Chapter 0

Introduction

Welcome. You will quickly discover that this book differs from many other C++ books. *Embracing Modern C++ Safely* is a *reference book* for experienced software engineering practitioners working in large organizations and developing complex systems at scale.

- We are practicing software engineering professionals employed by a global software-development enterprise. As senior developers with decades of real-world experience, we are primarily writing for software developers, team leads, and managers working in similar environments.
- We deliberately focus on the productive value that a given language feature affords (or fails to afford), particularly when the systems and the organization that employs the developers writing them are considered at scale. We do not focus on the aesthetics of language features that could hurt the "bottom line" when applied at scale in a large organization.
- Because we focus on only the features of modern C++ that have been part of the Standard and have been actively used for at least five years, we are able to provide you with a thorough and comprehensive treatment based on practical experience and worthy of your limited professional development time.
- By focusing on what is objectively true and relevant to making wise economic and design decisions — with an understanding of the tradeoffs that will arise in real-world practice — we steer clear of subjective opinions and recommendations.
- Finally, this book — a thorough catalog of consequential information written by expert C++ programmers experienced and adept at using certain (early) modern language features in practice — is explicitly and intentionally designed *not* as a tutorial of the latest features for beginners, but as a guide for senior C++ programmers less familiar with these specific features.

0.1 Scope for the First Edition

Given the vastness of C++'s already voluminous and rapidly growing standardized libraries, we have elected 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, exhibit, dissect, and elucidate some, but not all, of the modern language features introduced into the C++ Standard since 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 also decided to confine our attention to just the subset of C++ language features introduced in C++11 and C++14, rather than attempt to cover all the useful features available in C++98/03. Since we can assume with virtual certainty that most of you are quite familiar with essentially all the basic and important special-purpose features of classical C++, we made the tough decision to focus exclusively on what you need to know today: how to safely incorporate C++11/14 language features into a predominately C++98/03 code base.

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[1] as a concise, practical treatment of many important and useful C++98/03 features.

0.2 What Makes This Book Different

Popular books and other material on modern C++ language features typically teach what their authors and/or the proponents of a given feature perceive to be good style and best practices, based largely on their own subjective experiences. These recommendations, in turn, are typically influenced heavily by (and sometimes limited to) the application domain with which the authors are familiar. The factual basis and objective reasoning upon which these recommendations are based can sometimes be tenuous if not entirely unsubstantiated, with the consequences at scale routinely omitted since the authors often have no such experience to share. Such books may be useful for beginners but are far less so for expert developers who are accustomed to — and rewarded for — drawing their own thoughtful conclusions based on objectively verifiable information.

In this atypical book, we instead focus exclusively on elucidating such truths, leaving the final analysis and its application to the reader. The fact-based, data-driven, objective approach — employed throughout this book — ensures that your understanding of what modern C++ has to offer is not skewed by our subjective opinions or domain-specific requirements, thereby facilitating choices more appropriate to your context. Hence, this book is — by design — explicitly *not* a C++ style or coding-standards guide; it would, however, provide valuable input to any development organization seeking to author one.

What's more, we examine modern C++ features through the lens of a large, for-profit company developing and using software in a real-world economy. In addition to showing

[1] **meyers97**

you how best to 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 commercial software-development organization. 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. The outcome of this analysis is our signature categorization of features in terms of *safety*, namely *safe*, *conditionally safe*, or *unsafe* features. (We'll explain what we mean by *safety* and by each of our categories later in this chapter.)

Finally, the information contained in *EMC++S* is the result of our analysis of millions of human hours of using C++11 and C++14 in the development of large-scale commercial software systems, combined with the wisdom and our expertise as senior software engineers actively working with modern C++ in the industry and participating in its continuous improvement on the Standard C++ ISO committee (Working Group 21, i.e., WG21).

0.2.1 The *EMC++S* Manifesto

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.

Facts (Not Opinions)

This book describes only beneficial usages and potential pitfalls of modern C++ features. The content presented is based on objectively verifiable facts; we explicitly avoid subjective opinion such as the relative merits of design tradeoffs. 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: While the analysis of each feature is itself completely objective, its chapter classification — indicting the relative *safety* of its quotidian use in a typical, large, corporate software-development environment — reflects our combined opinions, backed by decades of real-world, hands-on experience developing large-scale C++ software systems in various contexts.

Elucidation (Not Prescription)

We deliberately avoid prescribing any canned solutions to address specific feature pitfalls, and instead merely describe and characterize such concerns in sufficient detail to empower 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.

Brevity (Not Verbosity)

EMC++S is neither designed nor intended to be an introduction to modern C++. It is a handy reference for expert C++ programmers who have at least a passing knowledge of the 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 and sparing you the task of wading through introductory material. If you

are entirely unfamiliar with a feature, we suggest you start with a more elementary and language-centric text such as *The C++ Programming Language* by Bjarne Stroustrup.[2]

Real-World (Not Contrived) Examples

Our goal is that the examples in this book pull their weight in multiple ways. The primary purpose of examples in our book, unlike many books on C++ programing, is to illustrate productive use of the feature as it might occur *in practice* rather than in a contrived use case that merely exercises seldom used aspects of the feature. Hence, many of our examples are based on simplified code fragments extracted from real-world code bases. While we typically change the identifier names to be more appropriate to the shortened example, rather than the business process that led to the example, we keep the code structure as close as possible to the original real-world counterparts from which we derive our experience.

Large-Scale (Not "Toy") Programs

By scale, we are simultaneously capturing two distinct aspect of size: (1) the 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 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.[3]

What's more, powerful new language features that are handled perfectly well by a few expert programmers working together "in a garage" on a prototype for their new start-up simply 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 (see the next section), we do so with mindfulness that any given feature might be used (or misused) in very large programs and by a very large number of programmers having a wide range of knowledge, skill, and ability.

0.2.2 What Do We Mean by *Safely*?

We, the authors of this book, are each members of the ISO standards committee, and we would be remiss — and downright negligent — were we to allow any feature of the C++ language to be standardized if that feature would be other than reliably safe when used as intended.[4] Still, we have chosen the word *safely* as the moniker for the signature aspect of our book, and by *safely* we indicate a comparatively low risk-to-reward ratio for using a given feature widely in a large-scale development environment. In this way, we have contextualized (hijacked) the meaning of the term *safely* to apply to a real-world economy in which everything has a cost, including the risk and added maintenance burden associated

[2] **stroustrup13**

[3] **lakos96**, **lakos20**

[4] Unfortunately, such features do exist and have since C++ was first standardized in 1998. A specific example is local (per-object) memory allocation. An aggressive effort (by us and others) has been and continues to be underway to ameliorate this specific failing of C++; see **lakos22**.

with widespread use of a new feature in an older code base that is maintained by developers that might not be especially familiar with that feature.

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:

- Number and severity of known deficiencies
- Difficulty in teaching consistent proper use
- Experience level required for consistent proper use
- Risks associated with wide-spread misuse

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

A *Safe* Feature

Some of the new features of modern C++ add considerable value, are easy to use, and are decidedly hard to (unintentionally) misuse; 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 unflappable C++ features as *safe*.

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.

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 defects and deficiencies, requiring explicit training and extra care to circumnavigate their pitfalls.

We consider *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 some circumstances (e.g., prevent relink-only patching in production).

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.

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 would be counterproductive in a typical large-scale software-development enterprise.

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[5]) reuse, which is critically important to the success of a large organization.[6]

0.3 Modern C++ Language 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.

0.3.1 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*	Catalog of
Chapter 2: *Conditionally Safe Features*	Language
Chapter 3: *Unsafe Features*	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 out of scope for this book.

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

- Brief description of intended use and purpose
- Real-world motivating example(s)
- Objective analysis
  - Elucidation of less-than-obvious properties
  - Potential risks and undesirable consequences
  - Known feature deficiencies (if any)
  - Workarounds (as needed)

[5] **lakos20**, section 0.4, pp. 20–28
[6] **lakos20**, section 0.9, pp. 86–97

- Cross-references to related features
- External references for 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 features is sought.

0.4 How To Use This Book

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

1. **Read the entire book from front to back.** If you are an expert developer, 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.** If you are a less sure-footed developer, understanding and applying first the *safe* features of Chapter 1, followed in time by the *conditionally safe* features of Chapter 2, will allow you to grow 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.** 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 nonetheless provide timely access to all relevant details of whichever feature is of immediate interest.

The knowledge imbued into this book — irrespective of how it is consumed — can be valuable in many important ways. In addition to helping you become a more knowledgeable and therefore *safer* developer, this book aims to elucidate (to developers and managers alike) which features demand more training, attention to detail, experience, peer review, and so on. 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, after all, we all are). Finally, any C++ software development organization that adopts this book wholesale will be taking the first steps toward levering modern C++ in a way that maximizes reward while minimizing risks, i.e., by embracing modern C++ *safely*.

Bibliography

lakos20
John Lakos, *Large-Scale C++ — Volume I: Process and Architecture* (Reading, MA: Addison-Wesley, 2019)

lakos22
John Lakos and Joshua Berne, *C++ Allocators for the Working Programmer* (Boston, MA: Addison-Wesley, forthcoming

lakos96
John Lakos, *Large-Scale C++ Software Design* (Reading, MA: Addison-Wesley, 1996)

meyers97
Scott Meyers.*Effective C++*, second ed. (Boston, MA: Addison Wesley, 1997)

stroustrup13
Bjarne Stroustrup, *The C++ Programming Language*, 4th ed. (Boston, MA: Addison-Wesley, 2013)