Siddhartha Rao

NINTH EDITION

New C++20 C++23 Coverage

Sams **Teach Yourself**

C++ in **One Hour** a Day

Sams **Teach Yourself** C++ in **One Hour** a Day

PART I: **The Basics**						
1 Getting Started	**2** The Anatomy of a C++ Program	**3** Using Variables, Declaring Constants	**4** Managing Arrays and Strings	**5** Working with Expressions, Statements, and Operators	**6** Controlling Program Flow	**7** Organizing Code with Functions
	PART II: **Fundamentals of Object-Oriented C++ Programming**					
8 Pointers and Explained References	**9** Classes and Objects	**10** Implementing Inheritance	**11** Polymorphism	**12** Operator Types and Operator Overloading	**13** Casting Operators	**14** An Introduction to Macros and Templates
PART III: **Learning the Standard Template Library (STL)**						PART IV:
15 An Introduction to the The Standard Template Library	**16** The String Class	**17** STL Dynamic Array Classes	**18** STL `list` and `forward_list`	**19** STL `set` and `multiset`	**20** STL `map` and `multimap`	**21** Understanding Function Objects
Lambda Expressions and STL Algorithms				PART V: **Advanced C++ Concepts**		
22 Lambda Expressions	**23** STL Algorithms	**24** Adaptive Containers: Stack and Queue	**25** Working with Bit Flags Using the STL	**26** Understanding Smart Pointers	**27** Using Streams for Input and Output	**28** Exception Handling
29 C++20 Concepts, Ranges, Views, and Adaptors	**30** C++ Threads	**31** C++20 Modules and C++23				

Sams Teach Yourself C++ in One Hour a Day

Table of Contents

Cover

Title

Copyright

Dedication

Contents at a Glance

Table of Contents

Introduction

PART I: The Basics

- LESSON 1: Getting Started
  - A Brief History of C++
    - Connection to C
    - Advantages of C++
    - Evolution of the C++ Standard
    - Who Uses Programs Written in C++?
  - Programming a C++ Application
    - Steps in Building an Executable
    - Analyzing Errors and Debugging
    - Integrated Development Environments
    - Programming Your First C++ Application
    - Building and Executing Your First C++ Application
    - Understanding Compiler Errors
  - Whats New in C++20?
  - Summary

Pearson

Table of Contents

Q&A

Workshop

Quiz

Exercises

LESSON 2: The Anatomy of a C++ Program

Parts of the Hello World Program

Preprocessor Directive #include

The Body of Your Program: main()

Returning a Value

The Concept of Namespaces

Comments in C++ Code

Functions in C++

Basic Input Using std::cin and Output Using std::cout

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 3: Using Variables, Declaring Constants

What Is a Variable?

Memory and Addressing in Brief

Declaring Variables to Access and Use Memory

Declaring and Initializing Multiple Variables of a Type

Understanding the Scope of a Variable

Global Variables

Naming Conventions

Common Compiler-Supported C++ Variable Types

Using Type bool to Store Boolean Values

Using Type char to Store Character Values

The Concept of Signed and Unsigned Integers

Signed Integer Types short, int, long, and long long

Table of Contents

Unsigned Integer Types unsigned short, unsigned int, unsigned long, and unsigned long long

Avoiding Overflow Errors by Selecting Correct Data Types

Floating-Point Types float and double

Determining the Size of a Variable by Using sizeof()

Avoid Narrowing Conversion Errors by Using List Initialization

Automatic Type Inference Using auto

Using typedef to Substitute a Variables Type

What Is a Constant?

Literal Constants

Declaring Variables as Constants Using const

Constant Expressions Using constexpr

C++20 Immediate Functions Using consteval

Enumerations

Scoped Enumerations

Defining Constants by Using #define

Keywords You Cannot Use as Variable or Constant Names

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 4: Managing Arrays and Strings

What Is an Array?

The Need for Arrays

Declaring and Initializing Static Arrays

How Data Is Stored in an Array

Accessing Data Stored in an Array

Modifying Data Stored in an Array

Multidimensional Arrays

Declaring and Initializing Multidimensional Arrays

Accessing Elements in a Multidimensional Array

Table of Contents

Dynamic Arrays

C-Style Character Strings

C++ Strings: Using std::string

Summary

Q&A

Workshop

- Quiz
- Exercises

LESSON 5: Working with Expressions, Statements, and Operators

Statements

Compound Statements, or Blocks

Using Operators

- The Assignment Operator (=)
- Understanding l-Values and r-Values
- Operators to Add (+), Subtract (-), Multiply (*), Divide (/), and Modulo Divide (%)
- Operators to Increment (++) and Decrement (--)
- To Postfix or to Prefix?
- Equality Operators (== and !=)
- Relational Operators
- C++20 Three-Way Comparison Operator (<=>)
- Logical Operations NOT, AND, OR, and XOR
- Using C++ Logical Operators NOT (!), AND (&&), and OR (||)
- Bitwise NOT (~), AND (&), OR (|), and XOR (^) Operators
- Bitwise Right Shift (>>) and Left Shift (<<) Operators
- Compound Assignment Operators
- Using the sizeof() Operator to Determine the Memory Occupied by a Variable
- Operator Precedence and Associativity

Summary

Q&A

Workshop

- Quiz
- Exercises

Pearson

Table of Contents

LESSON 6: Controlling Program Flow

Conditional Execution Using if else

Conditional Programming Using if else

Conditional Execution of Statements Within a Block

Nested if Statements

Conditional Processing Using switch-case

Conditional Execution Using the ?: Operator

Getting Code to Execute in Loops

A Rudimentary Loop Using goto

The while Loop

The do while Loop

The for Loop

The Range-Based for Loop

Modifying Loop Behavior Using continue and break

Loops That Dont End: Infinite Loops

Controlling Infinite Loops

Programming Nested Loops

Using Nested Loops to Walk a Multidimensional Array

Using Nested Loops to Calculate Fibonacci Numbers

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 7: Organizing Code with Functions

The Need for Functions

What Is a Function Prototype?

What Is a Function Definition?

What Is a Function Call, and What Are Arguments?

Programming a Function with Multiple Parameters

Programming Functions with No Parameters or No Return Values

Function Parameters with Default Values

Table of Contents

Recursion: Functions That Invoke Themselves

Functions with Multiple Return Statements

Using Functions to Work with Different Forms of Data

Overloading Functions

Passing an Array of Values to a Function

Passing Arguments by Reference

How Function Calls Are Handled by the Microprocessor

Inline Functions

Automatic Return Type Deduction

Lambda Functions

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 8: Pointers and References Explained

What Is a Pointer?

Declaring a Pointer

Determining the Address of a Variable by Using the Reference Operator (&)

Using Pointers to Store Addresses

Accessing Pointed Data Using the Dereference Operator (*)

What Is the Size of a Pointer?

Dynamic Memory Allocation

Using the Operators new and delete to Allocate and Release Memory Dynamically

Effects of the Increment (++) and Decrement (--) Operators on Pointers

Using the const Keyword on Pointers

Passing Pointers to Functions

Similarities Between Arrays and Pointers

Common Programming Mistakes When Using Pointers

Memory Leaks

Pointers Pointing to Invalid Memory Locations

Dangling Pointers (Also Called Stray or Wild Pointers)

Pearson

Table of Contents

Checking Whether an Allocation Request Using new Succeeded

Pointer Programming Best Practices

What Is a Reference?

What Makes References Useful?

Using the Keyword const on References

Passing Arguments by Reference to Functions

Summary

Q&A

Workshop

Quiz

Exercises

PART II: Fundamentals of Object-Oriented C++ Programming

LESSON 9: Classes and Objects

The Concept of Classes and Objects

Declaring a Class

An Object as an Instance of a Class

Accessing Members by Using the Dot Operator (.)

Accessing Members by Using the Pointer Operator (->)

The Keywords public and private

Abstraction of Data via the Keyword private

Constructors

Declaring and Implementing a Constructor

When and How to Use Constructors

Overloading Constructors

A Class Without a Default Constructor

Constructor Parameters with Default Values

Constructors with Initialization Lists

Destructor

Declaring and Implementing a Destructor

When and How to Use a Destructor

The Copy Constructor

Table of Contents

Shallow Copying and Associated Problems

Ensuring a Deep Copy Using a Copy Constructor

Using Move Constructors to Improve Performance

Different Uses of Constructors and the Destructor

A Class That Does Not Permit Copying

A Singleton Class That Permits a Single Instance

A Class That Prohibits Instantiation on the Stack

Using Constructors to Convert Types

The this Pointer

Using sizeof() with a Class

The Keyword struct and Its Differences from class

Declaring a friend of a class

Union: A Special Data Storage Mechanism

Declaring a Union

Where Would You Use a Union?

Using Aggregate Initialization on Classes and structs

constexpr with Classes and Objects

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 10: Implementing Inheritance

Basics of Inheritance

Inheritance and Derivation

C++ Syntax of Derivation

The Access Specifier Keyword protected

Base Class Initialization: Passing Parameters to the Base Class

A Derived Class Overriding the Base Classs Methods

Invoking Overridden Methods of a Base Class

Invoking Methods of a Base Class in a Derived Class

A Derived Class Hiding the Base Classs Methods

Pearson

Table of Contents

- Order of Construction
- Order of Destruction

Private Inheritance

Protected Inheritance

The Problem of Slicing

Multiple Inheritance

Avoiding Inheritance Using final

Summary

Q&A

Workshop

- Quiz
- Exercises

LESSON 11: Polymorphism

Basics of Polymorphism

- Need for Polymorphic Behavior
- Polymorphic Behavior Implemented Using Virtual Functions
- Need for Virtual Destructors
- How Do Virtual Functions Work? Understanding the Virtual Function Table
- Abstract Base Classes and Pure Virtual Functions

Using Virtual Inheritance to Solve the Diamond Problem

Using the Specifier override to Indicate the Intention to Override

Using final to Prevent Function Overriding

Virtual Copy Constructors?

Summary

Q&A

Workshop

- Quiz
- Exercises

LESSON 12: Operator Types and Operator Overloading

What Are Operators in C++?

Unary Operators

Table of Contents

- Unary Increment (++) and Decrement (--) Operators
- Conversion Operators
- The Dereference Operator (*) and Member Selection Operator (->)

Binary Operators

- The Binary Addition (a+b) and Subtraction (a-b) Operators
- The Addition Assignment (+=) and Subtraction Assignment (-=) Operators
- The Equality (==) and Inequality (!=) Operators
- The <, >, <=, and >= Operators
- The C++20 Three-Way Comparison Operator (<=>)
- The Copy Assignment Operator (=)
- The Subscript Operator ([])

The Function Operator (())

- The Move Constructor and Move Assignment Operator for High-Performance Programming
- The Problem of Unwanted Copy Steps

Declaring a Move Constructor and Move Assignment Operator

User-Defined Literals

Operators That Cannot Be Overloaded

Summary

Q&A

Workshop

- Quiz
- Exercises

LESSON 13: Casting Operators

The Need for Casting

Why C-Style Casts Are Not Popular with Some C++ Programmers

The C++ Casting Operators

- Using static_cast
- Using dynamic_cast and Runtime Type Identification
- Using reinterpret_cast
- Using const_cast

Problems with the C++ Casting Operators

Summary

Table of Contents

- Q&A
- Workshop
  - Quiz
  - Exercises

LESSON 14: An Introduction to Macros and Templates

- The Preprocessor and the Compiler
- Using the Macro #define to Define Constants
  - Using Macros for Protection Against Multiple Inclusion
- Using #define to Write Macro Functions
  - Why All the Parentheses?
  - Using the assert Macro to Validate Expressions
  - Advantages and Disadvantages of Using Macro Functions
- An Introduction to Templates
  - Template Declaration Syntax
  - The Different Types of Template Declarations
  - Template Functions
  - Templates and Type Safety
  - Template Classes
  - Declaring Templates with Multiple Parameters
  - Declaring Templates with Default Parameters
  - Sample Template Class: HoldsPair
  - Template Instantiation and Specialization
  - Template Classes and static Members
  - Variable Templates
  - Using static_assert to Perform Compile-Time Checks
  - Using Templates in Practical C++ Programming
- Summary
- Q&A
- Workshop
  - Quiz
  - Exercises

PART III: Learning the Standard Template Library (STL)

Pearson

Table of Contents

LESSON 15: An Introduction to the Standard Template Library

- STL Containers
  - Sequential Containers
  - Associative Containers
  - Container Adapters
- STL Iterators
- STL Algorithms
- Interaction Between Containers and Algorithms Using Iterators
  - Using the Keyword auto to Let a Compiler Define Type
- Choosing the Right Container
- STL String Classes
- Summary
- Q&A
- Workshop
  - Quiz

LESSON 16: The STL String Class

- The Need for String Manipulation Classes
- Working with the STL string Class
  - Instantiating the STL string Class and Making Copies
  - Accessing Character Contents of std::string
  - Concatenating One String to Another
  - Finding a Character or Substring in a String
  - Truncating an STL String
  - String Reversal
  - String Case Conversion
- Template-Based Implementation of an STL String
- operator ""s in std::string
- Using std::string_view (Amended in C++20)
- Summary
- Q&A
- Workshop

Table of Contents

Quiz

Exercises

LESSON 17: STL Dynamic Array Classes

The Characteristics of std::vector

Typical Vector Operations

Instantiating a Vector

Inserting Elements at the End of a Vector by Using push_back()

List Initialization

Inserting Elements at a Given Position by Using insert()

Accessing Elements in a Vector by Using Array Semantics

Accessing Elements in a Vector by Using Pointer Semantics

Removing Elements from a Vector

Understanding the Concepts of Size and Capacity

The STL deque Class

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 18: STL list and forward_list

The Characteristics of std::list

Basic list Operations

Instantiating a std::list Object

Inserting Elements at the Front or Back of a List

Inserting Elements in the Middle of a List

Erasing Elements from a List

Reversing and Sorting Elements in a List

Reversing Elements by Using list::reverse()

Sorting Elements

Sorting and Removing Elements from a List That Contains Instances of a Class

std::forward_list

Summary

Table of Contents

Q&A
Workshop
Quiz
Exercises
LESSON 19: STL set and multiset
An Introduction to STL Set Classes
Basic STL set and multiset Operations
Instantiating a std::set Object
Inserting Elements in a Set or Multiset
Finding Elements in an STL set or multiset Container
Erasing Elements in an STL set or multiset Container
Pros and Cons of Using STL set and multiset
STL Hash Set Implementation: std::unordered_set and std::unordered_multiset
Summary
Q&A
Workshop
Quiz
Exercises
LESSON 20: STL map and multimap
An Introduction to STL Map Classes
Basic std::map and std::multimap Operations
Instantiating std::map or std::multimap
Inserting Elements in an STL Map or Multimap
Finding Elements in an STL map Container
Finding Elements in an STL multimap Container
Erasing Elements from an STL map or multimap Container
Supplying a Custom Sort Predicate
STLs Hash TableBased Key/Value Container
How Hash Tables Work
Using unordered_map and unordered_multimap

Pearson

Table of Contents

Summary
Q&A
Workshop
Quiz
Exercises
PART IV: Lambda Expressions and STL Algorithms
LESSON 21: Understanding Function Objects
Function Objects and Predicates
Typical Applications of Function Objects
Unary Functions
Unary Predicates
Binary Functions
Binary Predicates
Summary
Q&A
Workshop
Quiz
Exercises
LESSON 22: Lambda Expressions
What Is a Lambda Expression?
How to Define a Lambda Expression
Capturing Variables
Parameters
Return Types
A Lambda Expression for a Unary Function
A Lambda Expression for a Unary Predicate
A Lambda Expression with State via Capture Lists ([ ])
A Lambda Expression for a Binary Function
A Lambda Expression for a Binary Predicate
Summary

Table of Contents

Q&A

Workshop

Quiz

Exercises

LESSON 23: STL Algorithms

What Are STL Algorithms?

Classification of STL Algorithms

Non-mutating Algorithms

Mutating Algorithms

Usage of STL Algorithms

Finding Elements, Given a Value or a Condition

Counting Elements Given a Value or a Condition

Searching for an Element or a Range in a Collection

Initializing Elements in a Container to a Specific Value

Using std::generate() to Initialize Elements to a Value Generated at Runtime

Processing Elements in a Range by Using for_each()

Performing Transformations on a Range by Using std::transform()

Copy and Remove Operations

Replacing Values and Replacing Elements Given a Condition

Sorting and Searching in a Sorted Collection and Erasing Duplicates

Partitioning a Range

Inserting Elements in a Sorted Collection

Performing Fold Operations Using std::accumulate() in C++20

C++20 Constrained Algorithms

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 24: Adaptive Containers: Stack and Queue

The Behavioral Characteristics of Stacks and Queues

Stacks

Table of Contents

Queues

Using the STL stack Class

Instantiating a Stack

Stack Member Functions

Insertion and Removal at the Top, Using push() and pop()

Using the STL queue Class

Instantiating a Queue

Member Functions of the queue Class

Insertion at the End and Removal at the Beginning of a Queue via push() and pop()

Using the STL Priority Queue

Instantiating the priority_queue Class

Member Functions of priority_queue

Insertion at the End and Removal at the Beginning of a Priority Queue via push() and pop()

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 25: Working with Bit Flags Using the STL

The bitset Class

Instantiating std::bitset

Using std::bitset and Its Members

Useful Operators in std::bitset

std::bitset Member Methods

The vector<bool> Class

Instantiating vector<bool>

vector<bool> Functions and Operators

Summary

Q&A

Workshop

Quiz

Table of Contents

Exercises

PART V: Advanced C++ Concepts

LESSON 26: Understanding Smart Pointers

What Are Smart Pointers?

The Problem with Using Conventional (Raw) Pointers

How Do Smart Pointers Help?

How Are Smart Pointers Implemented?

Types of Smart Pointers

Deep Copy

Copy on Write

Reference-Counted Smart Pointers

Reference-Linked Smart Pointers

Destructive Copy

Using std::unique_ptr

Popular Smart Pointer Libraries

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 27: Using Streams for Input and Output

The Concept of Streams

Important C++ Stream Classes and Objects

Using std::cout for Writing Formatted Data to the Console

Changing the Display Number Format by Using std::cout

Aligning Text and Setting Field Width by Using std::cout

Using std::cin for Input

Using std::cin for Input into a Plain Old Data Type

Using std::cin::get for Input into the char* Buffer

Using std::cin for Input into std::string

Using std::fstream for File Handling

Table of Contents

Opening and Closing a File Using open() and close()

Creating and Writing a Text File by Using open() and the Operator <<

Reading a Text File by Using open() and the Operator >>

Writing to and Reading from a Binary File

Using std::stringstream for String Conversions

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 28: Exception Handling

What Is an Exception?

What Causes Exceptions?

Implementing Exception Safety via try and catch

Using catch( ) to Handle All Exceptions

Catching Exceptions of a Type

Throwing Exceptions of a Type by Using throw

How Exception Handling Works

Class std::exception

A Custom Exception Class Derived from std::exception

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 29: C++20 Concepts, Ranges, Views, and Adaptors

Concepts

Using Concepts Provided by the Standard Library

Defining Custom Concepts by Using the Keyword requires

Using Concepts with Classes and Objects

The Ranges Library, Views, and Adaptors

Table of Contents

Views and Adaptors

Adaptors Provided by the Ranges Library

Combining Multiple Adaptors

Summary

Q&A

Workshop

Quiz

Exercises

LESSON 30: C++20 Threads

Multithreading

What Is a Thread?

Why Program Multithreaded Applications?

Using the C++20 Thread Library

How Can Threads Transact Data?

Using Mutexes and Semaphores to Synchronize Threads

Summary

Q&A

Workshop

Exercise

LESSON 31: C++20 Modules and C++23

Modules

The Problem with #include<header>

C++20 Modules

Programming a Module

Consuming a Module

Why import Module; Is Superior to the Preprocessor #include<header>

C++23 Expected Features

Learning C++ Doesnt Stop Here!

Online Documentation

Communities for Guidance and Help

Summary

Pearson

Table of Contents

Q&A

Workshop

Exercise

PART VI: Appendixes

APPENDIX A: Working with Numbers: Binary and Hexadecimal

APPENDIX B: C++ Keywords

APPENDIX C: Writing Great C++ Code

APPENDIX D: ASCII Codes

APPENDIX E: Answers

Index

Pearson