Preface

Maybe I don't love C++ the same way I love my kids, nor even as much as climbing 10% smooth tarmac in 32° on the rivet,[1] although at times it does come close. But I do count myself blessed that I get to spend the parts of my life that are devoted to work in the practice of, to paraphrase Frederick P. Brooks, "creation by exertion of my imagination." I consider myself doubly blessed that I have at my disposal the singularly powerful, dangerous, and spellbinding language that is C++.

That all sounds very hearts and flowers, but you may have picked up this book because the title suggests that it will be a beat-up of C++. Indeed, you may be an aficionado of Java, or C, or another of the popular major languages, and have seized on a copy of *Imperfect C++* eager to find evidence to justify why you've given C++ a wide berth. If that is you, you may be disappointed, because this book is rather a critical celebration of C++. But stick around anyway; you might find some reasons why you should start looking towards C++ instead.

What You Will Learn

I wrote this book to empower fellow developers. It takes a critical, but constructive, look at C++ and its imperfections, and presents practical measures for avoiding or ameliorating them. When you've read it, I hope you'll have a better grasp of:

- How to overcome several of the deficiencies in C++'s type system
- The usefulness of template programming in increasing code flexibility and robustness
- How to survive in the realm of undefined behavior—that which is not addressed by the standard—including dynamic libraries, static objects, and threading
- The costs of implicit conversions, the troubles they bring, and the alternative of effective and manageable generalized programming via explicit conversions
- How to write software that is, or may more easily be made, compatible with other compilers, libraries, threading models, and the like
- What compilers do "behind the scenes" and how they may be influenced
- The tricky interoperability of arrays and pointers, and techniques by which they may be dissuaded from behaving like each other
- The power of C++ to support the *Resource Acquisition Is Initialization* mechanism and the variety of problem domains in which it can be applied
- How to minimize your effort by maximizing your compiler's ability to detect errors

You will certainly be equipped to write code that is more efficient, more maintainable, more robust, and more flexible.

It's my intention that even very experienced C++ practitioners will find new ideas and some new techniques with which to stimulate the mind and enhance their existing practice. Programmers with less experience will be able to appreciate the principles involved and to use the

[1]The cyclists among you will know of what I speak.

techniques within their own work, moving to fill any gaps in their understanding of the details of the techniques as their knowledge grows.

I don't expect any of you to agree with everything that I have to say, but I do expect even the most contentious material to stimulate your understanding of your use of this formidable language.

What I Assume You Know

Unless one wants to write a *very* large book, a good degree of knowledge must be assumed. It would be churlish to stipulate that you have read a precise set of texts, but I do assume that you have knowledge and experience sufficient to be comfortable with most of the concepts contained in Scott Meyer's *Effective C++* and Herb Sutter's *Exceptional C++* series. I also assume that you have a copy of the language bible: Bjarne Stroustrup's *The C++ Programming Language.* I don't assume you've read Stroustrup cover to cover—I haven't (yet)—but you should use it as the ultimate reference for the language, as there's a gem on every other page.

Imperfect C++ contains a fair amount of template code—which modern C++ book doesn't?—but I do not assume that you're a guru[2] or have advanced knowledge of meta-programming. Nonetheless, it's probably best if you're at least familiar with using templates, such as those that form the popular parts of the C++ standard library. I have tried to keep the template use down to a reasonable level, but it has to be acknowledged that the support for templates is the very thing that allows C++ to "self-repair," and it is that, therefore, which largely accounts for the existence of this book.

Since flexibility and practicality are big things with me, this is not a book whose code can only be used with a small minority of "bleeding edge" compilers; nearly everything in the book will work with just about any reasonably modern compiler (see Appendix A). Certainly there are good freely available compilers, and you can have confidence that your compiler will support the code.

Wherever possible, I've avoided reference to particular operating environments, libraries, and technologies. However, I do touch on several, so it would be useful, though by no means essential, to have some grounding in some of the following: COM and/or CORBA, dynamic libraries (UNIX and/or Win32), STL, threads (POSIX and/or Win32), UNIX, and Win32. The bibliography contains numerous references to good books on these and other subjects. It would also be useful to have familiarity with more than one machine architecture, though again this is not essential.

Since C remains the lingua franca of interlanguage and operating system development, it continues to be an extremely important language. Notwithstanding that this is a book about C++, there are many areas in which the common heritage of C and C++ comes into focus, and I make no apologies for addressing both languages in those circumstances. Indeed, as we see in Part Two, we need to fall back on C to support several advanced uses of C++.

There's one important assumption about you that I am making. I assume that you believe in doing quality work, and are motivated to finding new ways in which you can achieve this. This book cannot claim to be the sole source of such new ways of approaching C++. Rather it represents a practical, and in some cases heretical, look at the problems we all encounter with the language, and can at best form a part of your library of essential texts. The ultimate responsibility is yours. All the rest is just getting the best tools to back you up.

[2]There are several books listed in the Bibliography that can help you on the long journey to becoming one, if you're willing to invest the effort.

Organization

The main content of the book is divided into six parts. Each part is comprised of an introduction, followed by between five and seven chapters, each of which is further divided into relevant sections.

Inspired by the title of the book, I try to highlight the actual imperfections, so you will find them throughout the text. In the early parts of the book, the imperfections come thick and fast, reflecting the relatively straightforward nature both of the imperfections themselves and of their solutions. Each subsection refers to a particular feature of the language and generally describes an imperfection. Wherever possible, a specific technique and/or software technology is presented which either answers the problem, or provides the developer with control over it. As the book progresses, the imperfections become less discrete and more significant, with correspondingly lengthy and detailed discussions.

The book does not follow the modern "buffet" format, nor does it have a single contiguous thread requiring it to be read from front to back. Having said that, most of the later chapters are described in terms of, and will occasionally be built on, the content of earlier ones so, unless you're feeling perverse, you'll be better off reading it in order. However, once you've read it once, you should be able to come back to any point for reference without needing to read the whole thing again. Within the chapters, sections generally follow a sequential format, so I would recommend that you read each chapter in that vein.

In terms of difficulty, it's certainly the case that Parts One through Four follow a progression from reasonably straightforward to seriously demanding.[3] Although Parts Five and Six rely on some of the material from Parts Three and Four, they are considerably less challenging, and you should feel yourself cruising along to the appendixes.

Following the main content of the book are four short appendixes. Appendix A details the compilers and libraries used in the research for *Imperfect C++*. Appendix B regales you with some of the slack-jawed blunders of a young C++ engineer, taking his first steps in the land of the double crosses. Appendix C describes the Arturius project, a free, open-source compiler-multiplexer, which is also included on the CD. Appendix D describes the contents of the CD-ROM.

I have a very consistent, perhaps strict, coding style; you may well call it pedantic. Former colleagues and users of my libraries have certainly done so. But I do it the way I do because there are no unanswered questions as to where everything goes, which means I can come back to it years later and dive straight in. The downside is that I need a twenty-one-inch monitor and an industrial-strength laser printer.

In order to minimize the effects of my coding style on the readability of *Imperfect C++,* I've taken a few liberties in the code examples presented throughout the book. You'll see a lot of ellipses (. . .) in the examples, and this generally means something that's either been covered in a previous related example, or reflects boilerplate code that we all use (e.g., the proscription of methods from client code access—see section 2.2). Only the aspects of style that have manifest effects on reliability are discussed, in Chapter 17.[4]

---

[3]Parts of Parts Three and Four hurt my brain still.

[4]If you absolutely must sample the voluminous splendor of the rest of my coding style, there's plenty of exemplifying code in the libraries provided on the CD.

References

One of the things that irritates me when reading C++ books is when the author stipulates a fact but does not make reference to the relevant section of the standard. So in addition to including references to relevant articles and books, wherever I make a statement as to the behavior of the language I have attempted to provide references in the C (C99) or C++ (C++98) standards.

Supplementary Material

CD-ROM

The accompanying CD-ROM contains libraries, compilers (including many of the code techniques described in the book), test programs, tools, and other useful software, as well as a number of relevant articles from various publications. See Appendix D for more details on its contents.

Online Resources

Supplementary material will also be available online at *http://imperfectcplusplus.com.*[5]

Acknowledgments

In just about any book you'll ever pick up there are effusive acknowledgments to family and friends, and I can promise you they are heartfelt. Writing a book cannot be done without a huge amount of support.

Thanks to mum and Suzanne for enduring, financing, and raising the overstuffed cuckoo, who finally flitted the nest, to the other side of the world, at the overripe age of twenty-seven. Thanks to mum and Robert for helping said cuckoo and his family during a challenging but eventually fruitful couple of years. A special thanks goes to Robert for helping keep the mind at ease at a couple of important points during this time.

Thanks to Piker, for filling in the gaps in the extended family, and giving more than his share of babysitting, encouragement, and free lunches. Similar thanks to Dazzle, for always telling me he had ultimate confidence in me, and for tearing himself away from all those fascinating DBA-guru activities to put in a stalwart effort in helping with many of the boring review tasks; he'll never look at another Perl or Python script quite the same way again! Grunts of gratitude to Besso, for always showing interest, undue pride, and an encouraging perspective on my plans. And thanks to Al and Cynth (the in-laws!) for many free dinners and a limitless supply of free gourmet chocolate. (Now, where's that bike? . . .)

Essential thanks are due 808 State, Aim, Barry White, Billy Bragg, De La Soul, Fatboy Slim, George Michael, Level 42, Rush, Seal, Stevie Wonder, and The Brand New Heavies, without whom I could not possibly have survived being in "the bubble" for a year and a half.

[5]This site will also host an errata page, not that there'll be any errata, of course.

And most important, thanks to my beautiful wife, Sarah, who managed to suppress her reasonable concerns and doubts to present an almost unblemished veneer of support and confidence. A true star!

I'd like to recognize some remarkable people who've influenced my education and career thus far. Thanks are due (Professor) Bob Cryan for recognizing a fellow congenital overachiever and offering him the chance to dodge work (and ride his bike) for a further three years in postgraduate study.

I'd like to thank Richard McCormack for making me see beauty in the efficiency of code, not just elegance of concept. These days I'm sometimes accused of being too focused on efficiency; I say blame Richard! Also, Graham Jones ("Dukey") for `set -o vi`, and for six crazy months of friendship and good fun. Unrepeatable stuff!

Thanks also to Leigh and Scott Perry for introducing me to their "bolt-in" concept and other excellent techniques. They may wish to demur and instead confess to unabashed autogeneration of 16MB+ DLLs and a lamentable recent obsession with languages that run in virtual machines; I couldn't possibly comment.

Special thanks to Andy Thurling who showed generous faith in my potential when I hit the job market with Ph.D. in hand, and a level of software engineering skill in inverse proportion to my estimation thereof.[6] Andy taught me probably the single greatest lesson for anyone in this wonderful but scary profession: that we're all just "skegging it out."[7] Chuck Allison puts it more accessibly, with the ancient American Indian wisdom: "He who learns from one who is learning drinks from a running stream."

Crucial to the success of any book are the publishers, reviewers, and advisors. Thanks to my editor, Peter Gordon, who encouraged, calmed, and contained an ebullient and bullish author on the tortuously slow emotional rollercoaster that is a first book. Thanks also to Peter's able assistant Bernard Gaffney, who managed the process and endured multiple e-mails a day with patience and restraint, as well as the rest of the production and marketing staff at Addison-Wesley: Amy Fleischer, Chanda Leary-Coutu, Heather Mullane, Jacquelyn Doucette, Jennifer Andrews, Kim Boedigheimer, and Kristy Hart. Heartfelt thanks (and apologies) to my project manager, Jessica Balch, of Pine Tree Composition, who had the questionable pleasure of sifting through the text and weeding out all my poor grammar, weak attempts at humor, and British-English spelling (a myriad "ise"s instead of "ize"s). Also a special note of thanks to Debbie Lafferty for encouragement when *Imperfect C++* was nothing but a catchy phrase I dreamt up one night in 2002.

Thanks to my loyal band of reviewers—Chuck Allison, Dave Brooks, Darren Lynch, Duane Yates, Eugene Gershnik, Gary Pennington, George Frasier, Greg Peet, John Torjo, Scott Patterson, and Walter Bright—without whom I'd have a flat face from having fallen on it too many times. Some made me laugh, others made me question the sanity of the enterprise, but the feedback helped improve the final result no end. What a wonderful world we live in where friendships with people from a wide spectrum of countries, most of whom I've never physically met, can engender this level of helpfulness. Bravo!

[6] I didn't even know the difference between real mode and protected mode!

[7] This is a North Yorkshire term meaning "making the best of what information you have at the time."

Thanks also to the Addison-Wesley reviewers—Dan Saks, JC van Winkel, Jay Roy, Ron McCarty, Justin Shaw, Nevin Liber, and Steve Clamage—whose feedback was crucial in bringing the book in under 600 pages, and not riddled with turgid prose and careless mistakes. Having been on the other side of the writing/review process, I know what it takes to do a thorough review, and I really appreciate the efforts.

Thanks to Peter Dimov, for graciously allowing me to use his superb quote in Chapter 26, and for providing some excellent feedback on the chapters of Part Five. Thanks to Kevlin Henney for casting his eye over Chapter 19 and some interesting discussions on smart casts, to Joe Goodman for helping me sift through my original dross to present a decent discussion of C++ ABIs (Chapters 7 and 8), and to Thorsten Ottosen for performing similar *denonsensisination* duties on the *Design by Contract* aspects of Chapter 1.

Special thanks to Chuck Allison, Herb Sutter, Joe Casad, John Dorsey, and Jon Erickson for great support and encouragement in a variety of activities over the last few years.

Thanks to Bjarne Stroustrup for subtle encouragements, and a little history lesson here and there. Oh, and for inventing this marvelous language in the first place!

Thanks to Walter Bright, for constantly improving his excellent Digital Mars C/C++ compiler throughout the writing of the book, for having such an amenable nature in the face of a constant barrage from the world's least patient man, and for graciously involving me in the D language development, which has been one of the many sources of inspiration for this book. Similar thanks to Greg Comeau, albeit that he didn't have much improving to do with the Comeau compiler: it is the most conformant in the business! As well as being great and responsive vendors, both these excellent fellows have provided repeated encouragement, advice, and input on a whole range of issues.

Thanks to CMP publishing, for letting me use material from some of my articles in the book, and include several original articles on the CD (see Appendix D).

Thanks to Borland, CodePlay, Digital Mars, Intel, Metrowerks, and Microsoft for providing me with their compilers to support my research, writing, and open-source library activities.

Special thanks to Digital Mars, Intel, and Watcom for granting permission to include their compilers on the CD (see Appendix D). And Greg Peet deserves several manly slaps on the back for his invaluable help in designing the CD and helping me with its contents.

Thanks to the readers of *C/C++ User's Journal, Dr. Dobb's Journal, BYTE,* and *Windows Developer Network* who have been kind enough to provide feedback and support for my articles and columns.

Similar thanks are also due to the C++ community at large and the kind people who give of themselves on various C++-related newsgroups (see Appendix A). These include Attila Feher, Carl Young, Daniel Spangenberg, Eelis van der Weegen, Gabriel Dos Reis, Igor Tandetnik, John Potter, Massimiliano Alberti, Michal Necasek, Richard Smith, Ron Crane, Steven Keuchel, Thomas Richter, "tom_usenet," and probably a whole lot more whom I've missed. Particular thanks to Ilya Minkov for putting in a request for me to implement Properties for C++ in the STLSoft libraries, something I'd not previously considered. Without that suggestion one of my favorite techniques (see Chapter 35) would never have been.

And finally, thanks to all the STLSoft users, without whose feedback many of the features of the library would not be, and parts of this book would have been all the harder to research and to write.