Username: Pralay Patoria **Book:** The C++ Standard Library: A Tutorial and Reference, Second Edition. No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

Contents

Preface to the Second Edition
Acknowledgments for the Second Edition
Preface to the First Edition
Acknowledgments for the First Edition

1 About This Book
- 1.1 Why This Book
- 1.2 Before Reading This Book
- 1.3 Style and Structure of the Book
- 1.4 How to Read This Book
- 1.5 State of the Art
- 1.6 Example Code and Additional Information
- 1.7 Feedback

2 Introduction to C++ and the Standard Library
- 2.1 History of the C++ Standards
  - 2.1.1 Common Questions about the C++11 Standard
  - 2.1.2 Compatibility between C++98 and C++11
- 2.2 Complexity and Big-O Notation

3 New Language Features
- 3.1 New C++11 Language Features
  - 3.1.1 Important Minor Syntax Cleanups
  - 3.1.2 Automatic Type Deduction with `auto`
  - 3.1.3 Uniform Initialization and Initializer Lists
  - 3.1.4 Range-Based `for` Loops
  - 3.1.5 Move Semantics and Rvalue References
  - 3.1.6 New String Literals
  - 3.1.7 Keyword `noexcept`
  - 3.1.8 Keyword `constexpr`
  - 3.1.9 New Template Features
  - 3.1.10 Lambdas
  - 3.1.11 Keyword `decltype`
  - 3.1.12 New Function Declaration Syntax
  - 3.1.13 Scoped Enumerations
  - 3.1.14 New Fundamental Data Types
- 3.2 Old “New” Language Features
  - 3.2.1 Explicit Initialization for Fundamental Types
  - 3.2.2 Definition of `main()`

4 General Concepts
- 4.1 Namespace `std`
- 4.2 Header Files
- 4.3 Error and Exception Handling
  - 4.3.1 Standard Exception Classes
  - 4.3.2 Members of Exception Classes
  - 4.3.3 Passing Exceptions with Class `exception_ptr`
  - 4.3.4 Throwing Standard Exceptions

4.3.5 Deriving from Standard Exception Classes

4.4 Callable Objects

4.5 Concurrency and Multithreading

4.6 Allocators

5 Utilities

5.1 Pairs and Tuples

5.1.1 Pairs

5.1.2 Tuples

5.1.3 I/O for Tuples

5.1.4 Conversions between `tuple` s and `pair` s

5.2 Smart Pointers

5.2.1 Class `shared_ptr`

5.2.2 Class `weak_ptr`

5.2.3 Misusing Shared Pointers

5.2.4 Shared and Weak Pointers in Detail

5.2.5 Class `unique_ptr`

5.2.6 Class `unique_ptr` in Detail

5.2.7 Class `auto_ptr`

5.2.8 Final Words on Smart Pointers

5.3 Numeric Limits

5.4 Type Traits and Type Utilities

5.4.1 Purpose of Type Traits

5.4.2 Type Traits in Detail

5.4.3 Reference Wrappers

5.4.4 Function Type Wrappers

5.5 Auxiliary Functions

5.5.1 Processing the Minimum and Maximum

5.5.2 Swapping Two Values

5.5.3 Supplementary Comparison Operators

5.6 Compile-Time Fractional Arithmetic with Class `ratio<>`

5.7 Clocks and Timers

5.7.1 Overview of the Chrono Library

5.7.2 Durations

5.7.3 Clocks and Timepoints

5.7.4 Date and Time Functions by C and POSIX

5.7.5 Blocking with Timers

5.8 Header Files `<cstddef>` , `<cstdlib>` , and `<cstring>`

5.8.1 Definitions in `<cstddef>`

5.8.2 Definitions in `<cstdlib>`

5.8.3 Definitions in `<cstring>`

6 The Standard Template Library

6.1 STL Components

6.2 Containers

6.2.1 Sequence Containers

6.2.2 Associative Containers

6.2.3 Unordered Containers

6.2.4 Associative Arrays

6.2.5 Other Containers

6.2.6 Container Adapters

6.3 Iterators

6.3.1 Further Examples of Using Associative and Unordered Containers

6.3.2 Iterator Categories
6.4 Algorithms
6.4.1 Ranges
6.4.2 Handling Multiple Ranges
6.5 Iterator Adapters
6.5.1 Insert Iterators
6.5.2 Stream Iterators
6.5.3 Reverse Iterators
6.5.4 Move Iterators
6.6 User-Defined Generic Functions
6.7 Manipulating Algorithms
6.7.1 “Removing” Elements
6.7.2 Manipulating Associative and Unordered Containers
6.7.3 Algorithms versus Member Functions
6.8 Functions as Algorithm Arguments
6.8.1 Using Functions as Algorithm Arguments
6.8.2 Predicates
6.9 Using Lambdas
6.10 Function Objects
6.10.1 Definition of Function Objects
6.10.2 Predefined Function Objects
6.10.3 Binders
6.10.4 Function Objects and Binders versus Lambdas
6.11 Container Elements
6.11.1 Requirements for Container Elements
6.11.2 Value Semantics or Reference Semantics
6.12 Errors and Exceptions inside the STL
6.12.1 Error Handling
6.12.2 Exception Handling
6.13 Extending the STL
6.13.1 Integrating Additional Types
6.13.2 Deriving from STL Types

7 STL Containers
7.1 Common Container Abilities and Operations
7.1.1 Container Abilities
7.1.2 Container Operations
7.1.3 Container Types
7.2 Arrays
7.2.1 Abilities of Arrays
7.2.2 Array Operations
7.2.3 Using `array`s as C-Style Arrays
7.2.4 Exception Handling
7.2.5 Tuple Interface
7.2.6 Examples of Using Arrays
7.3 Vectors
7.3.1 Abilities of Vectors
7.3.2 Vector Operations
7.3.3 Using Vectors as C-Style Arrays
7.3.4 Exception Handling
7.3.5 Examples of Using Vectors
7.3.6 Class `vector<bool>`
7.4 Deques
7.4.1 Abilities of Deques
7.4.2 Deque Operations
7.4.3 Exception Handling

7.4.4 Examples of Using Deques

7.5 Lists

7.5.1 Abilities of Lists

7.5.2 List Operations

7.5.3 Exception Handling

7.5.4 Examples of Using Lists

7.6 Forward Lists

7.6.1 Abilities of Forward Lists

7.6.2 Forward List Operations

7.6.3 Exception Handling

7.6.4 Examples of Using Forward Lists

7.7 Sets and Multisets

7.7.1 Abilities of Sets and Multisets

7.7.2 Set and Multiset Operations

7.7.3 Exception Handling

7.7.4 Examples of Using Sets and Multisets

7.7.5 Example of Specifying the Sorting Criterion at Runtime

7.8 Maps and Multimaps

7.8.1 Abilities of Maps and Multimaps

7.8.2 Map and Multimap Operations

7.8.3 Using Maps as Associative Arrays

7.8.4 Exception Handling

7.8.5 Examples of Using Maps and Multimaps

7.8.6 Example with Maps, Strings, and Sorting Criterion at Runtime

7.9 Unordered Containers

7.9.1 Abilities of Unordered Containers

7.9.2 Creating and Controlling Unordered Containers

7.9.3 Other Operations for Unordered Containers

7.9.4 The Bucket Interface

7.9.5 Using Unordered Maps as Associative Arrays

7.9.6 Exception Handling

7.9.7 Examples of Using Unordered Containers

7.10 Other STL Containers

7.10.1 Strings as STL Containers

7.10.2 Ordinary C-Style Arrays as STL Containers

7.11 Implementing Reference Semantics

7.12 When to Use Which Container

8 STL Container Members in Detail

8.1 Type Definitions

8.2 Create, Copy, and Destroy Operations

8.3 Nonmodifying Operations

8.3.1 Size Operations

8.3.2 Comparison Operations

8.3.3 Nonmodifying Operations for Associative and Unordered Containers

8.4 Assignments

8.5 Direct Element Access

8.6 Operations to Generate Iterators

8.7 Inserting and Removing Elements

8.7.1 Inserting Single Elements

8.7.2 Inserting Multiple Elements

8.7.3 Removing Elements

8.7.4 Resizing

8.8 Special Member Functions for Lists and Forward Lists

8.8.1 Special Member Functions for Lists (and Forward Lists)

8.8.2 Special Member Functions for Forward Lists Only

8.9 Container Policy Interfaces

- 8.9.1 Nonmodifying Policy Functions
- 8.9.2 Modifying Policy Functions
- 8.9.3 Bucket Interface for Unordered Containers
- 8.10 Allocator Support
  - 8.10.1 Fundamental Allocator Members
  - 8.10.2 Constructors with Optional Allocator Parameters

9 STL Iterators

- 9.1 Header Files for Iterators
- 9.2 Iterator Categories
  - 9.2.1 Output Iterators
  - 9.2.2 Input Iterators
  - 9.2.3 Forward Iterators
  - 9.2.4 Bidirectional Iterators
  - 9.2.5 Random-Access Iterators
  - 9.2.6 The Increment and Decrement Problem of Vector Iterators
- 9.3 Auxiliary Iterator Functions
  - 9.3.1 `advance()`
  - 9.3.2 `next()` and `prev()`
  - 9.3.3 `distance()`
  - 9.3.4 `iter_swap()`
- 9.4 Iterator Adapters
  - 9.4.1 Reverse Iterators
  - 9.4.2 Insert Iterators
  - 9.4.3 Stream Iterators
  - 9.4.4 Move Iterators
- 9.5 Iterator Traits
  - 9.5.1 Writing Generic Functions for Iterators
- 9.6 Writing User-Defined Iterators

10 STL Function Objects and Using Lambdas

- 10.1 The Concept of Function Objects
  - 10.1.1 Function Objects as Sorting Criteria
  - 10.1.2 Function Objects with Internal State
  - 10.1.3 The Return Value of `for_each()`
  - 10.1.4 Predicates versus Function Objects
- 10.2 Predefined Function Objects and Binders
  - 10.2.1 Predefined Function Objects
  - 10.2.2 Function Adapters and Binders
  - 10.2.3 User-Defined Function Objects for Function Adapters
  - 10.2.4 Deprecated Function Adapters
- 10.3 Using Lambdas
  - 10.3.1 Lambdas versus Binders
  - 10.3.2 Lambdas versus Stateful Function Objects
  - 10.3.3 Lambdas Calling Global and Member Functions
  - 10.3.4 Lambdas as Hash Function, Sorting, or Equivalence Criterion

11 STL Algorithms

- 11.1 Algorithm Header Files
- 11.2 Algorithm Overview
  - 11.2.1 A Brief Introduction
  - 11.2.2 Classification of Algorithms
- 11.3 Auxiliary Functions
- 11.4 The `for_each()` Algorithm
- 11.5 Nonmodifying Algorithms
  - 11.5.1 Counting Elements

- 11.5.2 Minimum and Maximum
- 11.5.3 Searching Elements
- 11.5.4 Comparing Ranges
- 11.5.5 Predicates for Ranges

11.6 Modifying Algorithms
- 11.6.1 Copying Elements
- 11.6.2 Moving Elements
- 11.6.3 Transforming and Combining Elements
- 11.6.4 Swapping Elements
- 11.6.5 Assigning New Values
- 11.6.6 Replacing Elements

11.7 Removing Algorithms
- 11.7.1 Removing Certain Values
- 11.7.2 Removing Duplicates

11.8 Mutating Algorithms
- 11.8.1 Reversing the Order of Elements
- 11.8.2 Rotating Elements
- 11.8.3 Permuting Elements
- 11.8.4 Shuffling Elements
- 11.8.5 Moving Elements to the Front
- 11.8.6 Partition into Two Subranges

11.9 Sorting Algorithms
- 11.9.1 Sorting All Elements
- 11.9.2 Partial Sorting
- 11.9.3 Sorting According to the *n*th Element
- 11.9.4 Heap Algorithms

11.10 Sorted-Range Algorithms
- 11.10.1 Searching Elements
- 11.10.2 Merging Elements

11.11 Numeric Algorithms
- 11.11.1 Processing Results
- 11.11.2 Converting Relative and Absolute Values

12 Special Containers

12.1 Stacks
- 12.1.1 The Core Interface
- 12.1.2 Example of Using Stacks
- 12.1.3 A User-Defined Stack Class
- 12.1.4 Class `stack<>` in Detail

12.2 Queues
- 12.2.1 The Core Interface
- 12.2.2 Example of Using Queues
- 12.2.3 A User-Defined Queue Class
- 12.2.4 Class `queue<>` in Detail

12.3 Priority Queues
- 12.3.1 The Core Interface
- 12.3.2 Example of Using Priority Queues
- 12.3.3 Class `priority_queue<>` in Detail

12.4 Container Adapters in Detail
- 12.4.1 Type Definitions
- 12.4.2 Constructors
- 12.4.3 Supplementary Constructors for Priority Queues
- 12.4.4 Operations

12.5 Bitsets
- 12.5.1 Examples of Using Bitsets

12.5.2 Class bitset in Detail

13 Strings

13.1 Purpose of the String Classes

13.1.1 A First Example: Extracting a Temporary Filename

13.1.2 A Second Example: Extracting Words and Printing Them Backward

13.2 Description of the String Classes

13.2.1 String Types

13.2.2 Operation Overview

13.2.3 Constructors and Destructor

13.2.4 Strings and C-Strings

13.2.5 Size and Capacity

13.2.6 Element Access

13.2.7 Comparisons

13.2.8 Modifiers

13.2.9 Substrings and String Concatenation

13.2.10 Input/Output Operators

13.2.11 Searching and Finding

13.2.12 The Value npos

13.2.13 Numeric Conversions

13.2.14 Iterator Support for Strings

13.2.15 Internationalization

13.2.16 Performance

13.2.17 Strings and Vectors

13.3 String Class in Detail

13.3.1 Type Definitions and Static Values

13.3.2 Create, Copy, and Destroy Operations

13.3.3 Operations for Size and Capacity

13.3.4 Comparisons

13.3.5 Character Access

13.3.6 Generating C-Strings and Character Arrays

13.3.7 Modifying Operations

13.3.8 Searching and Finding

13.3.9 Substrings and String Concatenation

13.3.10 Input/Output Functions

13.3.11 Numeric Conversions

13.3.12 Generating Iterators

13.3.13 Allocator Support

14 Regular Expressions

14.1 The Regex Match and Search Interface

14.2 Dealing with Subexpressions

14.3 Regex Iterators

14.4 Regex Token Iterators

14.5 Replacing Regular Expressions

14.6 Regex Flags

14.7 Regex Exceptions

14.8 The Regex ECMAScript Grammar

14.9 Other Grammars

14.10 Basic Regex Signatures in Detail

15 Input/Output Using Stream Classes

15.1 Common Background of I/O Streams

15.1.1 Stream Objects

15.1.2 Stream Classes

15.1.3 Global Stream Objects

15.1.4 Stream Operators

15.1.5 Manipulators
15.1.6 A Simple Example
15.2 Fundamental Stream Classes and Objects
15.2.1 Classes and Class Hierarchy
15.2.2 Global Stream Objects
15.2.3 Header Files
15.3 Standard Stream Operators `<<` and `>>`
15.3.1 Output Operator `<<`
15.3.2 Input Operator `>>`
15.3.3 Input/Output of Special Types
15.4 State of Streams
15.4.1 Constants for the State of Streams
15.4.2 Member Functions Accessing the State of Streams
15.4.3 Stream State and Boolean Conditions
15.4.4 Stream State and Exceptions
15.5 Standard Input/Output Functions
15.5.1 Member Functions for Input
15.5.2 Member Functions for Output
15.5.3 Example Uses
15.5.4 `sentry` Objects
15.6 Manipulators
15.6.1 Overview of All Manipulators
15.6.2 How Manipulators Work
15.6.3 User-Defined Manipulators
15.7 Formatting
15.7.1 Format Flags
15.7.2 Input/Output Format of Boolean Values
15.7.3 Field Width, Fill Character, and Adjustment
15.7.4 Positive Sign and Uppercase Letters
15.7.5 Numeric Base
15.7.6 Floating-Point Notation
15.7.7 General Formatting Definitions
15.8 Internationalization
15.9 File Access
15.9.1 File Stream Classes
15.9.2 Rvalue and Move Semantics for File Streams
15.9.3 File Flags
15.9.4 Random Access
15.9.5 Using File Descriptors
15.10 Stream Classes for Strings
15.10.1 String Stream Classes
15.10.2 Move Semantics for String Streams
15.10.3 `char*` Stream Classes
15.11 Input/Output Operators for User-Defined Types
15.11.1 Implementing Output Operators
15.11.2 Implementing Input Operators
15.11.3 Input/Output Using Auxiliary Functions
15.11.4 User-Defined Format Flags
15.11.5 Conventions for User-Defined Input/Output Operators
15.12 Connecting Input and Output Streams
15.12.1 Loose Coupling Using `tie()`
15.12.2 Tight Coupling Using Stream Buffers
15.12.3 Redirecting Standard Streams
15.12.4 Streams for Reading and Writing

- 15.13 The Stream Buffer Classes
  - 15.13.1 The Stream Buffer Interfaces
  - 15.13.2 Stream Buffer Iterators
  - 15.13.3 User-Defined Stream Buffers
- 15.14 Performance Issues
  - 15.14.1 Synchronization with C's Standard Streams
  - 15.14.2 Buffering in Stream Buffers
  - 15.14.3 Using Stream Buffers Directly

16 Internationalization

- 16.1 Character Encodings and Character Sets
  - 16.1.1 Multibyte and Wide-Character Text
  - 16.1.2 Different Character Sets
  - 16.1.3 Dealing with Character Sets in C++
  - 16.1.4 Character Traits
  - 16.1.5 Internationalization of Special Characters
- 16.2 The Concept of Locales
  - 16.2.1 Using Locales
  - 16.2.2 Locale Facets
- 16.3 Locales in Detail
- 16.4 Facets in Detail
  - 16.4.1 Numeric Formatting
  - 16.4.2 Monetary Formatting
  - 16.4.3 Time and Date Formatting
  - 16.4.4 Character Classification and Conversion
  - 16.4.5 String Collation
  - 16.4.6 Internationalized Messages

17 Numerics

- 17.1 Random Numbers and Distributions
  - 17.1.1 A First Example
  - 17.1.2 Engines
  - 17.1.3 Engines in Detail
  - 17.1.4 Distributions
  - 17.1.5 Distributions in Detail
- 17.2 Complex Numbers
  - 17.2.1 Class `complex<>` in General
  - 17.2.2 Examples Using Class `complex<>`
  - 17.2.3 Operations for Complex Numbers
  - 17.2.4 Class `complex<>` in Detail
- 17.3 Global Numeric Functions
- 17.4 Valarrays

18 Concurrency

- 18.1 The High-Level Interface: `async()` and Futures
  - 18.1.1 A First Example Using `async()` and Futures
  - 18.1.2 An Example of Waiting for Two Tasks
  - 18.1.3 Shared Futures
- 18.2 The Low-Level Interface: Threads and Promises
  - 18.2.1 Class `std::thread`
  - 18.2.2 Promises
  - 18.2.3 Class `packaged_task<>`
- 18.3 Starting a Thread in Detail
  - 18.3.1 `async()` in Detail
  - 18.3.2 Futures in Detail

18.3.3 Shared Futures in Detail

18.3.4 Class `std::promise` in Detail

18.3.5 Class `std::packaged_task` in Detail

18.3.6 Class `std::thread` in Detail

18.3.7 Namespace `this_thread`

18.4 Synchronizing Threads, or the Problem of Concurrency

18.4.1 Beware of Concurrency!

18.4.2 The Reason for the Problem of Concurrent Data Access

18.4.3 What Exactly Can Go Wrong (the Extent of the Problem)

18.4.4 The Features to Solve the Problems

18.5 Mutexes and Locks

18.5.1 Using Mutexes and Locks

18.5.2 Mutexes and Locks in Detail

18.5.3 Calling Once for Multiple Threads

18.6 Condition Variables

18.6.1 Purpose of Condition Variables

18.6.2 A First Complete Example for Condition Variables

18.6.3 Using Condition Variables to Implement a Queue for Multiple Threads

18.6.4 Condition Variables in Detail

18.7 Atomics

18.7.1 Example of Using Atomics

18.7.2 Atomics and Their High-Level Interface in Detail

18.7.3 The C-Style Interface of Atomics

18.7.4 The Low-Level Interface of Atomics

19 Allocators

19.1 Using Allocators as an Application Programmer

19.2 A User-Defined Allocator

19.3 Using Allocators as a Library Programmer

Bibliography

Newsgroups and Forums

Books and Web Sites

Index