C++ Interview Questions and Answers (2025)

Last Updated : 03 Jun, 2025

C++ - the must-known and all-time favourite programming language of coders. It is still relevant as it was in the mid-80s. As a general-purpose and object-oriented programming language is extensively employed mostly every time during coding. As a result, some job roles demand individuals be fluent in C++. It is utilized by top IT companies such as **Evernote, LinkedIn, Microsoft, Opera, NASA**, and **Meta** because of its dependability, performance, and wide range of settings in which it may be used. So, to get into these companies, you need to be thorough in these **top 50 C++ interview questions** which can make you seem like an expert in front of recruiters.

To make you interview-ready, we have brought the **Top 50 C++ interview questions for beginner, intermediate and experienced** which you must definitely go through in order to get yourself placed at top MNCs. The **C++ Course** includes a section dedicated to interview questions and answers, helping you get ready for your upcoming interviews with confidence.

C++ Interview Questions for Freshers

1. What is C++? What are the advantages of C++?

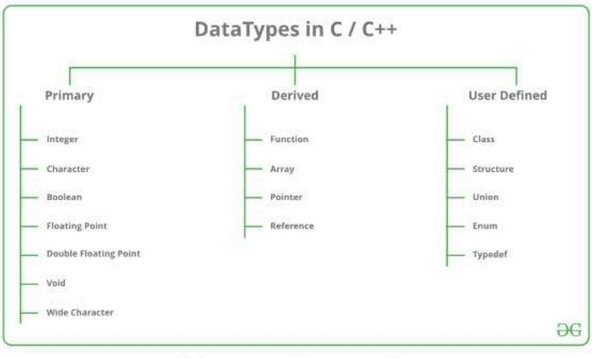
C++ is an object-oriented programming language that was introduced to overcome the jurisdictions where C was lacking. By object-oriented we mean that it works with the concept of *polymorphism*, *inheritance*, *abstraction*, *encapsulation*, *object, and class*.

Advantages of C++:

1. C++ is an OOPs language that means the data is considered as objects.
2. C++ is a multi-paradigm language; In simple terms, it means that we can program the logic, structure, and procedure of the program.
3. Memory management is a key feature in C++ as it enables dynamic memory allocation
4. It is a Mid-Level programming language which means it can develop games, desktop applications, drivers, and kernels

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

Following is the list of data types in C++:



Different types of data types in C++

3. Define 'std'?

'std' is also known as Standard or it can be interpreted as a namespace. The command "*using namespace std*" informs the compiler to add everything under the *std namespace* and inculcate them in the *global namespace*. This all inculcation of global namespace benefits us to use "**cout**" and "**cin**" without using "**std::_operator_**".

4. What are references in C++?

In C++, references are an alternative way to create an alias for another variable. A reference acts as a synonym for a variable, allowing you to access the variable directly without any additional syntax. They must be initialized when created and cannot be changed to refer to another variable afterward. This feature makes it easier to manipulate variables in functions while avoiding the overhead of copying large objects. A reference variable is preceded with a **'&'** symbol.

Syntax:

and "cin" without using "std::_operator_".

4. What are references in C++?

In C++, references are an alternative way to create an alias for another variable. A reference acts as a synonym for a variable, allowing you to access the variable directly without any additional syntax. They must be initialized when created and cannot be changed to refer to another variable afterward. This feature makes it easier to manipulate variables in functions while avoiding the overhead of copying large objects. A reference variable is preceded with a **'&'** symbol.

Syntax:

```
int GFG = 10;

// reference variable
int& ref = GFG;
```

5. What do you mean by Call by Value and Call by Reference?

In this programming language to call a function we have 2 methods: **Call by Value** and **Call by Reference**

Call by Value	Call by Reference
A copy of a variable is passed.	A variable itself is passed fundamentally.
Calling a function by sending the values by copying variables.	Calling a function by sending the address of the passed variable.
The changes made in the function are never reflected outside the function on the variable. In short, the original value is never altered in Call by Value.	The changes made in the functions can be seen outside the function on the passed function. In short, the original value is altered in Call by reference.
Passed actual and formal parameters are stored in different memory locations. Therefore, making Call by Value a little memory inefficient.	Passed actual and formal parameters are stored in the same memory location. Therefore, making Call by Reference a little more memory efficient.

6. Define token in C++

A token is the smallest individual element of a program that is understood by a compiler. A token comprises the following:

1. **Keywords** - That contain a special meaning to the compiler
2. **Identifiers** - That hold a unique value/identity
3. **Constants** - That never change their value throughout the program
4. **Strings** - That contains the homogenous sequence of data
5. **Special Symbols** - They have some special meaning and cannot be used for another purpose; eg: **[] () {}, ; * = #**
6. **Operators** - Who perform operations on the operand

7. What is the difference between C and C++?

The following table lists the major differences between C and C++:

C	C++
It is a procedural programming language. In simple words, it does not support classes and objects	It is a mixture of both procedural and object-oriented programming languages. In simple words, it supports classes and objects.
It does not support any OOPs concepts like polymorphism, data abstraction, encapsulation, classes, and objects.	It supports all concepts of data
It does not support Function and Operator Overloading	It supports Function and Operator Overloading respectively
It is a function-driven language	It is an object-driven language

8. What is the difference between struct and class?

It is a function-driven language	It is an object-driven language

8. What is the difference between struct and class?

Following table lists the primary difference between struct and class:

Aspect	struct	class
Default Access Modifier	Members are **public** by default.	Members are **private** by default.
Memory Allocation	Can be allocated on the **stack** or **heap**.	Can be allocated on the **stack** or **heap**.
Inheritance	Supports inheritance (with public, protected, or private access).	Supports inheritance (with public, protected, or private access).
Use Case	Often used for Plain Old Data (POD) structures, or simple data grouping.	Suitable for complex objects that may include methods, constructors, and destructors.

9. What is the difference between reference and pointer?

Following are the main difference between reference and pointer:

Reference	Pointer
The value of a reference cannot be reassigned	The value of a pointer can be reassigned
It can never hold a *null* value as it needs an existing value to become an alias of	It can hold or point at a *null* value and be termed as a *nullptr* or *null pointer*
To access the members of class/struct it uses a ' . '	To access the members of class/struct it uses a ' -> '
The memory location of reference can be accessed easily or it can be used directly	The memory location of a pointer cannot be accessed easily as we have to use a dereference ' * '

10. What is the difference between function overloading and operator overloading?

Following is the main difference operator overloading and function overloading:

Function Overloading	Operator Overloading
It is basically defining a function in numerous ways such that there are many ways to call it or in simple terms you have multiple versions of the same function	It is basically giving practice of giving a special meaning to the existing meaning of an operator or in simple terms redefining the pre-redefined meaning
Parameterized Functions are a good example of Function Overloading as just by changing the argument or parameter of a function you make it useful for different purposes	Polymorphism is a good example of an operator overloading as an object of allocations class can be used and called by different classes for different purposes
Example of Function Overloading: 1. int GFG(int X, int Y); 2. int GFG(char X, char Y);	Example of Operator Overloading: 1. int GFG() = X() + Y(); 2. int GFG() = X() - Y();

11. What is the difference between an array and a list?

The major differences between the arrays and lists are:

Arrays	Lists
Array are contiguous memory locations of homogenous data types stored in a fixed location or size.	Lists are classic individual elements that are linked or connected to each other with the help of pointers and do not have a fixed size.
Arrays are static in nature.	Lists are dynamic in nature.

11. What is the difference between an array and a list?

The major differences between the arrays and lists are:

Arrays	Lists
Array are contiguous memory locations of homogenous data types stored in a fixed location or size.	Lists are classic individual elements that are linked or connected to each other with the help of pointers and do not have a fixed size.
Arrays are static in nature.	Lists are dynamic in nature
Uses less memory than linked lists.	Uses more memory as it has to store the value and the pointer memory location

12. What is the difference between a while loop and a do-while loop?

Following are the major difference between while and do-while loop:

While Loop	do-while Loop
While loop is also termed an entry-controlled loop	The do-while loop is termed an exit control loop
If the condition is not satisfied the statements inside the loop will not execute	Even if the condition is not satisfied the statements inside the loop will execute for at least one time
Example of a while loop: while(condition) {statements to be executed;};	**Example** of a do-while loop: do { statements to be executed; } while(condition or expression);

13. Discuss the difference between prefix and postfix?

Following are the major difference between prefix and postfix:

prefix	postfix
It simply means putting the operator before the operand	It simply means putting the operator after the operand
It executes itself before '; '	It executes itself after '; '
Associativity of prefix ++ is right to left	Associativity of postfix ++ is left to right

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

Following are the major difference between new and malloc()

new	malloc()
new is an operator which performs an operation	malloc is a function that returns and accepts values
new calls the constructors	malloc cannot call a constructor
new is faster than malloc as it is an operator	malloc is slower than new as it is a function
new returns the exact data type	malloc returns void*

15. What is the difference between virtual functions and pure virtual functions?

Following are the major difference between virtual functions and pure virtual functions

15. What is the difference between virtual functions and pure virtual functions?

Following are the major difference between virtual functions and pure virtual functions

Virtual Function	Pure Virtual Function
A Virtual Function is a member function of a base class that can be redefined in another derived class.	A Pure Virtual Function is a member function of a base class that is only declared in a base class and defined in a derived class to prevent it from becoming an abstract class.
A virtual Function has its definition in its respective base class.	There is no definition in Pure Virtual Function and is initialized with a pure specifier **(= 0).**
The base class has a virtual function that can be represented or instanced; In simple words, its object can be made.	A base class having pure virtual function becomes abstract that cannot be represented or instanced; In simple words, it means its object cannot be made.

16. What are classes and objects in C++?

A class is a user-defined data type where all the member functions and data members are tailor-made according to demands and requirements in addition to which these all can be accessed with the help of an **object**. To declare a user-defined data type we use a keyword ***class.***

An object is an instance of a class and an entity with value and state; In simple terms, it is used as a catalyst or to represent a class member. It may contain different parameters or none.

> ***Note:*** *A class is a blueprint that defines functions which are used by an object.*

17. What is Function Overriding?

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.

18. What are the various OOPs concepts in C++?

Following are the OOPs concepts in C++:

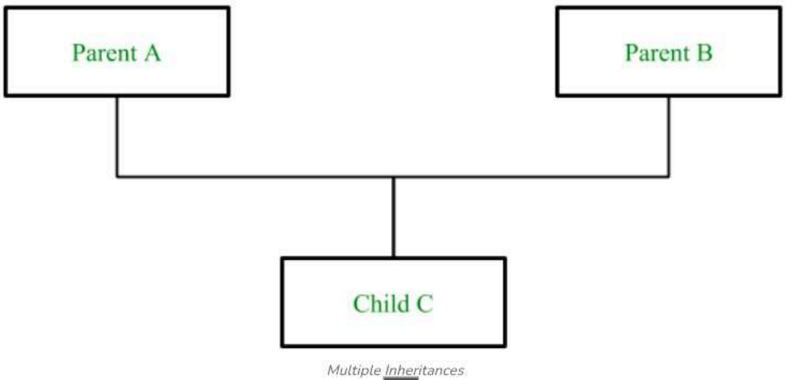
- ***Classes***: It is a user-defined datatype
- ***Objects***: It is an instance of a class
- **Abstraction:** It is a technique of showing only necessary details
- **Encapsulation:** Wrapping of data in a single unit
- **Inheritance:** The capability of a class to derive properties and characteristics from another class
- **Polymorphism:** Polymorphism is known as many forms of the same thing

19. Explain inheritance

The capability or ability of a class to derive properties and characteristics from another class is known as inheritance. In simple terms, it is a system or technique of reusing and extending existing classes without modifying them.

20. When should we use multiple inheritance?

Multiple inheritances mean that a derived class can inherit two or more base/parent classes. It is useful when a derived class needs to combine numerous attributes/contracts and inherit some, or all, of the implementation from these attributes/contracts. To take a real-life example consider your Parents where Parent A is your DAD Parent B is your MOM and Chid C is you.



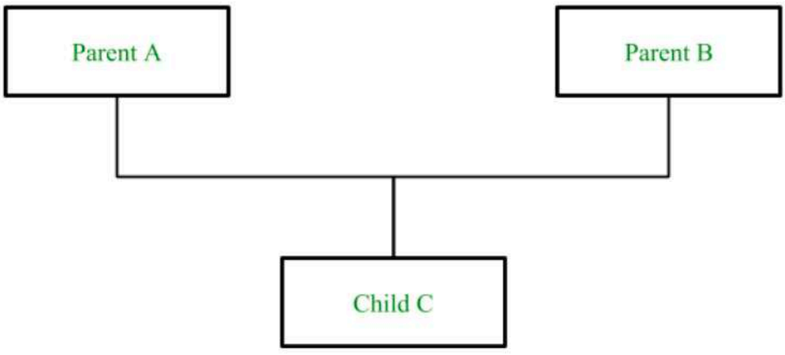
Multiple Inheritances

21. What is virtual inheritance?

classes without modifying them.

20. When should we use multiple inheritance?

Multiple inheritances mean that a derived class can inherit two or more base/parent classes. It is useful when a derived class needs to combine numerous attributes/contracts and inherit some, or all, of the implementation from these attributes/contracts. To take a real-life example consider your Parents where Parent A is your DAD Parent B is your MOM and Chid C is you.



Multiple Inheritances

21. What is 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.

22. What is polymorphism in C++?

Polymorphism is known as many forms of the same thing. In simple terms, we can say that Polymorphism 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.

There is 2 type of polymorphism:

1. ***Compile Time Polymorphism***
   - Function Overloading
   - Operator Overloading

2. ***Run Time Polymorphism***
   - Function Overriding
   - Virtual Function

23. What are the different types of polymorphism in C++?

There is 2 type of polymorphism

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***:

Function Overloading: When there are multiple functions with the same name but different parameters then this is known as function overloading.

```
// same name different arguments
int GFG() {}
int GFG(int a) {}
float GFG(double a) {}
int GFG(int a, double b) {}
```

Operator Overloading: It is basically giving practice of giving a special meaning to the existing meaning of an operator or in simple terms redefining the pre-redefined meaning

```
class GFG {
    // private and other modes
    statements public returnType
    operator symbol(arguments){ statements } statements
};
```

Run-Time Polymorphism or Late Binding

Run-time polymorphism takes place when [illegible] invoked during run time.

```
int GFG() {}
int GFG(int a) {}
float GFG(double a) {}
int GFG(int a, double b) {}
```

Operator Overloading: It is basically giving practice of giving a special meaning to the existing meaning of an operator or in simple terms redefining the pre-redefined meaning

```
class GFG {
    // private and other modes
    statements public returnType
    operator symbol(arguments){ statements } statements
};
```

Run-Time Polymorphism or Late Binding

Run-time polymorphism takes place when functions are invoked during run time.

Function Overriding: Function overriding occurs when a base class member function is redefined in a derived class with the same arguments and return type.

```
// C++ program to demonstrate
// Function overriding
#include
<iostream> using namespace std;
class GFG {
public:
    virtual void display()
    {
        cout << "Function of base class" << endl;
    }
};
class derived_GFG : public GFG {
public:
    void display()
    {
        cout << "Function of derived class" << endl;
    }
};
int main()
{
    derived_GFG dg;
    dg.display();
    return 0;
}
```

Output:

```
Function of derived class
```

24. Compare compile-time polymorphism and Runtime polymorphism

Following are the major differences between the runtime and compile time polymorphism:

Compile-Time Polymorphism	Runtime Polymorphism
It is also termed static binding and early binding.	It is also termed Dynamic binding and Late binding.
It is fast because execution is known early at compile time.	It is slow as compared to compile-time because execution is known at runtime.
It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and function overriding.

25. Explain the constructor in C++.

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.

There are 3 types of constructors:

A. 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.

Example:

```
class Class_name {
public:
    Class_name() { cout << "I am a default constructor"; }
};
```

25. Explain the constructor in C++.

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.

There are 3 types of constructors:

A. 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.

Example:

```
class Class_name {
public:
    Class_name() { cout << "I am a default constructor"; }
};
```

B. 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.

Also, It is used to overload constructors.

Example:

```
// CPP program to demonstrate
// parameterized constructors
#include
<iostream> using namespace std;
class GFG {
private:
    int x, y;

public:
    // Parameterized Constructor
    GFG(int x1, int y1)
    {
        x = x1;
        y = y1;
    }
    int getX() { return x; }
    int getY() { return y; }
};
int main()
{
    // Constructor called
    GFG G(10, 15);
    // Access values assigned by constructor
    cout << "G.x = " << G.getX() << ", G.y = " << G.getY();
    return 0;
}
```

Output

```
G.x = 10, G.y = 15
```

C. 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.

Example:

```
Sample(Sample& t) { id = t.id; }
```

26. What are destructors in C++?

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.

Syntax:

```
~constructor_name(); // tilde sign signifies that it is a destructor
```

27. What is a virtual destructor?

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

```
~constructor_name(); // tilde sign signifies that it is a destructor
```

27. What is a virtual destructor?

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.

28. Is destructor overloading possible? If yes then explain and if no then why?

The simple answer is **NO** we cannot overload a destructor. It is mandatory to only destructor per class in C++. Also to mention, Destructor neither take arguments nor they have a parameter that might help to overload.

C++ Interview Questions - Intermediate Level

29. Which operations are permitted on pointers?

Pointers are the variables that are used to store the address location of another variable. Operations that are permitted to a pointer are:

1. Increment/Decrement of a Pointer
2. Addition and Subtraction of integer to a pointer
3. Comparison of pointers of the same type

30. What is the purpose of the "*delete*" operator?

The delete operator is used to delete/remove all the characteristics/properties from an object by deallocating its memory; furthermore, it returns true or false in the end. In simple terms, it destroys or deallocates array and non-array(pointer) objects which are created by new expressions.

```
int GFG = new int[100];
// uses GFG for deletion
delete[] GFG;
```

31. How delete [] is different from delete?

delete[]	delete
It is used for deleting a whole array	It is used to delete only one single pointer
It is used for deleting the objects of **new[];** By this, we can say that **delete[]** is used to delete an array of objects	It is used for deleting the objects of **new;** By this, we can say that **delete** is used to delete a single object
It can call as many destructors it wants	It can only call the destructor of a class once

32. What do you know about friend class and friend function?

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.

Example of friend class:

```
class Class_1st {
    // ClassB is a friend class of ClassA
    friend class Class_2nd;
    statements;
} class Class_2nd {
    statements;
}
```

It can call as many destructors it wants	It can only call the destructor of a class once

32. What do you know about friend class and friend function?

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.

Example of friend class:

```
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.

Example of friend function:

```
class GFG {
    statements;
    friend datatype function_Name(arguments);
    statements;
} OR class GFG {
    statements' friend int divide(10, 5);
    statements;
}
```

33. What is an Overflow Error?

Overflow Error occurs when the number is too large for the data type to handle. In simple terms, it is a type of error that is valid for the defined but exceeds used the defined range where it should coincide/lie.

For example, the range of int data type is **–2,147,483,648** to **2,147,483,647** and if we declare a variable of size **2,247,483,648** it will generate a overflow error.

34. What does the Scope Resolution operator do?

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

35. What are the C++ access modifiers?

The access restriction specified to the class members (whether it is member function or data member) is known as access modifiers/specifiers.

Access Modifiers are of 3 types:

1. **Private -** It can neither be accessed nor be viewed from outside the class
2. **Protected -** It can be accessed if and only if the accessor is the derived class
3. **Public -** It can be accessed or be viewed from outside the class

36. Can you compile a program without the main function?

Yes, it is absolutely possible to compile a program without a main(). For example Use Macros that defines the main

```
// C++ program to demonstrate the
// a program without main()
#include
<stdio.h>
#define fun main
int fun(void)
{
```

3. **Public -** It can be accessed or be viewed from outside the class

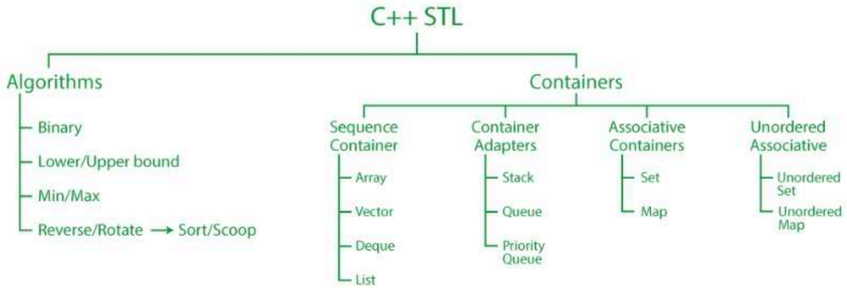
36. Can you compile a program without the main function?

Yes, it is absolutely possible to compile a program without a main(). For example Use Macros that defines the main

```
// C++ program to demonstrate the
// a program without main()
#include
<stdio.h>
#define fun main
    int fun(void)
{
    printf("Geeksforgeeks");
    return 0;
}
```

37. What is STL?

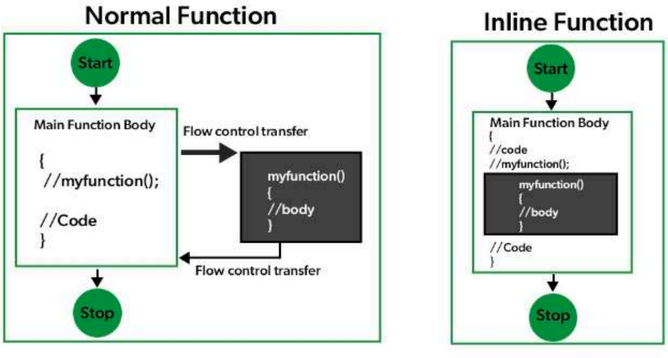
STL is known as Standard Template Library, it is a library that provides 4 components like container, algorithms, and iterators.



C++ STL

38. Define inline function. Can we have a recursive inline function in C++?

An inline function is a form of request not an order to a compiler which results in the inlining of our function to the main function body. An inline function can become overhead if the execution time of the function is less than the switching time from the caller function to called function. To make a function inline use the keyword **inline** before and define the function before any calls are made to the function.



Inline Function Explanation

Syntax:

```
inline data_type function_name()
{
Body;
}
```

The answer is **No;** It cannot be recursive.

An inline function cannot be recursive because in the case of an inline function the code is merely placed into the position from where it is called and does not maintain a piece of information on the stack which is necessary for recursion.

Plus, if you write an inline keyword in front of a recursive function, the compiler will automatically ignore it because the inline is only taken as a suggestion by the compiler.

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

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. ▲ declare a pure virtual function by using a ***pure specifier*** (= 0) in the declaration of a virtual function in the class declaration.

Plus, if you write an inline keyword in front of a recursive function, the compiler will automatically ignore it because the inline is only taken as a suggestion by the compiler.

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

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.

40. What are the static data members and static member functions?

The static data member of a class is a normal data member but preceded with a static keyword. It executes before main() in a program and is initialized to 0 when the first object of the class is created. It is only visible to a defined class but its scope is of a lifetime.

Syntax:

```
static Data_Type Data_Member;
```

The static member function is the member function that is used to access other static data members or other static member functions. It is also defined with a static keyword. We can access the static member function using the class name or class objects.

Syntax:

```
classname::function name(parameter);
```

C++ Interview Questions - Expert Level

41. What is the main use of the keyword “Volatile”?

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.

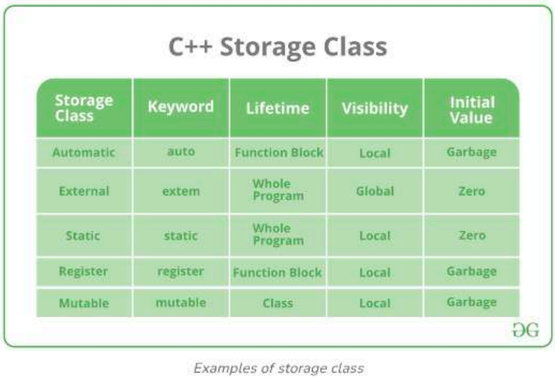
42. Define storage class in C++ and name some

Storage class is used to define the features(lifetime and visibility) of a variable or function. These features usually help in tracing the existence of a variable during the runtime of a program.

Syntax:

```
storage_class var_data_type var_name;
```

Some types of storage classes:



Examples of storage class

43. What is a mutable storage class specifier? How can they be used?

Just like its name, the mutable storage class specifier is used only on a class data member to make it modifiable even though the member is part of an object declared as const. Static or const, or reference members cannot use the mutable specifier. When we declare a function as const, this pointer passed to the function becomes const.

Examples of storage class

43. What is a mutable storage class specifier? How can they be used?

Just like its name, the mutable storage class specifier is used only on a class data member to make it modifiable even though the member is part of an object declared as const. Static or const, or reference members cannot use the mutable specifier. When we declare a function as const, this pointer passed to the function becomes const.

44. Define the Block scope variable.

So the scope of a variable is a region where a variable is accessible. There are two scope regions, A global and block or local.

A block scope variable is also known as a local scope variable. A variable that is defined inside a function (like main) or inside a block (like loops and if blocks) is a local variable. It can be used ONLY inside that particular function/block in which it is declared. a block-scoped variable will not be available outside the block even if the block is inside a function.

45. What is the function of the keyword "Auto"?

The auto keyword may be used to declare a variable with a complex type in a straightforward fashion. You can use auto to declare a variable if the initialization phrase contains templates, pointers to functions, references to members, etc. With type inference capabilities, we can spend less time having to write out things the compiler already knows. As all the types are deduced in the compiler phase only, the time for compilation increases slightly but it does not affect the runtime of the program.

46. Define namespace in C++.

Namespaces enable us to organize named items that would otherwise have global scope into smaller scopes, allowing us to give them namespace scope. This permits program parts to be organized into distinct logical scopes with names. The namespace provides a place to define or declare identifiers such as variables, methods, and classes.

Or we could say that A namespace is a declarative zone that gives the identifiers (names of types, functions, variables, and so on) within it a scope. Namespaces are used to arrange code into logical categories and to avoid name clashes, which might happen when you have many libraries in your code base.

47. When is void() return type used?

The void keyword, when used as a function return type, indicates that the function does not return a value. When used as a parameter list for a function, void indicates that the function takes no parameters. Non-Value Returning functions are also known as void functions. They're called "void" since they're not designed to return anything. True, but only partially. We can't return values from void functions, but we can certainly return something. Although void functions have no return type, they can return values.

48. What is the difference between shallow copy and deep copy?

Following are the primary differences between the shallow copy VS deep copy:

Shallow Copy	Deep Copy
In Shallow copy, a copy of the original object is stored and only the reference address is finally copied. In simple terms, Shallow copy duplicates as little as possible	In Deep copy, the copy of the original object and the repetitive copies both are stored. In simple terms, Deep copy duplicates everything
A shallow copy of a collection is a copy of the collection structure, not the elements. With a shallow copy, two collections now share individual elements.	A deep copy of a collection is two collections with all of the elements in the original collection duplicated.
A shallow copy is faster	Deep copy is comparatively slower.

49. Can we call a virtual function from a constructor?

Yes, we can call a virtual function from a constructor. But it can throw an exception of overriding.

50. What are void pointers?

functions, variables, and so on) within it a scope. Namespaces are used to arrange code into logical categories and to avoid name clashes, which might happen when you have many libraries in your code base.

47. When is void() return type used?

The void keyword, when used as a function return type, indicates that the function does not return a value. When used as a parameter list for a function, void indicates that the function takes no parameters. Non-Value Returning functions are also known as void functions. They're called "void" since they're not designed to return anything. True, but only partially. We can't return values from void functions, but we can certainly return something. Although void functions have no return type, they can return values.

48. What is the difference between shallow copy and deep copy?

Following are the primary differences between the shallow copy VS deep copy:

Shallow Copy	Deep Copy
In Shallow copy, a copy of the original object is stored and only the reference address is finally copied. In simple terms, Shallow copy duplicates as little as possible	In Deep copy, the copy of the original object and the repetitive copies both are stored. In simple terms, Deep copy duplicates everything
A shallow copy of a collection is a copy of the collection structure, not the elements. With a shallow copy, two collections now share individual elements.	A deep copy of a collection is two collections with all of the elements in the original collection duplicated.
A shallow copy is faster	Deep copy is comparatively slower.

49. Can we call a virtual function from a constructor?

Yes, we can call a virtual function from a constructor. But it can throw an exception of overriding.

50. What are void pointers?

Just like its name a void pointer is a pointer that is not associated with anything or with any data type. Nevertheless, a void pointer can hold the address value of any type and can be converted from one data type to another.

Bonus Question:

What is '*this*' pointer in C++?

this pointer enables every object to have access to its own address through an essential pointer. All member functions take ***this*** pointer as an implicit argument. ***this*** pointer may be used to refer to the calling object within a member function.

- *this* pointer is used to pass an object as a parameter to another method.
- Each object gets its own copy of the data member.
- *this* pointer is used to declare indexers.

Comment More info Campus Training Program

Next Article
Top C++ STL Interview Questions and Answers

Top C++ STL Interview Questions and Answers

Last Updated : 11 Dec, 2024

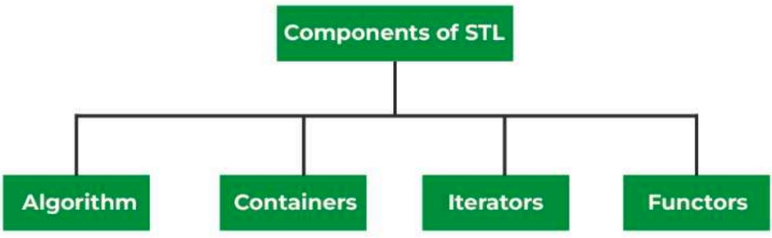
The Standard Template Library (STL) is a set of C++ template classes that are used to implement widely popular algorithms and data structures such as vectors, lists, stacks, and queues. It is part of the C++ Language ISO standard. STL is a popular topic among interviewers, so it is useful for both freshers and experienced to learn the commonly asked interview question on STL in C++.

In this article, we will see the **top 50 most important and most frequently asked interview questions on C++ STL.**

STL Interview Questions and Answers

1. What is STL?

STL stands for Standard template library and is the collection of some most commonly used algorithms and data structures such as vectors, maps, etc. It is a generalized library that works for all data types. STL has 4 components which are as follows:



For more information, refer to the article - STL in C++.

2. What is a template?

C++ gives a feature that allows functions and classes to operate with generic types in the form of templates. We only have to write the code of a function or a class once and it will be able to operate on different data types.

3. Why we use <bits/stdc++.h>?

The <bits/stdc++.h> is a header file that is used to include all the standard header files at once in the C++ program. It is mostly used in programming contests to save the time required for including header files one by one. But remember that it itself is a non-standard header file of the GNU C++ Library so not all compilers have it.

4. Why do we need STL when we can perform all the operations using a user-defined data structure and functions?

We prefer STL over user-defined data structure and functions because:

- It saves time spent writing code for user-defined data structures.
- It is tried, efficient, and debugged code tested over a long period of time.
- STL is the part of C++ Language Standard so it is easy to understand for other programmers.
- STL allows us to parameterize the code with the required data type.

5. What are containers in STL?

the C++ program. It is mostly used in programming contests to save the time required for including header files one by one. But remember that it itself is a non-standard header file of the GNU C++ Library so not all compilers have it.

4. Why do we need STL when we can perform all the operations using a user-defined data structure and functions?

We prefer STL over user-defined data structure and functions because:

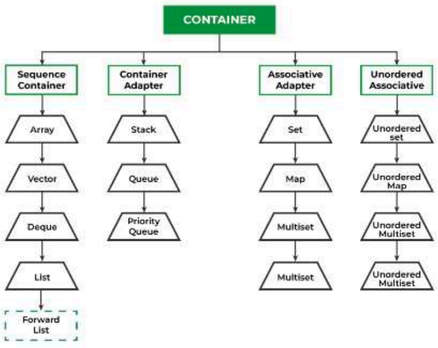
- It saves time spent writing code for user-defined data structures.
- It is tried, efficient, and debugged code tested over a long period of time.
- STL is the part of C++ Language Standard so it is easy to understand for other programmers.
- STL allows us to parameterize the code with the required data type.

5. What are containers in STL?

The Containers in STL are class templates using which we implement data structures like a queue, stack, etc. These containers act similarly to their corresponding data structure, manage the storage space for its elements and provide member functions to access them.
For Example, a Vector acts similarly to a dynamic array and has methods like push_back( ), pop_back( ), size( ), etc.

Containers are of three types:

- Sequence container
- Associative container
- Derived container



Different Types of Containers in STL

For more information, refer to the article - Containers in STL.

6. What are Algorithms in STL?

Algorithms in STL is a library that contains some commonly used methods predefined as function templates. They generally work on containers taking their iterators as arguments. They are defined inside the <algorithm> header file.

7. What are Functors in STL?

A functor (or function object) is a C++ class that acts like a function. Functors are called using the same old function called syntax. To create a functor, we overload the operator( ).

Example:

```
MyFunctor(10);
MyFunctor.operator( )(10);       //Both act same
```

8. What is a vector?

A vector is a type of container that holds the same properties as a dynamic array. It is a sequential container in which we can randomly access any element using an index number but can only insert or delete elements from the end in constant time using push_back( ) and pop_back( ) respectively.

Syntax:

```
vector<object_name> vector_name;
```

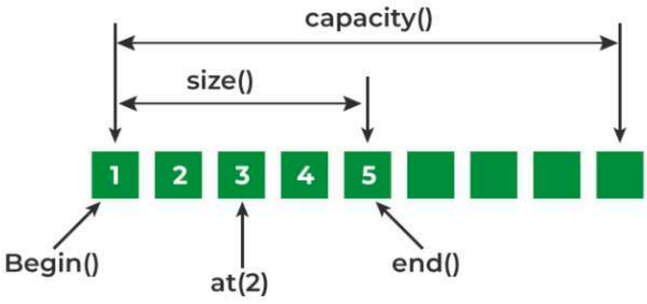
```
MyFunctor(10);
MyFunctor.operator( )(10);       //Both act same
```

8. What is a vector?

A vector is a type of container that holds the same properties as a dynamic array. It is a sequential container in which we can randomly access any element using an index number but can only insert or delete elements from the end in constant time using push_back( ) and pop_back( ) respectively.

Syntax:

```
vector<object_name> vector_name;
```



Vector in C++

9. What is an iterator?

An iterator is a variable that points to an element in an STL container and can be used to traverse through the elements in the container.

Syntax:

```
vector<int>::iterator itr;
```

Here itr is the iterator that can be used for iteration over a vector<int>.

10. What is the range in terms of vectors?

A range is a sequence of consecutive elements in a container. The range can be specified from begin( ) to end( ), containing all the elements of a container.

11. What is the difference between an array and a vector?

The differences between an array and a vector in C++ are as follows:

Array	Vector
The array is a type of data structure.	Vector is a type of Container
Its size is fixed after declaration.	Vector is dynamically resizeable that can change its size when needed
Elements of an array are stored in stack memory.	Elements of the vector are stored in the free store.
The size can be obtained by traversing.	Size can be easily determined using the size() member function.
Syntax: *data_type arr_name[size];*	**Syntax:** *vector< object_type > name;*

11. What is the difference between an array and a vector?

The differences between an array and a vector in C++ are as follows:

Array	Vector
The array is a type of data structure.	Vector is a type of Container
Its size is fixed after declaration.	Vector is dynamically resizeable that can change its size when needed
Elements of an array are stored in stack memory.	Elements of the vector are stored in the free store.
The size can be obtained by traversing.	Size can be easily determined using the size() member function.
Syntax: *data_type arr_name[size];*	**Syntax:** *vector< object_type > name;*
Example: *int arr[5];*	**Example:** *vector<int> arr;*

12. How can we insert elements in a vector?

We can insert elements using 4 methods:

- Using push_back( ) function
- Using insert() function
- Using emplace() function
- Using [ ] (array subscript)

A. Using push_back():

This method is used when we want to insert an element at the last position. The size of the vector will be increased using push_back(). It is the most used method of insertion in vectors.

Example:

```
vect.push_back(12);   //here arr is int vector and 12 is value inserted.
```

B. Using insert() function

We can also use the insert() member function to insert elements in a vector at some particular position.

Example:

```
vect.insert(vect.begin(), 20);
```

This will insert the 20 at the index 0 of the vector.

C. Using emplace() function

We can also use the emplace() member function to insert elements in a vector at some particular position in a similar way to insert() fuctions.

Example:

```
vect.emplace(vect.begin(), 20);
```

This will insert the 20 at the index 0 of the vector.

D. Using [ ] (Array Subscript):

If the size of the vector is predeclared, then we can directly insert elements using [ ] operators.

Example:

```
vect[i] =12;       // value of ith index will be 12 now.
```

13. How can we remove elements in a vector?

We can remove elements in vectors using two methods:

- Using pop_back() function
- Using erase() function

Example:

```
vect[i] =12;       // value of ith index will be 12 now.
```

13. How can we remove elements in a vector?

We can remove elements in vectors using two methods:

- Using pop_back() function
- Using erase() function

A. Using pop_back() function

pop_back() is a member function of the vector class and it is used to remove the last element from the vector.

Example:

```
vect.pop_back();   // last element will be removed.
```

B. Using erase() function

erase() is also a member function of the vector class. It is used to remove the elements at a particular position in the vector.

Example:

```
vect.erase(vect.begin() + 2);   // last element will be removed.
```

14. What is the time complexity of insertion and deletion in vector?

Insertion: If the size of the vector is N, then the time complexity of insertion of an element in the vector is:

- At the end: **O(1)**
- At M index: **O(N - M)**

Deletion: If the size of the vector is N, then the time complexity of deletion of an element in the vector is:

- At the end: **O(1)**
- At M index: **O(N-M)**

15. What is the use of auto keyword in C++?

The **auto** keyword specifies that the type of 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 the return type expression at runtime. Good use of auto is to avoid long initializations when creating iterators for containers.

16. How can we traverse a vector?

We can traverse in a vector with the following methods:

- Using Index
- Using an Iterator
- Using auto Keyword

i) Using index

We can access the elements of the vector using the index number in a similar way to the arrays.

```
for (int i = 0; i < v1.size(); i++)
{
    cout << v1[i] << " ";
}
```

ii) Using an iterator

Iterators are the objects just like pointers that point to the elements inside the containers. Iterators are used to iterate over the elements of the containers.

```
vector<int>::iterator itr;

 for (itr = v1.begin(); itr < v1.end(); itr++)
 {
     cout << *itr << " ";
 }
```

deducted from its initializer. In the case of functions, if their return type is auto then that will be evaluated by the return type expression at runtime. Good use of auto is to avoid long initializations when creating iterators for containers.

16. How can we traverse a vector?

We can traverse in a vector with the following methods:

- Using Index
- Using an Iterator
- Using auto Keyword

i) Using index

We can access the elements of the vector using the index number in a similar way to the arrays.

```
for (int i = 0; i < v1.size(); i++)
{
    cout << v1[i] << " ";
}
```

ii) Using an iterator

Iterators are the objects just like pointers that point to the elements inside the containers. Iterators are used to iterate over the elements of the containers.

```
vector<int>::iterator itr;

 for (itr = v1.begin(); itr < v1.end(); itr++)
 {
     cout << *itr << " ";
 }
```

iii) Using auto

The auto keyword specifies the type of variable that is being declared will be automatically deducted from its initializer.

```
for (auto it:v)
{
    cout << it << " ";
}
```

17. How to print vectors in C++?

We can print vectors using multiple ways:

- Using overloading << Operator
- Comma separated manner
- Using indexing
- One line without for loop
- Using (experimental::make_ostream_joiner) without providing element type
- One line using the lambda function

For more information, refer to the article - Print vector in C++.

18. How can we convert the array into a vector?

There are a few methods to convert array into a vector:

- While index iteration of array push elements
- Range-based Assignment during Initialization
- Using Inbuilt Function Insert( )
- Using Inbuilt Function Copy( )
- Using Inbuilt Function Assign( )
- Using Inbuilt Function Transform( )

For more information, refer to the article - Convert the array into a vector.

19. How can we convert vectors into arrays?

We can convert the vector into an array using multiple ways:

- By copying items one by one
- Using STL Algorithm copy()
- Using STL Algorithm transform()
- Using vector::data()

For more information, refer to the article - Convert the array into a vector.

19. How can we convert vectors into arrays?

We can convert the vector into an array using multiple ways:

- By copying items one by one
- Using STL Algorithm copy()
- Using STL Algorithm transform()
- Using vector::data()

20. How to initialize a 2-D vector in C++?

2-D vector can be initialized using multiple methods:

Syntax:

```
vector < vector<object_type> > vector_name;
```

There are some instances like:

i) If we want to insert elements during initialization

```
//v vector containing these values
vector<vector<int>> v={{1,2,3},{4,5,6},{7,8,9}};
```

ii) If the number of rows is given:

```
//rows is the number of rows
vector<vector<int>> v(rows);
```

iii) If the number of rows and columns both are given:

```
//rows is the number of rows
//cols is the number of columns
vector<vector<int>> v(rows, vector<int> (cols));
```

iv) If all the values of the vector should be initialized by x

```
//rows is the number of rows
//cols is the number of columns
vector<vector<int>> v(rows,vector<int> (cols,x));
```

21. What is the time complexity of the sorting done in vector using the sort( ) function?

STL provides the sort( ) function, which operates with the best time complexity possible that can be obtained out of every sorting algorithm. So, the time complexity is **O(N logN).**

22. What is the use of lower_bound() and upper_bound()?

STL algorithms provide the functionality to find lower and upper bounds.

- A lower bound of x is the number whose value is at least x.
- An upper bound of x is the number that comes just after x.

These functions are only effective when the vector is sorted because they use the binary search algorithm.

Example:

```
vector<int> a={2,3,4,5,6,7,8,8,10};

auto x=lower_bound(a.begin(),a.end(),5);
auto y=upper_bound(a.begin(),a.end(),5);
```

Output:

```
*x=5 , *y=6
```

23. What is a pair in STL?

Pair is a type of container that can store two elements of the same or different data types.

Syntax:

obtained out of every sorting algorithm. So, the time complexity is **O(N log N)**.

22. What is the use of lower_bound() and upper_bound()?

STL algorithms provide the functionality to find lower and upper bounds.

- A lower bound of x is the number whose value is at least x.
- An upper bound of x is the number that comes just after x.

These functions are only effective when the vector is sorted because they use the binary search algorithm.

Example:

```
vector<int> a={2,3,4,5,6,7,8,8,10};

auto x=lower_bound(a.begin(),a.end(),5);
auto y=upper_bound(a.begin(),a.end(),5);
```

Output:

```
*x=5 , *y=6
```

23. What is a pair in STL?

Pair is a type of container that can store two elements of the same or different data types.

Syntax:

```
pair<data_type,data_type> pair_name;
```

Elements of pair can be accessed by using first and second keywords. The first keyword is used to access the first element and the second keyword is used for accessing the second element.

24. Explain different methods to insert elements in a pair.

We can insert elements in a pair using three ways:

1. Directly inserting using the first and second keywords
2. Using make_pair() function
3. Using { } braces

Consider a pair:

```
pair<int,string> x;
```

i). Using first and second keywords

```
x.first = 1 ;
x.second= "Geeks";
```

ii). Using make_pair( )

```
x=make_pair(1,"Geeks");
```

iii). Using curly brackets

```
x={1, "Geeks"};
```

25. In which header file is the std::pair defined?

The std::pair class is defined inside the <**utility**> header file.

26. What is a List?

A list is a type of container in C++ that implements the doubly linked list data structure. The list provides non-contiguous storage with only sequential element access i.e we can't randomly access any element using an index number.

Syntax:

```
list <object_name> list_name;
```

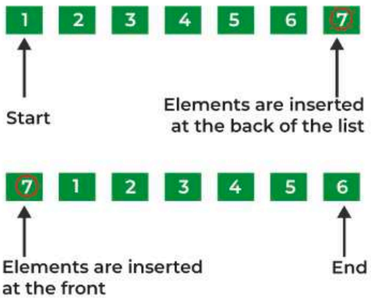
The std::pair class is defined inside the <**utility**> header file.

26. What is a List?

A list is a type of container in C++ that implements the doubly linked list data structure. The list provides non-contiguous storage with only sequential element access i.e we can't randomly access any element using an index number.

Syntax:

```
list <object_name> list_name;
```



List in C++

27. What is the time complexity of insertion and deletion in the list?

Insertion: Suppose the size of the list is N, then the time complexity of insertion of an element in the list is:

- **At the end:** O(1)
- **At the beginning:** O(1)
- **At M index:** O(M)

Deletion: Suppose the size of the list is N, then the time complexity of deletion of an element in the list is:

- **At the end:** O(1)
- **At the beginning:** O(1)
- **At M index:** O(M)

28. Difference between a vector and a list?

Vector	List
Insertion at the end requires constant time but insertion elsewhere is costly.	Insertion is cheap no matter where in the list it occurs once the element is found.
Random access to elements is possible.	Random access to elements is not possible.
It has contiguous memory.	While it has non-contiguous memory.
A vector may have a default size.	The list does not have a default size.
Iterators become invalid if elements are added to or removed from the vector.	Iterators are valid if elements are added to or removed from the list.
Syntax: vector<data_type> vector_name;	**Syntax:** list<data_type> list_name;

For more information, refer to the article - Vector vs List.

29. How can we remove elements from the list?

There are a few methods to remove elements from the list:

- Using list::erase()
- Using list::pop_front() and list::pop_back()
- Using remove() and remove_if()

For more information, refer to the article - Remove elements from the list.

30. What is a stack?

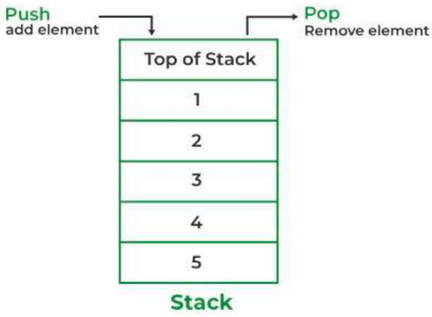
29. How can we remove elements from the list?

There are a few methods to remove elements from the list:

- Using list::erase()
- Using list::pop_front() and list::pop_back()
- Using remove() and remove_if()

For more information, refer to the article - Remove elements from the list.

30. What is a stack?



A stack is a container adapter that implements the stack data structure in STL. It has the following properties:

1. Elements follow the LIFO (Last In First Out) order of operation.
2. Only the element at the top can be accessed.
3. Insertion and Removal can be done only from the top.

Syntax:

```
stack <data_type> name;
```

Example: Chairs put one above the other so we can now either put, remove or access the chair at the top.

31. What are the basic functions associated with the STL stack?

The basic functions associated with the STL stack are as follows:

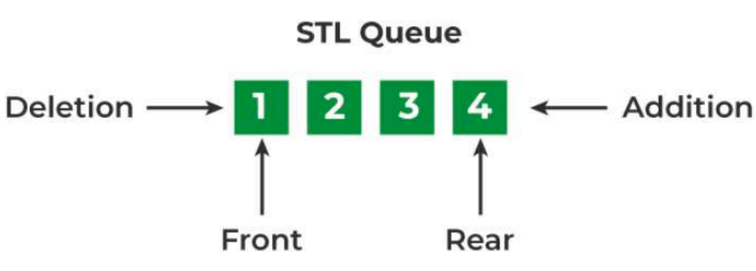
- **push()**: This function is used for inserting elements at the top of the stack.
- **pop()**: The pop() function is used for removing elements from the top.
- **size()**: It returns the size of the stack.
- **top()**: The top() function returns the top element of the stack.
- **empty()**: It checks if the stack is empty or not.

32. What is the time complexity of insertion and deletion in the stack?

Insertion and Deletion are only possible at the end of the container with the time complexity of **O(1)**. At any other position if you want to insert an element then we need to use another container or can use another stack, copy the element and remove it from the main stack then push the element at that position after then we need to push all elements again.

So, adding or removing at the M position we need to remove N-M elements and then add them again so, O(2*(N-M)) approx **O(N)**.

33. What is a queue in STL?

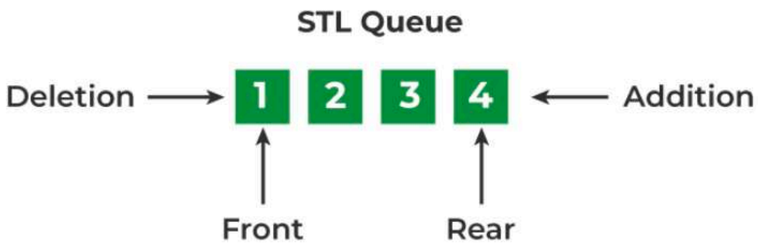


A queue in C++ STL is a container adapter that implements the queue data structure. It has the following properties:

- Follows FIFO (First In First Out) order of operation
- Queue allows insertion from one side and removal from another side.
- Insertion and removal both take O(1) time complexity.
- Only the element at the front is accessible

again so, O(2*(N-M)) approx **O(N)**.

33. What is a queue in STL?



A queue in C++ STL is a container adapter that implements the queue data structure. It has the following properties:

- Follows FIFO (First In First Out) order of operation
- Queue allows insertion from one side and removal from another side.
- Insertion and removal both take O(1) time complexity.
- Only the element at the front is accessible.

Syntax:

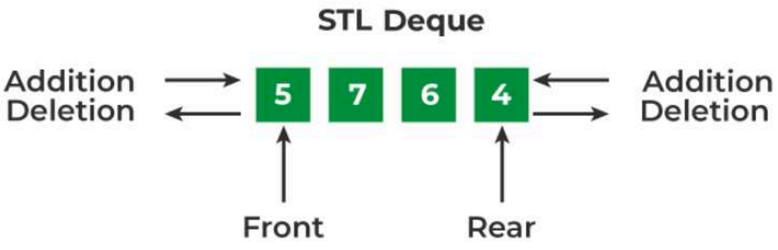
```
queue <data_type> name;
```

34. What are the commonly used member functions of the STL Queue?

Some commonly used functions associated with queue:

- **push(x)**: It is used to insert **x** in the queue.
- **pop( )**: Removes the least recent element from the list i.e. element at the front.
- **front( )**: It is used for accessing the front element.
- **size( )**: This function returns the size of the queue.

35. What is a deque?



Deque is also known as a Double-ended queue. Deque is a type of container that can insert and remove elements from both ends. It can push using push_back( ) and push_front( ) for inserting elements from the back and front respectively and can remove using pop_back( ) and pop_front( ) for removing elements from the back and front respectively.

Syntax:

```
deque <object_type> deque_name;
```

36. What is the time complexity of insertion and deletion in the deque?

Insertion and Removal of elements are possible from both sides of the deque, both insertion and deletion take O(1) time complexity from either side.

For inserting elements at index M somewhere inside the deque, the time complexity will be linear i.e O(N).

37. What is Set & How can we change the sorting order of a set?

A set is a type of associative container which stores value in a sorted way without duplication. It is sorted in increasing order by default.

Syntax:

```
set <object_type> name;
```

The set is implemented on a *Binary Search Tree (generally red-black tree)*, because of which time complexity of the elements that are stored in the set, thus the complexity of insertion, find and removal is *O(log N)*.

i.e O(N).

37. What is Set & How can we change the sorting order of a set?

A set is a type of associative container which stores value in a sorted way without duplication. It is sorted in increasing order by default.

Syntax:

```
set <object_type> name;
```

The set is implemented on a *Binary Search Tree (generally red-black tree)*, because of which time complexity of the elements that are stored in sorted form has the complexity of insertion, find and removal is *O(log N)*

For more information, refer to the article - Set in C++.

In the set, all the elements stored are sorted in increasing order, but we can change the sorting order of the set to decreasing by using a greater<int> (STL functor) as a comparator in the set declaration.

Syntax:

```
set <data_type, greater<data_type>> name;
```

Similarly, we can use any comparator function, functors, or lambda expressions in place of greater<int> for custom sorting order.

38. How to access elements in a set by index?

We can access the element at the Nth index in a set by:

- Using Iterator
- Using std::advance() method
- Using std::next() method

To know more about these methods, please refer to this article - How to Access Elements in Set by Index in C++?

39. How to iterate over the set?

We can iterate over the STL set using the following methods:

- Iterate over a set using an iterator.
- Iterate over a set in a backward direction using reverse_iterator.
- Iterate over a set using a range-based for loop.
- Iterate over a set using for_each loop.

For more information, refer to the article - Iterate over the set.

40. What is a multiset in STL?

A multiset is an associative container that acts the same way as a set but allows duplicate values.

Syntax:

```
multiset<object_type> name;
```

The multiset is also implemented using *Binary Search Tree*. The time complexity of the elements stored in sorted form has the complexity of insertion, find and removal is *O(log N)*.

For more information, refer to the article - Multiset in C++.

41. What is a unordered_set?

The unordered_set is an unordered associative container that stores values in an unsorted way without duplication.

Syntax:

```
unordered_set <object_type> name;
```

The unordered_set is implemented using the *Hash table data structure*. The time complexity of the elements stored in sorted form has the complexity of insertion, find and removal is *O(1) for average cases and O(n) time in the worst case.*

For more information, refer to the article - unordered_set in C++.

42. What is a unordered_multiset in S

For more information, refer to the article - Multiset in C++.

41. What is a unordered_set?

The unordered_set is an unordered associative container that stores values in an unsorted way without duplication.

Syntax:

```
unordered_set <object_type> name;
```

The unordered_set is implemented using the *Hash table data structure*. The time complexity of the elements stored in sorted form has the complexity of insertion, find and removal is *O(1) for average cases and O(n) time in the worst case.*

For more information, refer to the article - unordered_set in C++.

42. What is a unordered_multiset in STL?

The unordered_multiset is an unordered associative container that stores values in unsorted way and allows duplicate values.

Syntax:

```
unordered_multiset <object_type> name;
```

The unordered_multiset is based on the Hash-table data structure. The time complexity of the elements stored in sorted form has the complexity of insertion, find and removal is *O(1) for average cases and O(n) for the worst case.*

For more information, refer to the article - unordered_multiset in C++.

43. What is a map?

The map in STL is an associative container that stores the data in the form of key-value pairs. It is sorted according to keys and each key is unique.

Syntax:

```
map <object_type> name;
```

The map is generally implemented on the *Red-Black Tree (Self Balancing B.S.T.) data structure*. The time complexity for search, insert and delete operations is O(logN), where N is the number of key-value pairs in the map.

For more information, refer to the article - Map in C++.

44. What is a multimap?

A Multimap is a type of associative container that is similar to a map i.e storing key-value pairs and sorting according to keys, but the difference is that there can be multiple values associated with a single key.

Syntax:

```
multimap <object_type> name;
```

The multimap is also based on the *Balanced Binary Tree data structure*. The time complexity of the search, insert, and delete operations is logarithmic i.e. O(logN).

For more information, refer to the article - Multimap in C++.

45. What is a unordered_map?

An unordered_map is a type of unordered associative container that is similar to a map but the values are not sorted in any order.

Syntax:

```
unordered_map <object_type> name;
```

The unordered_map is based on the *Hash Table*. The time complexity of the insert, delete, and search operations is *O(1) for average cases and O(N) for the worst case.*

For more information, refer to the article - unordered_map in C++.

46. What is a unordered_multimap?

An unordered_multimap is also an unordered [illegible] that stores key-value pairs in an unsorted

An unordered_map is a type of unordered associative container that is similar to a map but the values are not sorted in any order.

Syntax:

```
unordered_map <object_type> name;
```

The unordered_map is based on the *Hash Table*. The time complexity of the insert, delete, and search operations is *O(1) for average cases and O(N) for the worst case.*

For more information, refer to the article - unordered_map in C++.

46. What is a unordered_multimap?

An unordered_multimap is also an unordered container that stores key-value pairs in an unsorted way. We can map multiple values to the same key in this container.

Syntax:

```
unordered_multimap <object_type> name;
```

The unordered_multimap is based on the *Hash Table*. The time complexity of the elements that are stored in sorted form has the complexity of insertion, find and removal is *O(1) average time and O(n) worst time.*

For more information, refer to the article - unordered_multimap in C++.

47. What is priority_queue?

A priority_queue is a container adapter that is used to create a queue whose order of operation is based on the priority of the element. The higher priority element will come out first instead of the least recent one.

It is implemented using heap data structures in C++.

Syntax:

```
priority_queue < object_type > heap_name;
```

By default, this will create a max-heap in which the largest key will pop first. For more information, refer to the article - Priority_queue in C++.

48. How to create a min-heap using STL priority_queue?

We can create a min-heap using priority_queue by the following method:

Syntax for Min heap:

```
priority_queue < object_type , vector<object_type> , greater<object_type> >
heap_name;
```

The third argument is the comparator function or functor with a boolean return type. We have used greater<> which is a built-in functor of STL for comparing two values to check which one is greater.

Another way to create a min-heap using priority_queue is to just multiply the keys by -1 before inserting them into the queue.

49. What are the basic operations on priority_queue in C++?

The basic operations that can be performed on the priority_queue are as follows:

- **push( )**: used for insertion of the element
- **pop( )**: used for removing top element
- **top( )**: used for checking either min for min-heap and max for max-heap.
- **size( )**: used for getting the size of the heap.

We can only access a top element of the heap.

50. How is priority_queue implemented in C++ STL ? What is the time complexity of basic operations in it?

The priority_queue is implemented as a heap data structure in C++. As heap is implemented using an array, the priority_queue in STL is also implemented using STL vectors which are nothing but dynamic arrays.

The time complexity of basic operations in priority_queue is as follows:

- **push( )**: O(logN)
- **pop( )**: O(logN)

We can only access a top element of the heap.

50. How is priority_queue implemented in C++ STL ? What is the time complexity of basic operations in it?

The priority_queue is implemented as a heap data structure in C++. As heap is implemented using an array, the priority_queue in STL is also implemented using STL vectors which are nothing but dynamic arrays.

The time complexity of basic operations in priority_queue is as follows:

- **push( )**: O(logN)
- **pop( )**: O(logN)
- **top( )**: O(1)
- **size( )**: O(1)

Bouns Questions

1. Write code to iterate from the first element to the last element when the vector is given

```
void iter(vector<int> v)
{
    vector<int>::iterator it1,it2;

    for(it1=v.begin(),it2=v.end(),it2;it1!=it2;it1++)
    {
        cout<<*it1<<" ";
    }

    cout<<endl;
}
```

2. Write code to sort and reverse the vector.

```
vector<int> sortVector(vector<int> v)
{
    sort(v.begin(), v.end());
    return v;
}

vector<int> reverseVector(vector<int> v)
{
    for(int i = 0, j = v.size() - 1; i < j; i++, j--)
        swap(v[i],v[j]);

    return v;
}
```

4. Declare, Insert and Print a pair of strings paired with another pair of int paired with int.

```
//Declaration of Pair
pair<string,pair<int,int>> x;

//Inserting element
x= { "Geeks" , { 1, 2} };

//Printing element
cout<<x.first<<" "<<x.second.first<<" "<<x.second.second;
```

5. Find the index of a given element in a vector, such that the next element to it is 2, and if there exists more than one such index returns the larger one if there is no such index then return -1.

```
int find_index(vector<int> &V,int a)
{
    stack<int> temp;

    for(int i=0;i<V.size();i++)
    {
        if(V[i]==a)
```

4. Declare, Insert and Print a pair of strings paired with another pair of int paired with int.

```
//Declaration of Pair
pair<string,pair<int,int>> x;

//Inserting element
x= { "Geeks" , { 1, 2} };

//Printing element
cout<<x.first<<" "<<x.second.first<<" "<<x.second.second;
```

5. Find the index of a given element in a vector, such that the next element to it is 2, and if there exists more than one such index returns the larger one if there is no such index then return -1.

```
int find_index(vector<int> &V,int a)
{
    stack<int> temp;

    for(int i=0;i<V.size();i++)
    {
        if(V[i]==a){
            temp.push(i);
        }

        if(i==temp.top()+1){
            if(V[i]==2)
                continue;
            else if(V[i]==a){
                temp.pop();
                temp.push(i);
            }
            else{
                temp.pop();
            }
        }
    }

    if(temp.top()==V.size()-1)
        temp.pop();

    if(temp.size()==0)
        return -1;

    return temp.top();
}
```

6. Write a Class stack to implement a stack using a queue.

```
class Stack {
    queue<int> q1, q2;

public:
    void push(int x)
    {
        q2.push(x);

        while (!q1.empty()) {
            q2.push(q1.front());
            q1.pop();
        }

        queue<int> q = q1;
        q1 = q2;
        q2 = q;
    }

    void pop()
    {
        if (q1.empty())
            return;
        q1.pop();
```

```
}
```

6. Write a Class stack to implement a stack using a queue.

```
class Stack {
    queue<int> q1, q2;

public:
    void push(int x)
    {
        q2.push(x);

        while (!q1.empty()) {
            q2.push(q1.front());
            q1.pop();
        }

        queue<int> q = q1;
        q1 = q2;
        q2 = q;
    }

    void pop()
    {
        if (q1.empty())
            return;
        q1.pop();
    }

    int top()
    {
        if (q1.empty())
            return -1;
        return q1.front();
    }

    int size()
    {
        return q1.size();
    }
};
```

7. Write a code to perform push from the back, pop from the front, get the size, get back and get front operations.

```
void push(queue<int> &q,int x)
{
    q.push_back(x);
}

int pop(queue<int> &q)
{
    int x=getFront(q);
    q.pop_front();

    return x;
}


int getSize(queue<int> &q)
{
    return q.size();
}

int getBack(queue<int> &q)
{
    return q.back();
}

int getFront(queue<int> &q)
{
    return q.front();
}
```

```
};
```

7. Write a code to perform push from the back, pop from the front, get the size, get back and get front operations.

```
void push(queue<int> &q,int x)
{
    q.push_back(x);
}

int pop(queue<int> &q)
{
    int x=getFront(q);
    q.pop_front();

    return x;
}


int getSize(queue<int> &q)
{
    return q.size();
}

int getBack(queue<int> &q)
{
    return q.back();
}

int getFront(queue<int> &q)
{
    return q.front();
}
```

8. Write a code to check duplicate values in the vector and print them with the best possible time complexity.

```
void dupicate(vector<int> &V)
{
    unordered_set<int> store;

    for(auto it:V)
    {
        if(store.find(it)!=store.end())
        {
            cout<<it<<" ";
        }
        else
        {
            store.insert(it);
        }
    }
}
```

Comment More info Campus Training Program

Next Article
30 OOPs Interview Questions and Answers [2025 Updated]

C++ Interview Questions For Freshers

1. What is the difference between C and C++?

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.
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.

2. Explain inheritance

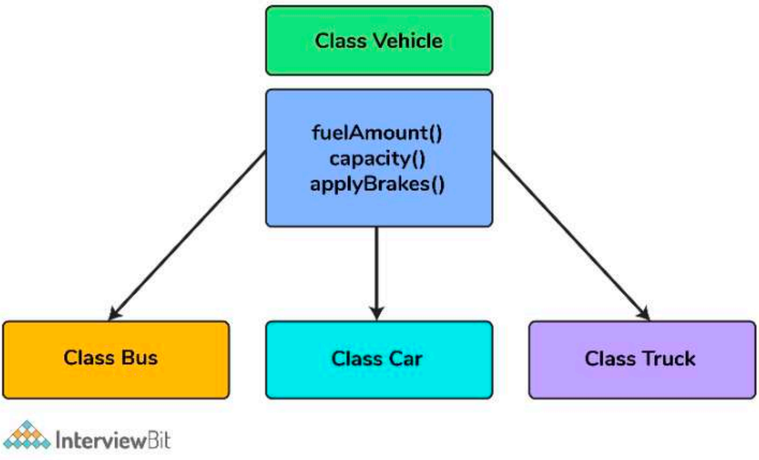
Inheritance is the process of creating new classes, called derived classes, from existing classes. These existing classes are called base classes. The derived classes inherit all the capabilities of the base class but can add new features and refinements of their own.

Example-

2. 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.

Example-



Inheritance in C++

Class Bus, Class Car, and Class Truck inherit the properties of Class Vehicle.

The most important thing about inheritance is that it permits code reusability.

3. 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 ::

4. What are destructors in C++?

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:

```
class A{
 private:
  int val;
 public:
  A(int x){
   val=x;
  }
  A(){
  }
  ~A(){          //de
  }
```

4. What are destructors in C++?

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:

```
class A{
 private:
  int val;
 public:
  A(int x){
   val=x;
  }
  A(){
  }
  ~A(){              //destructor
  }
}
int main(){
 A a(3);
 return 0;
}
```

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

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.

6. What do you mean by call by value and call by reference?

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.

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

No destructor overloading is not possible. Destructors take no arguments, so there's only one way to destroy an

access the actual argument used in the function call. So changes made in the parameter alter the passing argument.

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

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.

8. What do you mean by abstraction in C++?

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.

9. What is a reference in C++?

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.

For example-

```
int x=10;
int &ref=x;        //reference variable
```

If we change the value of ref it will be reflected in x. Once a reference variable is initialized it cannot refer to any other variable. We can declare an array of pointers but an array of references is not possible.

10. Define inline function

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.

11. What do you know about friend class and friend function?

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

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.

11. What do you know about friend class and friend function?

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.

For example -

```
class A{
 private:
  int data_a;
 public:
  A(int x){
   data_a=x;
  }
  friend int fun(A, B);
}
class B{
 private:
  int data_b;
 public:
  A(int x){
   data_b=x;
  }
  friend int fun(A, B);
}
int fun(A a, B b){
 return a.data_a+b.data_b;
}
int main(){
 A a(10);
 B b(20);
 cout<<fun(a,b)<<endl;
 return 0;
}
```

Here we can access the private data of class A and class B.

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

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.

13. What are class and object in C++?

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:
```

- User-defined data types. Example- structure, class, etc.

13. What are class and object in C++?

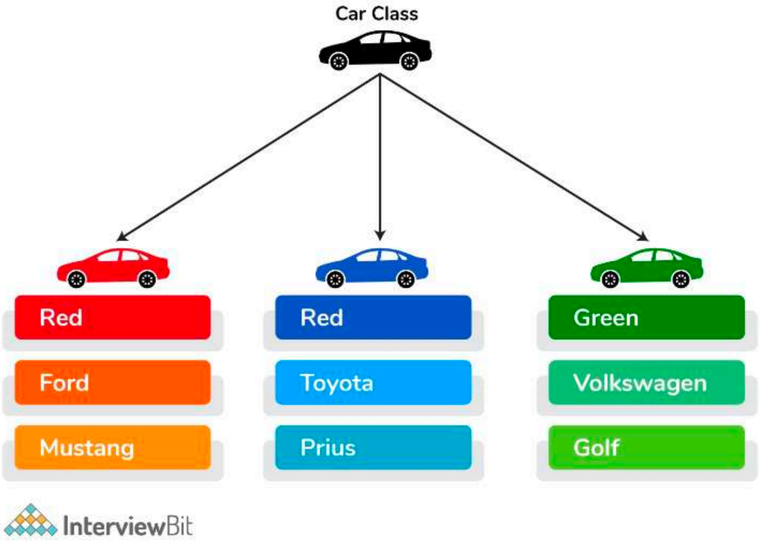
A class is a user-defined data type that has data members and member functions. Data members are the data variables and member functions are the functions that are used to perform operations on these variables.

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(){

 }
};
```



Class and Object in C++

For example, the following is a class car that can have properties like name, color, etc. and they can have methods like speed().

14. What is the difference between struct and class?

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.

15. What is operator overloading?

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 -

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.

15. What is operator overloading?

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-

```
class complex{
private:
 float r, i;
public:
 complex(float r, float i){
  this->r=r;
  this->i=i;
 }
 complex(){}
 void displaydata(){
  cout<<"real part = "<<r<<endl;
  cout<<"imaginary part = "<<i<<endl;
 }
 complex operator+(complex c){
  return complex(r+c.r, i+c.i);
 }
};
int main(){
complex a(2,3);
complex b(3,4);
complex c=a+b;
c.displaydata();
return 0;
}
```

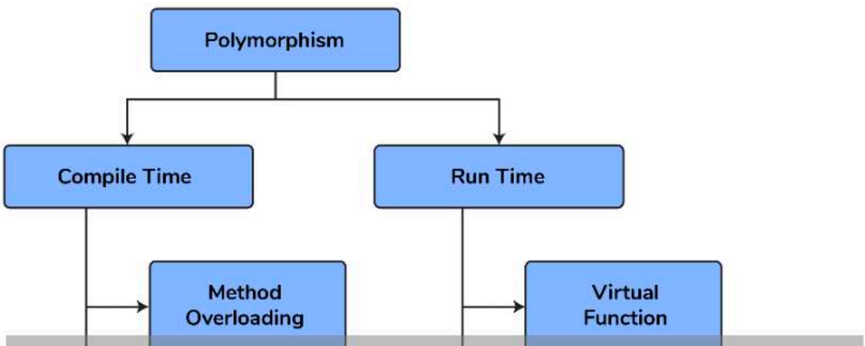
16. What is polymorphism in C++?

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

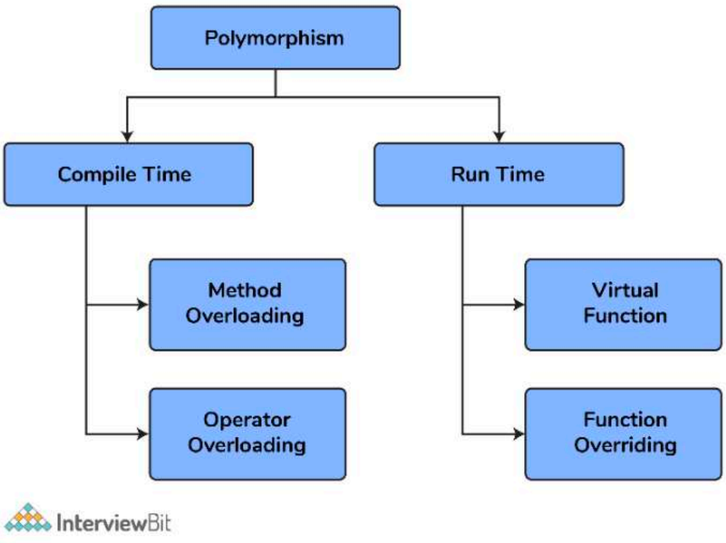
The two types of polymorphism in c++ are:

- Compile Time Polymorphism
- Runtime Polymorphism



The two types of polymorphism in c++ are:

- Compile Time Polymorphism
- Runtime Polymorphism



Polymorphism in C++

Here is a Free C++ course with certification that can help clear your basics of C++ programming

17. Explain constructor in C++

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.

Example:

```
class A{
 private:
  int val;
 public:
  A(int x){          //one argument constructor
   val=x;
  }
  A(){               //zero argument constructor
  }
}
int main(){
 A a(3);

 return 0;
}
```

18. Tell me about virtual function

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.

19. Compare compile time polymorphism and Runtime polymorphism

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

Compile-time polymorphism	Run time polymorphism
In this me	

determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.

19. Compare compile time polymorphism and Runtime polymorphism

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.
Example - `int add(int a, int b){ return a+b; } int add(int a, int b, int c){ return a+b+c; } int main(){ cout<<add(2,3)<<endl; cout<<add(2,3,4)<<endl; return 0; }`	Example - `class A{ public: virtual void fun(){ cout<<"base "; } }; class B: public A{ public: void fun(){ cout<<"derived "; } }; int main(){ A *a=new B; a->fun(); return 0; }`

20. What are the C++ access specifiers?

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.

C++ Interview Questions For Experienced

1. What is a copy constructor?

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

Example-

```
class A{
int x,y;
A(int x, int y){
 this->x=x;
```

derived class.

Private: All data members and member functions are not accessible outside the class.

C++ Interview Questions For Experienced

1. What is a copy constructor?

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

Example-

```
class A{
int x,y;
A(int x, int y){
 this->x=x;
 this->y=y;
}

};
int main(){
A a1(2,3);
A a2=a1;     //default copy constructor is called
return 0;
}
```

We can define our copy constructor. If we don't define a copy constructor then the default copy constructor is called.

2. What is the difference between shallow copy and deep copy?

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 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

3. What is the difference between virtual functions and pure virtual functions?

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.

Example-

```
class base{
public:
 virtual void fun(){

 }
};
```

A pure virtual function is a function that has no implementation and is declared by assigning 0. It has no body.

Example-

```
class base{
public:
```

the original object.	original object

3. What is the difference between virtual functions and pure virtual functions?

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.

Example-

```
class base{
public:
 virtual void fun(){

 }
};
```

A pure virtual function is a function that has no implementation and is declared by assigning 0. It has no body.

Example-

```
class base{
public:
 virtual void fun()=0;
};
```

Here, = sign has got nothing to do with the assignment, and value 0 is not assigned to anything. It is used to simply tell the compiler that a function will be pure and it will not have anybody.

4. 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?

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.

5. Can we call a virtual function from a constructor?

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-

```
class base{
 private:
  int value;
 public:
  base(int x){
   value=x;
  }
  virtual void fun(){

  }
}
```

So the first destructor of class D will be called and then the destructor of class B will be called.

5. Can we call a virtual function from a constructor?

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-

```
class base{
 private:
  int value;
 public:
  base(int x){
   value=x;
  }
  virtual void fun(){

  }
}

class derived{
 private:
  int a;
 public:
  derived(int x, int y):base(x){
   base *b;
   b=this;
   b->fun();     //calls derived::fun()
  }
  void fun(){
   cout<<"fun inside derived class"<<endl;
  }
}
```

6. What are void pointers?

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

For example-

```
void *ptr;
char *str;
p=str;          // no error
str=p;          // error because of type mismatch
```

We can assign a pointer of any type to a void pointer but the reverse is not true unless you typecast it as

```
str=(char*) ptr;
```

7. What is this pointer in C++?

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

```
class A{
 private:
  int value;
 public:
  void setvalue(int x
```

typecast it as

```
str=(char*) ptr;
```

7. What is this pointer in C++?

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

```
class A{
 private:
  int value;
 public:
  void setvalue(int x){
   this->value=x;
  }
};

int main(){
 A a;
 a.setvalue(5);
 return 0;
}
```

8. How do you allocate and deallocate memory in C++?

The new operator is used for memory allocation and deletes operator is used for memory deallocation in C++.

For example-

```
int value=new int;       //allocates memory for storing 1 integer
delete value;            // deallocates memory taken by value

int *arr=new int[10];      //allocates memory for storing 10 int
delete []arr;           // deallocates memory occupied by arr
```

Additional Resources

Practice Coding

C++ MCQ

C++ Tutorials

C Interview Questions

Difference Between C and C++

Difference Between C++ and Java

Online C++ Compiler

Features of C++

Coding Problems

0/4

Intermediate Problems

C++ Introduction Structure & Classes