C++ Evolution: 98 → 11 → 14 → 17 → 20 → 23 → 26  
  
C++ has evolved tremendously from C++98 to modern C++23 — bringing safer memory management, expressive syntax, and powerful features for performance-critical and high-level development alike.  
 Here’s a quick snapshot of its journey:  
  
🔹 C++98 (1998)  
 The foundation of modern C++ — introducing templates, exceptions, and the Standard Template Library (STL).  
 Set the stage for generic programming and object-oriented design.  
  
🔹 C++11 (2011 – “The Big One”)  
 ✅ Smart Pointers (unique\_ptr, shared\_ptr)  
 ✅ Move Semantics & Rvalue References  
 ✅ Lambda Expressions & auto keyword  
 ✅ Range-based for loops, nullptr, and multithreading library  
 → The release that modernized C++ forever.  
  
🔹 C++14 (2014)  
 Polished C++11 with small but powerful refinements:  
 ✅ Generic Lambdas  
 ✅ Binary Literals, std::make\_unique()  
 ✅ Relaxed constexpr functions  
 → A refinement release focusing on usability and compile-time power.  
  
🔹 C++17 (2017)  
 ✅ Structured Bindings (auto [x, y] = …)  
 ✅ if constexpr and Fold Expressions  
 ✅ std::optional, std::variant, std::any  
 ✅ Filesystem Library  
 → Made C++ cleaner, more expressive, and robust.  
  
🔹 C++20 (2020 – A Game Changer)  
 ✅ Concepts (stronger template constraints)  
 ✅ Ranges Library  
 ✅ Coroutines & Modules  
 ✅ constexpr enhancements across the board  
 → The largest leap since C++11 — enabling safer, modular, and more maintainable code.  
  
🔹 C++23 (2023)  
 ✅ std::expected for safer error handling  
 ✅ std::print, std::mdspan, std::flat\_map  
 ✅ Improved constexpr containers & pattern matching groundwork  
 → Polishing the C++20 ecosystem with developer productivity in mind.  
  
🔹 C++26 (Upcoming)  
 🚀 Reflection & Metaclasses (planned)  
 🚀 Contracts, Enhanced Pattern Matching  
 🚀 Further compile-time programming power and safer concurrency  
 → Continuing the modernization toward clarity, safety, and performance.  
  
✨ With every iteration, C++ is becoming faster, safer, and more expressive — without losing its low-level power and performance edge.  
  
👉 If you’re still on C++11 or 14, it’s right time to explore C++17+ — your productivity and code clarity will thank you.

