C++ Quant & HFT Systems — Detailed Assignment Roadmap

This document contains rigorous, professor-level programming assignments designed to build elite C++ competence for Quantitative Development, HFT Systems Engineering, and Low-Latency Trading roles. You are expected to reason about correctness, performance, memory, and edge cases in every assignment.

LEVEL 0 — Computational & Syntax Foundations

Assignment 0.1 — Numeric Reasoning Engine:
Implement a program that takes an integer N ($1 \le N \le 10^9$) and outputs:
a) Number of digits
b) Sum of digits
c) Whether N is a palindrome
d) Whether N is prime
Constraints: No strings, no STL, no math library. All operations must be integer-based.
Evaluation focuses on edge cases (N=0, N=1, large primes).

Assignment 0.2 — Base Conversion Simulator:
Write a program that converts numbers between decimal, binary, and hexadecimal.
You must manually compute powers and remainders. Bitwise operators allowed only after first implementation.
Explain time complexity and overflow risks in comments.

Assignment 0.3 — Pattern Stress Test:
Generate 10 different numeric and geometric patterns based on input size.
Patterns must scale correctly for N up to 200.
Focus: loop invariants and boundary control.

LEVEL 1 — Functions, Recursion & Control Abstraction

Assignment 1.1 — Mathematical Utility Library:
Design and implement your own versions of pow, abs, min, max, gcd, lcm.
Each function must handle invalid input safely.
Include a test harness that validates correctness across 1000 random inputs.

Assignment 1.2 — Recursive vs Iterative Analysis:
Implement factorial, Fibonacci, and power functions using both recursion and iteration.
Measure stack depth, execution time, and explain tradeoffs.
Explicitly identify tail recursion opportunities.

Assignment 1.3 — Overflow Detection:
Write safe addition and multiplication functions that detect signed integer overflow without using larger data types.

LEVEL 2 — Arrays & Strings (No STL)

Assignment 2.1 — Array Rotation Engine:
Rotate an array of size N by K positions left and right.
Implement at least two algorithms and compare time and space complexity.

Assignment 2.2 — Custom String Library:
Implement strlen, strcpy, strcmp, strcat from scratch.
You must handle overlapping memory and null-termination correctly.

Assignment 2.3 — Substring Search:
Implement naive substring search.
Discuss worst-case complexity and provide pathological test cases.

LEVEL 3 — Pointers & Manual Memory Management

Assignment 3.1 — Dynamic Array Class:
Implement a resizable integer array using new/delete.
Support push_back, pop_back, resize, and operator[].
Demonstrate and fix memory leaks.

Assignment 3.2 — Memory Debugging Exercise:
You are given a deliberately buggy program.
Identify dangling pointers, double frees, and memory corruption.
Provide a written postmortem.

Assignment 3.3 — Pointer Arithmetic Lab:
Traverse 1D and 2D arrays exclusively using pointers.
Explain address calculations.

LEVEL 4 — Object-Oriented Design & Lifetime Semantics

Assignment 4.1 — Financial Instrument Class Hierarchy:
Design a base Instrument class and derive Equity and Future classes.
Demonstrate polymorphism and virtual destructors.

Assignment 4.2 — Copy Semantics Audit:
Implement copy constructor and copy assignment for a class managing heap memory.
Demonstrate object slicing and prevent it.

Assignment 4.3 — Const Correctness Enforcement:
Refactor an existing codebase to enforce const correctness everywhere.

LEVEL 5 — RAII & Operator Overloading

Assignment 5.1 — RAII Lock Guard:
Implement a RAII-based mutex lock guard.
Demonstrate exception safety.

Assignment 5.2 — Numeric Wrapper Type:
Create a class that overloads arithmetic and comparison operators.
Ensure no unnecessary temporaries are created.

Assignment 5.3 — Rule of Five Compliance:
Design a resource-owning class and correctly implement all five special member functions.

LEVEL 6 — Templates & STL Internals

Assignment 6.1 — Templated Vector:
Implement a simplified std::vector with iterator support.
Support custom allocators.

Assignment 6.2 — Compile-Time Polymorphism:
Use templates to eliminate virtual dispatch.
Measure performance difference.

Assignment 6.3 — STL Complexity Analysis:
Analyze and document time/space complexity of vector, map, unordered_map under various workloads.

LEVEL 7 — Algorithms & Data Structures for Trading

Assignment 7.1 — Order Book Core:
Implement a price-time priority order book.
Support add, cancel, and match operations.

Assignment 7.2 — Tree-Based Index:
Implement a balanced BST and benchmark against sorted vectors.

Assignment 7.3 — Graph Latency Model:
Model network hops using graphs and compute shortest paths.

LEVEL 8 — Modern C++ & Concurrency

Assignment 8.1 — Move Semantics Lab:
Instrument constructors to count copies vs moves.
Eliminate unnecessary copies.

Assignment 8.2 — Thread-Safe Counter:
Implement and benchmark mutex vs atomic versions.
Analyze false sharing.

Assignment 8.3 — Async Task Engine:
Design a small task scheduler using std::async and futures.

LEVEL 9 — HFT Systems & Performance (God Level)

Assignment 9.1 — Custom Memory Allocator:
Implement a fixed-size block allocator optimized for cache lines.

Assignment 9.2 — Event-Driven Trading Loop:
Build a single-threaded event loop handling market data and orders.
Measure end-to-end latency.

Assignment 9.3 — Assembly & Cache Analysis:
Compile hot paths with different optimization levels.
Inspect generated assembly and explain differences.