Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

C++ and Object-Oriented Programming

Class Duration

35 hours of live training delivered over 5 days.

Student Prerequisites

- Professional programming experience in at least one language (C, C#, Java, Python, JavaScript, or equivalent)
- Familiarity with basic data structures and functions
- No prior C++ or formal object-oriented design experience required

Target Audience

Professional developers who need to become productive C++ programmers, whether coming from C, a managed language like C# or Java, or a scripting background. The course teaches C++ as it is written today, with object-oriented design as the organizing thread, and is ideal for teams inheriting C++ codebases, building performance-critical components, or integrating with native libraries. Developers already comfortable in C++ who want to modernize their habits should consider Effective Modern C++ instead.

Description

This hands-on course teaches modern C++ from the ground up, against the C++26 standard, with object-oriented programming taught the way contemporary C++ actually practices it: value semantics and RAII first, inheritance where it earns its place. Participants progress from the language core and C++'s distinctive memory model through class design, polymorphism, generic programming with templates and concepts, and the standard library's containers and algorithms, finishing with the testing and tooling practices of professional C++ teams. Every topic is exercised against realistic code, with agentic AI coding assistants used throughout for explaining unfamiliar constructs, generating tests, and refactoring, the way modern C++ teams actually work.

C++26 was finalized by the ISO committee in March 2026. Coverage of individual C++26 features reflects compiler support at delivery time.

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Learning Outcomes

- Explain C++'s compilation model, memory model, and the philosophy behind value semantics.
- Write idiomatic C++ using references, smart pointers, and RAII-based resource management.
- Design robust classes with correct construction, destruction, copying, and moving behavior.
- Apply inheritance and runtime polymorphism appropriately, and recognize when composition is the better tool.
- Build generic code with templates and constrain it with concepts.
- Use standard library containers, algorithms, and ranges effectively.
- Handle errors with exceptions, std::optional, and std::expected, and know when each fits.
- Set up and use professional C++ tooling: CMake, a package manager, sanitizers, and a unit testing framework.
- Direct agentic AI coding assistants productively in C++ work: explaining code, generating tests, and proposing refactorings for review.

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Software Requirements

Visual Studio 2026 (Windows) or VS Code with a current GCC or Clang toolchain (any platform), plus CMake and Git. Portions of the course covering C++26-specific features use GCC or Clang where MSVC support lags. Access to an agentic AI coding assistant is required; a free tier is sufficient.

Training Topics

C++ Foundations

- The compilation model: headers, modules, translation units
- Toolchain setup: Visual Studio 2026, GCC, Clang, CMake
- Types, variables, and initialization
- Value semantics vs. reference semantics
- Agentic AI coding assistants in C++ workflows

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Language Core

- Functions, overloading, and default arguments
- References and const correctness
- Namespaces and code organization
- Enumerations and type-safe alternatives to macros
- Error handling fundamentals

Memory and Resource Management

- Stack, heap, and object lifetime
- Pointers, references, and ownership
- RAII: resources tied to scope
- unique_ptr, shared_ptr, and weak_ptr
- Avoiding leaks, dangling pointers, and double frees

Classes and Encapsulation

- Class design: invariants and interfaces
- Constructors, destructors, and member initialization
- Copy and move semantics; the rule of zero and rule of five
- Operator overloading done tastefully
- Static members and friend functions

Inheritance and Polymorphism

- Base classes and virtual functions
- Abstract classes and interface design
- Overriding, final, and virtual destructors
- Object slicing and how to prevent it
- The costs of virtual dispatch

Composition and Object-Oriented Design

- Composition over inheritance
- SOLID principles in C++ terms
- Dependency injection without a framework
- Interfaces as seams for testing
- Common design patterns in modern C++

Templates and Generic Programming

- Function and class templates
- Template argument deduction and CTAD

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

- Concepts and constrained templates
- Generic programming vs. OO polymorphism: choosing between them

The Standard Library

- Containers: vector, map, unordered_map, and friends
- Iterators and the algorithms library
- Ranges and views
- Strings, string_view, and text handling
- Utility types: optional, variant, expected

Error Handling and Robustness

- Exception safety guarantees
- noexcept and its implications
- std::optional and std::expected as alternatives
- Contracts in C++26: pre, post, and contract_assert
- Defensive coding vs. design by contract

Testing and Professional Tooling

- Unit testing with GoogleTest or Catch2
- CMake project structure and presets
- Dependency management with vcpkg or Conan
- Sanitizers: address, undefined behavior, threads
- Static analysis with clang-tidy

Object-Oriented Design in Practice

- Designing class hierarchies for real domains
- Refactoring toward better OO structure
- API design and header hygiene
- Reviewing AI-generated C++ for correctness and style

