

# C++ OOP INTERVIEW CHECKLIST — EXTENDED EDITION

A comprehensive guide including:

- Complete OOP concepts
- C++-specific interview topics
- Code examples
- Modern C++ features
- Common pitfalls
- Scenario-based interview questions

## 1. BASICS OF OOP

- What is OOP?
- Class vs Object
- Structure vs Class
- Access Specifiers: public, private, protected
- Encapsulation, Abstraction
- Constructors & destructors

Example: Basic Class

```
class Student {  
private:  
    string name;  
    int age;  
public:  
    Student(string n, int a) : name(n), age(a) {}  
    void introduce() { cout << name << " (" << age << ")"; }  
};
```

## 2. CONSTRUCTORS & DESTRUCTORS

- Default / Parameterized constructors
- Copy constructor
- Move constructor
- Initialization lists
- Virtual destructor
- Order of construction/destruction

Example: Rule of Three

```
class A {  
private:  
    int* data;  
public:  
    A(int val) { data = new int(val); }  
    A(const A& other) { data = new int(*other.data); }
```

```

A& operator=(const A& other) {
    if(this != &other;) *data = *other.data;
    return *this;
}
~A() { delete data; }
};

```

### 3. MEMORY MANAGEMENT

- Raw pointers vs Smart pointers
- RAII (Resource Acquisition Is Initialization)
- unique\_ptr, shared\_ptr, weak\_ptr
- Dangling pointers & leaks

Example: Smart Pointers

```

unique_ptr p1 = make_unique(5);
shared_ptr p2 = make_shared(10);
weak_ptr pw = p2; // does not increase ref count

```

### 4. INHERITANCE

- Types: Single, Multiple, Multilevel, Hierarchical
- Virtual inheritance (diamond problem)
- override, final
- Constructors calling order

Example: Diamond Problem

```

class A {};
class B : virtual public A {};
class C : virtual public A {};
class D : public B, public C {};

```

### 5. POLYMORPHISM

Compile-time: Overloading, templates

Runtime: Virtual functions

- Late binding
- Pure virtual functions
- Abstract classes
- Vtable concept

Example: Runtime Polymorphism

```

class Base {
public:
    virtual void show() { cout << "Base"; }
};
class Derived : public Base {

```

```
public:  
void show() override { cout << "Derived"; }  
};
```

## 6. TEMPLATES & GENERICS

- Function templates
- Class templates
- Template specialization
- CRTP (advanced)

Example: Simple Template

```
template  
T add(T a, T b) { return a + b; }
```

## 7. EXCEPTION HANDLING

- try/catch/throw
- Stack unwinding
- RAII for exception safety
- noexcept

Example: Exception Safety

```
try {  
// risky code  
} catch(const exception& e) {  
cout << e.what();  
}
```

## 8. DESIGN & UML CONCEPTS

- Composition vs Aggregation
- IS-A vs HAS-A
- SOLID Principles
- Dependency inversion

Example: Composition

```
class Engine { };  
class Car {  
private:  
Engine eng; // HAS-A  
};
```

## 9. COMMON INTERVIEW PITFALLS

- Non-virtual destructor in base class
- Object slicing
- Incorrect copy semantics
- Memory leaks due to raw pointers
- Forgetting override
- Ambiguous multiple inheritance

## 10. TOP INTERVIEW QUESTIONS (WITH ANSWERS)

Q1. Why virtual destructor?

A. To ensure derived class destructor is called via base-pointer deletion.

Q2. Difference between override & overload?

A. Override = runtime polymorphism; Overload = compile-time.

Q3. What is object slicing?

A. Copying derived into base object, losing derived part.

Q4. How does vtable work?

A. Compiler builds an array of function pointers for virtual calls.

**END**

This concludes the extended PDF version of C++ OOP interview preparation.