**📘 High-Level Structure of the Final Guide**

Each section will be rich with:

* ✅ **Theory**
* 💡 **Interview Tips / Gotchas**
* 🔁 **Common Interview Questions**
* 🧠 **Deep Dives (Advanced Only)**
* 💻 **Code Examples**
* 📚 **C++ Version Notes (up to C++23)**

**🔖 Full Table of Contents (Grouped by Category)**

**1️⃣ C++ Fundamentals**

* Data Types, Sizes, Ranges (size\_t, signed, unsigned, int64\_t, int\_128\_int\_256, short, long, long long, float, double, long double, uint128\_t, wchar\_t, size\_t, ptrdiff\_t, intmax\_t, uintmax\_t, uintptr\_t, etc.)
* Size in bytes + range (signed/unsigned)
* How signedness impacts bit representation
* Overflow behavior
* Alignment & padding
* alignas, alignof, offsetof
* References vs Pointers. Pointer to pointer, pointer arithmetic. Null pointers, wild pointers
* Pass-by-Value vs Reference. const references for efficiency. Function argument behavior
* Memory Layout: Stack vs Heap
* Arrays vs Vectors. Fixed size vs dynamic. Memory layout differences.
* Function Basics, Overloading, Default Args
* Control Structures: if, for, while, switch
* Endl vs \n
* Cerr vs cin vs cout vs etc.

**2️⃣ OOP Concepts (Core & Advanced)**

* Encapsulation, Abstraction
* Inheritance (single, multiple, multilevel, hybrid, private/protected/public)
* Inheritance. Diamond problem. Virtual inheritance.
* Access Specifiers.
* Polymorphism (compile-time vs runtime, CRTP)
* Virtual Functions, vtables, vptr, cost of dynamic dispatch, Virtual destructors. What is the cost of dynamic dispatch? Can you remove virtual functions using CRTP?
* Improve performance in a polymorphic system with many virtual call
* Friend Classes. Funct Ions
* Rule of 0 / 3 / 5 (Constructors/Destructor/Assignment). Constructor delegation
* Destructor behavior in inheritance and RAII. Why my destructor is not called? What happens if base destructor isn’t virtual?
* Simulate polymorphism without using virtual functions.
* Operator Overloading (when & how to overload, Best practices (e.g., overloading <<, ==, [])
* SOLID Principles
* Abstract Classes vs Interfaces. Pure Virtual Functions and Abstract Classes (Interface implementation, Cannot instantiate abstract classes)
* Object Counting Patterns
* Constructor Order in Inheritance
* Private Inheritance & all other types of inheritance, Interface Segregation
* Class vs struct vs union vs enum vs typedef defined datatype
* Order of constructor calls: base → derived
* Calling base constructor from derived
* Implicit and explicit constructor inheritance using using
* Object slicing
* Assigning base class object to derived & vice versa
* Copy/move constructors across base/derived
* Virtual destructors and polymorphic cleanup

**3️⃣ C++ Language Features**

* this pointer
* constexpr, consteval, constinit
* Const Correctness. const methods. const arguments, return types, pointers
* decltype, auto, decltype(auto)
* std::optional, std::variant, std::any. Their usage and use cases
* Namespaces, Anonymous Namespaces. Namespace pollution, using directive
* Macros vs using vs define vs typedef vs Templates
* static (global, member, function). Static functions and variables. Class vs object-level members.
* #pragma, #define, #include
* Enum vs enum class
* size\_t and fixed-width types
* istream,ostream
* STL Debug mode: \_GLIBCXX\_DEBUG
* Exception Handling (try, catch, throw. Stack unwinding. RAII-based exception safety)
* Low-Level Performance Tuning. Cache-friendly data layouts. Inlining, loop unrolling, branch prediction.
* Ranges Library
* Casting. Static, dynamic, const, reinterpret etc.

**4️⃣ STL, Containers & Algorithms**

* vector, deque, list, map, unordered\_map, set, priority\_queue. Time and space complexity of operations
* Comparison of unordered\_map vs map performance and internal implementation
* iterator , ranges. Iterator invalidation
* push\_back vs emplace\_back
* passing map queue stack to a function - check if its passed by reference by default?
* assign, fill, iota, clear, erase, replace, substr
* find\_first\_not\_of, find\_last\_not\_of
* Passing containers to functions: deep vs shallow copy. Pass by value vs pass by reference
* next\_permutation, prev\_permutation
* Bit Manipulation (popcount, \_\_builtin , etc ops)
* stringstream, cin.clear(), buffer flushing
* tuple, pair, make\_pair, structured bindings
* All string Manipulation functions get/set
* std::span (C++20)
* std::execution (parallel algorithms)

**5️⃣ Templates & Meta Programming**

* Function/Class Templates
* Template Specialization & Partial Specialization
* Variadic Templates
* SFINAE & enable\_if
* C++20 Concepts
* Type Traits, Detection Idioms
* CRTP (Static Polymorphism) . Libraries like Eigen

**6️⃣ Memory Management**

* new/delete vs malloc/free
* Stack vs heap memory
* RAII Idiom (with custom classes)
* Smart Pointers: unique\_ptr, shared\_ptr, weak\_ptr
* Manual implementation of shared\_ptr and other smart pointer. What are the pitfalls?
* memset
* Deep vs Shallow Copy
* Copy/Move Semantics, std::move, std::forward.
* Move Semantics. Rvalue references (&&). std::move, move constructors, move assignments. Copy elision, NRVO
* Ownership semantics.
* Dangling Pointers
* Placement new, Alignment, Padding
* Memory Leaks: detection, analysis, fixes. Circular References

**7️⃣ Concurrency & Multithreading**

* std::thread, std::mutex, std::lock\_guard, std::scoped\_lock
* std::condition\_variable, std::atomic
* Thread-safe Singleton.
* Thread-safe design and OOP integration. Concurrent queue
* Object Pool Pattern
* Read/Write Locks
* Lock-Free Queues
* Semaphores, Barriers
* Data races & how to resolve and its types,
* Deadlocks & how to resolve and its types,
* False Sharing, Cache-line effects
* RAII + Thread Management
* Parallel Algorithms

**8️⃣ Advanced Design & System-Level Topics**

* Type Erasure (e.g., how std::function works or other variant). Implementing your own type-erased wrapper
* Custom Allocators & memory pooling.
* ABI Stability & PImpl Idiom. Why changing virtual function layout breaks ABI.
* Design Patterns in C++:
  + Singleton (thread-safe)
  + Factory, Strategy, Observer
  + PImpl Idion
  + Visitor, Prototype
* Polymorphism Without Virtual (via CRTP)
* Improving Virtual Dispatch Performance
* Compile vs Runtime Polymorphism
* std::function vs lambdas vs function pointers (Captures, Closures)
* Implement LRU Cache using OOP + STL.

**9️⃣ File Handling & Streams**

* ifstream, ofstream, fstream
* Input/output redirection
* Flushing, syncing, clearing input buffer
* cerr vs cout

**🔟 Build Systems & Toolchain**

* g++, gcc, clang usage
* Compiler flags:
  + -Wall, -Wextra, -O2, -g, -std=c++XX, -fsanitize=address
* gdb basics
* Build automation (Makefile, CMake)
* Debug builds vs Release builds