

# ◆ Advanced & Less Commonly Read C++ Concepts Asked in Interviews

## 1 Function Pointers & Callbacks

- Returning function pointers from functions
- Member function pointers (`void (Class::*ptr)()`)
- `std::function` and `std::bind` for function wrappers

## 2 Copy & Move Semantics (Deep Questions on Constructors)

- **Difference between copy and move constructors**
- When does the compiler **delete** move constructor?
- What happens when returning a temporary object? (RVO & NRVO)
- Implicitly deleted functions

## 3 Operator Overloading (Rare but Tricky Operators)

- Overloading `[]`, `()`, `->` (Function call operator & smart pointers)
- Why can't we overload `::`, `.*`, `sizeof`, `??`
- Friend function vs. member function for operator overloading

## 4 Memory Management (Deep-Dive)

- Custom **memory allocators** (overloading `new` and `delete`)
- Memory leaks and how to detect them (`valgrind`, sanitizers)
- Smart pointers (`std::unique_ptr`, `std::shared_ptr`, `std::weak_ptr`)

## 5 Virtual Functions & Polymorphism (Edge Cases)

- **Virtual destructor importance**
- **Pure virtual destructor – Is it possible?**
- Virtual table (VTable) and vptr (How does polymorphism work internally?)
- Overriding vs. hiding vs. shadowing in inheritance

## 6 Templates & Metaprogramming (Less Common but Tricky)

- **SFINAE (Substitution Failure Is Not An Error)**
- `enable_if` and type traits
- **Variadic templates** (`template<typename... Args>`)
- CRTP (Curiously Recurring Template Pattern)

## 7 Type System & Casting

- `static_cast`, `dynamic_cast`, `reinterpret_cast`, `const_cast`
- When does `dynamic_cast` fail?
- `typeid` and `RTTI` (Run-Time Type Information)

## 8 Concurrency & Multithreading (Tricky But Important)

- Mutex types (`std::mutex`, `std::recursive_mutex`, `std::timed_mutex`)
- Deadlocks & how to prevent them
- Atomic operations (`std::atomic`)
- Thread pools and `std::async` vs. `std::thread`

## 9 STL Internals (Rarely Read but Frequently Asked)

- How does `std::map` work internally? (Red-Black Tree)
- How does `std::unordered_map` work? (Hashing & collision resolution)
- When to use `std::vector` vs. `std::list` vs. `std::deque`?
- How does `std::sort()` work? (Introsort)

## 10 Undefined Behavior & Edge Cases

- Accessing a moved-from object – What happens?
- Integer overflow in signed vs. unsigned types
- Dangling references and use-after-free issues
- Alignment and padding in structs (`#pragma pack`)

---

## Conclusion

Many programmers focus only on **syntax and common patterns**, but **deep C++ knowledge** includes:

- ✓ Move semantics
- ✓ Memory management
- ✓ STL internals
- ✓ Multithreading
- ✓ Advanced polymorphism

 Would you like me to provide tricky interview questions on these topics? 😊

You said:

yes please

ChatGPT said:

Sure! Here are some **tricky C++ interview questions** based on **rare but important** concepts.

---

## ◆ Tricky C++ Interview Questions by Concept

### 1 Function Pointers & Callbacks

#### ◆ Question:

What is the difference between:

```
cpp
CopyEdit
void (*funcPtr)();
void (MyClass::*memberFuncPtr)();
```

✓ **Follow-up:** How do you call `memberFuncPtr` using an object?

#### ◆ Question:

What is the output of this code?

```
cpp
CopyEdit
#include <iostream>
using namespace std;

void hello() { cout << "Hello\n"; }

int main() {
    void (*func)() = hello;
    (*func)();
}
```

✓ **Follow-up:** Can we use `std::function<void()>` instead of function pointers?

---

### 2 Copy & Move Semantics

#### ◆ Question:

What will be the output of this code?

```
cpp
CopyEdit
#include <iostream>
using namespace std;
```

```

class Test {
public:
    Test() { cout << "Default Constructor\n"; }
    Test(const Test&) { cout << "Copy Constructor\n"; }
    Test(Test&&) { cout << "Move Constructor\n"; }
};

Test fun() {
    Test t;
    return t;
}

int main() {
    Test obj = fun();
}

```

✅ **Follow-up:** What happens if we enable **RVO (Return Value Optimization)**?

◆ **Question:**

Why does the **move constructor get deleted** in the following case?

```

cpp
CopyEdit
class Test {
public:
    Test(const Test&) { } // Explicit copy constructor
};

```

✅ **Follow-up:** How to explicitly enable the move constructor?

---

### 3 Operator Overloading

◆ **Question:**

What is the output of this program?

```

cpp
CopyEdit
#include <iostream>
using namespace std;

class A {

```

```

public:
    void operator()() { cout << "Function call operator\n"; }
};

int main() {
    A obj;
    obj();
}

```

✅ **Follow-up:** When would you use the function call operator `()` in real-world scenarios?

♦ **Question:**

Can we overload `operator->()`? If yes, how does it work internally?

---

## 4 Memory Management

♦ **Question:**

What happens if we **delete a pointer twice**?

```

cpp
CopyEdit
int* ptr = new int(5);
delete ptr;
delete ptr; // What happens here?

```

✅ **Follow-up:** How can we **prevent double deletion** issues?

♦ **Question:**

What is the **difference** between `new` and `malloc()` in C++?

✅ **Follow-up:** Can we override `new` and `delete` operators?

---

## 5 Virtual Functions & Polymorphism

♦ **Question:**

Why should the **base class destructor be virtual**?

```

cpp
CopyEdit
class Base {
public:

```

```

    ~Base() { cout << "Base Destructor\n"; }
};

class Derived : public Base {
public:
    ~Derived() { cout << "Derived Destructor\n"; }
};

int main() {
    Base* obj = new Derived();
    delete obj;
}

```

✅ **Follow-up:** What happens if the destructor is **not virtual**?

♦ **Question:**

Can a **pure virtual destructor** exist in C++? If yes, how to implement it?

---

## 6 Templates & Metaprogramming

♦ **Question:**

What is **SFINAE** in templates? Give an example.

✅ **Follow-up:** How does `std::enable_if` help with template specialization?

♦ **Question:**

What will be the output of this **variadic template** function?

```

cpp
CopyEdit
#include <iostream>
using namespace std;

template<typename T>
void print(T t) {
    cout << t << endl;
}

template<typename T, typename... Args>
void print(T t, Args... args) {
    cout << t << " ";
    print(args...);
}

```

```

}

int main() {
    print(1, 2, 3, "Hello");
}

```

✓ **Follow-up:** How does **parameter pack expansion** work?

---

## 7 Type System & Casting

♦ **Question:**

What is the difference between **static\_cast**, **dynamic\_cast**, **reinterpret\_cast**, and **const\_cast**?

♦ **Question:**

What will happen in this **dynamic\_cast** case?

```

cpp
CopyEdit
class Base { virtual void fun() {} };
class Derived : public Base {};
int main() {
    Base* b = new Derived();
    Derived* d = dynamic_cast<Derived*>(b);
}

```

✓ **Follow-up:** What happens if **b** is **not actually a Derived object**?

---

## 8 Multithreading & Concurrency

♦ **Question:**

What is the difference between:

```

cpp
CopyEdit
std::mutex mtx;
std::recursive_mutex rmtx;

```

✓ **Follow-up:** When should you use **std::condition\_variable**?

♦ **Question:**

What does `std::atomic<int>` do?

✓ **Follow-up:** How is `std::atomic` different from using `volatile`?

---

## 9 STL Internals

♦ **Question:**

How does `std::map` work internally?

✓ **Follow-up:** What is the time complexity for `std::unordered_map::find()`?

♦ **Question:**

Why is `std::vector<bool>` **not** a true STL container?

---

## 10 Undefined Behavior & Edge Cases

♦ **Question:**

What happens in this **use-after-move** scenario?

```
cpp
CopyEdit
#include <iostream>
using namespace std;

class A {
public:
    A() { cout << "Constructor\n"; }
    A(A&&) { cout << "Move Constructor\n"; }
};

int main() {
    A a = A();
    A b = move(a);
}
```

✓ **Follow-up:** What happens if we try to **access a** after `move()`?

♦ **Question:**

What is the output of this **integer overflow case**?

```
cpp
```



CopyEdit

```
#include <iostream>
using namespace std;

int main() {
    int a = INT_MAX;
    cout << a + 1;
}
```

✅ **Follow-up:** What happens if you use **unsigned int** instead?

---

## 🔥 Bonus:

**Interviewer Favorite Trick:** They give a code snippet and ask you to find a bug or undefined behavior.

---

## 🚀 Conclusion

These **rarely studied** but **frequently asked** interview questions cover:

- ✓ Move Semantics
- ✓ Operator Overloading
- ✓ Memory Management
- ✓ Advanced Polymorphism
- ✓ STL Internals
- ✓ Undefined Behavior