Here's a **detailed C++ Developer Syllabus** tailored for **getting a job at Google**, especially for roles involving **system-level programming, performance-critical applications, or large-scale infrastructure**. Google expects deep understanding of both **C++ language internals** and **computer science fundamentals**.

□ 1. Computer Science Fundamentals

Google prioritizes strong fundamentals before language-specific skills.

📌 Data Structures

- Arrays, Linked Lists (Singly, Doubly)
- Stacks, Queues, Deques
- Trees (Binary Tree, BST, AVL, Segment Tree, Fenwick Tree)
- Graphs (Adjacency List/Matrix, DFS, BFS)
- Heaps (Min/Max Heap, Priority Queue)
- Hashing (unordered_map, map, set)
- Tries, Disjoint Set Union (Union-Find)

📌 Algorithms

- Sorting (Quick, Merge, Heap, Counting)
- Searching (Binary Search, Ternary Search)
- Recursion & Backtracking
- Greedy Algorithms
- Dynamic Programming (Knapsack, LIS, LCS, Memoization, Tabulation)
- Graph Algorithms:
    - Dijkstra, Bellman-Ford, Floyd-Warshall
    - Prim’s and Kruskal’s MST
    - Topological Sort, Tarjan’s/Kosaraju’s SCC
- Bit Manipulation
- Sliding Window, Two Pointers
- Divide & Conquer
- Math:
    - GCD, LCM
    - Modular Arithmetic
    - Sieve of Eratosthenes
    - Combinatorics (nCr, Catalan Numbers, Pigeonhole, Inclusion-Exclusion)
    - Matrix Exponentiation

💻 2. C++ Language Mastery

□ Core Language Concepts

- Data types, Operators, Control Flow
- Functions (inline, default args, overloading)
- Pointers and References
- Memory Allocation (new/delete, malloc/free)
- RAII and smart pointers (`unique_ptr`, `shared_ptr`)
- Const correctness
- Namespaces and Scope Resolution
- Preprocessor directives (`#define`, `#ifdef`, macros)

🔄 Object-Oriented Programming

- Classes and Objects
- Access Specifiers (`private`, `protected`, `public`)
- Constructors/Destructors
- Inheritance (single, multiple, multilevel, virtual)
- Polymorphism (compile-time: overloading, run-time: virtual functions)
- Encapsulation, Abstraction
- Operator Overloading
- Virtual Tables and Pure Virtual Functions
- Abstract Classes
- Diamond Problem and Virtual Inheritance

📚 STL (Standard Template Library)

- Containers: `vector`, `deque`, `list`, `map`, `set`, `unordered_map`, `unordered_set`, `stack`, `queue`, `priority_queue`
- Iterators
- Algorithms: `sort`, `find`, `lower_bound`, `upper_bound`
- Function Objects & Lambda Functions
- Custom comparators and hashing

Templates and Meta-Programming

- Function and Class Templates
- Template Specialization
- Variadic Templates
- SFINAE (Substitution Failure Is Not An Error)
- `decltype`, `typeid`, `auto`
- Concepts (C++20)

⚙ Advanced C++

- Memory Management (heap/stack, leaks, valgrind)
- Move Semantics and Rvalue References
- Rule of 3 / Rule of 5
- Copy elision and Return Value Optimization (RVO)
- C++11 to C++23 features (lambdas, `constexpr`, `nullptr`, `auto`, smart pointers, ranges, coroutines, modules)
- Multithreading and Concurrency:

    - `std::thread`, `mutex`, `condition_variable`
    - Atomic variables
    - Race conditions, deadlocks, thread safety

---

3. System Design & Low-Level Knowledge

For infra-related roles:

- Operating Systems (process, thread, scheduling, memory layout, syscalls)
- Computer Networks (TCP/IP, DNS, HTTP basics)
- Linux programming (file descriptors, sockets)
- Compiler Design Basics (compilation phases, optimization)
- Assembly/C basics are a bonus
- Performance Optimization:
    - Time/Space tradeoffs
    - Cache locality, paging
    - Profiling tools (`gprof`, `perf`)
    - Code size & binary size tuning

---

4. Tooling & Development Practices

- Git and version control
- Build systems (`make`, `cmake`, `bazel`)
- Unit Testing frameworks (`gtest`, `catch2`)
- Debugging tools (`gdb`, `valgrind`)
- Code style and code reviews
- Continuous Integration (CI/CD basics)

---

5. Preparation Resources

Books

- *Effective C++* by Scott Meyers
- *C++ Primer* by Lippman
- *C++ Concurrency in Action* by Anthony Williams
- *Design Patterns* by GoF
- *Cracking the Coding Interview* by Gayle Laakmann McDowell
- *Elements of Programming Interviews* (C++ Edition)

Practice Platforms

- LeetCode (must)
- Codeforces / AtCoder (competitive programming)
- HackerRank / GeeksforGeeks
- InterviewBit (DSA practice)

□ Google Interview Process (Typically)

1. **Online Assessment / Phone Screen**
    - 2–3 DSA problems, typically on LeetCode Hard/Medium level.
2. **Technical Interviews (2–4 rounds)**
    - DSA-focused, with coding in C++.
    - Some rounds may test C++ internals or system-level concepts.
3. **Onsite (now virtual)**
    - System design for senior roles.
    - Behavioral/Googliness round.