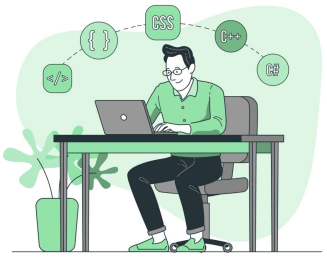
C++ Interview Questions and Answers

Top 25 Most Asked C++ Interview Questions

Here, you can find the most asked C++ Interview Questions and Answers on this page. Prepare for your interview with Prepinsta. PrepInsta provides you with all the latest updated information about C++ and all the C ++ Interview Questions and answers for freshers and experienced questions for candidates.



Page Highlights:

Read more: C++ programming

Introduction to C++ Programming

C++ is a cross-platform language or powerful general programming language that can be used to create high-performance applications. It was developed by Bjarne Stroustrup, as an extension to the C language. It is used to develop OS,browsers,games and so on. Popular operating systems include in C++ programming is: Microsoft and Linux.

Top 25 C++ Interview Questions

1. What is C++ programming?

- C was a procedural language i.e. it didn't have the concept of classes and objects.
- C++ was made to cover this and additional functionalities of Object-Oriented Programming were added.
- C++ features –
  - Object-Oriented
  - Procedural
  - Case Sensitive
  - Generic

2.Difference between C and C++ ?

1. C++ can be considered as a superset of C, most C programs except some exceptions, work in C++ and C.
2. C programming is a little bit limited and is a procedural programming language, but C++ supports both procedural and Object-Oriented programming
3. Since C++ supports object-oriented programming, it is capable of performing tasks like function overloading, templates, inheritance, virtual functions, friend functions. These features are not present

in C.

4. C++ supports exception handling at the language level, in C exception handling is done in the traditional if-else style.
5. C++ supports references, C doesn't.
6. In C, scanf() and printf() are mainly used input/output. C++ mainly uses streams to perform input and output operations. cin is a standard input stream and cout is a standard output stream.

3.What is role of static keyword on class member variable?

Static is a keyword in C++ used to give special characteristics to an element. Static elements are allocated storage only once in a program lifetime in the static storage area. And they have a scope till the program's lifetime. Static Keyword can be used with the following,

1. Static variable in functions
2. Static Class Objects
3. Static member Variable in class
4. Static Methods in class

A static variable does exit through the objects for the respective class are not created. Static member variables share a common memory across all the objects created for the respective class. A static member variable can be referred to using the class name itself.

4. What are the advantages of a C++ program?

Some of the main advantages of the C++ programming language are:

- C++ includes the concept of inheritance. Through inheritance, one can reduce redundancy in the code and can reuse the existing classes.
- Message passing is a technique used for communication between objects.
- C++ is a highly portable language means that the software developed using the C++ language can run on any platform.
- C++ contains a rich function library.
- C++ is an object-oriented programming language that includes the concepts such as classes, objects, inheritance, polymorphism, abstraction.
- Data hiding helps the programmer to build secure programs so that the program cannot be attacked by the invaders.

5. Explain what are Operators and explain with an example?

Operators are the basic concept of any programming language, used to build a foundation in programming for freshers. Operators can be defined as basic symbols that help us work on logical and mathematical operations. Operators in C and C++, are tools or symbols that are used to perform mathematical operations concerning arithmetic, logical, conditional, and, bitwise operations. The different types of operators available for C++ are Assignment Operator, Compound Assignment Operator, Arithmetic Operator, Increment Operator, and so on.

For example arithmetic operators, you want to add two values a+b

```
#include
Using namespace std;

main ()
{
int a= 21 ;
int b= 10 ;
int c;
c= a + b;
cout << "Line 1- Value of c is : " << c << endl ;
return 0;
}
```

It will give the output as 31.

6. If You Want To Share Several Functions Or Variables In Several Files Maintaining The Consistency How Would You Share It?

To maintain the consistency between several files firstly place each definition in the '.c' file then using external declarations put it in the '.h' file after it is included .h file we can use it in several files using #include as it will be in one of the header files, thus to maintain the consistency we can make our header file and include it where ever needed.

7.What is the error in the code below and how should it be corrected?

```
my_struct_t *bar;
/* ... do stuff, including setting bar to point to a defined my_struct_t object ... */
memset(bar, 0, sizeof(bar));
```

sizeof(*bar) should be the last argument of memset instead of sizeof(bar). sizeof(bar) calculates the size of the bar (i.e., the pointer *itself*) rather than the size of the structure pointed to by the bar.

The code can therefore be corrected by using sizeof(*bar) as the last argument in the call to memset.

It might be thought that the *bar will cause a dereferencing error if the bar has not been assigned. Therefore an even safer solution would be to use sizeof(my_struct_t). However, an even sharper candidate must know that in this case using *bar is safe within the call to size, even if the bar has not been initialized yet since sizeof is a compile-time construct.

8. A C++ class has a constructor and overloaded new and delete operator function. If we create a class object dynamically using new then out of constructor and overloaded new operator function, which one get called first?

If we have an overloaded new operator in the class then on creation of object dynamically, the overloaded new operator will be called first and then the class constructor will be called.

And the reverse is the case for overloaded delete and destructor i.e. destructor will be called first then overloaded delete operator in C++ program

Below is the C++ program illustrating the overloading of new and delete operators. In this program, we have a class Book and this class has constructor Book(), overloaded new and delete operator, and a destructor ~Book().

On creation of the object of Book class in main () function, first overloaded new function will be called then the class constructor.

```
#include
using namespace std;

class Book{
public:
        Book(){
                cout<<"constructor"<<endl;
        }
        //Overloaded new operator
        void* operator new(size_t size){
                cout<<"Overloaded new operator"<<endl;
                return malloc(size);
        }
        //overloaded delete operator
        void operator delete(void* ptr){
                cout<<"overloaded delete operator"<<endl;
                free(ptr);
        }
        ~Book(){
                cout<<"Destructor"<<endl;
        }
};

int main(){

        //This will call overloaded new function from
        //class Book, not from OS.
        Book *ob = new Book();
        delete ob;

        return 0;
}
```

OUTPUT:

1. Overloaded new operator
2. constructor
3. Destructor
4. overloaded delete operator

NOTE:
If we overload the new and delete operator in a class, then on creation of object using new, memory will not be directly requested from the operating system, but class new and delete overloaded function will be called in which we write custom memory allocation or request memory to the operating system.

9. What are virtual functions – Write an example?

Virtual functions are used with inheritance, they are called according to the type of the object pointed or referred to, not according to the type of pointer or reference. A virtual function is a member function that is declared within a base class and is re-defined(Overriden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function. In other words, virtual functions are resolved late, at runtime. The virtual keyword is used to make a function virtual.

Following things are necessary to write a C++ program with runtime polymorphism (use of virtual functions)
1) A base class and a derived class.
2) A function with the same name in the base class and derived class.
3) A pointer or reference of base class type pointing or referring to an object of the derived class.

For example, in the following program bp is a pointer of type Base, but a call to bp->show() calls show() function of Derived class because bp points to an object of Derived class.

```
#include
using namespace std;

class Base {
public:
virtual void show() { cout<<" In Base \n"; }
};

class Derived: public Base {
public:
void show() { cout<<"In Derived \n"; } }; int main(void) { Base *bp = new Derived; bp-
>show(); // RUN-TIME POLYMORPHISM
return 0;
}
```

Output:

```
In Derived
```

10. Comment on Assignment Operator in C++?

An assignment operator in C++ is used to assign a value to another variable.

x = 5;

This line of code assigns the integer value **5** to variable **a**.

The part at the left of the = operator is known as an ***lvalue*** (left value) and the right as an ***rvalue*** (right value).

The **L*value*** must always be a variable whereas the ***value*** can be a constant, a variable, the result of an operation, or any combination of them.

The assignment operation always takes place from the right to the left and never at the inverse.

One property which C++ has over the other programming languages is that the assignment operator can be used as the *value* (or part of an *rvalue*) for another assignment.

C++ Interview Questions and Answers for Experienced

1. Define namespace in C++?

A **namespace** is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it. **Namespaces** are used to organize code into logical groups and to prevent name collisions that can occur especially when your codebase includes multiple libraries.

- The namespace is a logical division of the code that is designed to stop the naming conflict.
- The namespace defines the scope where the identifiers such as variables, class, functions are declared.
- C++ consists of a standard namespace, i.e., std which contains inbuilt classes and functions. So, by using the statement “using namespace std;” includes the namespace “std” in our program.
- The main purpose of using namespace in C++ is to remove ambiguity. Ambiguity occurs when a different task occurs with the same name.

For example: if two functions exist with the same name such as add(). To prevent this ambiguity, the namespace is used. Functions are declared in different namespaces.
Syntax of the namespace:

```
namespace namespace_name
{
//body of namespace;
}
```

2. Explain what is C++ exceptional handling?

One of the advantages of C++ over C is Exception Handling. Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution.

The problem that arises during the execution of a program is referred to as exceptional handling. The exceptional handling in C++ is done by three keywords.

- ***try*:** represents a block of code that can throw an exception.
- ***catch*:** represents a block of code that is executed when a particular exception is thrown.
- ***throw*:** Used to throw an exception. Also used to list the exceptions that a function throws, but doesn't handle itself.

3. Define Pointers?

- Pointes is a special type of variables that are used to store the memory address of the other variables.
- Pointers are declared normally as other variables with a difference of * that is present in front of the pointer identifier.
- There are two operators that are used with the pointers one is '&' and another one is '*'. & is known as the address of operator and * is known as dereferencing operator, both are prefix unary operators.

4. What is the output of the following code:

```
#include

class A {
public:
    A() {}
    ~A() {
        throw 42;
    }
};

int main(int argc, const char * argv[]) {
    try {
        A a;
        throw 32;
    } catch(int a) {
        std::cout << a;
    }
}
```

Solution:-

This program will terminate abnormally. ***throw 32*** will start unwinding the stack and destroy class A.

The class A destructor will throw another exception during the exception handling, which will cause the program to crash.

5. What is references and pointers?

Pointers: A pointer is a variable that holds the memory address of another variable. A pointer needs to be dereferenced with the * operator to access the memory location it points to.

References: A reference variable is an alias, that is, another name for an already existing variable. A reference, like a pointer, is also implemented by storing the address of an object.
A reference can be thought of as a constant pointer (not to be confused with a pointer to a constant value!) with automatic indirection, i.e the compiler will apply the * operator for you.

1. ***References are less powerful than pointers***
   1) Once a reference is created, it cannot be later made to reference another object; it cannot be reseated. This is often done with pointers.
   2) References cannot be NULL. Pointers are often made NULL to indicate that they are not pointing to any valid thing.
   3) A reference must be initialized when declared. There is no such restriction with pointers
2. ***References are safer and easier to use***:
   1) Safer: Since references must be initialized, wild references like wild pointers are unlikely to exist. It is still possible to have references that don't refer to a valid location (See questions 5 and 6 in the below exercise)
   2) Easier to use: References don't need dereferencing operator to access the value. They can be used like normal variables. '&' operator is needed only at the time of declaration. Also, members of an object reference can be accessed with dot operator ('.'), unlike pointers where arrow operator (->) is needed to access members.

6. What is equal to (==) and Assignment Operator (=)?

In C++, equal to (==) and assignment operator (=) are two completely different operators.

- The assignment operator (=) is used to assign a value to a variable. Hence, we can have a complex assignment operation inside the equality relational operator for evaluation.
- Equal to (==) is an equality relational operator that evaluates two expressions to see if they are equal and returns true if they are equal and false if they are not.

7. Which operations are permitted on pointers?

In C++ programming the following operations are allowed to perform on pointers:
Incrementing or decrementing a pointer: Incrementing a pointer means that we can increment the pointer by the size of a data type to which it points.

There are two types of increment/decrement pointers:

1. Pre-increment/decrement pointer: The pre-increment/decrement operator increments the operand by 1, and the value of the expression becomes the resulting value of the incremented/decremented. Suppose ptr is a pointer then pre-increment/decrement pointer is represented as ++ptr.

2.Post-increment/decrement pointer: The post-increment/decrement operator increments the operand by 1, but the value of the expression will be the value of the operand prior to the incremented/decremented value of the operand. Suppose ptr is a pointer then post-increment/decrement pointer is represented as ptr++.

8. Mention what are the decision-making statements in C++?

The major decision making statements in C++ are

- if statement
- switch statement
- conditional operator

For example, we want to implement if condition in C++

```
#include<iostream>
int main ( )
{
  int, x, y;
  X= 10;
  Y= 5;
  if (x > y)
  {
    Cout << "x is greater than y";
   }
}
```

9. What is Between Global Variables And Static Variables?

- **Global variables**: Global variables are variables that are defined outside the function. The scope of global variables begins at the point where they are defined and lasts till the end of the file/program. They have external linkage, which means that in other source files, the same name refers to the same location in memory. The scope of the global variables remains throughout the program also the life span of these variables is throughout the program.

- **Static variables:** Static variables are private to the source file where they are defined and do not conflict with other variables in other source files which would have the same name. The scope of the variable describes whether the variable is accessible at a certain point in the program or not.

10. How many times will this loop execute? Explain your answer.

```
unsigned char half_limit = 150;

for (unsigned char i = 0; i < 2 * half_limit; ++i)
{
    // do something;
}
```

Solution:- **This code will result in an infinite loop**.
Here's why: The expression `2 * half_limit` will get promoted to an `int` (based on C++ conversion rules) and will have a value of 300. However, since `i` is an `unsigned char`, it is represented by an 8-bit value which, after reaching 255, will overflow (so it will go back to 0) and the loop will therefore go on forever.

11. What is the role of protected access specifier?

A class member is accessible in the inherited class if the class member is protected. However, outside both the private and protected members are not accessible.

12. Distinguish between shallow copy and deep copy.

Shallow copy performs bit-by-bit memory dumping from one object to another. Deep copy is the process of copying an object field by field from one object to another. The copy function Object() { [native code] } and the overloading assignment operator are used to achieve deep copy.

13. Mention the storage classes names in C++.

The following are storage classes supported in C++ :

- auto
- static
- extern
- register
- mutable

14. What is the role of a mutable storage class specifier?

The member variable of a constant class object may be used by declaring it with the mutable storage class specifier. Just applies to the class's non-static and non-constant member variables.

15. What is a class template?

A template class is a prototype class. A template class can be described using the keyword template.

16. What Is The Memory Structure Of An Object?

Usually, C++ objects are made by concatenating member variables.

For example;

```
class Test
{
int I;
float j;
};
It is represented by an int followed by a float.
class TestSub: public Test
{
int k;
};
```

The above class is represented by Test and then an int(for int k). So finally it will be int, float, and int.

In addition to this, each object will have the vptr(virtual pointer) if the class has the virtual function, usually as the first element in a class.

17. Define Encapsulation in C++?

The process of encapsulating data and functions in a class is called encapsulation. For security purposes, it is used to prevent direct access to data. This is accomplished by using class functions.

For example, the net banking facility for customers allows only the approved person with the proper login id and password to access the information in the bank data source, and only for his or her part of the information.

18. What Is The Difference Between Exit And Abort?

- Exit performs a graceful process termination by invoking the destructors for all constructed objects, while abort does not.
- Exiting the local The destructors of the calling functions and its callers' variables will not be called.

19. Write C++ code to create an array of objects using a new keyword and delete these objects. Also, proof that all the objects are deleted properly.

C++ not only preserves all elements of the C language, but it also simplifies memory management and introduces many new features, such as:

We know that when we generate a class object dynamically from heap memory with the new keyword, we must specifically delete it after we are finished with its operations to prevent memory leaks in C++ programs.

As an example, consider the class Pen below.

```
class Pen {

public:
       Pen() {
          cout << "Constructor..." <<endl;
       }

       ~Pen() {
          cout << "Destructor...!" <<endl;
       }
            void write(){
                    cout << "Writing...!" <<endl;
            }
};
```

Below is an example, of how we create an object of the class Pen dynamically and delete it after we are done with operations.

```
int main()
{
        //create an object dynamically
   Pen* pen = new Pen();
   pen->write();//operations

   delete pen;//de-allocate the memory

   return 0;
}
```

How to create an array of objects of a class in C?

Let's see how we create an array of objects in the below C++ program example. In this example, we will create an array of 3 objects, perform operations and delete a dynamic array of pointers properly.

```
int main()
{
        //create an array of objects
   Pen* pen = new Pen[3];

   pen[0].write();
   pen[1].write();
   pen[2].write();

   delete [] pen;//de-allocate array of objects

   return 0;
}
```

Output:

Constructor…
Constructor…
Constructor…
Writing…!
Writing…!
Writing…!
Destructor…!
Destructor…!
Destructor…!

Proof of proper deletion of an array of objects in C++

First, note that if we create an object of a class and delete the object then class constructor and destructor will be called respectively.

In the above example, we have created an array of 3 objects and deleted it using statement delete [] pen; if we look at the output the class constructor and destructor were called 3 times each. means, we have deleted objects properly.

Now, If we delete the object using the statement “delete pen” (This is a general mistake a programmer makes) then the destructor will be called only once, and for the rest 2 objects destructor will not be called and the program will crash. and this will prove that objects are not deleted properly.

Let’s see the output if we use the statement “delete pen”

```
int main()
{
        //create an array of objects
   Pen* pen = new Pen[3];

   pen[0].write();
   pen[1].write();
   pen[2].write();

   //this is not a prper way to delete
   //array of objects
   delete pen;

   return 0;
}
```

Output:

Constructor…

Constructor…

Constructor…

Writing…!

Writing…!

Writing…!

Destructor…!

and then the program will crash at run time.

Conclusion:

In C++, a single object of a class created dynamically is deleted using the statement “delete pointer” and an array of objects are deleted using the statement “delete [ ] pointer”.

20. How many ways are there to initialize an int with a Constant?

There are two ways:

- The first format uses traditional C notation.
  int result = 10;
- The second format uses the constructor notation.
  int result (10);

21. Explain The Is-a And Has-a Class Relationships. How Would You Implement Each In A Class Design?

A specialized class “is” a specialization of another class and therefore shares an IS-A relationship with it. An ISA person who works for the business. Inheritance is the only way to implement this partnership. The word “employee” comes from the word “person.” A class can contain an instance of another class. Employees, for example, “have” salaries, because the Employee class has a HAS-A relationship with the Wage class. The best way to enforce this relationship is to embed a Salary class object in the Employee class.

22. Draw a comparison between C++ and Java

- Destructors are called automatically when an object is destroyed in C++. Automatic garbage collection is a function of Java.
- Multiple inheritance, operator overloading, pointers, structures, templates, and unions are all supported in C++. None of them are available in Java.
- To build a new thread, Java has a Thread class that is inherited. Threads aren’t supported by C++ by default.
- In C++, a goto statement offers a way to jump from a location to some labeled statement in the same function. There is no goto statement in Java
- C++ runs and compiles using the compiler, which converts the source code into machine-level language. Hence, it is platform-dependent. Java compiler, on the other hand, converts the source code into JVM bytecode, which is platform-independent.

23. Why do we need the Friend class and function?

It’s often necessary to grant specific class access to a class’s private or protected members. A friend class is a solution since it can access both protected and private members of the class in which it is declared as a friend. A friend feature, like the friend class, has access to private and safe class members.

- Friendship is not something you inherit.
- Friendship is not reciprocal, because if one class is a friend of another, such as NotAFriend, it does not immediately become a friend of the Friend class.
- The total number of friend classes and friend functions in a program should be restricted because too many of them will degrade the principle of encapsulation of separate classes, which is an intrinsic and desirable quality of object-oriented programming.

24. Difference between Declaration and Definition of a variable.

The name of a variable and its data type is also specified in the declaration of a variable. As a result of the declaration, we tell the compiler to set aside memory space for a variable of the specified data type.

Example:

```
int Result;
char c;
int a,b,c;
```

All of the preceding declarations are right. Also, notice that the variable's value is unknown as a consequence of the declaration.

A description, on the other hand, is an implementation/instantiation of the declared variable in which we bind appropriate values to the declared variable so that the linker can link references to the correct entities.

From above Example,

Result = 10;

C = 'A';

These are valid definitions.

25. When there are a Global variable and Local variable with the same name, how will you access the global variable?

When two variables have the same name but different scopes, for example, one is a local variable and the other is a global variable, the compiler would favor the local variable.

We use a "scope resolution operator (::)" to get access to the global variable. We can get the value of the global variable with this operator.

Example:

```
#include<iostream>
int x= 10;
int main()
{
int x= 2;
cout<<"Global Variable x = "<<::x;
cout<<"\nlocal Variable x= "<<x;
}
```

Output:

Global Variable x = 10

local Variable x= 2

Also Check: Most asked Technical Interview Questions

We help students to prepare for placements with the best study material, online classes, Sectional Statistics for better focus and Success stories & tips by Toppers on PrepInsta.