. known as polymorphism.

**Hiding internal details and showing functionality** is known as abstraction.

**Binding (or wrapping) code and data together into a single unit** is known as encapsulation.

In dynamic binding, a **decision is made at runtime regarding the code that will be run in response to a function call**. For this, C++ supports virtual functions.

**Polymorphism also has a subset known as overloading**. An existing operator or function is said to be overloaded when it is forced to operate on a new data type.

**Advantages of OOPs:**

OOPs makes development and maintenance easier.

OOPs provide data hiding.

OOPs provide ability to simulate real-world event much more effectively.

OOPs provide reusability, modularity and extensibility to the code.

POPs has global data access.

**C++ is a partial object-oriented programming language:**

1. The main function must always be outside the class in C++ and is required. This means that we may do without classes and objects and have a single main function in the application.
2. Global variables are a feature of the C++ programming language that can be accessed by any other object within the program and are defined outside of it.

**C++ Constructors:**

In C++, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C++ has the same name as class or structure.

1. <**class**-name> (list-of-parameters);
2. <**class**-name> (list-of-parameters) {// constructor definition}
3. <class-name>: :<class-name> (list-of-parameters) {// constructor definition}

(Scope resolution operator: used to access the constructor outside the class)

There can be two types of constructors in C++.

* Default constructor/No Argument constructor

It is always invoked at the time of object creation.

* Parameterized constructor

Used to initialize the values at the time of the object creation.

* Copy constructor

Example:

Give me a marker. (Default constructor)

Give me a red marker of XYZ brand. (Parameterized constructor)

Give me a similar marker as that of I have in my hands. (Copy constructor)

Some important points about constructors:

1. Constructors are special member functions invoked at the time of the object creation and they are used to initialize the values of the object generally.
2. Constructors can be private, but it is not recommended to declare them in the private region.
3. Constructors have no return type.
4. Constructors cannot be inherited but can be overloaded.
5. Virtual constructors are not allowed.
6. Constructor addresses cannot be accessed.
7. Constructor makes implicit calls to new and delete operator.

Constructors cannot be inherited but child class can call the base class constructor.

class Base {

public:

Base (int a) {//constructor for base class}

}

class Derived: public Base {

public:

Derived (int a, int b): Base (int a) {//constructor for derived class}

}

class Base {

Base (int a) {//constructor for base class}

}

class Derived extends Base {

Derived (int a, int b) {

//constructor for derived class

super(a);

}

}

class Base:

def \_\_init\_\_(self,a):

self.a = a

class Derived (Base):

def \_\_init\_\_(self,a,b):

super().\_\_init\_\_(a)

Copy constructor:

A member function known as a copy constructor initializes an item using another object from the same class

Sample (Sample &t) {

id = t.id

}

**C++ Destructors:**

A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically when the object goes out of scope or is explicitly deleted.

It must have same name as class. But it is prefixed with a tilde sign (~).

C++ destructor cannot have parameters.

In C++, destructors are invoked in the reverse order of object creation. This is often referred to as the "last in, first out" (LIFO) order.

**‘this’ keyword:**

this keyword is used to refer the current instance of the class.

this keyword is used to pass the current object as the parameter to the class method.

this keyword is used to declare indexers.

Indexers is a special type of property in which the instances of the class or structs can be accessed using overloaded [] operator or they can be indexed just like arrays.

class Employee {

public:

int id;

string name;

Employee (int id, string name) {

this->id = id;

this->name = name;

}

};

**‘static’ keyword:**

static keyword is used to declare the class variables, the variables that belong to the type and not the instance.

static variables share a common memory among all the instances.

There is no need to create an instance to access the static variables.

Example:

rateOfInterest in case of Account class

companyName in case of Employee class

Constructors cannot be static because constructor is used to initialize the object and it doesn’t make sense to make the constructor static as static fields are associated with the type and not the instance.

But we can have a static factory method which is a very used pattern to create objects to have more control over the object creation. Constructor is made in private section and static factory method is used for object creation.

**C++ ‘structs’:**

structs are used for light weight objects like Rectangle, Color, Point, etc.

structs are used as the value type instead of the reference type.

structs are used generally when data is not required to be modified after the struct creation and it is used to group the data of different data types.

The default access modifiers for a struct is public while for a class it is private.

**C++ ‘enumeration’:**

Enum is a data type having a fixed set of constants. Enum is used to provide type safety.

The constants of C++ enums are static and final implicitly.

enum week {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}

enum direction {NORTH, EAST, WEST, SOUTH}

week day = Friday;

For modern C++:

enum class Color {RED, GREEN, BLUE}

Color myColor = Color::RED;

**C++ ‘friend’ function:**

friend function is used to access the private or public members of the class.

friend function cannot directly access the data, instead the object of the same class is to be passed and then it is accessed using dot operator.

friend function can be declared in the private or public section of the class.

Friend function is not in the scope of the class in which it has been declared as a friend.

Similarly, friend class is used to access the private and protected members of the class.

Example:

In a company, the accounts department keeps track of the money of the company. HR department needs to access the salary component of every employee. So, the HR class may be declared as friend of the first class to access the private data.

**C++ Inheritance: (is-a relationship)**

Reusing the members of the parent class in the child class is called inheritance.

Types of inheritance:

Single inheritance

Multiple inheritance

Multilevel inheritance

Hierarchial inheritance

Hybrid inheritance

Visibility mode: private, public, protected

The default mode of visibility is private and therefore the members of the base class can only be inherited using the member functions of the derived class in case of private visibility mode.

In multiple inheritance, when the same class inherits from two parent classes then there can be a situation of ambiguity when there is same member function available in both the classes.

A: display ()

B: display ()

C: public A, public B

void view () {

A::display();

B::display();

}

**C++ Aggregation: (has-a relationship)**

Aggregation is a process in which one class defines another class as any entity reference.

class Address {

};

class Employee {

public:

Address\* address;

};

**Object Slicing:**

Object slicing happens when an object of a derived class is assigned to an object of the base class, causing the derived part of the object to be "sliced off."

Animal: Parent class

Dog: Child class

Animal a = Dog ();

**C++ Polymorphism:**

When a task is executed in different ways, it is known as polymorphism.

Example: Lady behaves as a teacher in classroom, Mother or daughter in her home and Customer in a market.

Polymorphism is of two types: Compile-time polymorphism (Function overloading and Operator Overloading)

Run-time polymorphism (Virtual functions)

Compile-time polymorphism is known as static binding or early binding and Run-time polymorphism is known as dynamic binding or late binding.

In overloading, the function to be executed is known at the compile time as the number or the order of arguments is different for functions but the prototype is same in the case of method overriding and the execution is decided at run-time.

**C++ Virtual Function:**

Virtual function is a function that is declared within the base class and it is overridden by the derived class.

When base class pointer contains the address of the derived class object, always executes the base class function. This issue can only be resolved by using the 'virtual' function.

It is used to achieve run-time polymorphism or dynamic linkage or late binding.

Virtual functions cannot be static.

A class may have virtual destructor but it cannot have a virtual constructor.

A virtual function can be friend of another class.

It is not mandatory to have the method overridden in child class. In that case, the base class method will be called.

The prototypes of a virtual function of the base class and all the derived classes must be identical. If the two functions with the same name but different prototypes, C++ will consider them as the overloaded functions.

class Base {

public:

void show () {cout<<”Show base class”<<endl;}//this will be invoked

virtual void print () {cout<<”Print base class”<<endl;}

};

class Derived: public Base {

public:

void show () {cout<<”Show derived class”<<endl;}

void print () {cout<<”Print derived class”<<endl;}//this will be invoked

};

Derived d;

Base\* bptr = d;

bptr->show ();

bptr->print ();

**C++ Function Overloading:**

Ambiguity while overloading is the situation when the compiler is unable to decide which function to invoke from the overloaded functions.

Ambiguity arises in the following situations:

1. Type conversion
2. Functions with default arguments
3. Functions with pass by reference

**C++ Operator Overloading:**

Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type.

Operator overloading is used to overload or redefines most of the operators available in C++.

The operators that cannot be overloaded are:

Sizeof operator

Scope resolution operator (::)

Member selector operator (.)

Member pointer selector operator (->)

Ternary operator (?:)

void operator + (A a) {

x + a.x;

}

void operator ++ () {

num = num+2;

}

**C++ Function Overriding:**

If derived class defines same function as defined in its base class, it is known as function overriding in C++. It is used to achieve runtime polymorphism.

**Pure Virtual Function (Do -nothing function):**

A pure virtual function is not meant for performing any task. It only serves as a placeholder. It serves as a contract that the derived class must implement.

A class containing the pure virtual function cannot be used to create their own objects, that classes are known as abstract classes.

virtual void show () = 0;

**Interfaces: (contract like structure)**

***Interfaces*** play an important role in designing clean and well-organized code structures in C++.

It acts as a blueprint that enforces consistent behavior among its implementers while shielding the details of the actual implementation.

There is no specific interface keyword available in C++. It is achieved using abstract classes and pure virtual functions.

Classes that implement the interface must implement all the pure virtual methods of the interface.

class Interface {

public:

virtual void func1() = 0;

virtual void func2() = 0;

};

class Concrete: public Interface {

public:

void func1() const override {//Implementation for method1}

void func2() const override {//Implementation for method2}

};

Example: different types of buttons (simple, radio, toggle) share a same behaviour that is click event.

**Data Abstraction in C++:**

Data Abstraction is a process of providing only the essential details to the outside world and hiding the internal details, i.e., representing only the essential details in the program.

Example: AC can be turned ON or OFF, temperature can be changed, mode can be changed, but the internal mechanism is not known to us.

Example: pow() function in C++ is implemented but we don’t know the algorithm which is used for the implementation of the function.

Data Abstraction can be achieved in two ways:

Abstraction using header files

Abstraction using classes

Advantages of abstraction:

1. Implementation details are hidden from the user.
2. Code duplication can be avoided.
3. Reusability can be increased.
4. Internal implementation can be changed without affecting the user-level code.

Remaining topics:

Memory allocation in C++ objects

Object a;

Object a = Object();

Object\* a = new Object();

Use of const in C++

Virtual function memory related details

Class and structs memory related details