Since you're learning **C++17**, here’s a structured list of topics you should cover to master the language from basics to advanced levels.

**1. Basics of C++**

* Syntax and structure
* Variables and data types
* Input and output (cin, cout)
* Operators (arithmetic, logical, bitwise)
* Control structures (if-else, switch-case)
* Loops (for, while, do-while)
* Functions and function overloading

**2. Memory Management**

* Pointers and references
* Dynamic memory allocation (new, delete)
* Smart pointers (unique\_ptr, shared\_ptr, weak\_ptr)
* Stack vs Heap memory

**3. Object-Oriented Programming (OOP)**

* Classes and objects
* Constructors and destructors
* Access specifiers (public, private, protected)
* Encapsulation, Inheritance, Polymorphism, Abstraction
* Virtual functions and function overriding
* Friend functions and classes
* Operator overloading

**4. C++ Standard Library (STL)**

* Containers:
  + Vector, List, Deque
  + Set, Multiset
  + Map, Multimap
  + Stack, Queue, Priority Queue
* Iterators and Algorithms
* Lambda functions
* Sorting and Searching algorithms in STL

**5. File Handling**

* Reading/Writing files (ifstream, ofstream, fstream)
* Binary file handling
* Error handling in file operations

**6. Templates & Generic Programming**

* Function templates
* Class templates
* Template specialization
* Variadic templates

**7. Exception Handling**

* Try, Catch, Throw
* Custom exception classes
* Stack unwinding

**8. Multi-threading and Concurrency**

* Threads (std::thread)
* Mutex and locks (std::mutex, std::lock\_guard, std::unique\_lock)
* Condition variables
* Atomic operations

**9. Advanced C++17 Features**

* Structured bindings
* std::optional, std::variant, std::any
* std::filesystem
* std::string\_view
* Fold expressions
* Parallel algorithms (std::execution::par)

**10. Debugging & Profiling**

* Debugging using GDB or VS Code debugger
* Memory leak detection (Valgrind, AddressSanitizer)
* Performance optimization techniques

**11. System Programming & Embedded C++**

* Interfacing with hardware
* Low-level bit manipulation
* Real-time constraints

**12. Design Patterns & Best Practices**

* Singleton, Factory, Observer, Strategy, etc.
* RAII (Resource Acquisition Is Initialization)
* Modern C++ best practices (Avoiding raw pointers, using smart pointers, etc.)

**13. Competitive Programming (Optional)**

* Fast input/output techniques
* Graph algorithms (DFS, BFS, Dijkstra)
* Dynamic programming
* Bit manipulation tricks

**14. Build Systems & Tools**

* CMake, Makefiles
* Version control (Git, GitHub)
* Continuous Integration (CI/CD)
* Unit testing frameworks (Google Test, Catch2)

This roadmap covers **everything from beginner to expert level**. Let me know if you want a **day-wise** or **project-based** learning plan! 🚀