



DEITEL® DEVELOPER SERIES

Modern C++

Functional-Style Programming

Concepts

Templates

Copy/Move Semantics

Spaceship Operator

Smart Pointers

Text Formatting

Security

Modules

Standard Library

C++20

for

Programmers

An Objects-Natural Approach

Executors

Design Patterns

Coroutines

Ranges/Views

Performance

Contracts

Open Source Libraries

jthread

Parallel Algorithms

Lambdas

Concurrency

PAUL DEITEL • HARVEY DEITEL



# C++20 for Programmers: An Objects-Natural Approach

## Table of Contents

Cover

Half Title

Title Page

Copyright Page

Contents

Preface

Before You Begin

1 Intro and Test-Driving Popular, Free C++ Compilers

1.1 Introduction

1.2 Test-Driving a C++20 Application

1.2.1 Compiling and Running a C++20 Application with Visual Studio 2022  
Community Edition on Windows

1.2.2 Compiling and Running a C++20 Application with Xcode on macOS

1.2.3 Compiling and Running a C++20 Application with GNU C++ on Linux 11

1.2.4 Compiling and Running a C++20 Application with g++ in the GCC Docker  
Container

1.2.5 Compiling and Running a C++20 Application with clang++ in a Docker  
Container

1.3 Moores Law, Multi-Core Processors and Concurrent Programming

1.4 A Brief Refresher on Object Orientation

1.5 Wrap-Up

2 Intro to C++20 Programming

2.1 Introduction

2.2 First Program in C++: Displaying a Line of Text

2.3 Modifying Our First C++ Program

# **Table of Contents**

2.4 Another C++ Program: Adding Integers

2.5 Arithmetic

2.6 Decision Making: Equality and Relational Operators

2.7 Objects Natural: Creating and Using Objects of Standard-Library Class string

2.8 Wrap-Up

## **3 Control Statements: Part 1**

3.1 Introduction

3.2 Control Structures

3.2.1 Sequence Structure

3.2.2 Selection Statements

3.2.3 Iteration Statements

3.2.4 Summary of Control Statements

3.3 if Single-Selection Statement

3.4 ifelse Double-Selection Statement

3.4.1 Nested ifelse Statements

3.4.2 Blocks

3.4.3 Conditional Operator (?:)

3.5 while Iteration Statement

3.6 Counter-Controlled Iteration

3.6.1 Implementing Counter-Controlled Iteration

3.6.2 Integer Division and Truncation

3.7 Sentinel-Controlled Iteration

3.7.1 Implementing Sentinel-Controlled Iteration

3.7.2 Converting Between Fundamental Types Explicitly and Implicitly

3.7.3 Formatting Floating-Point Numbers

3.8 Nested Control Statements

3.8.1 Problem Statement

3.8.2 Implementing the Program

3.8.3 Preventing Narrowing Conversions with Braced Initialization

3.9 Compound Assignment Operators

3.10 Increment and Decrement Operators

# **Table of Contents**

- 3.11 Fundamental Types Are Not Portable
- 3.12 Objects-Natural Case Study: Arbitrary-Sized Integers
- 3.13 C++20: Text Formatting with Function format
- 3.14 Wrap-Up

## **4 Control Statements: Part 2**

- 4.1 Introduction
- 4.2 Essentials of Counter-Controlled Iteration
- 4.3 for Iteration Statement
- 4.4 Examples Using the for Statement
- 4.5 Application: Summing Even Integers
- 4.6 Application: Compound-Interest Calculations
- 4.7 dowhile Iteration Statement
- 4.8 switch Multiple-Selection Statement
- 4.9 C++17 Selection Statements with Initializers
- 4.10 break and continue Statements
- 4.11 Logical Operators
  - 4.11.1 Logical AND (&&) Operator
  - 4.11.2 Logical OR (||) Operator
  - 4.11.3 Short-Circuit Evaluation
  - 4.11.4 Logical Negation (!) Operator
  - 4.11.5 Example: Producing Logical-Operator Truth Tables
- 4.12 Confusing the Equality (==) and Assignment (=) Operators
- 4.13 Objects-Natural Case Study: Using the miniz-cpp Library to Write and Read ZIP files
- 4.14 C++20 Text Formatting with Field Widths and Precisions
- 4.15 Wrap-Up

## **5 Functions and an Intro to Function Templates**

- 5.1 Introduction
- 5.2 C++ Program Components
- 5.3 Math Library Functions

# **Table of Contents**

5.4 Function Definitions and Function Prototypes

5.5 Order of Evaluation of a Functions Arguments

5.6 Function-Prototype and Argument-Coercion Notes

5.6.1 Function Signatures and Function Prototypes

5.6.2 Argument Coercion

5.6.3 Argument-Promotion Rules and Implicit Conversions

5.7 C++ Standard Library Headers

5.8 Case Study: Random-Number Generation

5.8.1 Rolling a Six-Sided Die

5.8.2 Rolling a Six-Sided Die 60,000,000 Times

5.8.3 Seeding the Random-Number Generator

5.8.4 Seeding the Random-Number Generator with `random_device`

5.9 Case Study: Game of Chance; Introducing Scoped enums

5.10 Scope Rules

5.11 Inline Functions

5.12 References and Reference Parameters

5.13 Default Arguments

5.14 Unary Scope Resolution Operator

5.15 Function Overloading

5.16 Function Templates

5.17 Recursion

5.18 Example Using Recursion: Fibonacci Series

5.19 Recursion vs. Iteration

5.20 Lnfylun Lhqtomh Wjtz Qarcv: Qjwazkrplm xzz Xndmwwqhlz

5.21 Wrap-Up

6 arrays, vectors, Ranges and Functional-Style Programming

6.1 Introduction

6.2 arrays

6.3 Declaring arrays

6.4 Initializing array Elements in a Loop

# **Table of Contents**

- 6.5 Initializing an array with an Initializer List
- 6.6 C++11 Range-Based for and C++20 Range-Based for with Initializer
- 6.7 Calculating array Element Values and an Intro to constexpr
- 6.8 Totaling array Elements
- 6.9 Using a Primitive Bar Chart to Display array Data Graphically
- 6.10 Using array Elements as Counters
- 6.11 Using arrays to Summarize Survey Results
- 6.12 Sorting and Searching arrays
- 6.13 Multidimensional arrays
- 6.14 Intro to Functional-Style Programming
  - 6.14.1 What vs. How
  - 6.14.2 Passing Functions as Arguments to Other Functions: Introducing Lambda Expressions
  - 6.14.3 Filter, Map and Reduce: Intro to C++20s Ranges Library
- 6.15 Objects-Natural Case Study: C++ Standard Library Class Template vector
- 6.16 Wrap-Up
- 7 (Downplaying) Pointers in Modern C++
  - 7.1 Introduction
  - 7.2 Pointer Variable Declarations and Initialization
    - 7.2.1 Declaring Pointers
    - 7.2.2 Initializing Pointers
    - 7.2.3 Null Pointers Before C++11
  - 7.3 Pointer Operators
    - 7.3.1 Address (&) Operator
    - 7.3.2 Indirection (\*) Operator
    - 7.3.3 Using the Address (&) and Indirection (\*) Operators
  - 7.4 Pass-by-Reference with Pointers
  - 7.5 Built-In Arrays
    - 7.5.1 Declaring and Accessing a Built-In Array
    - 7.5.2 Initializing Built-In Arrays

# **Table of Contents**

- 7.5.3 Passing Built-In Arrays to Functions
- 7.5.4 Declaring Built-In Array Parameters
- 7.5.5 C++11 Standard Library Functions begin and end
- 7.5.6 Built-In Array Limitations
- 7.6 Using C++20 `to_array` to Convert a Built-In Array to a `std::array`
- 7.7 Using `const` with Pointers and the Data Pointed To
  - 7.7.1 Using a Nonconstant Pointer to Nonconstant Data
  - 7.7.2 Using a Nonconstant Pointer to Constant Data
  - 7.7.3 Using a Constant Pointer to Nonconstant Data
  - 7.7.4 Using a Constant Pointer to Constant Data
- 7.8 `sizeof` Operator
- 7.9 Pointer Expressions and Pointer Arithmetic
  - 7.9.1 Adding Integers to and Subtracting Integers from Pointers
  - 7.9.2 Subtracting One Pointer from Another
  - 7.9.3 Pointer Assignment
  - 7.9.4 Cannot Dereference a `void*`
  - 7.9.5 Comparing Pointers
- 7.10 Objects-Natural Case Study: C++20 `spans` Views of Contiguous Container Elements
- 7.11 A Brief Intro to Pointer-Based Strings
  - 7.11.1 Command-Line Arguments
  - 7.11.2 Revisiting C++20s `to_array` Function
- 7.12 Looking Ahead to Other Pointer Topics
- 7.13 Wrap-Up
- 8 strings, `string_views`, Text Files, CSV Files and Regex
  - 8.1 Introduction
  - 8.2 string Assignment and Concatenation
  - 8.3 Comparing strings
  - 8.4 Substrings
  - 8.5 Swapping strings
  - 8.6 string Characteristics



# **Table of Contents**

- 8.7 Finding Substrings and Characters in a string
- 8.8 Replacing and Erasing Characters in a string
- 8.9 Inserting Characters into a string
- 8.10 C++11 Numeric Conversions
- 8.11 C++17 string\_view
- 8.12 Files and Streams
- 8.13 Creating a Sequential File
- 8.14 Reading Data from a Sequential File
- 8.15 C++14 Reading and Writing Quoted Text
- 8.16 Updating Sequential Files
- 8.17 String Stream Processing
- 8.18 Raw String Literals
- 8.19 Objects-Natural Case Study: Reading and Analyzing a CSV File  
Containing Titanic Disaster Data
  - 8.19.1 Using rapidcsv to Read the Contents of a CSV File
  - 8.19.2 Reading and Analyzing the Titanic Disaster Dataset
- 8.20 Objects-Natural Case Study: Intro to Regular Expressions
  - 8.20.1 Matching Complete Strings to Patterns
  - 8.20.2 Replacing Substrings
  - 8.20.3 Searching for Matches
- 8.21 Wrap-Up

## **9 Custom Classes**

- 9.1 Introduction
- 9.2 Test-Driving an Account Object
- 9.3 Account Class with a Data Member and Set and Get Member Functions
  - 9.3.1 Class Definition
  - 9.3.2 Access Specifiers private and public
- 9.4 Account Class: Custom Constructors
- 9.5 Software Engineering with Set and Get Member Functions
- 9.6 Account Class with a Balance

# **Table of Contents**

## **9.7 Time Class Case Study: Separating Interface from Implementation**

9.7.1 Interface of a Class

9.7.2 Separating the Interface from the Implementation

9.7.3 Class Definition

9.7.4 Member Functions

9.7.5 Including the Class Header in the Source-Code File

9.7.6 Scope Resolution Operator (::)

9.7.7 Member Function setTime and Throwing Exceptions

9.7.8 Member Functions to24HourString and to12HourString

9.7.9 Implicitly Inlining Member Functions

9.7.10 Member Functions vs. Global Functions

9.7.11 Using Class Time

9.7.12 Object Size

## **9.8 Compilation and Linking Process**

## **9.9 Class Scope and Accessing Class Members**

## **9.10 Access Functions and Utility Functions**

## **9.11 Time Class Case Study: Constructors with Default Arguments**

9.11.1 Class Time

9.11.2 Overloaded Constructors and C++11 Delegating Constructors

## **9.12 Destructors**

## **9.13 When Constructors and Destructors Are Called**

## **9.14 Time Class Case Study: A Subtle Trap Returning a Reference or a Pointer to a private Data Member**

## **9.15 Default Assignment Operator**

## **9.16 const Objects and const Member Functions**

## **9.17 Composition: Objects as Members of Classes**

## **9.18 friend Functions and friend Classes**

## **9.19 The this Pointer**

9.19.1 Implicitly and Explicitly Using the this Pointer to Access an Objects Data Members

9.19.2 Using the this Pointer to Enable Cascaded Function Calls

## **9.20 static Class Members: Classwide Data and Member Functions**

# **Table of Contents**

## **9.21 Aggregates in C++20**

### **9.21.1 Initializing an Aggregate**

### **9.21.2 C++20: Designated Initializers**

## **9.22 Objects-Natural Case Study: Serialization with JSON**

### **9.22.1 Serializing a vector of Objects Containing public Data**

### **9.22.2 Serializing a vector of Objects Containing private Data**

## **9.23 Wrap-Up**

# **10 OOP: Inheritance and Runtime Polymorphism**

## **10.1 Introduction**

## **10.2 Base Classes and Derived Classes**

### **10.2.1 CommunityMember Class Hierarchy**

### **10.2.2 Shape Class Hierarchy and public Inheritance**

## **10.3 Relationship Between Base and Derived Classes**

### **10.3.1 Creating and Using a SalariedEmployee Class**

### **10.3.2 Creating a SalariedEmployeeSalariedCommissionEmployee Inheritance Hierarchy**

## **10.4 Constructors and Destructors in Derived Classes**

## **10.5 Intro to Runtime Polymorphism: Polymorphic Video Game**

## **10.6 Relationships Among Objects in an Inheritance Hierarchy**

### **10.6.1 Invoking Base-Class Functions from Derived-Class Objects**

### **10.6.2 Aiming Derived-Class Pointers at Base-Class Objects**

### **10.6.3 Derived-Class Member-Function Calls via Base-Class Pointers**

## **10.7 Virtual Functions and Virtual Destructors**

### **10.7.1 Why virtual Functions Are Useful**

### **10.7.2 Declaring virtual Functions**

### **10.7.3 Invoking a virtual Function**

### **10.7.4 virtual Functions in the SalariedEmployee Hierarchy**

### **10.7.5 virtual Destructors**

### **10.7.6 final Member Functions and Classes**

## **10.8 Abstract Classes and Pure virtual Functions**

### **10.8.1 Pure virtual Functions**

### **10.8.2 Device Drivers: Polymorphism in Operating Systems**

# **Table of Contents**

## **10.9 Case Study: Payroll System Using Runtime Polymorphism**

- 10.9.1 Creating Abstract Base Class Employee
- 10.9.2 Creating Concrete Derived Class SalariedEmployee
- 10.9.3 Creating Concrete Derived Class CommissionEmployee
- 10.9.4 Demonstrating Runtime Polymorphic Processing

## **10.10 Runtime Polymorphism, Virtual Functions and Dynamic Binding Under the Hood**

## **10.11 Non-Virtual Interface (NVI) Idiom**

## **10.12 Program to an Interface, Not an Implementation**

- 10.12.1 Rethinking the Employee Hierarchy Compensation Model Interface
- 10.12.2 Class Employee
- 10.12.3 CompensationModel Implementations
- 10.12.4 Testing the New Hierarchy
- 10.12.5 Dependency Injection Design Benefits

## **10.13 Runtime Polymorphism with std::variant and std::visit**

## **10.14 Multiple Inheritance**

- 10.14.1 Diamond Inheritance
- 10.14.2 Eliminating Duplicate Subobjects with virtual Base-Class Inheritance

## **10.15 protected Class Members: A Deeper Look**

## **10.16 public, protected and private Inheritance**

## **10.17 More Runtime Polymorphism Techniques; Compile-Time Polymorphism**

- 10.17.1 Other Runtime Polymorphism Techniques
- 10.17.2 Compile-Time (Static) Polymorphism Techniques
- 10.17.3 Other Polymorphism Concepts

## **10.18 Wrap-Up**

# **11 Operator Overloading, Copy/Move Semantics and Smart Pointers**

## **11.1 Introduction**

## **11.2 Using the Overloaded Operators of Standard Library Class string**

## **11.3 Operator Overloading Fundamentals**

- 11.3.1 Operator Overloading Is Not Automatic

# **Table of Contents**

- 11.3.2 Operators That Cannot Be Overloaded
- 11.3.3 Operators That You Do Not Have to Overload
- 11.3.4 Rules and Restrictions on Operator Overloading
- 11.4 (Downplaying) Dynamic Memory Management with new and delete
- 11.5 Modern C++ Dynamic Memory Management: RAII and Smart Pointers
  - 11.5.1 Smart Pointers
  - 11.5.2 Demonstrating unique\_ptr
  - 11.5.3 unique\_ptr Ownership
  - 11.5.4 unique\_ptr to a Built-In Array
- 11.6 MyArray Case Study: Crafting a Valuable Class with Operator Overloading
  - 11.6.1 Special Member Functions
  - 11.6.2 Using Class MyArray
  - 11.6.3 MyArray Class Definition
  - 11.6.4 Constructor That Specifies a MyArrays Size
  - 11.6.5 C++11 Passing a Braced Initializer to a Constructor
  - 11.6.6 Copy Constructor and Copy Assignment Operator
  - 11.6.7 Move Constructor and Move Assignment Operator
  - 11.6.8 Destructor
  - 11.6.9 toString and size Functions
  - 11.6.10 Overloading the Equality (==) and Inequality (!=) Operators
  - 11.6.11 Overloading the Subscript ([]) Operator
  - 11.6.12 Overloading the Unary bool Conversion Operator
  - 11.6.13 Overloading the Preincrement Operator
  - 11.6.14 Overloading the Postincrement Operator
  - 11.6.15 Overloading the Addition Assignment Operator (+=)
  - 11.6.16 Overloading the Binary Stream Extraction (>>) and Stream Insertion (<<) Operators
  - 11.6.17 friend Function swap
- 11.7 C++20 Three-Way Comparison Operator (<=>)
- 11.8 Converting Between Types
- 11.9 explicit Constructors and Conversion Operators

# **Table of Contents**

11.10 Overloading the Function Call Operator ()

11.11 Wrap-Up

## **12 Exceptions and a Look Forward to Contracts**

12.1 Introduction

12.2 Exception-Handling Flow of Control

12.2.1 Defining an Exception Class to Represent the Type of Problem That Might Occur

12.2.2 Demonstrating Exception Handling

12.2.3 Enclosing Code in a try Block

12.2.4 Defining a catch Handler for DivideByZeroExceptions

12.2.5 Termination Model of Exception Handling

12.2.6 Flow of Control When the User Enters a Nonzero Denominator

12.2.7 Flow of Control When the User Enters a Zero Denominator

12.3 Exception Safety Guarantees and noexcept

12.4 Rethrowing an Exception

12.5 Stack Unwinding and Uncaught Exceptions

12.6 When to Use Exception Handling

12.6.1 assert Macro

12.6.2 Failing Fast

12.7 Constructors, Destructors and Exception Handling

12.7.1 Throwing Exceptions from Constructors

12.7.2 Catching Exceptions in Constructors via Function try Blocks

12.7.3 Exceptions and Destructors: Revisiting noexcept(false)

12.8 Processing new Failures

12.8.1 new Throwing bad\_alloc on Failure

12.8.2 new Returning nullptr on Failure

12.8.3 Handling new Failures Using Function set\_new\_handler

12.9 Standard Library Exception Hierarchy

12.10 C++s Alternative to the finally Block: Resource Acquisition Is Initialization (RAII)

12.11 Some Libraries Support Both Exceptions and Error Codes

# **Table of Contents**

12.12 Logging

12.13 Looking Ahead to Contracts

12.14 Wrap-Up

## **13 Standard Library Containers and Iterators**

13.1 Introduction

13.2 Introduction to Containers

13.2.1 Common Nested Types in Sequence and Associative Containers

13.2.2 Common Container Member and Non-Member Functions

13.2.3 Requirements for Container Elements

13.3 Working with Iterators

13.3.1 Using `istream_iterator` for Input and `ostream_iterator` for Output

13.3.2 Iterator Categories

13.3.3 Container Support for Iterators

13.3.4 Predefined Iterator Type Names

13.3.5 Iterator Operators

13.4 A Brief Introduction to Algorithms

13.5 Sequence Containers

13.6 `vector` Sequence Container

13.6.1 Using vectors and Iterators

13.6.2 `vector` Element-Manipulation Functions

13.7 `list` Sequence Container

13.8 `deque` Sequence Container

13.9 Associative Containers

13.9.1 `multiset` Associative Container

13.9.2 `set` Associative Container

13.9.3 `multimap` Associative Container

13.9.4 `map` Associative Container

13.10 Container Adaptors

13.10.1 `stack` Adaptor

13.10.2 `queue` Adaptor

13.10.3 `priority_queue` Adaptor

# **Table of Contents**

13.11 bitset Near Container

13.12 Optional: A Brief Intro to Big O

13.13 Optional: A Brief Intro to Hash Tables

13.14 Wrap-Up

## **14 Standard Library Algorithms and C++20 Ranges & Views**

14.1 Introduction

14.2 Algorithm Requirements: C++20 Concepts

14.3 Lambdas and Algorithms

14.4 Algorithms

14.4.1 fill, fill\_n, generate and generate\_n

14.4.2 equal, mismatch and lexicographical\_compare

14.4.3 remove, remove\_if, remove\_copy and remove\_copy\_if

14.4.4 replace, replace\_if, replace\_copy and replace\_copy\_if

14.4.5 Shuffling, Counting, and Minimum and Maximum Element Algorithms

14.4.6 Searching and Sorting Algorithms

14.4.7 swap, iter\_swap and swap\_ranges

14.4.8 copy\_backward, merge, unique, reverse, copy\_if and copy\_n

14.4.9 inplace\_merge, unique\_copy and reverse\_copy

14.4.10 Set Operations

14.4.11 lower\_bound, upper\_bound and equal\_range

14.4.12 min, max and minmax

14.4.13 Algorithms gcd, lcm, iota, reduce and partial\_sum from Header <numeric>

14.4.14 Heapsort and Priority Queues

14.5 Function Objects (Functors)

14.6 Projections

14.7 C++20 Views and Functional-Style Programming

14.7.1 Range Adaptors

14.7.2 Working with Range Adaptors and Views

14.8 Intro to Parallel Algorithms

14.9 Standard Library Algorithm Summary

14.10 A Look Ahead to C++23 Ranges



# **Table of Contents**

## 14.11 Wrap-Up

## 15 Templates, C++20 Concepts and Metaprogramming

### 15.1 Introduction

### 15.2 Custom Class Templates and Compile-Time Polymorphism

### 15.3 C++20 Function Template Enhancements

#### 15.3.1 C++20 Abbreviated Function Templates

#### 15.3.2 C++20 Templated Lambdas

### 15.4 C++20 Concepts: A First Look

#### 15.4.1 Unconstrained Function Template multiply

#### 15.4.2 Constrained Function Template with a C++20 Concepts requires Clause

#### 15.4.3 C++20 Predefined Concepts

### 15.5 Type Traits

### 15.6 C++20 Concepts: A Deeper Look

#### 15.6.1 Creating a Custom Concept

#### 15.6.2 Using a Concept

#### 15.6.3 Using Concepts in Abbreviated Function Templates

#### 15.6.4 Concept-Based Overloading

#### 15.6.5 requires Expressions

#### 15.6.6 C++20 Exposition-Only Concepts

#### 15.6.7 Techniques Before C++20 Concepts: SFINAE and Tag Dispatch

### 15.7 Testing C++20 Concepts with `static_assert`

### 15.8 Creating a Custom Algorithm

### 15.9 Creating a Custom Container and Iterators

#### 15.9.1 Class Template ConstIterator

#### 15.9.2 Class Template Iterator

#### 15.9.3 Class Template MyArray

#### 15.9.4 MyArray Deduction Guide for Braced Initialization

#### 15.9.5 Using MyArray and Its Custom Iterators with `std::ranges` Algorithms

### 15.10 Default Arguments for Template Type Parameters

### 15.11 Variable Templates

### 15.12 Variadic Templates and Fold Expressions

# **Table of Contents**

15.12.1 tuple Variadic Class Template

15.12.2 Variadic Function Templates and an Intro to C++17 Fold Expressions

15.12.3 Types of Fold Expressions

15.12.4 How Unary-Fold Expressions Apply Their Operators

15.12.5 How Binary-Fold Expressions Apply Their Operators

15.12.6 Using the Comma Operator to Repeatedly Perform an Operation

15.12.7 Constraining Parameter Pack Elements to the Same Type

## **15.13 Template Metaprogramming**

15.13.1 C++ Templates Are Turing Complete

15.13.2 Computing Values at Compile-Time

15.13.3 Conditional Compilation with Template Metaprogramming and constexpr if

15.13.4 Type Metafunctions

## **15.14 Wrap-Up**

## **16 C++20 Modules: Large-Scale Development**

### **16.1 Introduction**

### **16.2 Compilation and Linking Before C++20**

### **16.3 Advantages and Goals of Modules**

### **16.4 Example: Transitioning to ModulesHeader Units**

### **16.5 Modules Can Reduce Translation Unit Sizes and Compilation Times**

### **16.6 Example: Creating and Using a Module**

16.6.1 module Declaration for a Module Interface Unit

16.6.2 Exporting a Declaration

16.6.3 Exporting a Group of Declarations

16.6.4 Exporting a namespace

16.6.5 Exporting a namespace Member

16.6.6 Importing a Module to Use Its Exported Declarations

16.6.7 Example: Attempting to Access Non-Exported Module Contents

### **16.7 Global Module Fragment**

### **16.8 Separating Interface from Implementation**

16.8.1 Example: Module Implementation Units

16.8.2 Example: Modularizing a Class

16.8.3 :private Module Fragment

# **Table of Contents**

## **16.9 Partition**

16.9.1 Example: Module Interface Partition Units

16.9.2 Module Implementation Partition Units

16.9.3 Example: Submodules vs. Partitions

## **16.10 Additional Modules Examples**

16.10.1 Example: Importing the C++ Standard Library as Modules

16.10.2 Example: Cyclic Dependencies Are Not Allowed

16.10.3 Example: imports Are Not Transitive

16.10.4 Example: Visibility vs. Reachability

## **16.11 Migrating Code to Modules**

## **16.12 Future of Modules and Modules Tooling**

## **16.13 Wrap-Up**

# **17 Parallel Algorithms and Concurrency: A High-Level View**

## **17.1 Introduction**

## **17.2 Standard Library Parallel Algorithms (C++17)**

17.2.1 Example: Profiling Sequential and Parallel Sorting Algorithms

17.2.2 When to Use Parallel Algorithms

17.2.3 Execution Policies

17.2.4 Example: Profiling Parallel and Vectorized Operations

17.2.5 Additional Parallel Algorithm Notes

## **17.3 Multithreaded Programming**

17.3.1 Thread States and the Thread Life Cycle

17.3.2 Deadlock and Indefinite Postponement

## **17.4 Launching Tasks with `std::jthread`**

17.4.1 Defining a Task to Perform in a Thread

17.4.2 Executing a Task in a `jthread`

17.4.3 How `jthread` Fixes thread

## **17.5 ProducerConsumer Relationship: A First Attempt**

## **17.6 ProducerConsumer: Synchronizing Access to Shared Mutable Data**

17.6.1 Class `SynchronizedBuffer`: Mutexes, Locks and Condition Variables

17.6.2 Testing `SynchronizedBuffer`

# **Table of Contents**

17.7 ProducerConsumer: Minimizing Waits with a Circular Buffer

17.8 Readers and Writers

17.9 Cooperatively Canceling jthreads

17.10 Launching Tasks with `std::async`

17.11 Thread-Safe, One-Time Initialization

17.12 A Brief Introduction to Atomics

17.13 Coordinating Threads with C++20 Latches and Barriers

17.13.1 C++20 `std::latch`

17.13.2 C++20 `std::barrier`

17.14 C++20 Semaphores

17.15 C++23: A Look to the Future of C++ Concurrency

17.15.1 Parallel Ranges Algorithms

17.15.2 Concurrent Containers

17.15.3 Other Concurrency-Related Proposals

17.16 Wrap-Up

## **18 C++20 Coroutines**

18.1 Introduction

18.2 Coroutine Support Libraries

18.3 Installing the `conurrencpp` and `generator` Libraries

18.4 Creating a Generator Coroutine with `co_yield` and the `generator` Library

18.5 Launching Tasks with `conurrencpp`

18.6 Creating a Coroutine with `co_await` and `co_return`

18.7 Low-Level Coroutines Concepts

18.8 C++23 Coroutines Enhancements

18.9 Wrap-Up

A Operator Precedence and Grouping

B Character Set

Index

Online Chapters and Appendices

# **Table of Contents**

19 Stream I/O and C++20 Text Formatting

C Number Systems

D Preprocessor