

C++ Cookbook

How to write great code with the latest C++ releases

Wayne Murphy



www.bpbonline.com

First Edition 2024

Copyright © BPB Publications, India

ISBN: 978-93-55515-377

All Rights Reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means or stored in a database or retrieval system, without the prior written permission of the publisher with the exception to the program listings which may be entered, stored and executed in a computer system, but they can not be reproduced by the means of publication, photocopy, recording, or by any electronic and mechanical means.

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

To View Complete
BPB Publications Catalogue
Scan the QR Code:



www.bpbonline.com

[Kup książkę](#)

Dedicated to

*My family and friends
who have supported me,
during this book, and for everything else.*

and

*My managers at work that believed in me
ever since I first started working in software,
and those along the way.*

*Thanks **Murray, Ed, Shane**, et al.*

About the Author

Wayne Murphy grew up in Brampton, Ontario, Canada. Graduating from Sheridan College in town. He started work at his first programming job in 1987, and over the years been at many companies, in several different roles, with various technologies. For most of the past 20 years, Wayne has been consulting in his company, Great Leap Forwards Inc. When Wayne is not at work, he is working on some of his own code, and spending time with his family.

About the Reviewers

- ❖ **Maxim Chetrușca** is a Software Engineer with almost a decade of professional experience in C++. Throughout his career, he focused on real-time, latency sensitive systems. During his work, he used C++ for the backend of the busiest websites, distributed databases as well as squeezing microseconds from high frequency trading systems. A firm believer of C++ philosophy, he had the chance to use the language versions ranging from 03 to 20 in production, running the code on 4 different platforms. Outside of working hours he spends time learning about new car models with his 2-year-old son.
- ❖ **Kris Jusiak** is a passionate Software Engineer with experience across various industries, including telecommunications, gaming, and most recently, finance. He specializes in modern C++ development, with a keen focus on performance and quality. Kris is also an active conference speaker and open-source enthusiast, having created multiple open-source libraries.
- ❖ **Chetan Sachdeva** is a Tech Enthusiast, with over 13 years of expertise in C++ and Python in software development. With a dynamic career spanning Automotive, AR/VR, IoT, Android native development, Typography, and Printing RIP. His true passion lies in solving complex problems with data structures and algorithms. His entrepreneurial spirit has led to the successful design of software and products for startups. Beyond his professional endeavors, Chetan is an avid reader of non-fiction and IT-related books, staying at the forefront of industry trends.

Acknowledgement

I remember that warm summer day in the mid-1970s, when my father's father told me *You should get into computers*, and me not knowing what one was. I remember my dad taking me into his work, because they just got their first computer there. I later recall when my parents got us our first family computer, I thought to myself, *I will never be bored again*. I have to thank them for that, because I have not been bored yet.

I am very grateful to all those publishing material (books, web-pages, videos) in the goal of wanting people to learn what they know. I hope I have done your efforts justice.

My gratitude also goes to the team at BPB Publications, for giving me a chance to do this.

Preface

C++ Cookbook walks you through all the recent new features. In a cookbook-type style that talks about each new class or function, it shows you in simple terms how to use it. Authored by a software professional, with over 3 decades of experience, their passion for coding shows in the book. This book takes the reader on a tour of what they will need to know to be up to date with the latest C++ abilities.

The book outlines new features, with lots of code examples. They can use it as a reference guide, or progress through each how-to recipe to maximize their knowledge. Sometimes it will give suggestions on the best approach, but mainly wants to inform you of options, and lets you take the right path for your individual situation. The book should help those that have not kept up to date with recent C++ releases. Whether your company was sticking to an older standard, or you are starting with a new product. There are a lot of great new features available, and this book will help you working with the latest and greatest functionality. You can walk through all the chapters with their code examples, or use it as a quick-reference guide for something specific.

After reading it, you will be up to date and will make you and your project work better.

Chapter 1: Working with Concepts – It defines what a concept is, how to use one, and how to create your own. It discusses the different styles how concepts are defined; whether creating a single constraint on a variable or a function, making a requires clause, or combining concepts to make a conjunction or compound requirement.

Chapter 2: Using the New Core Language Concepts – The chapter initiates a long journey to learn what existing concepts are included in C++. There are concepts for comparison, assignment, checking on the type, and hierarchy.

Chapter 3: Using the New Comparison Concepts – The chapter continues delving into predefined concepts. We look at those relating to equality, and comparison.

Chapter 4: Using the New Iterator Concepts – Our next step into concepts looks at all of the concepts defined for iterators, and also talks about the progression of functionality with types of iterators. We also touch on some other concepts relating to moving or copying values.

Chapter 5: Using the New Object Concepts – We pick up where we left off, talking about concepts relating to the constraints of moving or copying values. We also discuss the similarities and differences between the concepts semiregular and regular.

Chapter 6: Using the New Callable Concepts – We will talk about concepts relating functions, or other mechanisms that we can call to execute some code.

Chapter 7: Const Related Specifiers – With this chapter, we end our lengthy talk about concepts and discuss specifiers relating to defining what is constant. We go over previously existing specifiers, and talk about newer ones that will help your code be more efficient.

Chapter 8: Concurrent Processing – The chapter discusses threads, callbacks, and different ways for your code to be executed safely by different threads at the same time.

Chapter 9: Coroutines – The chapter presents a new mechanism for executing code. We get into the details of the parts of a coroutine, how to create them, and how to use them.

Chapter 10: Organizing Your Code with Modules – The chapter covers a great new addition to the language. Modules will help you organize your code easier. We get into the details of its components and go through how to using other modules, and learn how to write our own code.

Chapter 11: Introduction to Ranges and Views – The chapter starts discussion about what ranges, views, and spans are. The differences between them, and how to use them are discussed. Since ranges relate to concepts, we will talk about some of the existing range concepts.

Chapter 12: Range Access and Non-Modifying Sequence Functions for Ranges – The chapter continues to discuss ranges, and some of the functionality relating to subranges, sizes, iterating, and comparing.

Chapter 13: Range Algorithms: Sort, Search and More – The next leg of our journey talks about ranges deals with searching and sorting. We can perform functions on a range to get a value, or even a new range by doing a permutation.

Chapter 14: Range Algorithms: Memory and Modification Functions – We focus on memory; moving values around, and doing transformations.

Chapter 15: Views and Range Adaptors – The chapter talks about the many functions that exist so we can get values in a view by calling a range with an adaptor.

Chapter 16: Range Factories and Utilities – We conclude our talk about ranges by showing some exiting new functionality. We can create an infinite view of values, whether by using a simple function or a generator. There is code showing how to even format your range for output.

Chapter 17: New Features for Containers – The chapter covers a lot of ground, dealing with new container types, and simple new functions.

Chapter 18: Making it Easier to Code – There are a number of new features discussed that you will be happy to see such as easier output, new enhancements for strings and ranges, plus an easier way to use enums and bits. There is also new types for dates, times and time zones.

Chapter 19: Making Your Code Cleaner – The chapter covers many recent features for lambdas, new suffixes, optional and expected arguments, plus new preprocessor directives for cleaner code.

Chapter 20: Making Your Code Safer – The chapter touches on some important new features to make code less fragile.

Chapter 21: Making Your Code Faster and Easier to Debug – We conclude by talking about ways to make your code run faster; including some new functions, and new attributes. We also talk about new types that could help you debug your code.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the *Coloured Images* of the book:

<https://rebrand.ly/de3g86d>

The code bundle for the book is also hosted on GitHub at

<https://github.com/bpbpublications/C-Plus-Plus-Cookbook>.

In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at
<https://github.com/bpbpublications>. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at :

business@bpbonline.com for more details.

At **www.bpbonline.com**, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



Table of Contents

1. Working with Concepts.....	1
Introduction	1
Structure	3
Objectives	3
Recipe 1.1: Creating your first concept	3
Recipe 1.2: Using a concept	4
<i>Suggestion</i>	5
Recipe 1.3: Creating a constraint.....	6
Recipe 1.4: Creating a conjunction.....	6
<i>Suggestion</i>	7
Recipe 1.5: Making a requires clause.....	8
Recipe 1.6: Constraining a function.....	9
Recipe 1.7: Creating a requires expression	9
Recipe 1.8: Multiple requires expressions	10
Recipe 1.9: Nested requirements.....	11
Recipe 1.10: Function templates.....	11
Recipe 1.11: Compound requirements	13
Recipe 1.12: Different types of expressions	13
Recipe 1.13: Specializations	14
Conclusion	17
Points to remember.....	17
References.....	17
2. Using the New Core Language Concepts	19
Introduction	19
Structure	19
Objectives	20
Recipe 2.1: same_as	21

Recipe 2.2: convertible_to	24
Recipe 2.3: derived_from	26
Recipe 2.4: integral, signed_integral, unsigned_integral	31
Recipe 2.5: floating_point.....	32
<i>Suggestion</i>	34
Recipe 2.6: assignable_from.....	34
<i>Suggestion</i>	36
Recipe 2.7: swappable.....	36
Recipe 2.8: destructible	38
Recipe 2.9: constructible_from	41
Recipe 2.10: default_initializable	43
Recipe 2.11: move_constructible, copy_constructible	46
Recipe 2.12: common_with and common_reference_with	51
Conclusion	54
Points to remember	54
References.....	54
3. Using the New Comparison Concepts	55
Introduction	55
Structure	55
Objectives	56
Recipe 3.1: equality_comparable	56
Recipe 3.2: equality_comparable_with	60
Recipe 3.3: totally_ordered	60
<i>Suggestion</i>	63
Recipe 3.4: totally_ordered_with	64
Recipe 3.5: three_way_comparable	64
<i>Suggestion</i>	67
Recipe 3.6: three_way_comparable with custom types	68
<i>Suggestion</i>	78
Recipe 3.7: three_way_comparable_with	78
Conclusion	81

Points to remember	81
References.....	81
4. Using the New Iterator Concepts	83
Introduction	83
Structure	84
Objectives	85
Recipe 4.1: weakly_incrementable.....	85
Recipe 4.2: input_or_output_iterator	87
Recipe 4.3: sentinel_for.....	89
Recipe 4.4: sized_sentinel_for.....	91
Recipe 4.5: indirectly_readable.....	92
Recipe 4.6: indirectly_writable	92
Recipe 4.7: incrementable.....	92
Recipe 4.8: input_iterator	93
Recipe 4.9: output_iterator.....	93
Recipe 4.10: forward_iterator	93
Recipe 4.11: bidirectional_iterator.....	96
Recipe 4.12: random_access_iterator.....	98
Recipe 4.13: contiguous_iterator	103
Recipe 4.14: indirectly_movable, and indirectly_movable_storable	108
Recipe 4.15: indirectly_copyable, and indirectly_copyable_storable	109
Recipe 4.16: indirectly_swappable.....	109
Recipe 4.17: indirectly_comparable	109
Recipe 4.18: permutable	109
Recipe 4.19: mergeable	110
Conclusion	110
Points to remember	110
References.....	110
5. Using the New Object Concepts	111
Introduction	111
Structure	111

Objectives	112
Recipe 5.1: movable	112
<i>Suggestion</i>	112
Recipe 5.2: copyable.....	118
Recipe 5.3: semiregular.....	120
Recipe 5.4: regular	122
Conclusion	124
Points to remember	124
References.....	124
6. Using the New Callable Concepts	125
Introduction	125
Structure	126
Objectives	126
Recipe 6.1: invocable.....	127
<i>Suggestion</i>	131
Recipe 6.2: regular_invocable.....	131
Recipe 6.3: predicate	133
Recipe 6.4: relation and equivalence_relation.....	134
Recipe 6.5: strict_weak_order	134
Recipe 6.6: indirectly_unary_invocable, and indirectly_regular_unary_invocable. 135	
Recipe 6.7: indirect_binary_predicate	136
Recipe 6.8: indirect_equivalence_relation	136
Recipe 6.9: indirect_strict_weak_order	137
Conclusion	137
Points to remember	137
References.....	137
7. Const Related Specifiers	139
Introduction	139
Structure	139
Objectives	141
Recipe 7.1: Looking at the constexpr enhancements	141

Recipe 7.2: constexpr for transient allocation	150
Recipe 7.3: consteval	153
Recipe 7.4: constexpr	157
Recipe 7.5: constexpr if	160
Recipe 7.6: if consteval	166
Recipe 7.7: Some hidden problems with constexpr	173
Conclusion	175
Points to remember	175
References	175
8. Concurrent Processing	177
Introduction	177
Structure	177
Objectives	178
Recipe 8.1: std::jthread is the new thread	179
Recipe 8.2: std::stop_token	190
Recipe 8.3: std::stop_callback	193
Recipe 8.4: std::stop_source	196
Recipe 8.5: std::counting_semaphore, and std::binary_semaphore	200
Recipe 8.6: std::atomic<std::shared_ptr>, and std::atomic<std::weak_ptr>	204
Recipe 8.7: std::atomic<T>::wait, and std::atomic<T>::notify*	207
Recipe 8.8: std::latch	210
Recipe 8.9: std::barrier	211
Recipe 8.10: std::atomic_ref	214
Conclusion	216
Points to remember	217
References	217
9. Coroutines	219
Introduction	219
Structure	220
Objectives	220
Recipe 9.1: What is a coroutine	221

Recipe 9.2: Components of a coroutine: The promise.....	224
Recipe 9.3: Components of a coroutine: The state, and the handle	227
Recipe 9.4: Coroutine restrictions	242
Recipe 9.5: Generators	244
Conclusion	248
Points to remember	249
References.....	249
10. Organizing Your Code with Modules	251
Introduction	251
Structure	251
Objectives	252
Recipe 10.1: What is a module.....	252
Recipe 10.2: What the compiler does with a module.....	260
Recipe 10.3: How to use a module.....	261
Recipe 10.4: Visibility and reachability	266
Recipe 10.5: Layout of a module.....	268
Recipe 10.6: Cyclical modules	271
Recipe 10.7: Importing the standard modules.....	273
Recipe 10.8: What are module partitions.....	274
Conclusion	275
Points to remember	275
References.....	276
11. Introduction to Ranges and Views.....	277
Introduction	277
Structure	277
Objectives	278
Recipe 11.1: What is a range	278
Recipe 11.2: What is a view	281
Recipe 11.3: What is a span.....	282
Recipe 11.4: Range concepts	284
Recipe 11.5: Range primitives.....	287

Recipe 11.6: What is a subrange	287
Recipe 11.7: What is <code>ranges::dangling</code>	289
Recipe 11.8: What is a <code>view_interface</code>	292
Recipe 11.9: What is an <code>owning_view</code>	295
Conclusion	295
Points to remember	295
References.....	295
12. Range Access and Non-Modifying Sequence Functions for Ranges	297
Introduction	297
Structure	297
Objectives	298
Recipe 12.1: Review the basic range access functions.....	298
Recipe 12.2: The <code>for_each</code> functions	303
Recipe 12.3: Looking at <code>count</code> , <code>any_of</code> , <code>all_of</code> , and <code>none_of</code> functions	306
Recipe 12.4: Looking at comparison functions	310
Recipe 12.5: Search functionality	314
Conclusion	327
Points to remember	327
References.....	327
13. Range Algorithms: Sort, Search and More	329
Introduction	329
Structure	329
Objectives	330
Recipe 13.1: Minimum and maximum functions	330
Recipe 13.2: Sorting and partitioning functions	334
Recipe 13.3: Binary search functions, and set functions	342
Recipe 13.4: Permutation functions	347
Recipe 13.5: Fold functions	351
Conclusion	353
Points to remember	353
References.....	353

14. Range Algorithms: Memory and Modification Functions	355
Introduction	355
Structure	355
Objectives	356
Recipe 14.1: Heap functions.....	356
Recipe 14.2: Uninitialized memory functions.....	360
Recipe 14.3: Modifying sequence functions	361
Conclusion	376
Points to remember	376
References.....	377
15. Views and Range Adaptors	379
Introduction	379
Structure	379
Objectives	380
Recipe 15.1: Progressing from views and ranges, to range adaptors	381
Recipe 15.2: Some simple range adaptors	382
Recipe 15.3: Using take() and drop() adaptors	385
Recipe 15.4: Using range adaptors with composition	387
Recipe 15.5: Using filter() and transform() adaptors.....	389
Recipe 15.6: Working with the join() and split() adaptors.....	392
Recipe 15.7: Working with the keys() and values() adaptors	395
Recipe 15.8: Using chunk() and slide() adaptors	397
Recipe 15.9: Using zip() adaptor	403
Recipe 15.10: A quick look at reverse().....	404
Recipe 15.11: Looking at special adaptors	405
Conclusion	408
Points to remember	409
References.....	409
16. Range Factories and Utilities	411
Introduction	411
Structure	411

Objectives	412
Recipe 16.1: Factory overview, and the <code>ranges::iota_view</code> factory	412
Recipe 16.2: The <code>views::repeat</code> , and <code>views::cartesian_product</code>	415
Recipe 16.3: Creating your range factory	416
Recipe 16.4: Formatting ranges	417
Recipe 16.5: Swapping ranges	420
Recipe 16.6: The utility <code>ranges::to</code>	424
Conclusion	425
Points to remember	425
References	426
17. New Features for Containers	427
Introduction	427
Structure	427
Objectives	428
Recipe 17.1: <code>Contains</code>	429
Recipe 17.2: Looking at <code>std::span</code>	431
Recipe 17.3: <code>std::counted_iterator</code>	432
Recipe 17.4: <code>std::is_bounded_array</code> and <code>std::is_unbounded_array</code>	435
Recipe 17.5: <code>std::to_array</code> converts to <code>std::array</code>	436
Recipe 17.6: <code>std::erase</code> and <code>std::erase_if</code>	438
Recipe 17.7: <code>std::flat_map</code> and <code>std::flat_set</code>	440
Recipe 17.8: Iterators pair constructors for stack and queue	441
Recipe 17.9: Allow default arguments for pair's forwarding constructors	442
Recipe 17.10: The <code>push_range()</code> function for queue, stack, and priority_queue	443
Recipe 17.11: What is <code>std::mdspan</code>	444
Conclusion	446
Points to remember	447
References	447
18. Making it Easier to Code	449
Introduction	449
Structure	449

Objectives	450
Recipe 18.1: String formatting	451
Recipe 18.2: <code>std::print</code>	456
Recipe 18.3: <code>starts_with</code> and <code>ends_with</code> for strings.....	458
Recipe 18.4: Other enhancements for strings and ranges	460
Recipe 18.5: using <code>enum</code> reduces typing for enums	461
Recipe 18.6: New date features	464
Recipe 18.7: New time features	470
Recipe 18.8: Timezone library	474
Recipe 18.9: <code>std::midpoint</code>	477
Recipe 18.10: <code><numbers></code> includes <code>pi</code> and <code>e</code>	479
Recipe 18.11: Bit manipulation.....	481
Recipe 18.12: Designated initializer for aggregates.....	489
Conclusion	491
Points to remember	491
References.....	492
19. Making Your Code Cleaner	493
Introduction	493
Structure	493
Objectives	495
Recipe 19.1: Familiar template syntax Lambdas	495
Recipe 19.2: Lambda parameter packs.....	497
Recipe 19.3: Literal class types in non-type template parameters	499
Recipe 19.4: Multidimensional subscript operator.....	501
Recipe 19.5: <code>std::make_shared</code> supports arrays.....	503
Recipe 19.6: CTAD improvements.....	505
Recipe 19.7: Deducing <code>this</code>	507
Recipe 19.8: Transforming <code>auto</code>	510
Recipe 19.9: <code>char8_t</code> and <code>std::u8string</code> types	512
Recipe 19.10: Suffix for <code>std::size_t</code>	512
Recipe 19.11: Named universal character escapes.....	513

Recipe 19.12: Monadic operations for <code>std::optional</code>	514
Recipe 19.13: Using <code>std::expected</code>	522
Recipe 19.14: Pre-processing directives <code>elifdef</code> and <code>elifndef</code>	524
Recipe 19.15: <code>invoke_r<T></code>	526
Recipe 19.16: <code><spanstream></code>	526
Recipe 19.17: Bit initialization	528
Recipe 19.18: Extend <code>init-statement</code> to allow alias-declaration	529
Conclusion	530
Points to remember	530
References.....	531
20. Making Your Code Safer.....	533
Introduction	533
Structure	533
Objectives	534
Recipe 20.1: Fixed width floating-point types	534
Recipe 20.2: Conditionally explicit constructor	535
Recipe 20.3: <code>#warning</code>	541
Recipe 20.4: <code>std::start_lifetime_as</code>	542
Recipe 20.5: <code>std::out_ptr()</code> , <code>std::inout_ptr()</code>	543
Recipe 20.6: Overloads of <code>std::to_chars()</code> and <code>std::from_chars()</code>	544
Recipe 20.7: <code>constexpr std::bitset</code>	548
Recipe 20.8: <code>std::to_underlying</code>	549
Recipe 20.9: Direct initialization	551
Conclusion	553
Points to remember	553
References.....	553
21. Making Your Code Faster and Easier to Debug	555
Introduction	555
Structure	555
Objectives	556

Recipe 21.1: <code>[[likely]]</code> and <code>[[unlikely]]</code>	556
Recipe 21.2: <code>[[assume()]]</code>	559
Recipe 21.3: <code>std::unreachable</code>	561
Recipe 21.4: <code>string::substr</code> is faster.....	562
Recipe 21.5: <code>string::resize_and_overwrite</code>	563
Recipe 21.6: PMR containers.....	564
Recipe 21.7: Comparing integrals	566
Recipe 21.8: <code>std::endl</code> vs <code>"\n"</code>	567
Recipe 21.9: Synchronized stream output	569
Recipe 21.10: <code>std::source_location</code>	573
Recipe 21.11: <code>std::stacktrace</code>	575
Conclusion	578
Points to remember	578
References.....	578
Index	579-586

Welcome to C++

C++ is a wonderful language. You can often be working with high-level abstractions, but still be able to dig down deep and work with raw memory. C++ has been around for decades, and with the efforts of many good people out there, is still evolving. We can assume that this means that it will probably be around for decades more. It is on lists of most used languages (sometimes in the top 10, and if not, then certainly the top 20), as well C++ is on lists of programmers' most liked languages. C++ is not owned by a big company, has multiple companies making compilers for it, and many companies making tools and libraries for it. It runs on different operating systems, for purposes big and small. The governing body meets regularly to discuss, debate, and decide what will go into the next release. Releases were decided to be every three years, which they have done in the age of modern C++ (11, 14, 17, 20, 23, and they are already talking about C++26). To be precise, the releases are defined every three years. At the time of writing this book, at the end of December, C++23 was still being voted on whether it is officially released.

We could not talk about everything to do with C++ just in one book, and so we are here to talk about the latest couple of releases; C++20 and C++23.

Recipe 1: Who is this book for?

Problem: We want to know who should read this book.

Solution: It can appeal to different types of audiences.

Of course, we would like to say anybody and everybody can read this book. However, this is a cookbook focusing on the newest releases. If you are just starting out knowing little about C++, then this probably is not the best first book for you. Having said that though, today's world lets us learn in different media in different ways. If you have a handful of websites and other learning video channels that teach you the basics, then yes, you can buy the book as a secondary source of learning.

This book does not only discuss the basics but focuses on new features. In that regard, it can be used by any who knows the basics but is not kept up to date with all the latest features. If you look at the following figure, we would say that we are trying to focus on what you will want to know to get caught up on the latest features.

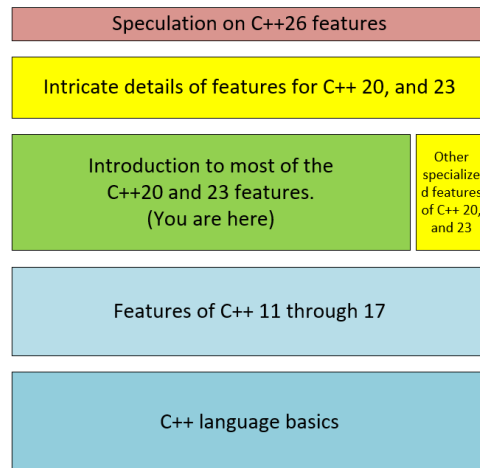


Figure 1: What this book is about

Recipe 2: What is the format for each chapter?

Problem: How is each chapter planned out?

Solution: It is a cookbook-type design, with many problem and solution topics for various Recipes.

Each chapter has a particular focus:

- We have an **Introduction**, or overview of what the chapter is about.
- The **Structure** is given, which lists the features or topics that we will talk about in detail.
- We have a brief statement on the **Objectives** of the chapter.

- Then, we iterate over our **Recipes**, which are individual topics in the chapter. Sometimes, a recipe can be about an idea, but often it is about a particular class or function. There can be between 5-15 recipes in a chapter. We have tried to provide lots of code samples to help illustrate what the recipe is about.
- Next, we give a **Conclusion**, which summarizes what you have learned in the chapter.
- We give reminders of some important facets in the **Points to remember** section.
- Lastly, we provide **References**, aside from needing to say where the information came from, which is also a huge and sincere thank-you to those who provided information and helped inform others. There are probably more resources that we could have mentioned. For brevity's sake, if we saw similar information on a handful of different websites and videos, then we jotted down what we thought best presented the information.

Recipe 3: Is C++23 worth reading about?

Problem: If there have been so many releases, can we just learn it as we go?

Solution: You probably can learn a bit as you go, however the good and bad part about C++, is that it has evolved over the decades. So yes, you could continue using nothing but new and delete, and maybe you could get by and use a few new classes as you trip over them. The result though, is that the code, and coder, will not be as good as they could be. Some things get deprecated, but a lot of code is still around due to historical use. Everybody says you should not use function **A()**, but now use function **B()**. And in truth, we are probably on function **D()** in some cases. You should want to at least be aware of the latest features. You may not want to, or need to, rewrite your entire code-base because new features exist, but for new functionality, you should understand what is in the latest release(s), to make an informed decision on when to use them. Compilers are smarter, computers are faster, learning tools are better, and so using newer features will not make your job more difficult. Releases are created based on input like yours, to make coding with new features safer, smarter, faster, and easier.

To get into slightly more detail, the C++20 release was huge. It was arguably the biggest C++ update ever in terms of features. What is more, it helped to define a direction on where the language should head. C++20's *Big Four* features will influence other features for years to come. C++23 is not as big as C++20, partially, because of the global pandemic, and admitted partially because there were multiple changes from C++20. Having said that, C++23 is an impressive release on its own merit, which will change how coders create software.

Refer to the following figure:

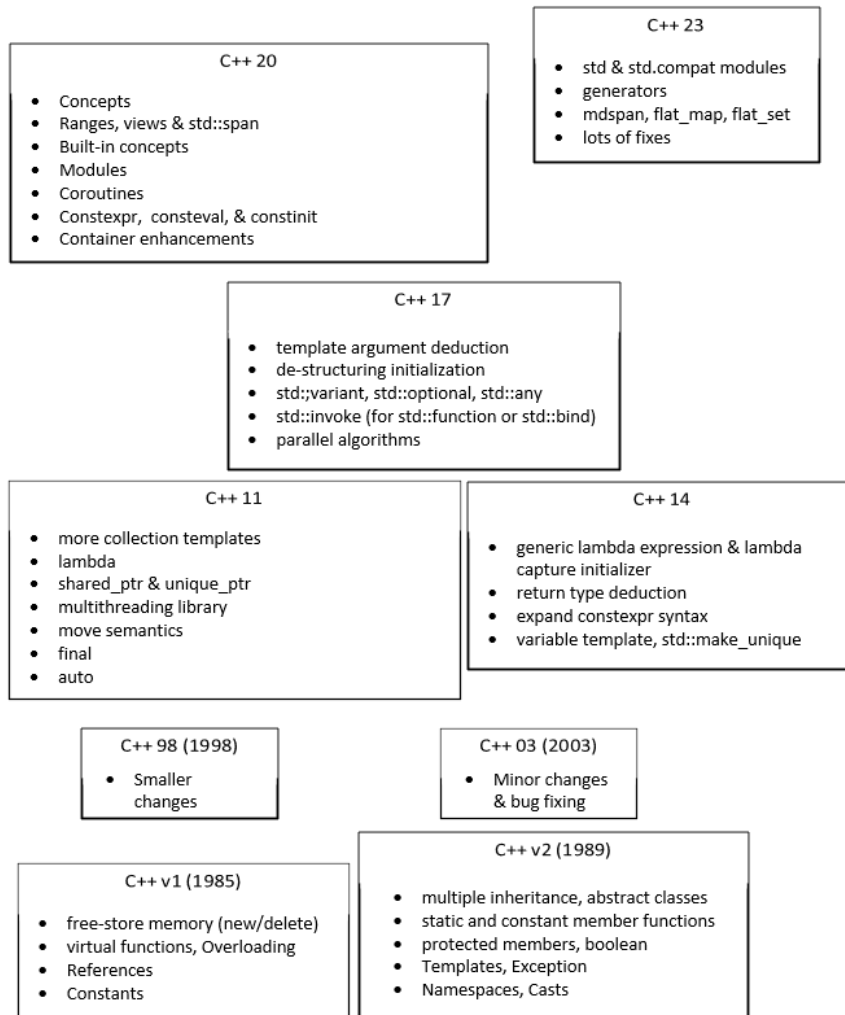


Figure 2: Overview of C++ releases

Recipe 4: Is the book just about C++23?

Problem: We want to learn more about the latest version, but have not focused too much on other recent releases.

Solution: This book talks wants you to learn the latest, but understands if you might have missed something along the way.

The book has tried to include everything that has been defined in the C++23 release. A couple of features might be vague or short, partially because some features are more important than others, but also a couple of C++23 features are still a bit vague. They are included in the release, but no one has implemented the feature, and there is not much information about them outside of the spec. As we have said, C++20 was such an important release, that some parts of C++23 are tough to understand if you do not know about the related C++20 features. We do talk quite a bit about the C++20 release, and even some parts of the C++17, but that is not to distract you from learning about C++23, but here to help you understand how we got to C++23.

Recipe 5: How to work with the code

Problem: How do you use the code samples provided?

Solution: We can walk you through some of that.

Most of the code samples were built and run in Visual Studio 2022. We have used Visual Studio Code as well. There are two main steps to use C++23 in Visual Studio. The first is to run the Visual Studio Installer program and enable using the *Latest* C++ tools. Refer to the following figure:

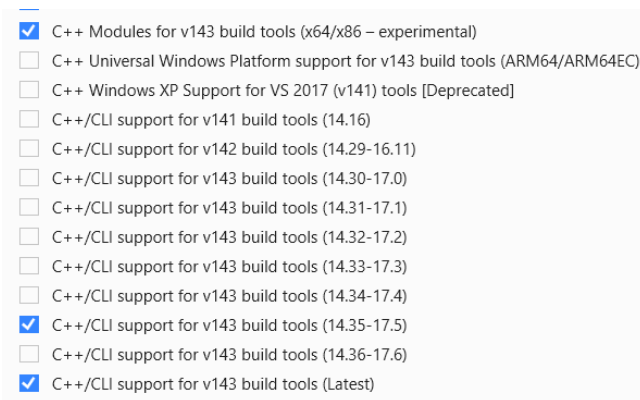


Figure 3: Showing we are using the "Latest" build tools, in the Visual Studio Installer

The next is to go into your project settings and select if you want to use the *Preview* release of the C++ Standard. This would equate to having `/std:c++latest` in your makefile. Refer to the following figure:

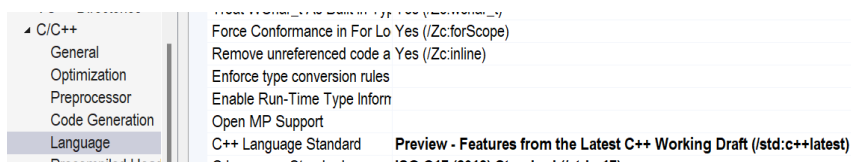


Figure 4: Visual Studio project setting to use the latest C++ standard