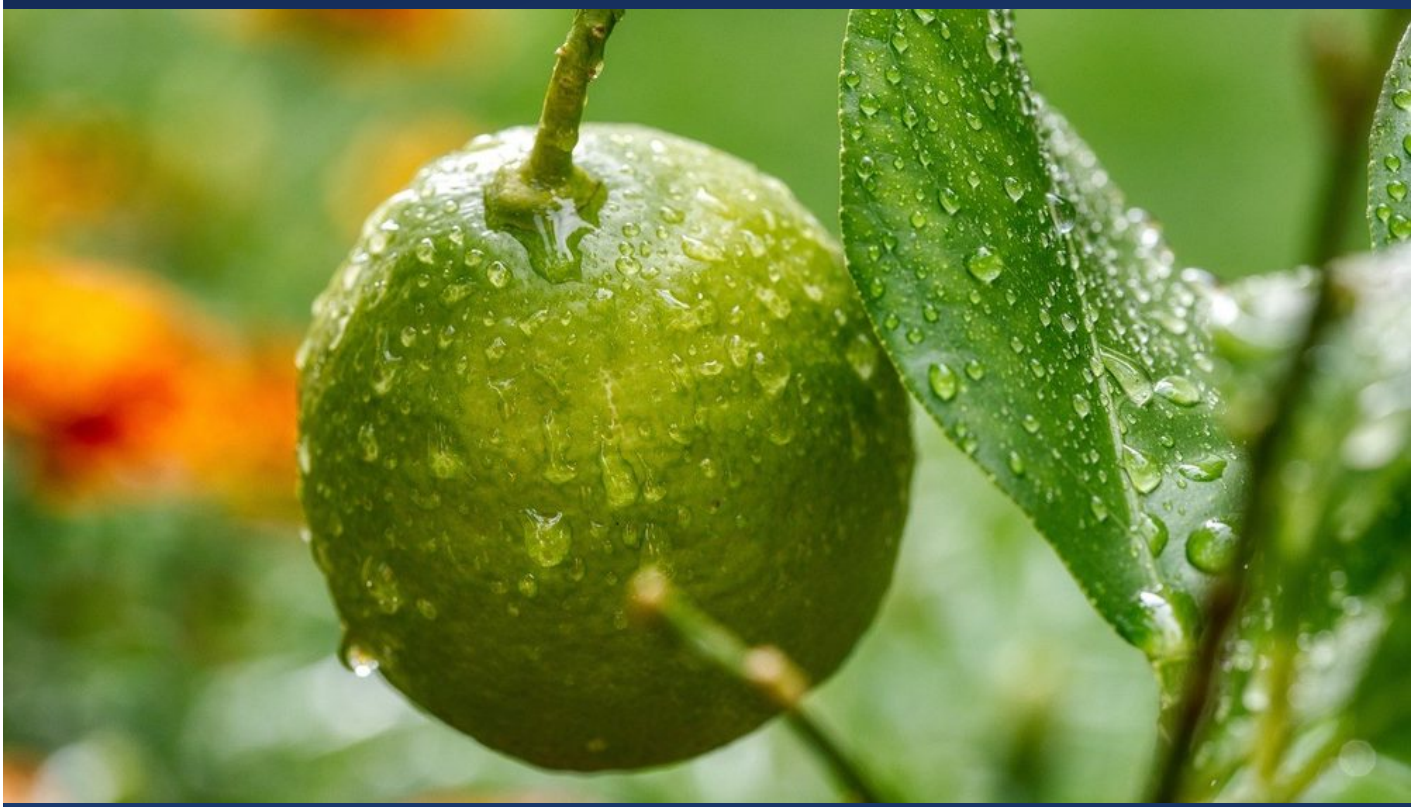
LAPT

LONDON ACADEMY OF PROFESSIONAL TRAINING

CERTIFICATE

ISO 14882 — C++ Programming Language Standard

ISO Certification Programme



LAPT — London Academy of Professional Training

Address 85 Great Portland Street, W1W 7LT, London, United Kingdom

Email info@lapt.org

Telephone +44 7513 283044

Web lapt.org

Reference IIT-COD-14882 • ISO IT & Related Technologies

UKPRN 10067351 • Registered in England and Wales, company 4984798

6	20	108	430	60%	Lifetime
SUBJECTS	CHAPTERS	CLASSES	TOTAL MARKS	PASS MARK	VALIDITY

AT A GLANCE

REFERENCE	**IIT-COD-14882**
AWARD	**Certificate**
ASSESSMENT	**430 marks — 258 to pass (60%)**
PREREQUISITES	**None**
VALIDITY	**Lifetime**
AWARDING BODY	**LAPT — London Academy of Professional Training**

CURRICULUM

1 Advanced C++ Programming Techniques

5 chapters · 30 classes **85 marks**

• 1 Mastering Advanced Template Programming in C++ — 6 classes

- › 1.1 Understanding Template Basics in C++
- › 1.2 Exploring Function Templates and Their Applications
- › 1.3 Implementing Class Templates for Reusable Components
- › 1.4 Leveraging Variadic Templates for Enhanced Flexibility
- › 1.5 Specializing Templates: Full and Partial Specialization Techniques
- › 1.6 Applying Template Metaprogramming for Compile-Time Computation

• 2 Understanding and Implementing Smart Pointers — 6 classes

- › 2.1 Explain the Concept of Smart Pointers
- › 2.2 Identify Different Types of Smart Pointers in C++
- › 2.3 Implement std::unique_ptr in Practical Scenarios
- › 2.4 Utilize std::shared_ptr for Shared Resource Management
- › 2.5 Apply std::weak_ptr to Handle Circular References
- › 2.6 Evaluate Performance and Safety Benefits of Smart Pointers

• 3 Implementing Move Semantics and Perfect Forwarding — 6 classes

- › 3.1 Understand Move Semantics: Concepts and Benefits
- › 3.2 Implementing Move Constructors and Move Assignment Operators
- › 3.3 Exploring Rvalue References: Syntax and Usage
- › 3.4 Introduction to Perfect Forwarding and Its Importance
- › 3.5 Utilizing std::forward for Perfect Forwarding in Functions
- › 3.6 Real-World Applications: Performance Optimization through Move Semantics

• 4 Exploring Concurrency and Multithreading in C++ — 6 classes

- › 4.1 Understand Concurrency Concepts in C++
- › 4.2 Explore the C++ Standard Library's Thread Support
- › 4.3 Implement Basic Multithreading with std::thread
- › 4.4 Manage Shared Data with Mutexes and Locks
- › 4.5 Use Condition Variables for Thread Synchronization
- › 4.6 Optimize Multithreaded Applications for Performance

• 5 Building Robust C++ Applications with Design Patterns — 6 classes

- › 5.1 Identify Common Design Patterns in C++ Applications
- › 5.2 Analyze the Singleton Pattern for Resource Management
- › 5.3 Implement the Factory Method for Object Creation
- › 5.4 Utilize the Observer Pattern for Event Handling
- › 5.5 Apply the Strategy Pattern for Flexible Algorithms
- › 5.6 Evaluate Design Patterns for Robust Application Architecture

2 Innovative Software Solutions

0 chapters **65 marks**

3 Project Management in Software Development

5 chapters · 18 classes **65 marks**

• 1 Fundamentals of Project Management in Software Development — 6 classes

- › 1.1 Define Project Management in Software Development
- › 1.2 Identify Key Phases of Software Development Projects
- › 1.3 Analyze the Role of Stakeholders in Project Success
- › 1.4 Apply Project Planning Techniques for Software Projects
- › 1.5 Evaluate Risk Management Strategies in Software Development
- › 1.6 Implement Effective Communication Practices in Project Teams

• 2 Agile Methodologies and C++ Development Practices — 6 classes

- › 2.1 Define and Explore Agile Methodologies in Software Development
- › 2.2 Identify Key Principles of Agile that Enhance C++ Development
- › 2.3 Compare Agile and Traditional Project Management Approaches in C++ Projects
- › 2.4 Implement Scrum Framework for Managing C++ Development Teams
- › 2.5 Apply Test-Driven Development (TDD) Practices in C++ Agile Projects
- › 2.6 Evaluate Tools and Technologies Supporting Agile C++ Development Practices

• 3 Project Planning and Resource Allocation in Software Engineering — 6 classes

- › 3.1 Define Project Scope and Objectives in Software Development
- › 3.2 Identify Key Stakeholders and Their Roles in Project Management
- › 3.3 Develop a Comprehensive Project Timeline Using Gantt Charts
- › 3.4 Assess and Allocate Resources Effectively for Software Projects
- › 3.5 Implement Risk Management Strategies in Software Project Planning
- › 3.6 Evaluate Project Progress and Adjust Plans as Necessary

• 4 Quality Assurance and Risk Management in Software Projects

• 5 Advanced Project Tracking and Performance Evaluation Techniques

4 Software Design Patterns

5 chapters · 30 classes **85 marks**

• 1 Understanding Design Patterns and Their Importance in Software Development — 6 classes

- › 1.1 Define Design Patterns and Their Relevance in Software Development
- › 1.2 Identify the Common Types of Design Patterns in C++
- › 1.3 Explore the Benefits of Using Design Patterns for Software Efficiency
- › 1.4 Examine Real-World Examples of Design Patterns in Action
- › 1.5 Apply a Specific Design Pattern to a Sample C++ Project
- › 1.6 Evaluate the Impact of Design Patterns on Software Maintenance and Scalability

• 2 Creational Patterns: Building Blocks of Object Creation — 6 classes

- › 2.1 Understand the Importance of Creational Patterns in C++
- › 2.2 Explore the Singleton Pattern: Implementing a Unique Instance
- › 2.3 Apply the Factory Method Pattern for Object Creation
- › 2.4 Differentiate Between Abstract Factory and Factory Method Techniques
- › 2.5 Utilize the Builder Pattern for Complex Object Construction
- › 2.6 Analyze Real-World Applications of Creational Patterns in Software Design

• 3 Structural Patterns: Organizing Code for Clarity and Efficiency — 6 classes

- › 3.1 Identify Key Structural Patterns in Software Design
- › 3.2 Analyze the Benefits of Using Structural Patterns
- › 3.3 Compare and Contrast Adapter and Bridge Patterns
- › 3.4 Implementing the Composite Pattern for Hierarchical Structures
- › 3.5 Applying the Decorator Pattern to Enhance Functionality
- › 3.6 Evaluate Real-World Examples of Structural Patterns in Code

• 4 Behavioral Patterns: Defining Interactions and Responsibilities — 6 classes

- › 4.1 Identify Behavioral Patterns in Software Design
- › 4.2 Analyze the Role of Context in Behavioral Patterns
- › 4.3 Evaluate Key Responsibilities of Behavioral Patterns
- › 4.4 Compare and Contrast Strategy and Command Patterns
- › 4.5 Implement Observer Pattern in a C++ Application
- › 4.6 Case Study: Applying Behavioral Patterns to Solve Real-World Problems

• 5 Implementing Real-World Applications of Design Patterns in C++ — 6 classes

- › 5.1 Understand Core Design Patterns in C++
- › 5.2 Identify Use Cases for Design Patterns in Real-World Applications
- › 5.3 Implement the Singleton Pattern in a C++ Project
- › 5.4 Create a Simple Observer Pattern Example in C++
- › 5.5 Develop a Strategy Pattern for a Real-World Scenario
- › 5.6 Evaluate and Refactor Code Using Design Patterns in C++

5 Software Development Best Practices

5 chapters · 30 classes **65 marks**

- **1 Understanding Software Development Methodologies** — 6 classes
  - 1.1 Identify Key Software Development Methodologies
  - 1.2 Compare Agile and Waterfall Approaches
  - 1.3 Explore Lean Software Development Principles
  - 1.4 Assess the Role of DevOps in Software Development
  - 1.5 Analyze the Benefits of Iterative Development
  - 1.6 Implement a Mini Project Using Agile Practices
- **2 Version Control Systems and Best Practices** — 6 classes
  - 2.1 Understand the Fundamentals of Version Control Systems
  - 2.2 Explore Popular Version Control Systems: Git vs. SVN
  - 2.3 Implement Basic Git Operations: Clone, Commit, and Push
  - 2.4 Establish Best Practices for Branching and Merging
  - 2.5 Manage Collaborative Workflows with Pull Requests
  - 2.6 Analyze and Resolve Merge Conflicts Effectively
- **3 Code Quality and Static Analysis Tools** — 6 classes
  - 3.1 Understand Code Quality Metrics and Their Importance
  - 3.2 Identify Common Code Smells and Anti-Patterns
  - 3.3 Explore Static Analysis Tools: Features and Benefits
  - 3.4 Integrate Static Analysis into Development Workflow
  - 3.5 Analyze Static Analysis Reports for Code Improvement
  - 3.6 Implement Best Practices for Maintaining Code Quality
- **4 Testing Strategies and Automated Testing** — 6 classes
  - 4.1 Understand Different Testing Strategies
  - 4.2 Identify and Define Test Cases
  - 4.3 Explore Automated Testing Tools and Frameworks
  - 4.4 Implement Unit Testing in C++
  - 4.5 Analyze Test Results and Debugging Techniques
  - 4.6 Integrate Continuous Testing into Development Workflow
- **5 Continuous Integration and Continuous Deployment (CI/CD)** — 6 classes
  - 5.1 Define Continuous Integration and Continuous Deployment Concepts
  - 5.2 Identify Benefits of Implementing CI/CD in Software Development
  - 5.3 Explore CI/CD Tools and Technologies Used in the Industry
  - 5.4 Outline Best Practices for Setting Up a CI/CD Pipeline
  - 5.5 Demonstrate Automated Testing in a CI/CD Workflow
  - 5.6 Implement a Basic CI/CD Pipeline Using Sample Code

6 Team Leadership and Communication

0 chapters **65 marks**