Embracing Modern C++ Safely

This is simply a placeholder. Your production team will replace this page with the real series page.

Embracing Modern C++ Safely

John Lakos

Vittorio Romeo

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
Sao Paulo • Sidney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Library of Congress Cataloging-in-Publication Data

LIBRARY OF CONGRESS CIP DATA WILL GO HERE; MUST BE ALIGNED AS INDICATED BY LOC

ISBN-13: NUMBER HERE
ISBN-10: NUMBER HERE
Text printed in the United States on recycled paper at PRINTER INFO HERE.
First printing, MONTH YEAR

This is John's dedication to Vittorio for being so great and writing this book so well.
JL

This is Vittorio dedication to something else.
VR

This is Slava's dedication to something else.
RK

This is Alisdair's dedication to something else.
AM

Contents

Foreword

The text of the foreword will go here.

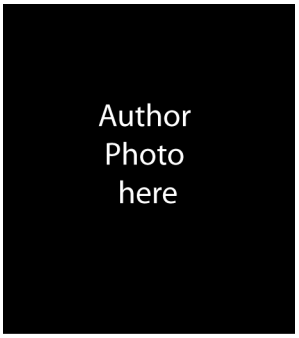
Preface

The text of the preface will go here.

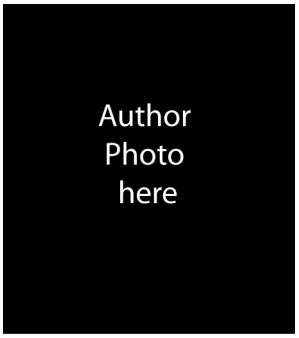
Acknowledgements

The text of the author's acknowledgements will go here.

About the Authors



John Lakos, author of *Large-Scale C++ Software Design* (Addison-Wesley, 1996) and *Large-Scale C++ Volume I: Process and Architecture* (Addison-Wesley, 2019), serves at Bloomberg in New York City as a senior architect and mentor for C++ software development worldwide. He is also an active voting member of the C++ Standards Committee's Evolution Working Group. From 1997 to 2001, Dr. Lakos directed the design and development of infrastructure libraries for proprietary analytic financial applications at Bear Stearns. From 1983 to 1997, Dr. Lakos was employed at Mentor Graphics, where he developed large frameworks and advanced ICCAD applications for which he holds multiple software patents. His academic credentials include a Ph.D. in Computer Science (1997) and an Sc.D. in Electrical Engineering (1989) from Columbia University. Dr. Lakos received his undergraduate degrees from MIT in Mathematics (1982) and Computer Science (1981).



Vittorio Romeo (B.Sc., Computer Science, 2016) is a senior software engineer at Bloomberg in London, working on mission-critical C++ middleware and delivering modern C++ training to hundreds of fellow employees. He began programming at the age of 8 and quickly fell in love with C++. Vittorio has created several open-source C++ libraries and games, has published many video courses and tutorials, and actively participates in the ISO C++ standardization process. He is an active member of the C++ community with an ardent desire to share his knowledge and learn from others: He presented more than 20 times at international C++ conferences (including CppCon, C++Now, ++it, ACCU, C++ On Sea, C++ Russia, and Meeting C++), covering topics from game development to template metaprogramming. Vittorio maintains a website (https://vittorioromeo.info/) with advanced C++ articles and a YouTube channel (https://www.youtube.com/channel/UC1XihgHdkNOQd5IBHnIZWbA) featuring well received modern C++11/14 tutorials. He is active on StackOverflow, taking great care in answering interesting C++ questions (75k+ reputation). When he is not writing code, Vittorio enjoys weightlifting and fitness-related activities as well as computer gaming and sci-fi movies.

Chapter 0

Introduction

Welcome! *Embracing Modern C++ Safely* is a *reference book* dedicated to professionals who want to leverage modern C++ features in the development and maintenance of large-scale, complex C++ software systems.

This book deliberately concentrates on the productive value afforded by each new language feature added by C++ starting with C++11, particularly when the systems and organizations involved are considered at scale. We left aside ideas and idioms, however clever and intellectually intriguing, that could hurt the bottom line when applied at large. Instead, we focus on what is objectively true and relevant to making wise economic and design decisions, with an understanding of the inevitable tradeoffs that arise in any engineering discipline. In doing so, we do our best to steer clear of subjective opinions and recommendations.

Richard Feynman famously said: "If it disagrees with experiment, it's wrong. In that simple statement is the key to science."[1] There is no better way to experiment with a language feature than letting time do its work. We took that to heart by dedicating *Embracing Modern C++ Safely* to only the features of Modern C++ that have been part of the Standard for at least five years, which grants enough perspective to properly evaluate its practical impact. Thus, we are able to provide you with a thorough and comprehensive treatment based on practical experience and worthy of your limited professional development time. If you're out there looking for tried and true ways to better use modern C++ features for improving your productivity, we hope this book will be the one you'll reach for.

What's missing from a book is as important as what's present. *Embracing Modern C++ Safely* is not a tutorial on programming, on C++, or even on new features of C++. We assume you are an experienced developer, team lead, or manager, that you already have a good command of "classic" C++98/03, and that you are looking for clear, goal-driven ways to integrate modern C++ features within your and your team's toolbox.

0.1 What Makes This Book Different

The book you're now reading aims very strongly at being objective, empirical, and practical. We simply present features, their applicability, and their potential pitfalls as reflected by the analysis of millions of human-hours of using C++11 and C++14 in the development of varied large-scale software systems; personal preference matters have been neutralized to our, and our reviewers', best ability. We wrote down the distilled truth that remains, which

[1] Richard Feynman, lecture at Cornell University, 1964. Video and commentary available at https://fs.blog/2009/12/mental-model-scientific-method.

should shape your understanding of what modern C++ has to offer to you without being skewed by our subjective opinions or domain-specific inclinations.

The final analysis and interpretation of what is appropriate for your context is left to you, the reader. Hence, this book is, by design, not a C++ style or coding-standards guide; it would, however, provide valuable input to any development organization seeking to author or enhance one.

Practicality is a topic very important to us, too, and in a very real-world, economic sense. We examine modern C++ features through the lens of a large company developing and using software in a competitive environment. In addition to showing you how to best utilize a given C++ language feature in practice, our analysis takes into account the costs associated with having that feature employed routinely in the ecosystem of a software development organization. (We believe that costs of using language features are sadly neglected by most texts.) In other words, we weigh the benefits of successfully using a feature against the risk of its widespread ineffective use (or misuse) and/or the costs associated with training and code review required to reasonably ensure that such ill-conceived use does not occur. We are acutely aware that what applies to one person or small crew of like-minded individuals is quite different from what works with a large, distributed team. The outcome of this analysis is our signature categorization of features in terms of safety of adoption — namely *safe*, *conditionally safe*, or *unsafe* features.

We are not aware of any similar text amid the rich offering of C++ textbooks; in a very real sense, we wrote it because we needed it.

0.2 Scope for the First Edition

Given the vastness of C++'s already voluminous and rapidly growing standardized libraries, we have chosen to limit this book's scope to just the language features themselves. A companion book, *Embracing Modern C++ Standard Libraries Safely*, is a separate project that we hope to tackle in the future. However, to be effective, this book must remain small, concise, and focused on what expert C++ developers need to know well to be successful right now.

In this first of an anticipated series of periodically extended volumes, we characterize, dissect, and elucidate most of the modern language features introduced into the C++ Standard starting with C++11. We chose to limit the scope of this first edition to only those features that have been in the language Standard and widely available in practice for at least five years. This limited focus enables us to more fully evaluate the real-world impact of these features and to highlight any caveats that might not have been anticipated prior to standardization and sustained, active, and widespread use in industry.

We assume you are quite familiar with essentially all of the basic and important special-purpose features of classic C++98/03, so in this book we confined our attention to just the subset of C++ language features introduced in C++11 and C++14. This book is best for you if you need to know how to safely incorporate C++11/14 language features into a predominately C++98/03 code base, today.

Over time, we expect, and hope, that practicing senior developers will emerge entirely from the postmodern C++ era. By then, a book that focuses on all of the important

features of modern C++ would naturally include many of those that were around before C++11. With that horizon in mind, we are actively planning to cover pre-C++11 material in future editions. For the time being, however, we highly recommend *Effective C++* by Scott Meyers[2] as a concise, practical treatment of many important and useful C++98/03 features.

0.3 The *EMC++S* White Paper

Throughout the writing of *Embracing Modern C++ Safely*, we have followed a set of guiding principles, which collectively drive the style and content of this book.

0.3.1 Facts (Not Opinions)

This book describes only beneficial uses and potential pitfalls of modern C++ features. The content presented is based on objectively verifiable facts, either derived from standards documents or from extensive practical experience; we explicitly avoid subjective opinion such as our evaluation of the relative merits of design tradeoffs (restraint that admittedly is a good exercise in humility). Although such opinions are often valuable, they are inherently biased toward the author's area of expertise.

Note that *safety* — the rating we use to segregate features by chapter — is the one exception to this objectivity guideline. Although the analysis of each feature aims at being entirely objective, its chapter classification — indicating the relative safety of its quotidian use in a large software-development environment — reflects our combined accumulated experience totaling decades of real-world, hands-on experience with developing a variety of large-scale C++ software systems.

0.3.2 Elucidation (Not Prescription)

We deliberately avoid prescribing any cut-and-dried solutions to address specific feature pitfalls. Instead, we merely describe and characterize such concerns in sufficient detail to equip you to devise a solution suitable for your own development environment. In some cases, we might reference techniques or publicly available libraries that others have used to work around such speed bumps, but we do not pass judgment about which workaround should be considered a best practice.

0.3.3 Brevity (Not Verbosity)

Embracing Modern C++ Safely is neither designed nor intended to be an introduction to modern C++. It is a handy reference for experienced C++ programmers who may have a passing knowledge of the recently added C++ features and a desire to perfect their understanding. Our writing style is intentionally tight, with the goal of providing you with facts, concise objective analysis, and cogent, real-world examples. By doing so we spare you the task of wading through introductory material. If you are entirely unfamiliar with

[2] **meyers05**

a feature, we suggest you start with a more elementary and language-centric text such as *The C++ Programming Language* by Bjarne Stroustrup.[3]

0.3.4 Real-World (Not Contrived) Examples

We hope you will find the examples in this book useful in multiple ways. The primary purpose of examples is to illustrate productive use of each feature as it might occur in practice. We stay away from contrived examples that give equal importance to seldom-used aspects of the feature, as to the intended, idiomatic uses. Hence, many of our examples are based on simplified code fragments extracted from real-world codebases. Though we typically change identifier names to be more appropriate to the shortened example (rather than the context and the process that led to the example), we keep the code structure of each example as close as possible to its original real-world counterpart.

0.3.5 At Scale (Not Overly Simplistic) Programs

By scale, we attempt to simultaneously capture two distinct aspects of size: (1) the sheer product size (e.g., in bytes, source lines, separate units of release) of the programs, systems, and libraries developed and maintained by a software organization; and (2) the size of an organization itself as measured by the number of software developers, quality assurance engineers, site reliability engineers, operators, and so on that the organization employs. As with many aspects of software development, what works for small programs simply doesn't scale to larger development efforts.

What's more, powerful new language features that are handled perfectly well by a few expert programmers working together in the archetypal garage on a prototype for their new start-up don't always fare as well when they are wantonly exercised by numerous members of a large software development organization. Hence, when we consider the relative safety of a feature, as defined in the next section, we do so with mindfulness that any given feature might be used, and occasionally misused, in very large programs and by a very large number of programmers having a wide range of knowledge, skill, and ability.

0.4 What Do We Mean by *Safely*?

The ISO C++ Standards Committee, of which we are members, would be remiss — and downright negligent — if it allowed any feature of the C++ language to be standardized if that feature were not reliably safe when used as intended. Still, we have chosen the word "safely" as the moniker for the signature aspect of our book, by which we indicate a comparatively favorable risk-to-reward ratio for using a given feature in a large-scale development environment. By contextualizing the meaning of the term "safe," we get to apply it to a real-world economy in which everything has a cost in multiple dimensions: risk of misuse, added maintenance burden borne by using a new feature in an older code base, and training needs for developers who might not be familiar with that feature.

[3] **stroustrup13**

Several aspects conspire to offset the value added by the adoption and widespread use of any new language feature, thereby reducing its intrinsic safety. By categorizing features in terms of safety, we strive to capture an appropriately weighted combination of the following factors:

1. Number and severity of known deficiencies
2. Difficulty in teaching consistent proper use
3. Experience level required for consistent proper use
4. Risks associated with widespread misuse

Bottom line: In this book, the degree of safety of a given feature is the relative likelihood that widespread use of that feature will have positive impact and no adverse effect on a large software company's codebase.

0.5 A *Safe* Feature

Some of the new features of modern C++ add considerable value, are easy to use, and are decidedly hard to misuse unintentionally; hence, ubiquitous adoption of such features is productive, relatively unlikely to become a problem in the context of a large-scale development organization, and to be generally encouraged — even without training. We identify such staunchly helpful, unflappable C++ features as *safe*.

For example, we categorize the `override` contextual keyword as a safe feature because it prevents bugs, serves as documentation, cannot easily be misused, and has no serious deficiencies. If someone has heard of this feature and tried to use it and the software compiles, the code base is likely better for it. Using `override` wherever applicable is always a sound engineering decision.

0.6 A *Conditionally Safe* Feature

The preponderance of new features available in modern C++ has important, frequently occurring, and valuable uses, yet how these features are used appropriately, let alone optimally, might not be obvious. What's more, some of these features are fraught with inherent dangers and deficiencies, requiring explicit training and extra care to circumnavigate their pitfalls.

For example, we deem default member initializers a *conditionally safe* feature because, although they are easy to use per se, the perhaps less-than-obvious unintended consequences of doing so (e.g., tight compile-time coupling) might be prohibitively costly in certain circumstances (e.g., might prevent relink-only patching in production).

0.7 An *Unsafe* Feature

When an expert programmer uses any C++ feature appropriately, the feature typically does no direct harm. Yet other developers — seeing the feature's use in the code base but failing to appreciate the highly specialized or nuanced reasoning justifying it — might attempt to use it in what they perceive to be a similar way, yet with profoundly less desirable results. Similarly, maintainers may change the use of a fragile feature altering its semantics in subtle but damaging ways.

Features that are classified as unsafe are those that might have valid, and even very important, use cases, yet our experience indicates that routine or widespread use thereof would be counterproductive in a typical large-scale software-development enterprise.

For example, we deem the final contextual keyword an unsafe feature because the situations in which it would be misused overwhelmingly outnumber those vanishingly few isolated cases in which it is appropriate, let alone valuable. Furthermore, its widespread use would inhibit fine-grained (e.g., hierarchical) reuse, which is critically important to the success of a large organization.

0.8 Modern C++ Feature Catalog

As an essential aspect of its design, this first edition of *Embracing Modern C++ Safely* aims to serve as a comprehensive catalog of C++11 and C++14 language features, presenting vital information for each of them in a clear, concise, consistent, and predictable format to which experienced engineers can readily refer during development or technical discourse.

0.8.1 Organization

This book is divided into five chapters, the middle three of which form the catalog characterizing modern C++ language features grouped by their respective safety classifications:

- Chapter 0: Introduction
- Chapter 1: Safe Features
- Chapter 2: Conditionally Safe Features
- Chapter 3: Unsafe Features
- Chapter 4: Parting Thoughts

For this first edition, the language-feature chapters (1, 2, and 3) each consist of two sections containing, respectively, C++11 and C++14 features having the safety level (*safe*, *conditionally safe*, or *unsafe*) corresponding to that chapter. Recall, however, that Standard Library features are outside the scope of this book.

Each feature resides in its own subsection, rendered in a canonical format:

- Description
- Use Cases
- Potential Pitfalls
- Annoyances
- See Also
- Further Reading

By constraining our treatment of each individual feature to this canonized format, we avoid gratuitous variations in rendering, thereby facilitating rapid discovery of whatever particular aspects of a given language feature you are searching for.

0.9 How To Use This Book

Depending on your needs, *Embracing Modern C++ Safely* can be handy in a variety of ways.

1. **Read the entire book from front to back.** If you are conversant with classic C++, consuming this book in its entirety all at once will provide a complete and nuanced practical understanding of each of the language features introduced by C++11 and C++14.

2. **Read the chapters in order but slowly over time.** An incremental, priority-driven approach is also possible and recommended, especially if you're feeling less sure-footed. Understanding and applying first the safe features of Chapter 1 gets you the low-hanging fruit. In time, the conditionally safe features of Chapter 2 will allow you to ease into the breadth of useful modern C++ language features, prioritizing those that are least likely to prove problematic.

3. **Read the first sections of each of the three catalog chapters first.** If you are a developer whose organization uses C++11 but not yet C++14, you can focus on learning everything that can be applied now and then circle back and learn the rest later when it becomes relevant to your evolving organization.

4. **Use the book as a quick-reference guide if and as needed.** Random access is great, too, especially now that you've made it through Chapter 0. If you prefer not to read the book in its entirety (or simply want to refer to it periodically as a refresher), reading any arbitrary individual feature subsection in any order will provide timely access to all relevant details of whichever feature is of immediate interest.

We wish you would derive value in several ways from the knowledge imbued into *Embracing Modern C++ Safely*, irrespective of how you read it. In addition to helping you become a more knowledgeable and therefore safer developer, this book aims to clarify (whether you

are a developer, a lead, or a manager) which features demand more training, attention to detail, experience, peer review, and such. The factual, objective presentation style also makes for excellent input into the preparation of coding standards and style guides that suit the particular needs of a company, project, team, or even just a single discriminating developer (which, of course, we all aim at being). Finally, any C++ software development organization that adopts this book will be taking the first steps toward leveraging modern C++ in a way that maximizes reward while minimizing risks, i.e., by embracing modern C++ *safely*. We are very much looking forward to getting feedback and suggestions for future editions of *Embracing Modern C++ Safely* at www.TODOTODOTODO.com. Happy coding!

Todo list