OOPS in C++

**Q1. What is OOPS?**

**Ans.**

Object-Oriented Programming or OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

**Q2. Explain classes in C++.**

**Ans.**

It’s a blueprint from which objects are created.

1. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.
2. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behaviour of the objects in a Class.

**Q3. What are objects ?**

**Ans.**

It is a basic unit of Object-Oriented Programming and represents the real-life entities. An object refers to the instance of the class, which contains the instance of the members and behaviours defined in the class template. In the real world, an object is an actual entity to which a user interacts, whereas class is just the blueprint for that object. So, the objects consume space and have some characteristic behaviour.

**class** person

{

**char** name[20];

**int** id; // Attribute

**public**: // Access specifier

**void** getdetails(){} // this is a member function

};

**int** main()

{

   person p1; // p1 is an object

}

\*\* Size of the object is the size of the properties declared but in case of no properties/empty class the size comes as 1 not 0. This is done to keep track.

**Q4. What are data members?**

**Ans.**

Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behaviour of the objects in a Class.

\*\* we can declare member functions outside the class using ::

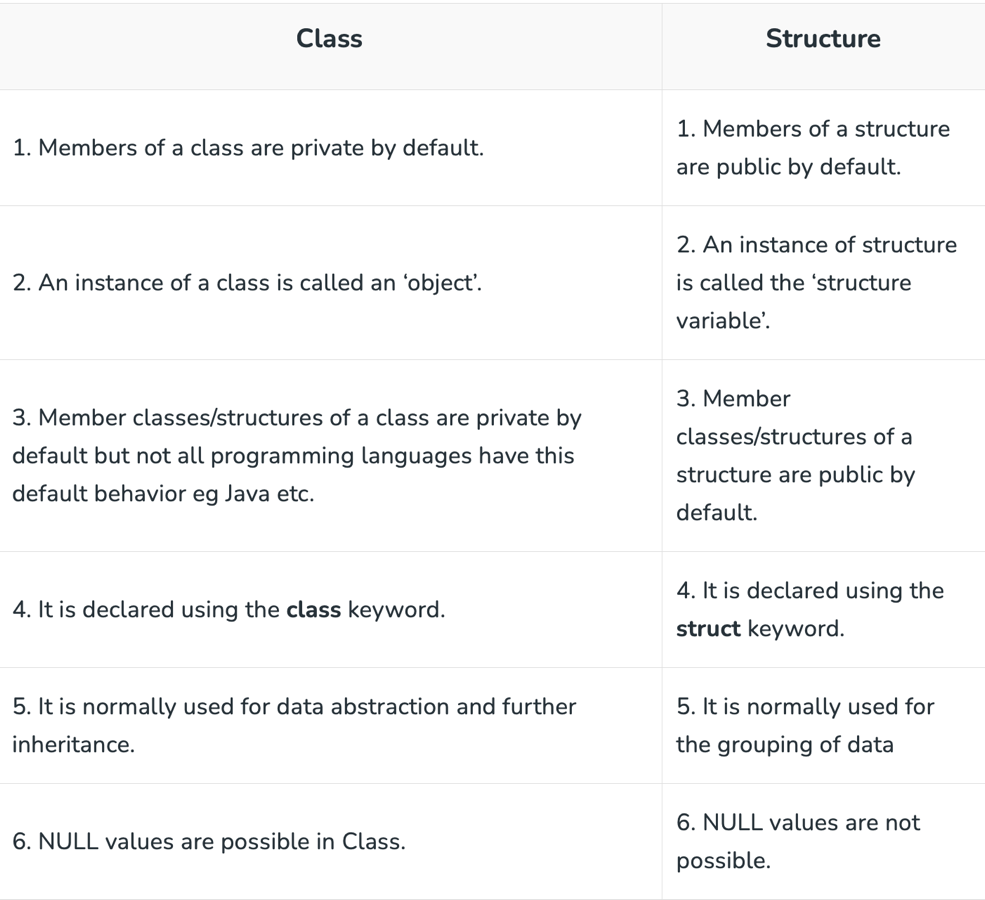
**Q5. What is need of OOPS?  
Ans.**

1. OOPs helps users to understand the software easily, although they don’t know the actual implementation.
2. With OOPs, the readability, understandability, and maintainability of the code increase multifold.
3. Even very big software can be easily written and managed easily using OOPs.
4. OOPs, promote code reuse, thereby reducing redundancy.
5. OOPs also helps to hide the unnecessary details with the help of Data Abstraction.
6. OOPs, are based on a bottom-up approach, unlike the Structural programming paradigm, which uses a top-down approach.
7. Polymorphism offers a lot of flexibility in OOPs

**Q6. Are class and structure the same? If not, what's the difference between a class and a structure?**

**Ans.**

In C++, a structure is the same as a class except for a few differences. The most important of them is security. A Structure is not secure and cannot hide its implementation details from the end-user while a class is secure and can hide its programming and designing details.

For example, the structure is saved in the stack memory, whereas the class is saved in the heap memory. Also, Data Abstraction cannot be achieved with the help of structure, but with class, Abstraction is majorly used.

**Q8. How can we declare the object dynamically?**

**Ans.**

Let us have a class named Student. A **class pointer** is a **pointer variable** that stores address of an **object** of a class. In case of pointer to a class object which is dynamically allocated to the heap. We will dereference the pointer and then use dot operator.

Student \*chan = new student(); // dynamically declared object

Chan ->marks; or (\*chan).marks; // we need to do like this to have its data

Delete chan; // we need to delete it explicitly

**Q9. How can we have an array of objects?**

**Ans.**

Students [] = {s1,s2,s3};

**Q10. What are scope resolution operators?**

**Ans.**

“::” is called scope resolution operators.

We need it for following 🡪

1. To define a function outside a class. Removing ambiguity in inheritance.

**void** A::fun() { }

1. **To access a class’s static variables.**

**int** Test::x = 1; // Test is class

1. **In case of multiple Inheritance**

A::x;

B::x;

1. **Refer to a class inside another class**

**Q11. What are access modifiers in C++?  
Ans.**

There are 3 types of access modifiers available in C++:

1. **Public🡪**

All the class members declared under the public specifier will be available to everyone. The data members and member functions declared as public can be accessed by other classes and functions too.

1. **Private 🡪**

The class members declared as *private* can be accessed only by the member functions inside the class. They are not allowed to be accessed directly by any object or function outside the class.

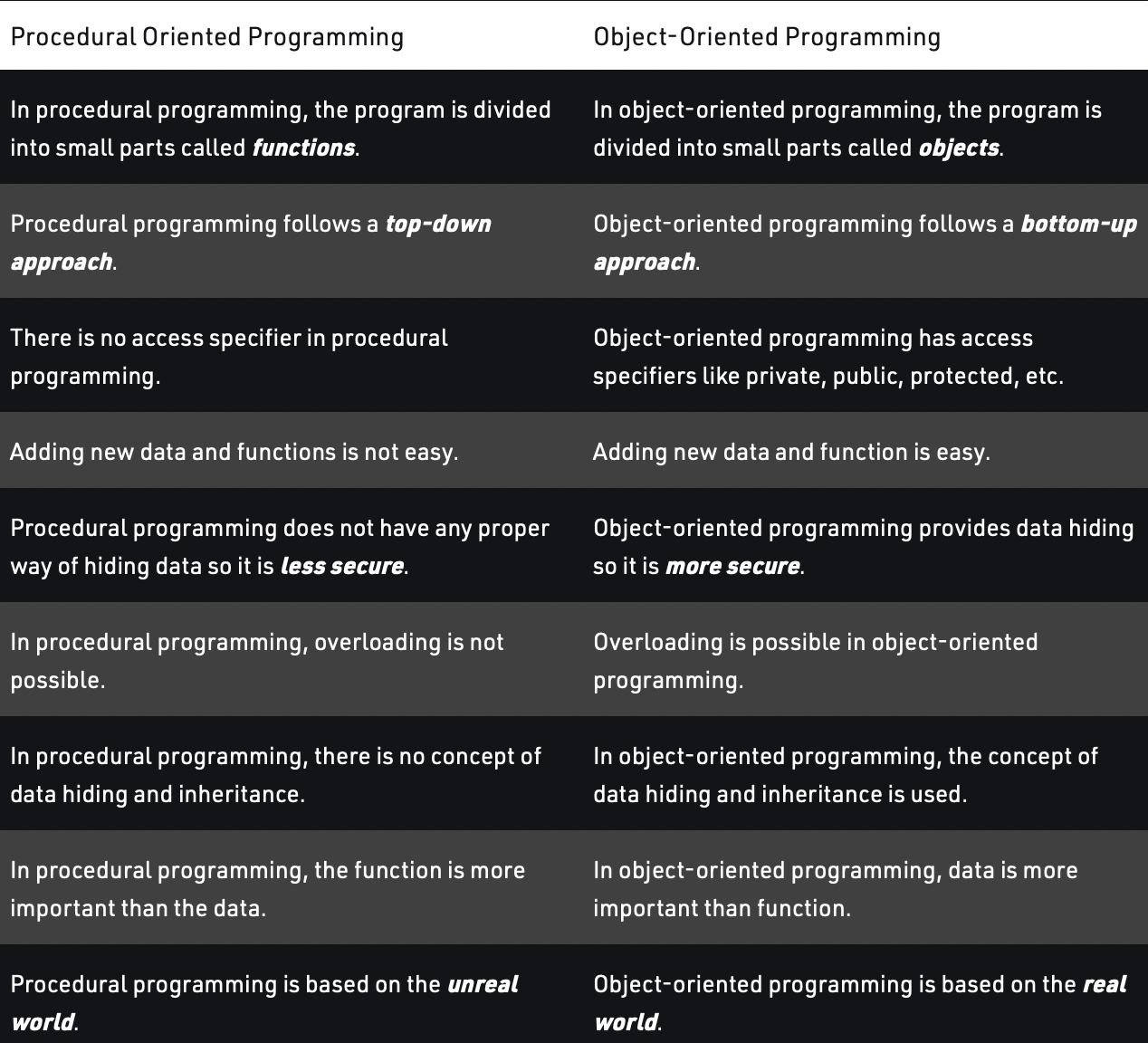
\*\* friend functions can access the private.

1. **Protected 🡪**

The protected access modifier is similar to the private access modifier in the sense that it can’t be accessed outside of its class unless with the help of a friend class. The difference is that the class members declared as Protected can be accessed by any subclass (derived class) of that class as well.

**Q12. What is procedural and object-oriented programming?**

**Ans.**

Procedural programming has multiple limitations like if the structure of the data is changed then all the functions need to be changed to work in accordance to it.

**Q13. What are Friend Class and Friend Functions?**

**Ans.**

A friend class is a class that can access both the protected and private variables of the classes where it is declared as a friend.

class Class\_1st {

// ClassB is a friend class of ClassA

friend class Class\_2nd;

statements;

}

class Class\_2nd {

statements;

}

A friend function is a function used to access the private, protected, and public data members or member functions of other classes. It is declared with a friend keyword. The advantage of a friend function is that it is not bound to the scope of the class and once it is declared in a class, furthermore to that, it cannot be called by an object of the class; therefore it can be called by other functions. Considering all the mentioned points we can say that a friend function is a global function**.**

**class GFG {**

**statements;**

**friend dataype function\_Name(arguments);**

**statements;**

**}**

**OR**

**class GFG{**

**statements'**

**friend int divide(10,5);**

**statements;**

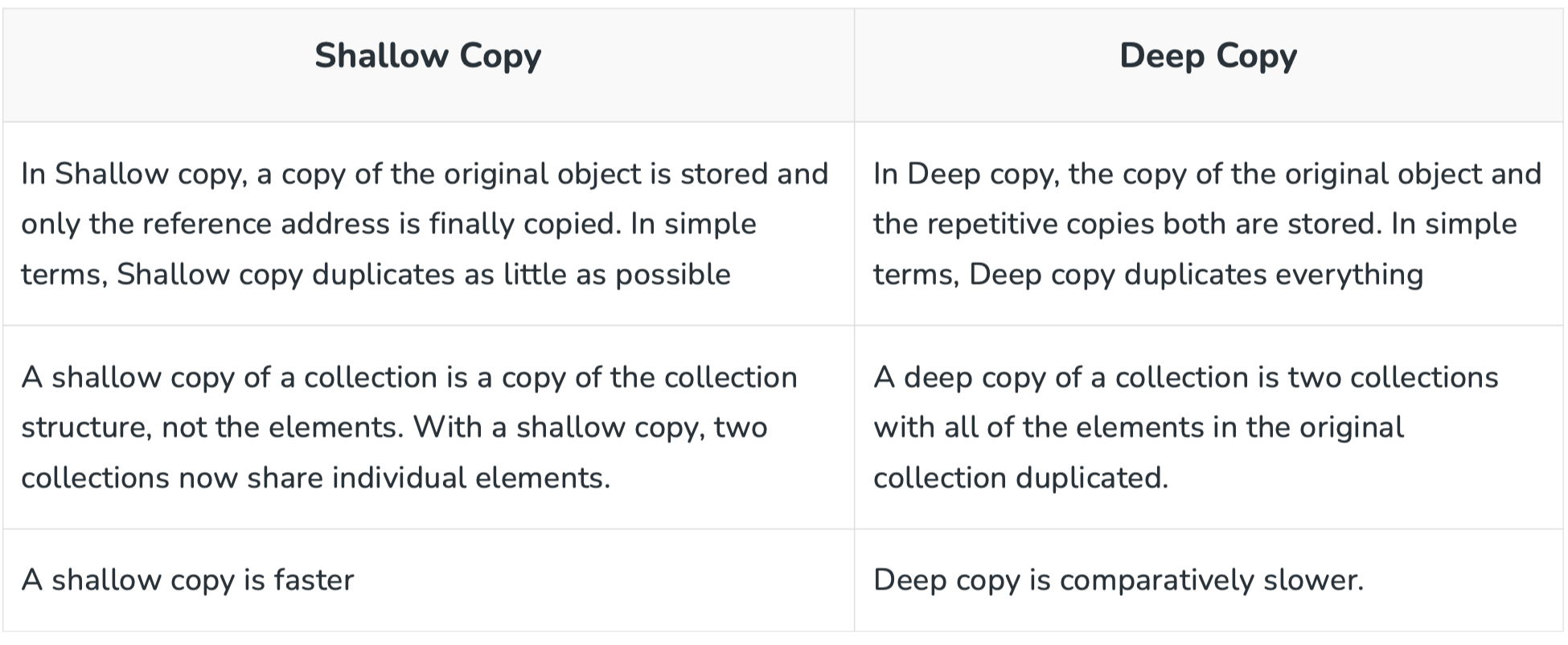
**}**

**Q14. What is the function of the keyword “Auto”?**

**Ans.**

The **auto** keyword specifies that the type of the variable that is being declared will be automatically deducted from its initializer. In the case of functions, if their return type is auto then that will be evaluated by return type expression at runtime. Good use of auto is to avoid long initializations when creating iterators for containers. We can spend less time having to write out things the compiler already knows.

**Q15. What is Shallow Copy and Deep Copy?**

**Ans.**

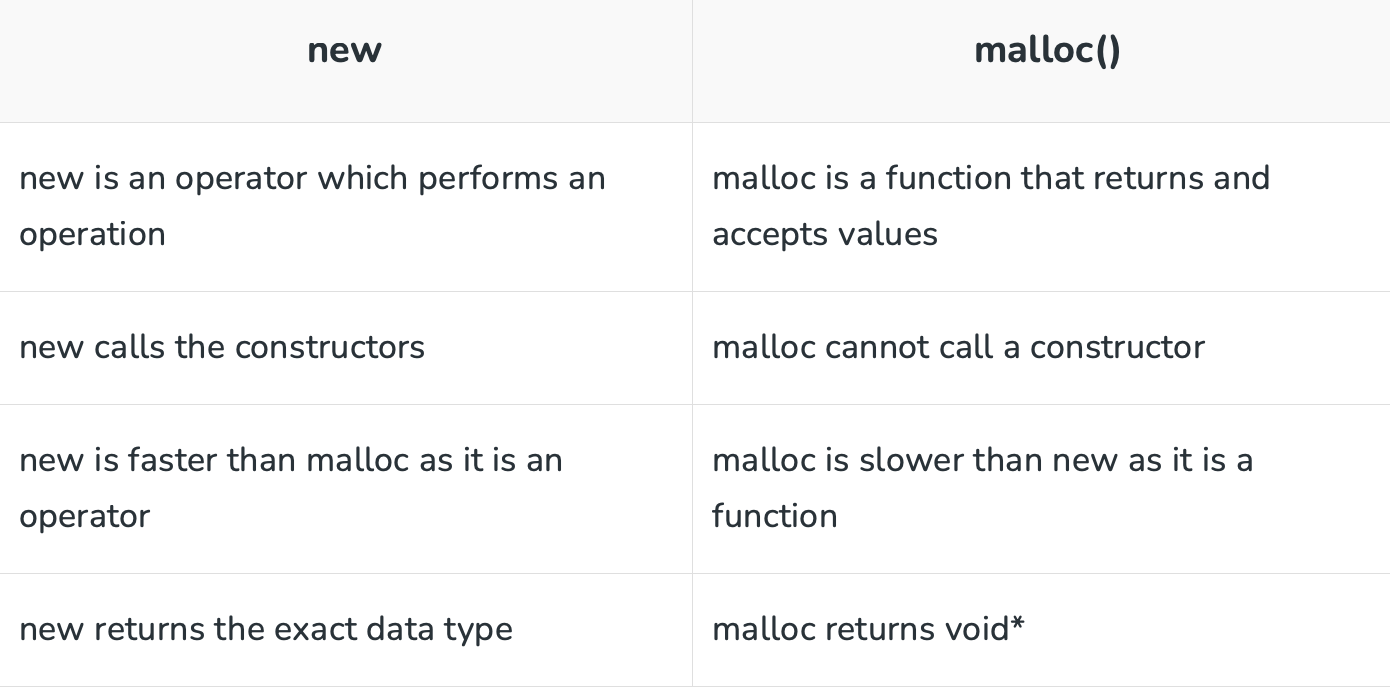
\*\*Copy constructors by default creates shallow copy

**Q16. What is ‘*this*‘ pointer in C++?**

**Ans.**

This is a pointer that holds address of the current object. If we print this we get the address of the object. This stores the address of current object. That’s why when we use this we refer to the object to avoid confusion.

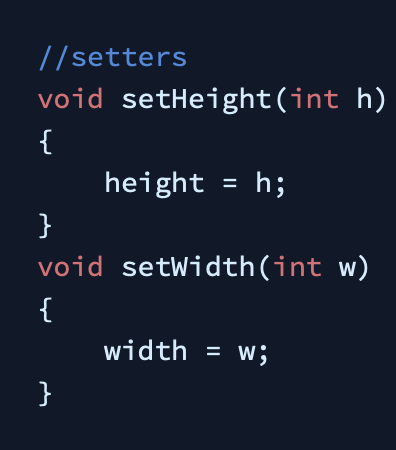
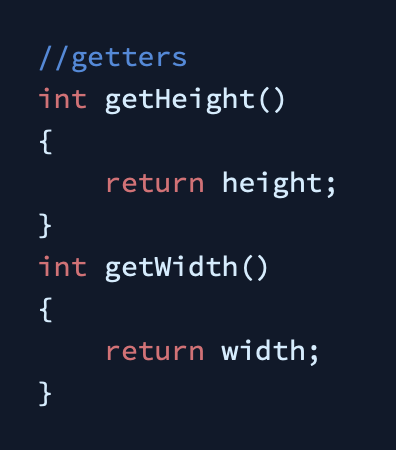
**Q17. What is the difference between new and malloc ()?**

**Ans.**

**Q.18 Why do we use getter and setter?**

**Ans.**

We use these for having control access to the private data member as we cannot access them directly outside the class. Getter and setter methods provide better encapsulation and abstraction of the class’s internal workings.

It also allows the class to modify its code without affecting the external code along with the data validation feature to protect the data member.

**Q19. What are constructors?**

**Ans.**

A constructor is a special type of member function of a class, whose name is the same as that of the class by whom it is invoked and initializes value to the object of a class. It is automatically called when an object of a class is created and can be overloaded.

There are 3 types of constructors:

1. **Default constructor:**

It is the most basic type of constructor which accepts no arguments or parameters. Even if it is not called the compiler calls it automatically when an object is created.

1. **Parameterized constructor:**

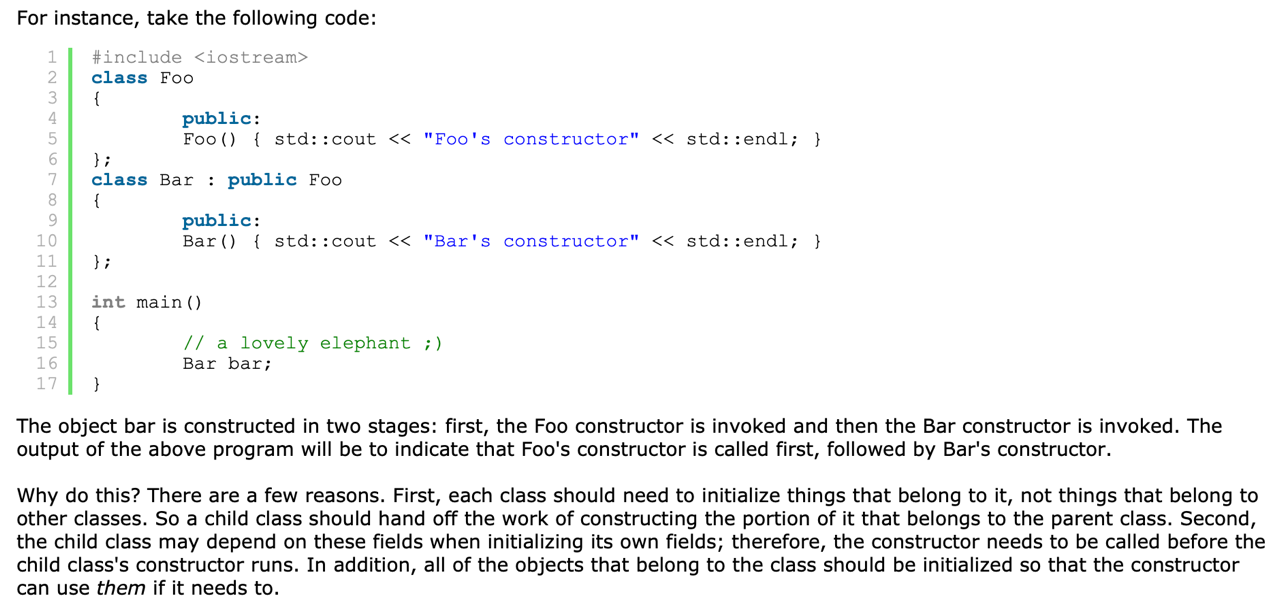
It is a type of constructor which accepts arguments or parameters. It has to be called explicitly by passing values in the arguments as these arguments help initialize an object when it is created. It also has the same name as that of the class.

1. **Copy Constructor**

A copy constructor is a member function that initializes an object using another object of the same class. Also, the Copy constructor takes a reference to an object of the same class as an argument.

// Copy Constructor

Geeks Obj1(Obj); //or

****Geeks Obj1 = Obj;

\*\*If at any time we make a constructor call, the default constructor ceases to exist.

**Q20. What are Destructors?**

**Ans.**

Destructors are members of functions in a class that delete an object when an object of the class goes out of scope. Destructors have the same name as the class preceded by a tilde (~) sign. Also, destructors follow a **down-to-top**approach, unlike constructors which follow a top-to-down.

~constructor\_name(); // tilde sign signifies that it is a destructor

destructor can-not be overloaded.

* It is automatically called when an object goes out of scope.
* Destructor release memory space occupied by the objects created by the constructor.
* In destructor, objects are destroyed in the reverse of an object creation.

The reason for reverse order is, an object created later may use the previously created object.

A a;

B b(a);

In the above code, the object ‘b’ (which is created after ‘a’), may use some members of ‘a’ internally. So, destruction of ‘a’ before ‘b’ may create problems. Therefore, object ‘b’ must be destroyed before ‘a’.

If the object is created statically the destructor is automatically called but we need to call it manually in case of dynamically created.

We do this by delete keyword. Eg. delete ob1;

I.e.  if the object is created by using new or the constructor uses new to allocate memory that resides in the heap memory or the free store, the destructor should use delete to free the memory.

Q21. What are virtual destructors.

Ans.

**Virtual Destructors:**

When destroying instances or objects of a derived class using a base class pointer object, a virtual destructor is invoked to free up memory space allocated by the derived class object or instance.

Virtual destructor guarantees that first the derived class destructor is called. Then the base class’s destructor is called to release the space occupied by both destructors in the inheritance class which saves us from the memory leak. It is advised to make your destructor virtual whenever your class is polymorphic.

\*\*Destructors can be private.

\*\*If the object is created statically the destructor is automatically called but we need to call it manually in case of dynamically created.

**Q21. What is Encapsulation?**

**Ans.**

Wrapping up data members and functions together

It is wrapping up data in a single unit. It is combination of data hiding and abstraction. It says the data members (variables) should be kept private and member functions should be used to manipulate data by this way the access of data member is limited to those who have access of the member functions

The member function which manipulates the data members should be labelled as public using the public access specifier.

Eg.1 for YouTube algorithm user has no access to the like count manipulation but can like or dislike which will change the value of total likes. Hence the data protection is covered.

**Q22. Fully encapsulated class?**

**Ans.** The class in which all the data members are private and have their functions that handles the change.

Q23. Why encapsulation?

Ans. “Data hiding”

1. Data Hiding which enhances the security.
2. We can have read only class using this property

**Q24.What is inheritance?**

**Ans.** The capability of a class to derive properties and characteristics from another class is called Inheritance.

****The class that inherits properties from another class is called Subclass or Derived Class. The class whose properties are inherited by a subclass is called Base Class or Superclass.

\*\*class ABC: XYZ     //private derivation by default

Q25. What are the types of inheritance?

Ans.

1. Single inheritance 🡪 A class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only.
2. Multilevel inheritance 🡪 A derived class is created from another derived class.
3. Multiple inheritance 🡪Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e one **subclass** is inherited from more than one **base class**.
4. Hierarchical inheritance 🡪 In this type of inheritance, more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.
5. Hybrid inheritance 🡪 Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.

syntax :

single 🡪 class child\_class : visibility\_mode parent\_class

multiple 🡪 class derived\_class: visibility\_mode\_1 base\_class\_1, visibility\_mode\_2 base\_class\_2

**Q26. What is Polymorphism?**

**Ans.**

Polymorphism is performing same thing in different ways. It is the ability to display a member function in multiple forms depending on the type of object that calls them.

In other words, we can also say that a man can be an employee to someone, a son of someone, a father of someone, and a husband of someone; this is how polymorphism can be projected in real life.

**Q28. What are the types of Polymorphism?**

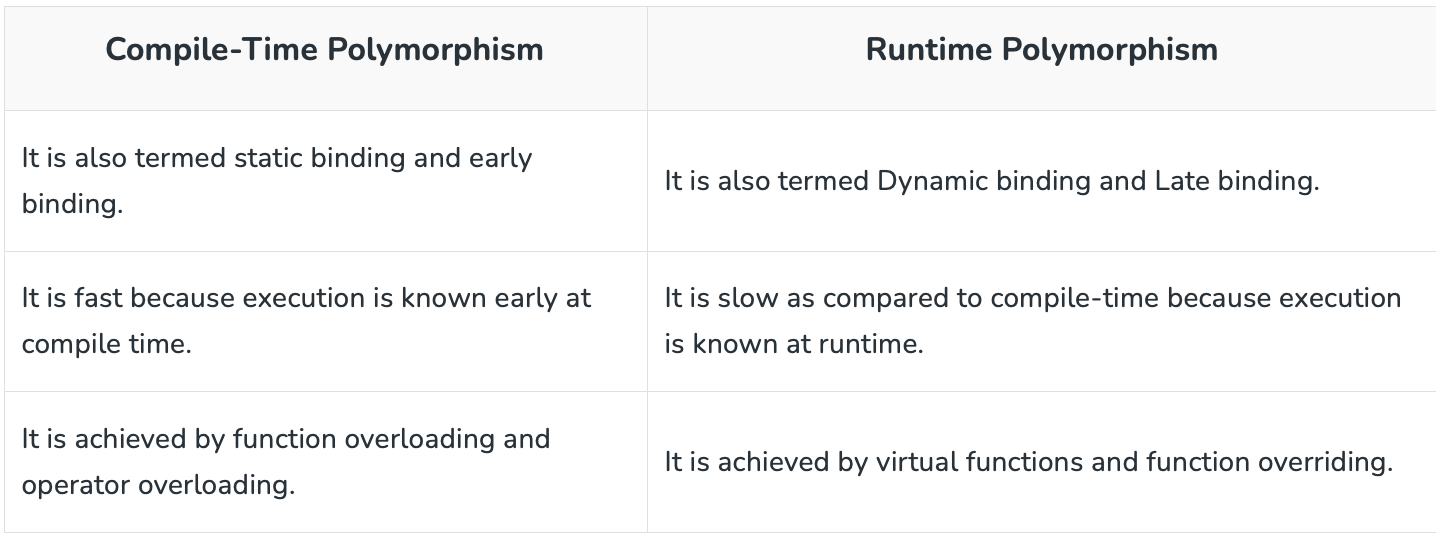
**Ans.**

There are 2 types of polymorphism:

1. **Compile Time Polymorphism or Static Binding**

This type of polymorphism is achieved during the compile time of the program which results in it making a bit faster than Run time. Also, Inheritance is not involved in it. It is comprised of ***2 further techniques***: (a) Operator Overloading (b) Function Overloading

1. **Run-Time Polymorphism or Late Binding**

****This type of polymorphism is achieved by **Function Overriding**. Late binding and dynamic polymorphism are other names for runtime polymorphism.The function call is resolved at runtime in runtime polymorphism. In contrast, with compile time polymorphism, the compiler determines which function call to bind to the object after deducing it at runtime. (a) Virtual Functions

**Q19. What is function overriding ?**

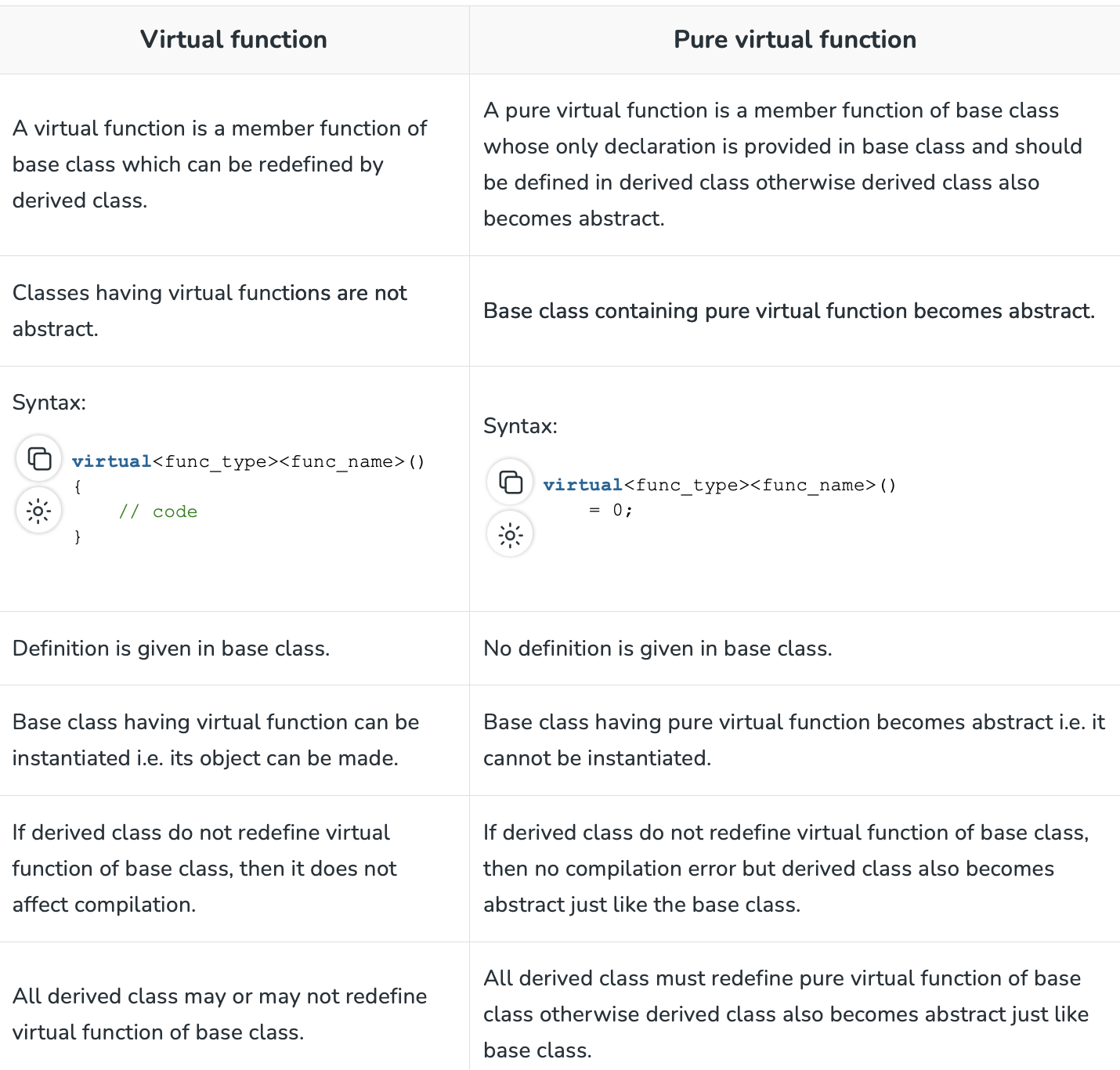
**Ans.**

When a function of the same name, same arguments or parameters, and same return type already present/declared in the base class is used in a derived class is known as Function Overriding. It is an example of Runtime Polymorphism or Late Binding which means the overridden function will be executed at the run time of the execution.

The best Real-life example of this concept is the Constitution of India. India took several features of other countries and implemented them on its own.

\*\* Cannot be executed without inheritance this is the main difference between function overloading and overriding.

**Q21. What is the difference between virtual functions and pure virtual functions?**

**Ans.**

**Q22. What is Data Abstraction? What are abstract classes?**

**Ans.**

Data Abstraction is the concept of OOP, it refers to providing only essential data to the outside world and hiding the background details. It is the concept of OOP, it refers to providing only essential data to the outside world and hiding the background details.

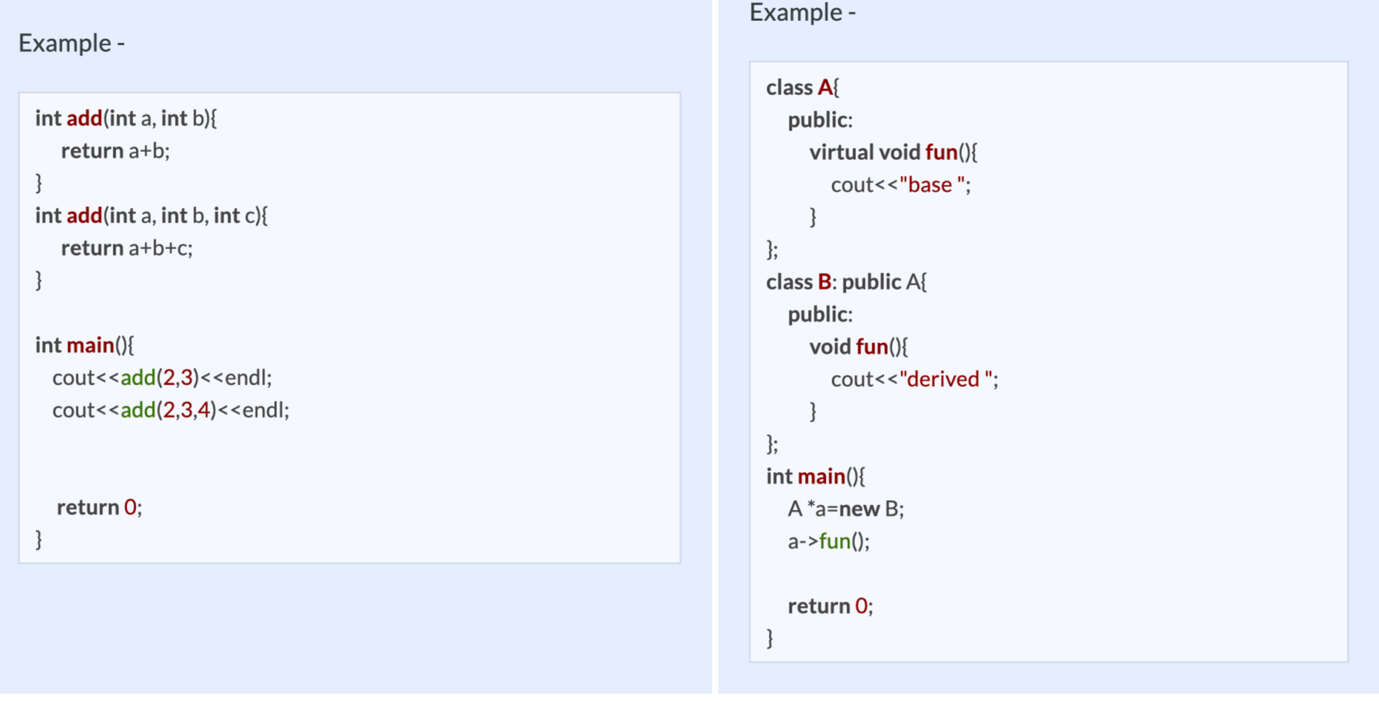
By definition, an **abstract class in C++** is a class that has at least one pure virtual function (i.e., a function that has no definition). The classes inheriting the abstract class must provide a definition for the pure virtual function; otherwise, the subclass would become an abstract class itself.

Abstract classes are essential to providing an abstraction to the code to make it reusable and extendable. For example, a *Vehicle* parent class with *Truck* and *Motorbike* inheriting from it is an abstraction that easily allows more vehicles to be added.

**Q25. What are Abstract Data Types?**

**Ans.** Abstract Data Types (ADT) 🡪 ( Meaning of abstract is existing in thought or idea but lacking concrete existence ) these are the data types that we don’t need to know the inner structure. E.g. In case of stack we use pop() and push() operations but we are totally unaware of the inner implementation. @2 a class we create with different variables and operations like insert delete display all combined represents a ADT.

\*\* A friend function can access private members of a class

****

**Q30. What is Inheritance?**

**Ans.**

Inheritance is the process of creating new classes, called derived classes, from existing classes. These existing classes are called base classes. The derived classes inherit all the capabilities of the base class but can add new features and refinements of their own.

**Virtual Inheritance**

Virtual inheritance is a technique that ensures only one copy of a base class’s member variables is inherited by grandchild-derived classes. Or in simple terms, virtual inheritance is used when we are dealing with a situation of multiple inheritances but want to prevent multiple instances of the same class from appearing in the inheritance hierarchy.

**Q34. What is Abstraction?**

**Ans.**

Abstraction means only show relevant data and details rest of others are hide. This is the most important pillar in OOP. This is mostly done by*interfaces rather than abstract class*.

Eg.1 we use functions inside header files without knowing their functionality like pow, sine etc.

Eg.2 we have different cars but ignition button will start the car we don’t need to know the real implementation of the technology used inside.

**\*\*** Interface is an abstract class but it only contains purely virtual functions means all functions in it should be initialized to 0.

**Q35. Differentiate between Abstraction and Encapsulation.**

**Ans.**

QQ

Inline functions

Friend functions