Perfect — cracking system design and AI roles at top companies (like Microsoft, NVIDIA, etc.) **requires mastery of algorithms and data structures (DSA)** alongside design patterns and C++. Let’s update your roadmap to include a full **DSA + C++ interview prep track**.

**🔁 Updated 14-Week Plan: C++ + Design Patterns + AI Infra + DSA**

**📘 Phase 1: Modern C++ Fundamentals (Weeks 1–3)**

Master the basics and intermediate concepts from Bjarne Stroustrup’s book.

**Focus Areas:**

* Syntax, types, memory model
* Classes, constructors, destructors
* Templates, smart pointers (unique\_ptr, shared\_ptr)
* STL (vector, map, set, unordered\_map, queue, stack)
* File I/O, error handling

**Mini Projects:**

* Bank simulator
* CLI Task manager
* Text-based menu system

**📦 Phase 2: Data Structures & Algorithms (Weeks 4–7)**

Prepare for **FAANG-level interviews** and system-intensive C++ roles.

**Topics (with C++ practice):**

| **Week** | **Topics** | **Sample Problems** |
| --- | --- | --- |
| 4 | Arrays, Strings, HashMaps | Two Sum, Group Anagrams |
| 5 | Recursion, Linked Lists | Merge K Lists, Detect Cycle |
| 6 | Trees, Binary Search Trees | Inorder, LCA, Serialize Tree |
| 7 | Graphs, Heaps, Greedy | Dijkstra, Topo Sort, Kth Largest |

**Platforms:**

* [LeetCode](https://leetcode.com/)
* [InterviewBit](https://www.interviewbit.com/)
* [GeeksForGeeks](https://www.geeksforgeeks.org/data-structures/)

**Target:** 5–10 high-quality problems per week in C++ with detailed comments

**🧱 Phase 3: Design Patterns in C++ (Weeks 8–9)**

Learn reusable system design logic and best practices.

**Focus Patterns:**

* Factory, Singleton, Builder
* Strategy, Observer, Command
* Decorator, Proxy, Composite

**Projects:**

* Plugin-based command line tool
* Event-driven trading simulation
* UI component renderer in C++

**🧠 Phase 4: AI Infra & System Design (Weeks 10–12)**

Combine your AI knowledge with system-level thinking.

**Topics:**

* Embedding services, LLM API wrappers
* Inference-serving pipelines (ONNX, Triton)
* Vector DBs (FAISS/Chroma), semantic search
* Redis, Message Queues, Azure Blob/S3
* Multi-threaded C++ programming

**Mini Projects:**

* AI Chatbot server: FastAPI + C++ backend
* C++ inference engine using ONNX
* FAISS-based search engine

**🏗️ Phase 5: Final Capstone Project (Weeks 13–14)**

Combine React + Python + C++ to build a full-stack AI application.

**Examples:**

* "AI PDF Summarizer": Upload PDF → C++ chunking → Python LLM → React UI
* "Voice Bot": C++ for audio I/O + NLP → FastAPI API → React dashboard

Deploy using Docker, and optionally Azure or AWS.

**🎯 Final Goals & Results**

| **Goal** | **Outcome** |
| --- | --- |
| Strong in C++, STL, and memory handling | Confidence in low-level & system design |
| Mastery of DSA | Ready for FAANG/Microsoft interviews |
| Expertise in design patterns | Architect-level coding and abstractions |
| Full-stack AI+Infra projects | Showcases modern enterprise experience |
| Blog + GitHub portfolio | Stand out to hiring managers & recruiters |

Would you like:

* A **Notion tracker**
* A **GitHub folder template**
* Or a **PDF roadmap** of this plan?

Also, I can give you **50 DSA questions to solve in C++** tailored for your level. Let me know!

Here’s a **detailed 3-week schedule for Phase 4: AI Infra & System Design**, integrating GPU programming, parallel computing, AI framework internals, and deployment—all tailored for your goals:

**🧠 Phase 6 Detailed Weekly Schedule (Weeks 10–12)**

**Week 10: GPU Programming Fundamentals (CUDA & HIP)**

| **Day** | **Focus Area** | **Tasks / Resources** | **Deliverables** |
| --- | --- | --- | --- |
| 1 | Introduction to GPU architecture | Read NVIDIA CUDA programming guide basics | Notes on GPU memory & compute model |
| 2 | CUDA programming basics | Write simple CUDA kernel: vector addition | Vector addition CUDA kernel |
| 3 | HIP introduction (AMD GPU analog) | Explore HIP programming guide, port CUDA kernel to HIP | HIP version of vector addition |
| 4 | GPU memory types & optimization | Study shared, global, constant memory & thread synchronization | Optimized CUDA kernel using shared mem |
| 5 | Simple AI-related kernel | Implement matrix multiplication (GEMM) on GPU | GEMM CUDA kernel |
| 6 | Debugging & profiling GPU code | Use NVIDIA Nsight or ROCm tools to profile kernels | Profiling report & kernel improvements |
| 7 | Recap and mini project integration | Combine vector add + GEMM kernels into one app | Mini app with multiple kernels |

**Week 11: Parallel & Distributed Computing in C++**

| **Day** | **Focus Area** | **Tasks / Resources** | **Deliverables** |
| --- | --- | --- | --- |
| 1 | Multithreading basics in C++ | Write multi-threaded CPU code using std::thread & mutex | Multi-threaded vector addition |
| 2 | Synchronization & thread safety | Use mutex, condition\_variable to build thread-safe queue | Thread-safe task queue |
| 3 | OpenMP introduction | Parallelize a for-loop with OpenMP | OpenMP-parallelized matrix multiply |
| 4 | MPI basics & setup | Write basic MPI "hello world" & simple communication program | MPI send/receive demo |
| 5 | Distributed vector search demo | Simulate distributed vector search across multiple processes | Distributed vector similarity search |
| 6 | Combining GPU + multithreading | Build multi-threaded host code launching GPU kernels | Multi-threaded GPU inference driver |
| 7 | Recap & documentation | Document key learnings, challenges, and performance results | Detailed project README |

**Week 12: AI Framework Internals & Deployment**

| **Day** | **Focus Area** | **Tasks / Resources** | **Deliverables** |
| --- | --- | --- | --- |
| 1 | PyTorch custom operator basics | Write a simple C++ custom operator | Custom PyTorch operator repo |
| 2 | Extending TensorFlow or JAX | Explore extending TF ops or JAX primitives | Sample TF op or JAX function |
| 3 | Profiling AI models | Use torch.profiler, NVIDIA Nsight for end-to-end profiling | Profiling report |
| 4 | Containerization basics | Dockerize a small AI inference service | Dockerfile + dockerized app |
| 5 | Kubernetes introduction | Setup local k8s with Minikube; deploy your dockerized service | Mini Kubernetes deployment |
| 6 | CI/CD pipelines for ML infra | Create simple GitHub Actions pipeline for automated tests | GitHub Actions YAML file |
| 7 | Capstone prep & presentation | Prepare README, docs, and demo script for your phase 4 projects | Complete project repo + presentation |

If you want, I can also prepare **daily task checklists, sample code snippets, and resource links** for each day.

Would you like me to do that?

Got it! Here’s an expanded **6-week detailed plan** for **Phase 1: Modern C++ Fundamentals**, paced for deeper understanding and more practice, including mini projects spread out for hands-on learning.

Absolutely! Here is your **complete and detailed 6-week Phase 1 plan for Modern C++ Fundamentals**, expanded slightly to **include all essential topics** a solid foundation requires, while keeping your original structure and flow intact.

Absolutely! Here’s your **6-week detailed plan** for Phase 1: Modern C++ Fundamentals — now with a **practice strategy included daily** to ensure hands-on learning and retention.

**Phase 1: Modern C++ Fundamentals (Weeks 1–6)**

**Each day includes a focused study topic + targeted practice tasks + mini challenges for mastery.**

**Week 1: C++ Basics & Syntax, Types, Memory Model**

Goal: Build a strong foundation on syntax, types, variables, memory management basics.

| **Day** | **Task & Topics** | **Practice Strategy** |
| --- | --- | --- |
| Day 1 | Setup environment, compile/run your first C++ program ("Hello World") | Write, compile & run "Hello World"; modify to print your name |
| Day 2 | Variables, primitive types (int, char, float, bool), constants | Declare variables of each type; print values; try changing values |
| Day 3 | Operators, expressions, basic input/output (cin, cout) | Build simple calculator: input 2 numbers, output sum/difference |
| Day 4 | Memory model: stack vs heap, pointers basics, pointer arithmetic | Write code to print addresses; pointer dereference; pointer arithmetic exercises |
| Day 5 | Pointer exercises & dynamic memory allocation (new, delete) | Allocate int array dynamically; write code to delete safely; null pointer checks |
| Day 6 | References: usage and difference from pointers | Write function to swap two variables using references; increment variables |
| Day 7 | Review: write small programs combining pointers & references | Mini-challenge: create linked list node with pointers and references |

**Week 2: Classes, Constructors, Destructors**

Goal: Learn OOP fundamentals and resource management.

| **Day** | **Task & Topics** | **Practice Strategy** |
| --- | --- | --- |
| Day 8 | Classes: data members, member functions, access specifiers | Create simple class (e.g., Person); add methods to set/get data |
| Day 9 | Constructors: default, parameterized, copy constructors | Implement each constructor; create multiple objects; print info |
| Day 10 | Destructors and resource cleanup | Add destructor freeing dynamic resources; test object lifecycle |
| Day 11 | Practice: classes representing real-world entities (Bank Account) | Build Bank Account with deposit, withdraw, balance methods |
| Day 12 | Static members, const functions, this pointer | Add static member tracking total accounts; const getter functions |
| Day 13 | Object lifetime: stack vs heap allocation | Create objects on stack and heap; observe destruction timing |
| Day 14 | Review & Mini project: start Bank Account class | Build main features, test with sample accounts |

**Week 3: Templates, Smart Pointers & Move Semantics**

Goal: Generic programming, modern memory management, efficient resource handling.

| **Day** | **Task & Topics** | **Practice Strategy** |
| --- | --- | --- |
| Day 15 | Function templates: syntax and examples | Write generic swap function; test with different types |
| Day 16 | Class templates: generic containers/utilities | Create simple templated container class; insert and retrieve data |
| Day 17 | unique\_ptr: ownership semantics & usage | Replace raw pointers with unique\_ptr in small programs |
| Day 18 | shared\_ptr and weak\_ptr: reference counting | Build shared ownership example; observe destruction timing |
| Day 19 | Practice: combine templates and smart pointers | Write templated class managing smart pointers; test different types |
| Day 20 | Exception safety & smart pointers | Write code that throws exceptions; use smart pointers safely |
| Day 21 | Move semantics & rvalue references | Implement move constructor/assignment; compare performance vs copy |

**Week 4: STL Containers Part 1 — Vector, Map, Set, Lambdas & Algorithms**

Goal: Master commonly used STL containers, lambdas, and algorithms.

| **Day** | **Task & Topics** | **Practice Strategy** |
| --- | --- | --- |
| Day 22 | STL overview: container types | Create examples using vector, map, set |
| Day 23 | std::vector: dynamic arrays, modifying elements | Write vector usage code; insert, erase, iterate |
| Day 24 | std::map & std::unordered\_map: key-value storage | Count word frequencies in text using map/unordered\_map |
| Day 25 | std::set & std::unordered\_set: unique elements | Remove duplicates from vector using sets; practice insert/search |
| Day 26 | Iterators and range-based for loops | Iterate with different methods; practice modifying container data |
| Day 27 | Practice programs combining vector, map, set | Write small projects using multiple containers |
| Day 28 | Lambdas and STL algorithms (std::sort, std::find\_if, std::accumulate) | Sort custom objects using lambdas; filter and accumulate values |

**Week 5: STL Containers Part 2 — Queue, Stack, File I/O, Error Handling**

Goal: Advanced STL containers, file I/O, exception handling.

| **Day** | **Task & Topics** | **Practice Strategy** |
| --- | --- | --- |
| Day 29 | std::queue and std::stack: usage & underlying containers | Implement queue/stack problems: task scheduling, parentheses match |
| Day 30 | Solve queue/stack based problems | Write balanced parentheses checker; simulate task queues |
| Day 31 | File I/O basics: reading/writing files (ifstream, ofstream) | Write programs saving/loading text data |
| Day 32 | Practice file I/O: Bank Simulator persistence | Save/load account data to file |
| Day 33 | Exception handling: try-catch, throw, standard exceptions | Write code with exception handling for invalid input or file errors |
| Day 34 | Write exception-safe code using RAII and smart pointers | Practice writing safe code with resources that might throw |
| Day 35 | Review & mini project: Add file I/O and error handling to Bank Simulator | Test and debug file handling with error scenarios |

**Week 6: Mini Projects & Integration, Optional Advanced Topics**

Goal: Complete projects, consolidate learning, optional multithreading and unit testing.

| **Day** | **Task & Topics** | **Practice Strategy** |
| --- | --- | --- |
| Day 36 | Start CLI Task Manager: define Task class and task container | Design classes; write add/remove/display functions |
| Day 37 | Implement adding, removing, displaying tasks | Interactive console input/output |
| Day 38 | Add persistence with file I/O | Save/load task list to file |
| Day 39 | Build text-based menu system | Write menu-driven program to call task manager functions |
| Day 40 | Test CLI Task Manager thoroughly | Write test cases, handle invalid input, edge cases |
| Day 41 | Finalize Bank Simulator: polish code, add comments | Refactor code for clarity, add documentation |
| Day 42 | Final review: summarize, document your learning | Write blog post or detailed notes; optionally start unit tests |
|  | **Optional:** Intro to multithreading basics (std::thread, mutex) | Create simple threaded programs |
|  | **Optional:** Basics of unit testing (Google Test, Catch2) | Write unit tests for Bank Simulator or Task Manager |

If you'd like, I can now create **Day 1 detailed lesson + exercises + sample code** — just say the word!

**Extra Notes**

* Aim to **code daily** for at least 1–2 hours.
* Use **Bjarne Stroustrup’s book** as the main theory resource.
* Supplement learning with **cppreference.com** for STL and modern C++ details.
* Version control your projects with **Git/GitHub** to track progress and build portfolio.
* Reach out with questions or code reviews anytime!

Would you want me to help you with daily practice problems or mini project starter code as you go?

**📦 Phase 2: Data Structures & Algorithms (Weeks 4–7)**

Sure! Here's a detailed **6-week daily schedule** for **Phase 2: Data Structures & Algorithms** in a table format, with specific daily tasks to guide your C++ practice and problem-solving.

| **Week** | **Day** | **Topics / Tasks** | **Sample Problems / Practice** |
| --- | --- | --- | --- |
| **4** | 1 | Study arrays: basics, declaration, traversal | Solve **Two Sum** (LeetCode) |
|  | 2 | Strings: operations, substrings, comparison | Solve **Valid Anagram** |
|  | 3 | HashMaps (unordered\_map in C++): usage, frequency counting | Solve **Group Anagrams** |
|  | 4 | More on hash maps: collisions, common patterns | Solve **Longest Substring Without Repeating Characters** |
|  | 5 | Practice mixed array/string/hashmap problems | Solve 2 easy-medium problems from LeetCode |
|  | 6 | Analyze time and space complexity of your solutions | Review all Week 4 solutions, write comments in code |
|  | 7 | Rest / Optional: revisit tough problems | Reread C++ STL docs on vectors, strings, and unordered\_map |
| **5** | 8 | Recursion basics: understand call stack and function calls | Solve simple recursive problem (e.g., factorial, fibonacci) |
|  | 9 | Linked Lists: singly linked list insertion/deletion | Implement basic linked list in C++ |
|  | 10 | Cycle detection in linked lists (Floyd’s algorithm) | Solve **Detect Cycle** problem |
|  | 11 | Merge K sorted linked lists: understand merge process | Solve **Merge K Sorted Lists** |
|  | 12 | Linked list reversal and pointer manipulation | Solve linked list reversal problem |
|  | 13 | Practice 2-3 linked list problems | Solve medium linked list problems |
|  | 14 | Review linked list and recursion concepts | Write comments explaining recursive calls and pointers |
| **6** | 15 | Trees basics: define tree node, simple traversal (inorder recursive) | Implement inorder traversal |
|  | 16 | Preorder and postorder traversals (recursive and iterative) | Implement preorder and postorder traversal |
|  | 17 | Binary Search Tree (BST): insertion, search, validation | Implement BST insertion/search, solve Validate BST problem |
|  | 18 | Lowest Common Ancestor (LCA) in BST | Solve LCA problem |
|  | 19 | Serialize and deserialize binary tree | Implement serialize/deserialize algorithms |
|  | 20 | Practice 3-4 tree problems | Solve tree traversal and BST problems |
|  | 21 | Review trees & BSTs, write detailed comments | Refactor and comment tree code |
| **7** | 22 | Graphs basics: adjacency list/matrix representation | Implement graph representation |
|  | 23 | Graph traversal: BFS and DFS | Implement BFS and DFS |
|  | 24 | Dijkstra’s shortest path algorithm | Implement Dijkstra’s algorithm |
|  | 25 | Topological sort (Kahn’s algorithm and DFS approach) | Solve Topological sort problem |
|  | 26 | Heaps and priority queues (priority\_queue in C++) | Solve Kth largest element problem |
|  | 27 | Greedy algorithms basics and patterns | Solve 2 greedy algorithm problems |
|  | 28 | Review and practice: graph + heap + greedy | Review all Week 7 problems and write comments |
| **8** | 29 | Dynamic Programming (DP) basics: memoization and tabulation | Solve Climbing Stairs (DP problem) |
|  | 30 | Sliding window technique and two pointers | Solve Longest Substring Without Repeating Characters |
|  | 31 | Union-Find / Disjoint Set Union (DSU) structure | Implement Union-Find structure |
|  | 32 | Practice problems using DP, sliding window, and Union-Find | Solve Number of Islands, Container With Most Water |
|  | 33 | Mixed practice day | Solve 2 medium-hard problems combining patterns |
|  | 34 | Review and optimize previous solutions | Refactor code for efficiency and clarity |
|  | 35 | Rest / Optional: revisit toughest problems | Prepare summary notes |
| **9** | 36 | Review toughest problems from Weeks 4–8 | Solve 2-3 hard problems under timed conditions |
|  | 37 | Mock interview: simulate coding interviews (use LeetCode or InterviewBit) | Time yourself solving problems |
|  | 38 | System design basics overview | Read about load balancing, caching, and sharding |
|  | 39 | Practice explaining your solutions aloud or writing detailed comments | Explain algorithms and trade-offs |
|  | 40 | Write a cheat sheet summarizing key algorithms and data structures | Compile notes for quick review |
|  | 41 | Final problem solving and polishing code | Refine all previous code with clean style |
|  | 42 | Rest and prepare for next learning phase | Reflect on progress and plan ahead |

If you want, I can provide detailed daily problem statements with starter C++ code and hints. Would you like to start with Week 4 Day 1 task now?