TOP 30

C++

INTERVIEW QUESTION

Created by- TOPPERWORLD

Q 1. What are the different data types present in C++?

Ans : The 4 data types in C++ are given below:

- Primitive Datatype(basic datatype). Example- char, short, int, float, long, double, bool, etc.
- Derived datatype. Example- array, pointer, etc.
- Enumeration. Example- enum
- User-defined data types. Example- structure, class, etc.

Q 2. What is the difference between C and C++?

Ans : The main difference between C and C++ are provided in the table below:

C	C++
C is a procedure-oriented programming language.	C++ is an object-oriented programming language.
C does not support data hiding.	Data is hidden by encapsulation to ensure that data structures and operators are used as intended.
C is a subset of C++	C++ is a superset of C.
Function and operator overloading are not supported in C	Function and operator overloading is supported in C++
Namespace features are not present in C	Namespace is used by C++, which avoids name collisions.
Functions can not be defined inside structures.	Functions can be defined inside structures.

C	C++
calloc() and malloc() functions are used for memory allocation and free() function is used for memory deallocation.	new operator is used for memory allocation and deletes operator is used for memory deallocation.

Q 3. What are class and object in C++?

Ans : A class is a user-defined data type that has data members and member functions. Data members are the data variables and member functions are the functions that are used to perform operations on these variables.

An object is an instance of a class. Since a class is a user-defined data type so an object can also be called a variable of that data type.

A class is defined as-

```
class A{
private:
Int data;
public:
void fun()
{
}
 };
```

Q 4. What is the difference between struct and class?

Ans : In C++ a structure is the same as a class except for a few differences like security. The difference between struct and class are given below:

Structure	Class
Members of the structure are public by default.	Members of the class are private by default.
When deriving a struct from a class/struct, default access specifiers for base class/struct are public.	When deriving a class, default access specifiers are private.

Q 5. What is operator overloading?

Ans : Operator Overloading is a very essential element to perform the operations on user-defined data types. By operator overloading we can modify the default meaning to the operators like +, -, *, /, <=, etc.

For example -

The following code is for adding two complex number using operator overloading-

```
#include <iostream>
class Complex {
private:
    double real;
    double imag;
public:
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}
    // Overloading the '+' operator to add two complex numbers
    Complex operator+(const Complex& other) const {
        return Complex(real + other.real, imag + other.imag);
    }
    // Overloading the '<<' operator for easy printing
    friend std::ostream& operator<<(std::ostream& out, const
Complex& c) {
        out << c.real << " + " << c.imag << "i";
        return out;
    }
};
int main() {
    Complex c1(2.5, 3.5);
    Complex c2(1.5, 2.5);
    Complex result = c1 + c2;
    std::cout << "c1 = " << c1 << std::endl;
    std::cout << "c2 = " << c2 << std::endl;
    std::cout << "Sum = " << result << std::endl;
    return 0;
}
```

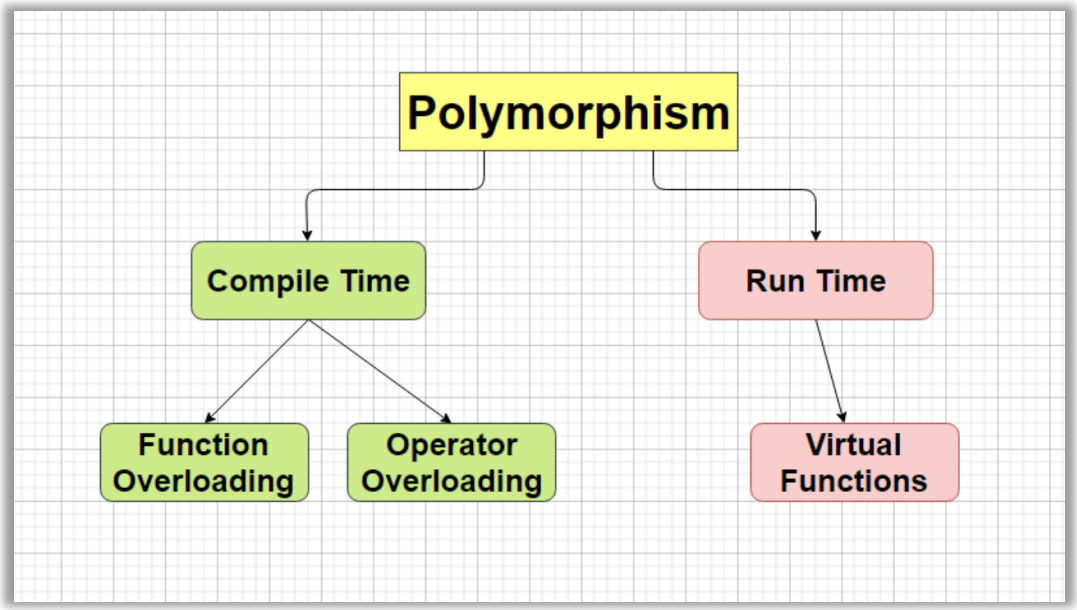
Q 6. What is polymorphism in C++?

Ans : Polymorphism in simple means having many forms. Its behavior is different in different situations. And this occurs when we have multiple classes that are related to each other by inheritance.

For example, think of a base class called a car that has a method called car brand(). Derived classes of cars could be Mercedes, BMW, Audi - And they also have their own implementation of a cars

The two types of polymorphism in c++ are:

1. Compile Time Polymorphism
2. Runtime Polymorphism



Q 7. Explain constructor in C++

Ans : The constructor is a member function that is executed automatically whenever an object is created. Constructors have the same name as the class of which they are members so that compiler knows that the member function is a constructor. And no return type is used for constructors.

```
#include <iostream>
class MyClass {
private:
    int myInt;
public:
    // Constructor with parameter
    MyClass(int val) {
        myInt = val;
        std::cout << "Constructor called with value: " << val << std::endl;
    }       // Function to display the value of myInt
    void display() {
        std::cout << "myInt = " << myInt << std::endl;
    }
};
int main() {
    MyClass obj1(10); // Constructor is called with value 10
    MyClass obj2(20); // Constructor is called with value 20
    // Displaying the values of myInt for both objects
    obj1.display(); // Output: myInt = 10
    obj2.display(); // Output: myInt = 20
    return 0;
}
```

Q 8. Tell me about virtual function

Ans : **Virtual function** is a member function in the base class that you redefine in a derived class. A virtual function is declared using the virtual keyword. When the function is made virtual, C++ determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.

Q 9. Compare compile time polymorphism and Runtime polymorphism

Ans : The main difference between compile-time and runtime polymorphism is provided below:

Compile-time polymorphism	Run time polymorphism
In this method, we would come to know at compile time which method will be called. And the call is resolved by the compiler.	In this method, we come to know at run time which method will be called. The call is not resolved by the compiler.
It provides fast execution because it is known at the compile time.	It provides slow execution compared to compile-time polymorphism because it is known at the run time.
It is achieved by function overloading and operator overloading.	It can be achieved by virtual functions and pointers.

Q 10. What do you know about friend class and friend function?

Ans : A friend class can access private, protected, and public members of other classes in which it is declared as friends.

Like friend class, friend function can also access private, protected, and public members. But, Friend functions are not member functions.

```
#include <iostream>

// Forward declaration of MyClass
class MyClass;

// Friend function declaration
void friendFunction(const MyClass&);

// Friend class declaration
class FriendClass {
public:
    void friendMethod(const MyClass&);
};
// Definition of MyClass
class MyClass {
private:
    int data = 10;
    // Declare friendFunction as a friend function
    friend void friendFunction(const MyClass&);
```

```
 // Declare FriendClass as a friend class
    friend class FriendClass;
public:
    void display() {
        std::cout << "Data: " << data << std::endl;
    }
};
// Definition of friend function
void friendFunction(const MyClass& obj) {
    std::cout << "Friend function accessing private data: " <<
obj.data << std::endl;
}
// Definition of friend class method
void FriendClass::friendMethod(const MyClass& obj) {
    std::cout << "Friend class method accessing private data: "
<< obj.data << std::endl;
}
int main() {
    MyClass obj;
    // Accessing private data using friend function
    friendFunction(obj);

    // Accessing private data using friend class method
    FriendClass fc;
    fc.friendMethod(obj);
    return 0;
}
```

Output:

```
Friend function accessing private data: 10
Friend class method accessing private data: 10
```

Q 11. What are the C++ access specifiers?

Ans : In C++ there are the following access specifiers:

- **Public:** All data members and member functions are accessible outside the class.
- **Protected:** All data members and member functions are accessible inside the class and to the derived class.
- **Private:** All data members and member functions are not accessible outside the class.

Q 12. Define inline function

Ans : If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

One of the important advantages of using an inline function is that it eliminates the function calling overhead of a traditional function.

Q 13. What is a reference in C++?

Ans : A reference is like a pointer. It is another name of an already existing variable. Once a reference name is initialized with a variable, that variable can be accessed by the variable name or reference name both.

Q 14. What do you mean by abstraction in C++?

Ans : Abstraction is the process of showing the essential details to the user and hiding the details which we don't want to show to the user or hiding the details which are irrelevant to a particular user.

Q 15. Is deconstructor overloading possible? If yes then explain and if no then why?

Ans : No destructor overloading is not possible. Destructors take no arguments, so there's only one way to destroy an object. That's the reason destructor overloading is not possible.

Q 16. What do you mean by call by value and call by reference?

Ans : In call by value method, we pass a copy of the parameter is passed to the functions.

For these copied values a new memory is assigned and changes made to these values do not reflect the variable in the main function.

In call by reference method, we pass the address of the variable and the address is used to access the actual argument used in the function call. So changes made in the parameter alter the passing argument.

Q 17. What is an abstract class and when do you use it?

Ans : A class is called an abstract class whose objects can never be created. Such a class exists as a parent for the derived classes.

We can make a class abstract by placing a pure virtual function in the class.

Q 18. What are destructors in C++?

Ans : A constructor is automatically called when an object is first created. Similarly when an object is destroyed a function called destructor automatically gets called.

A destructor has the same name as the constructor (which is the same as the class name) but is preceded by a tilde.

Example:

```
#include <iostream>

class MyClass {
public:
    // Constructor
    MyClass() {
        std::cout << "Constructor called." << std::endl;
    }

    // Destructor
    ~MyClass() {
        std::cout << "Destructor called." << std::endl;
    }
};

int main() {
    MyClass obj;
    // Destructor will be called automatically when obj goes out of scope
    return 0;
}
```

Output:

```
Constructor called.
Destructor called.
```

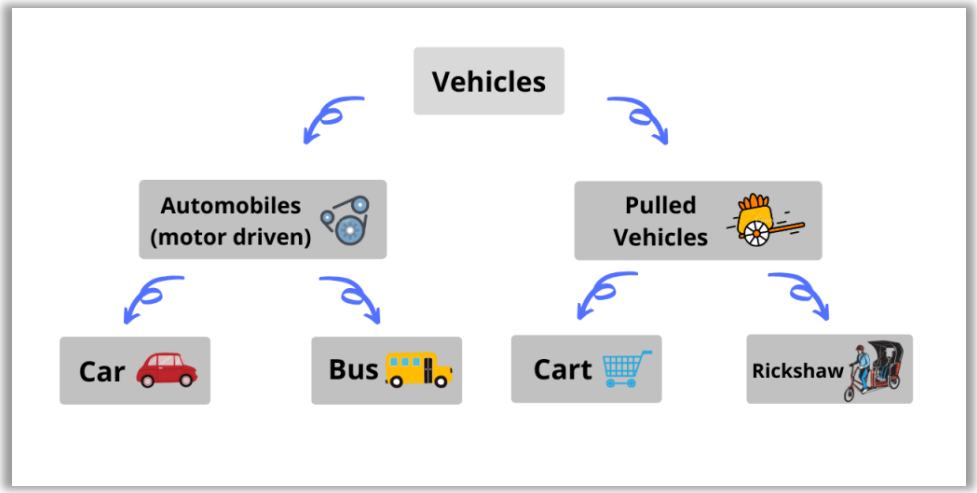
Q 19. What are the static members and static member functions?

When a variable in a class is declared static, space for it is allocated for the lifetime of the program. No matter how many objects of that class have been created, there is only one copy of the static member. So same static member can be accessed by all the objects of that class.

A static member function can be called even if no objects of the class exist and the static function are accessed using only the class name and the scope resolution operator ::

Q 20. Explain inheritance

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.



Q 21. What is a copy constructor?

Ans : A copy constructor is a member function that initializes an object using another object of the same class.

Example-

```
#include <iostream>

class MyClass {
private:
    int data;

public:
    // Constructor
    MyClass(int value) : data(value) {
        std::cout << "Constructor called." << std::endl;
    }

    // Copy constructor
    MyClass(const MyClass& other) : data(other.data) {
        std::cout << "Copy constructor called." << std::endl;
    }

    // Display function
    void display() {
        std::cout << "Data: " << data << std::endl;
    }
};
```

```
int main() {
    // Creating an object
    MyClass obj1(42);
    obj1.display(); // Output: Data: 42

    // Using copy constructor to create a new object
    MyClass obj2 = obj1;
    obj2.display(); // Output: Data: 42

    return 0;
}
```

Output:

```
Constructor called.
Data: 42
Copy constructor called.
Data: 42
```

Q 22. What is the difference between shallow copy and deep copy?

Ans : The difference between shallow copy and a deep copy is given below:

Shallow Copy	Deep Copy
Shallow copy stores the references of objects to the original memory address.	Deep copy makes a new and separate copy of an entire object with its unique memory address.

Shallow Copy	Deep Copy
Shallow copy is faster.	Deep copy is comparatively slower.
Shallow copy reflects changes made to the new/copied object in the original object.	Deep copy doesn't reflect changes made to the new/copied object in the original object

Q 23. What is the difference between virtual functions and pure virtual functions?

Ans : A virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword.

Virtual function	Pure virtual function
A virtual function is a member function of base class which can be redefined by derived class.	A pure virtual function is a member function of base class whose only declaration is provided in base class and should be defined in derived class otherwise derived class also becomes abstract.
Classes having virtual functions are not abstract.	Base class containing pure virtual function becomes abstract.

Syntax:
`virtual<func_type><func_name>()`
`{`
`  // code`
`}`
	Syntax:
	`virtual<func_type><func_name>()`
	`  = 0;`

Virtual function	Pure virtual function
Definition is given in base class.	No definition is given in base class.
Base class having virtual function can be instantiated i.e. its object can be made.	Base class having pure virtual function becomes abstract i.e. it cannot be instantiated.
If derived class do not redefine virtual function of base class, then it does not affect compilation.	If derived class do not redefine virtual function of base class, then no compilation error but derived class also becomes abstract just like the base class.
All derived class may or may not redefine virtual function of base class.	All derived class must redefine pure virtual function of base class otherwise derived class also becomes abstract just like base class.

Q 24. If class D is derived from a base class B. When creating an object of type D in what order would the constructors of these classes get called?

Ans : The derived class has two parts, a base part, and a derived part. When C++ constructs derived objects, it does so in phases. First, the most-base class(at the top of the inheritance tree) is constructed.

Then each child class is constructed in order until the most-child class is constructed last.
So the first Constructor of class B will be called and then the constructor of class D will be called.

During the destruction exactly reverse order is followed. That is destructor starts at the most-derived class and works its way down to base class.
So the first destructor of class D will be called and then the destructor of class B will be called.

Q 25. Can we call a virtual function from a constructor:

Ans : Yes, we can call a virtual function from a constructor. But the behavior is a little different in this case. When a virtual function is called, the virtual call is resolved at runtime. It is always the member function of the current class that gets called. That is the virtual machine doesn't work within the constructor.

For example-

```
#include <iostream>
class Base {
public:
    Base() {
        std::cout << "Base constructor called." << std::endl;
        // Call virtual function from constructor
        printMessage();
    }
    virtual void printMessage() {
        std::cout << "Message from Base class." << std::endl;
    }
};
class Derived : public Base {
public:
    Derived() {
        std::cout << "Derived constructor called." << std::endl;
    }
    void printMessage() override {
        std::cout << "Message from Derived class." << std::endl;
    }
};
int main() {
    Derived d;
    return 0;
}
```

Output:

```
Base constructor called.
Message from Base class.
Derived constructor called.
```

Q 26. What are void pointers?

Ans : A void pointer is a pointer which is having no datatype associated with it. It can hold addresses of any type.

For example-

```
#include <iostream>


int main() {
   int x = 10;
   float y = 3.14;
   void* ptr;
   // Assigning address of integer variable to void pointer
   ptr = &x;
   std::cout << "Value of x: " << *(static_cast<int*>(ptr)) <<
std::endl;
   // Assigning address of float variable to void pointer
   ptr = &y;
   std::cout << "Value of y: " << *(static_cast<float*>(ptr)) <<
std::endl;
   return 0;
}
```

Output:

```
Value of x: 10
Value of y: 3.14
```

Q 27. What is this pointer in C++?

Ans : The member functions of every object have a pointer named this, which points to the object itself.

The value of this is set to the address of the object for which it is called. It can be used to access the data in the object it points to.

Example :

```
#include <iostream>


class MyClass {
private:
   int data;
public:
   // Constructor
   MyClass(int value) : data(value) {}


   // Member function to display the value of data
   void display() {
      std::cout << "Data: " << this->data << std::endl;
   }
```

```
    // Member function to compare two objects
    bool isEqual(const MyClass& other) {
        return this == &other;
    }
};

int main() {
    MyClass obj1(10);
    MyClass obj2(20);

    obj1.display(); // Output: Data: 10
   obj2.display(); // Output: Data: 20

    if (obj1.isEqual(obj2)) {
        std::cout << "obj1 and obj2 are the same object." << std::endl;
    } else {
        std::cout << "obj1 and obj2 are different objects." <<
std::endl;
    }
    return 0;
}
```

Output:

```
Data: 10

Data: 20

obj1 and obj2 are different objects.
```

Q 28. What does the Scope Resolution operator do?

Ans : A scope resolution operator is denoted by a ‘:: ‘ symbol. Just like its name this operator resolves the barrier of scope in a program. A scope resolution operator is used to reference a member function or a global variable out of their scope furthermore to which it can also access the concealed variable or function in a program.

Scope Resolution is used for numerous amounts of tasks:

1) To access a global variable when there is a local variable with the same name
2) To define the function outside the class
3) In case of multiple inheritances
4) For namespace

Q 29. What is an abstract class and when do you use it?

Ans : An abstract class is a class that is specifically designed to be used as a base class. An abstract class contains at least one pure virtual function. You declare a pure virtual function by using a ***pure specifier(= 0)*** in the declaration of a virtual member function in the class declaration

You cannot use an abstract class as a parameter type, a function return type, or the type of an explicit conversion, nor can you declare an object of an abstract class. However, it can be used to declare pointers and references to an abstract class.

An abstract class is used if you want to provide a common, implemented functionality among all the implementations of the component. Abstract classes will allow you to partially implement your class, whereas interfaces

would have no implementation for any members whatsoever. In simple words, Abstract Classes are a good fit if you want to provide implementation details to your children but don't want to allow an instance of your class to be directly instantiated.

Q 30. What is the main use of the keyword "Volatile" ?

Ans : Just like its name, things can change suddenly and unexpectantly; So it is used to inform the compiler that the value may change anytime. Also, the volatile keyword prevents the compiler from performing optimization on the code. It was intended to be used when interfacing with memory-mapped hardware, signal handlers, and machine code instruction.

➢ Unleash Your Potential

- In an ever-evolving world, the pursuit of knowledge is essential. ToppperWorld serves as your virtual campus, where you can explore a diverse array of online resources tailored to your specific college curriculum.
- Whether you're studying science, arts, engineering, or any other discipline, we've got you covered.
- Our platform hosts a vast library of e-books, quizzes, and interactive study tools to ensure you have the best resources at your fingertips.

➢ The ToppperWorld Community

- Education is not just about textbooks and lectures; it's also about forming connections and growing together.
- ToppperWorld encourages you to engage with your fellow students, ask questions, and share your knowledge.
- We believe that collaborative learning is the key to academic success.

➢ Start Your Journey with ToppperWorld

- Your journey to becoming a top-performing college student begins with ToppperWorld.
- Join us today and experience a world of endless learning possibilities.
- Together, we'll help you reach your full academic potential and pave the way for a brighter future.
- Join us on this exciting journey, and let's make academic success a reality for every college student.

"UNLOCK YOUR POTENTIAL"

With- TOPPERWORLD

Explore More

topperworld.in

DSA TUTORIAL

C TUTORIAL

C++ TUTORIAL

JAVA TUTORIAL

PYTHON TUTORIAL

Follow Us On

E-mail

topperworld.in@gmail.com