C++ Interviewers Beware

of 20.5.2.4

C++ Under the Sea 2026 Lightning Talk
Roi Barkan

Istra Research

roi.barkan@gmail.com

what's a common job interview question about c++ virtual functions

what's a common job interview question about c++ virtual functions

These Slides

roi.barkan@gmail.com

Istra Research

Gemini

what's a common job interview question about c++ virtual functions

One of the most common and critical job interview questions about C++ virtual functions is: **"Why do we need virtual destructors, and what happens if a base class destructor is not declared as virtual?"** GeeksforGeeks +2

This question is a favorite among interviewers because it tests your practical understanding of **memory management**, **object lifecycles**, and **polymorphism** all at once. GeeksforGeeks +1

roi.barkan@gmail.com

Claude

what's a common job interview question about c++ virtual functions

A classic one is **"Why should a base class have a virtual destructor?"**

ChatGPT

Other common interview questions

1. What is the difference between a virtual function and a pure virtual function?
2. What is an abstract class in C++?
3. Why should a base class have a virtual destructor?
4. Can constructors be virtual in C++? Why or why not?
5. What is the difference between compile-time and runtime polymorphism?
6. What happens if you override a virtual function but omit the `override` keyword?

roi.barkan@gmail.com

C++ Core Guidelines

C.35: A base class destructor should be either public and virtual, or protected and non-virtual

Reason To prevent undefined behavior. If the destructor is public, then calling code can attempt to destroy a derived class object through a base class pointer, and the result is undefined if the base class's destructor is non-virtual. If the destructor is protected, then calling code cannot destroy through a base class pointer and the destructor does not need to be virtual; it does need to be protected, not private, so that derived destructors can invoke it. In general, the writer of a base class does not know the appropriate action to be done upon destruction.

Link

roi.barkan@gmail.com

Enter C++26

Cppcon
The C++ Conference

Cppcon.org

Agenda

cppcon the c++ conference

- C++26 Core Language
  - Static Reflection
  - Contracts
  - Unnamed Placeholder Variables (_)
  - User-Generated static_assert() Messages
  - = delete("reason") Syntax
  - Attributes on Structured Bindings
  - Structured Bindings in Conditions
  - constexpr Structured Bindings
  - Structured Bindings Introducing Packs
  - Pack Indexing
  - #embed
  - Variadic Friend Declarations
  - constexpr Casts from void*
  - constexpr Placement new
  - constexpr Exceptions
  - Uninitialized Read Formalization
- C++26 Standard Library
  - Execution Control Library
  - std::inplace_vector
  - std::hive
  - std::submdspan()
  - Saturation Arithmetic
  - string_view init for stringstreams/bitset
  - std::text_encoding
  - Native File Handles
  - More constexpr
  - New SI Prefixes
  - <debugging>
  - <linalg>
  - Tuple Protocol for std::complex
  - String Concatenation
  - Enabling List Initialization for Algorithms
  - views::concat
  - std::ranges::cache_latest_view
  - std::ranges::generate_random
  - std::optional Becomes a View
  - std::philox_engine
  - Filesystem Path Formatter
  - std::println() Blank Lines
  - std::simd
  - std::dynamic_format
  - std::indirect
  - std::polymorphic
  - std::function_ref

Link

5

C++26:
A Curated Tour of What's New

Marc Gregoire

Istra Research
roi.barkan@gmail.com

std::polymorphic

- Solves the following use case:
  - Allocate polymorphic types on the heap
  - Easily make copies
- Example:

```
class Base { /*...*/ };
class Derived1 : public Base { /*...*/ };
class Derived2 : public Base { /*...*/ };

std::vector<std::polymorphic<Base>> storedItems;
storedItems.emplace_back(Derived1{});
storedItems.emplace_back(Derived2{});
```

Link

roi.barkan@gmail.com

Istra Research

std::polymorphic

[P3019]

- Treat polymorphic (virtual) classes like values
- Mechanism: type-erase copy/assign/move members
- Allocator-aware
- Small-Buffer-Optimization
- And… [20.5.2.4]

20.5.2.4 Destructor **[polymorphic.dtor]**

```
constexpr ~polymorphic();
```

1 *Mandates*: T is a complete type.

2 *Effects*: If `*this` is not valueless, calls `allocator_traits<Allocator>::destroy(p)`, where p is a pointer of type `U*` to the owned object and U is the type of the owned object; then the storage is deallocated.

roi.barkan@gmail.com

Istra Research

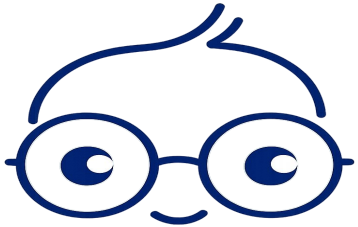
Summary

- The language evolves, things are easier, need to keep up
- Value semantics for the win
  - Also see **std::indirect**
- Cppcon 2026 Early Access is Free for Everyone
  cppcon.programmingarchive.com
- **Notes:**
  - **shared_ptr** also has some destructor erasure, discussed here
  - Virtual destructor has no downside for **std::is_polymorphic** classes

These Slides

roi.barkan@gmail.com

Istra Research



Thank You

roi.barkan@gmail.com