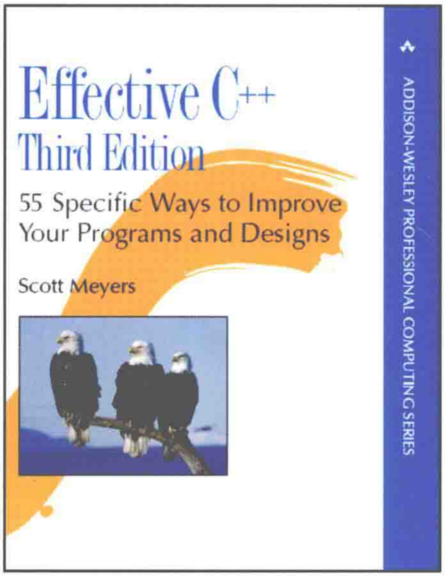
Broadview®
WWW.BROADVIEW.COM.CN

# Effective C++ Third Edition 英文版

55 Specific Ways to Improve Your Programs and Designs

改善程序技术与设计思维的55个有效做法

[美] Scott Meyers 著


Effective C++
Third Edition
55 Specific Ways to Improve Your Programs and Designs
Scott Meyers
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

# Effective C++, Third Edition 英文版

# Effective C++, Third Edition

Scott Meyers 著

電子工業出版社

**Publishing House of Electronics Industry**

北京 • BEIJING

## 内 容 简 介

有人说 C++程序员可以分成两类，读过 Effective C++的和没读过的。世界顶级 C++大师 Scott Meyers 成名之作的第三版的确当得起这样的评价。当您读过这本书之后，就获得了迅速提升自己 C++功力的一个契机。

在国际上，本书所引起的反响，波及整个计算技术出版领域，余音至今未绝。几乎在所有 C++书籍的推荐名单上，本书都会位于前三名。作者高超的技术把握力、独特的视角、诙谐轻松的写作风格、独具匠心的内容组织，都受到极大的推崇和仿效。这种奇特的现象，只能解释为人们对这本书衷心的赞美和推崇。

这本书不是读完一遍就可以束之高阁的快餐读物，也不是用以解决手边问题的参考手册，而是需要您去反复阅读体会的，C++是真正程序员的语言，背后有着精深的思想与无与伦比的表达能力，这使得它具有类似宗教般的魅力。希望这本书能够帮助您跨越 C++的重重险阻，领略高处才有的壮美风光，做一个成功而快乐的 C++程序员。

**图书在版编目（CIP）数据**

有效 C++：第 3 版＝Effective C++，Third Edition /（美）玛雅尔斯（Meyers,S.）著. —北京：电子工业出版社，2006.3
ISBN 7-121-00827-0
Ⅰ.有… Ⅱ.玛… Ⅲ.C 语言－程序设计－英文 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2006）第 004200 号

责任编辑：周　筠
印　　刷：北京市海淀区四季青印刷厂
出版发行：电子工业出版社
　　　　　北京市海淀区万寿路 173 信箱　邮编　100036
经　　销：各地新华书店
开　　本：787×980　1/16　印张：20.25　字数：500 千字
印　　次：2006 年 3 月第 1 次印刷
定　　价：45.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

# 序　言

1991 年我写下《*Effective C++*》第一版。1997 年撰写第二版时我更新了许多重要内容，但为了不让熟悉第一版的读者感到困惑，我竭尽所能保留原始结构：原先 50 个条款中的 48 个标题基本没有改变。如果把书籍视为一栋房屋，第二版相当于只是更换地毯灯饰，重新粉刷一遍而已。

到了第三版，修缮工作进一步深入壁骨墙筋（好几次我甚至希望能够翻新地基）。自 1991 年起，C++ 世界已经经历了巨大变革，而本书目标 —— 在一本小而有趣的书中确认最重要的一些 C++ 编程准则 —— 却已不再能够由 15 年前建立的那些条款体现出来。“C++ 程序员拥有 C 背景”这句话在 1991 年是个合理假设，

如今C++ 程序员却很可能转移自Java 或C# 阵营。Inheritance（继承）和objectoriented programming（面向对象编程）在1991 年对大多数程序员都很新鲜，如今的程序员则已经建立良好概念，exceptions（异常）、templates（模板）和generic programming（泛型编程）才是需要更多引导的领域。1991年没人听过所谓design patterns（设计模式），如今少了它很难讨论软件系统。1991 年C++ 正式标准才 刚要上路，如今C++ 标准规格已经8 岁，新版标准规格已经蓄势待发。

为了对付这些改变，我把所有条款抹得一干二净，然后问自己“2005 年什么是对C++ 程序员最重要的忠告？”答案便是第三版的这些条款。本书有两个新章，一个是resource management（资源管理），另一个是programming with templates（模板编程）。事实上template这东西遍布全书，因为它们几乎影响了C++ 的每个角落。本书的新素材还包括在exceptions的概念下编程、套用design patterns、以及使用新的TR1 程序库设施（TR1 描述于条款54）。众所周知任何在单线程系统（single-threaded systems）中运作良好的技术和解法有可能不适用于多线程系统（multithreaded systems）。本书半数以上的内容是新的。在此同时，第二版的大部分基础资讯仍然很重要，所以我找出一个保留它们的办法：你可以在附录B找到二、三两版的条款对映表。

我努力让本书达到我所能够达到的最佳状态，但这并不表示它已经完美。如果你认为某些条款不适合作为一般性忠告，或你有更好的办法完成本书所谈的某件工作，或书中某

些技术讨论不够清楚不够完全，甚或有所误导，请告诉我。如果你找出任何错误——技术上的、文法上的、排版印刷上的，不论哪一种——也请告诉我。我很乐意将第一位提出问题并吸引我注意的朋友加入下次印刷的致谢名单中。

即使本书条款数已扩充为55，这一整组编程准则还是谈不上完备。然而毕竟整理出优良准则——也就是几乎任何时间适用于几乎任何应用程序的准则——比想象中困难得多。如果你有其他编程准则的想法或建议，我将乐以与闻。

我维护着本书第一刷以来的变化清单，其中包括错误修订、进一步说明、以及技术更新。这份清单放在网址为`http://aristeia.com/BookErrata/ec++3e-errata.html`的“*Effective C++ Errata*”网页上。如果你希望在这份清单更新时获得通知，请加入我的邮件列表（mailing list）。我的邮件列表用来发布消息给可能对我的专业工作感兴趣的人士，详情请洽`http://aristeia.com/MailingList/`。

Scott Douglas Meyers Stafford.Oregon

http://aristeia.com/ April 2005

# Preface

I wrote the original edition of *Effective* C++ in 1991. When the time came for a second edition in 1997, I updated the material in important ways, but, because I didn't want to confuse readers familiar with the first edition, I did my best to retain the existing structure: 48 of the original 50 Item titles remained essentially unchanged. If the book were a house, the second edition was the equivalent of freshening things up by replacing carpets, paint, and light fixtures.

For the third edition, I tore the place down to the studs. (There were times I wished I'd gone all the way to the foundation.) The world of C++ has undergone enormous change since 1991, and the goal of this book — to identify the most important C++ programming guidelines in a small, readable package — was no longer served by the Items I'd established nearly 15 years earlier. In 1991, it was reasonable to assume that C++ programmers came from a C background. Now, programmers moving to C++ are just as likely to come from Java or C#. In 1991, inheritance and object-oriented programming were new to most programmers. Now they're well-established concepts, and exceptions, templates, and generic programming are the areas where people need more guidance. In 1991, nobody had heard of design patterns. Now it's hard to discuss software systems without referring to them. In 1991, work had just begun on a formal standard for C++. Now that standard is eight years old, and work has begun on the next version.

To address these changes, I wiped the slate as clean as I could and asked myself, "What are the most important pieces of advice for practicing C++ programmers in 2005?" The result is the set of Items in this new edition. The book has new chapters on resource management and on programming with templates. In fact, template concerns are woven throughout the text, because they affect almost everything in C++. The book also includes new material on programming in the presence of exceptions, on applying design patterns, and on using the new TR1 library facilities. (TR1 is described in Item 54.) It acknowledges that techniques and approaches that work well in single-threaded systems may not be appropriate in multithreaded systems. Well over half the material in the book is new. However, most of the fundamental information in the second edition continues to be important, so I found a way to retain it in one form or another. (You'll find a mapping between the second and third edition Items in Appendix B.)

I've worked hard to make this book as good as I can, but I have no illusions that it's perfect. If you feel that some of the Items in this book are inappropriate as general advice; that there is a better way to accomplish a task examined in the book; or that one or more of the technical discussions is unclear, incomplete, or misleading, please tell me. If you find an error of any kind — technical, grammatical, typographical, *whatever* — please tell me that, too. I'll gladly add to the acknowledgments in later printings the name of the first person to bring each problem to my attention.

Even with the number of Items expanded to 55, the set of guidelines in this book is far from exhaustive. But coming up with good rules — ones that apply to almost all applications almost all the time — is harder than it might seem. If you have suggestions for additional guidelines, I would be delighted to hear about them.

I maintain a list of changes to this book since its first printing, including bug fixes, clarifications, and technical updates. The list is available at the *Effective C++ Errata* web page, `http://aristeia.com/BookErrata/ ec++3e-errata.html`. If you'd like to be notified when I update the list, I encourage you to join my mailing list. I use it to make announcements likely to interest people who follow my professional work. For details, consult `http://aristeia.com/MailingList/`.

Scott Douglas Meyers

`http://aristeia.com/`

Stafford, Oregon

April 2005

# Acknowledgments

*Effective C++* has existed for fifteen years, and I started learning C++ about five years before I wrote the book. The "*Effective C++* project" has thus been under development for two decades. During that time, I have benefited from the insights, suggestions, corrections, and, occasionally, dumbfounded stares of hundreds (thousands?) of people. Each has helped improve *Effective C++*. I am grateful to them all.

I've given up trying to keep track of where I learned what, but one general source of information has helped me as long as I can remember: the Usenet C++ newsgroups, especially `comp.lang.c++.moderated` and `comp.std.c++`. Many of the Items in this book — perhaps most — have benefited from the vetting of technical ideas at which the participants in these newsgroups excel.

Regarding new material in the third edition, Steve Dewhurst worked with me to come up with an initial set of candidate Items. In Item 11, the idea of implementing `operator=` via copy-and-`swap` came from Herb Sutter's writings on the topic, e.g., Item 13 of his *Exceptional C++* (Addison- Wesley, 2000). RAII (see Item 13) is from Bjarne Stroustrup's *The C++ Programming Language* (Addison-Wesley, 2000). The idea behind Item 17 came from the "Best Practices" section of the Boost `shared_ptr` web page, http://boost.org/libs/smart_ptr/shared_ptr.htm#Best-Practices and was refined by Item 21 of Herb Sutter's *More Exceptional C++* (Addison-Wesley, 2002). Item 29 was strongly influenced by Herb Sutter's extensive writings on the topic, e.g., Items 8-19 of *Exceptional C++*, Items 17–23 of *More Exceptional C++*, and Items 11–13 of *Exceptional C++ Style* (Addison-Wesley, 2005); David Abrahams helped me better understand the three exception safety guarantees. The NVI idiom in Item 35 is from Herb Sutter's column, "Virtuality," in the September 2001 *C/C++ Users Journal*. In that same Item, the Template Method and Strategy design patterns are from *Design Patterns* (Addison- Wesley, 1995) by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. The idea of using the NVI idiom in Item 37 came

from Hendrik Schober. David Smallberg contributed the motivation for writing a custom set implementation in Item 38. Item 39's observation that the EBO generally isn't available under multiple inheritance is from David Vandevoorde's and Nicolai M. Josuttis' *C++ Templates* (Addison-Wesley, 2003). In Item 42, my initial understanding about typename came from Greg Comeau's C++ and C FAQ (`http://www.comeaucomputing.com/techtalk/#typename`), and Leor Zolman helped me realize that my understanding was incorrect. (My fault, not Greg's.) The essence of Item 46 is from Dan Saks' talk, "Making New Friends." The idea at the end of Item 52 that if you declare one version of `operator new`, you should declare them all, is from Item 22 of Herb Sutter's *Exceptional C++ Style*. My understanding of the Boost review process (summarized in Item 55) was refined by David Abrahams.

Everything above corresponds to who or where *I* learned about something, not necessarily to who or where the thing was invented or first published.

My notes tell me that I also used information from Steve Clamage, Antoine Trux, Timothy Knox, and Mike Kaelbling, though, regrettably, the notes fail to tell me how or where.

Drafts of the first edition were reviewed by Tom Cargill, Glenn Carroll, Tony Davis, Brian Kernighan, Jak Kirman, Doug Lea, Moises Lejter, Eugene Santos, Jr., John Shewchuk, John Stasko, Bjarne Stroustrup, Barbara Tilly, and Nancy L. Urbano. I received suggestions for improvements that I was able to incorporate in later printings from Nancy L. Urbano, Chris Treichel, David Corbin, Paul Gibson, Steve Vinoski, Tom Cargill, Neil Rhodes, David Bern, Russ Williams, Robert Brazile, Doug Morgan, Uwe Steinmüller, Mark Somer, Doug Moore, David Smallberg, Seth Meltzer, Oleg Shteynbuk, David Papurt, Tony Hansen, Peter McCluskey, Stefan Kuhlins, David Braunegg, Paul Chisholm, Adam Zell, Clovis Tondo, Mike Kaelbling, Natraj Kini, Lars Nyman, Greg Lutz, Tim Johnson, John Lakos, Roger Scott, Scott Frohman, Alan Rooks, Robert Poor, Eric Nagler, Antoine Trux, Cade Roux, Chandrika Gokul, Randy Mangoba, and Glenn Teitelbaum.

Drafts of the second edition were reviewed by Derek Bosch, Tim Johnson, Brian Kernighan, Junichi Kimura, Scott Lewandowski, Laura Michaels, David Smallberg, Clovis Tondo, Chris Van Wyk, and Oleg Zabluda. Later printings benefited from comments from Daniel Steinberg, Arunprasad Marathe, Doug Stapp, Robert Hall, Cheryl Ferguson, Gary Bartlett, Michael Tamm, Kendall Beaman, Eric Nagler, Max Hailperin, Joe Gottman, Richard Weeks, Valentin Bonnard, Jun He, Tim King, Don Maier, Ted Hill, Mark Harrison, Michael Rubenstein, Mark Rodgers, David Goh, Brenton Cooper, Andy Thomas-Cramer, Antoine Trux, John Wait, Brian Sharon, Liam Fitzpatrick, Bernd Mohr, Gary Yee, John O'Hanley, Brady Patterson, Christopher Peterson, Feliks Kluzniak, Isi Dunietz, Christopher Creutzi, Ian Cooper, Carl Harris, Mark Stickel, Clay Budin, Panayotis Matsinopoulos, David Smallberg, Herb Sutter, Pajo Misljencevic, Giulio Agostini, Fredrik Blomqvist, Jimmy Snyder, Byrial Jensen, Witold Kuzminski, Kazunobu Kuriyama, Michael Christensen, Jorge Yáñez Teruel, Mark Davis, Marty Rabinowitz, Ares Lagae, and Alexander Medvedev.

An early partial draft of this edition was reviewed by Brian Kernighan, Angelika Langer, Jesse Laeuchli, Roger E. Pedersen, Chris Van Wyk, Nicholas Stroustrup, and Hendrik Schober. Reviewers for a full draft were Leor Zolman, Mike Tsao, Eric Nagler, Gene Gutnik, David Abrahams, Gerhard Kreuzer, Drosos Kourounis, Brian Kernighan, Andrew Kirmse, Balog Pal, Emily Jagdhar, Eugene Kalenkovich, Mike Roze, Enrico Carrara, Benjamin Berck, Jack Reeves, Steve Schirripa, Martin Fallenstedt, Timothy Knox, Yun Bai, Michael Lanzetta, Philipp Janert, Guido Bartolucci, Michael Topic, Jeff Scherpelz, Chris Nauroth, Nishant Mittal, Jeff Somers, Hal Moroff, Vincent Manis, Brandon Chang, Greg Li, Jim Meehan, Alan Geller, Siddhartha Singh, Sam Lee, Sasan Dashtinezhad, Alex Marin, Steve Cai, Thomas Fruchterman, Cory Hicks, David Smallberg, Gunavardhan Kakulapati, Danny Rabbani, Jake Cohen, Hendrik Schober, Paco Viciana, Glenn Kennedy, Jeffrey D. Oldham, Nicholas Stroustrup, Matthew Wilson, Andrei Alexandrescu, Tim Johnson, Leon Matthews, Peter Dulimov, and Kevlin Henney. Drafts of some individual Items were reviewed by Herb Sutter and Attila F. Feher.

Reviewing an unpolished (possibly incomplete) manuscript is demanding work, and doing it under time pressure only makes it harder. I continue to be grateful that so many people have been willing to undertake it for me.

Reviewing is harder still if you have no background in the material being discussed and are expected to catch *every* problem in the manuscript. Astonishingly, some people still choose to be copy editors. Chrysta Meadowbrooke was the copy editor for this book, and her very thorough work exposed many problems that eluded everyone else.

Leor Zolman checked all the code examples against multiple compilers in preparation for the full review, then did it again after I revised the manuscript. If any errors remain, I'm responsible for them, not Leor.

Karl Wiegers and especially Tim Johnson offered rapid, helpful feedback on back cover copy.

Since publication of the first printing, I have incorporated revisions suggested by Jason Ross and Robert Yokota.

John Wait, my editor for the first two editions of this book, foolishly signed up for another tour of duty in that capacity. His assistant, Denise Mickelsen, adroitly handled my frequent pestering with a pleasant smile. (At least I think she's been smiling. I've never actually seen her.) Julie Nahil drew the short straw and hence became my production manager. She handled the overnight loss of six weeks in the production schedule with remarkable equanimity. John Fuller (her boss) and Marty Rabinowitz (his boss) helped out with production issues, too. Vanessa Moore's official job was to help with FrameMaker issues and PDF preparation, but she also added the entries to Appendix B and formatted it for printing on the inside cover. Solveig Haugland helped with index formatting. Sandra Schroeder and Chuti Prasertsith were responsible for cover design, though Chuti seems to have been the one who had to rework the cover each time I said, "But what about *this* photo with a stripe of *that* color...?" Chanda Leary-Coutu got tapped for the heavy lifting in marketing.

During the months I worked on the manuscript, the TV series *Buffy the Vampire Slayer* often helped me "de-stress" at the end of the day. Only with great restraint have I kept Buffyspeak out of the book.

Kathy Reed taught me programming in 1971, and I'm gratified that we remain friends to this day. Donald French hired me and Moises Lejter to create C++ training materials in 1989 (an act that led to my *really* knowing C++), and in 1991 he engaged me to present them at Stratus Computer. The students in that class encouraged me to write what ultimately became the first edition of this book. Don also introduced me to John Wait, who agreed to publish it.

My wife, Nancy L. Urbano, continues to encourage my writing, even after seven book projects, a CD adaptation, and a dissertation. She has unbelievable forbearance. I couldn't do what I do without her.

From start to finish, our dog, Persephone, has been a companion without equal. Sadly, for much of this project, her companionship has taken the form of an urn in the office. We really miss her.

# Contents

# Introduction

Learning the fundamentals of a programming language is one thing; learning how to design and implement *effective* programs in that language is something else entirely. This is especially true of C++, a language boasting an uncommon range of power and expressiveness. Properly used, C++ can be a joy to work with. An enormous variety of designs can be directly expressed and efficiently implemented. A judiciously chosen and carefully crafted set of classes, functions, and templates can make application programming easy, intuitive, efficient, and nearly error-free. It isn't unduly difficult to write effective C++ programs, *if* you know how to do it. Used without discipline, however, C++ can lead to code that is incomprehensible, unmaintainable, inextensible, inefficient, and just plain wrong.

The purpose of this book is to show you how to use C++ *effectively*. I assume you already know C++ as a *language* and that you have some experience in its use. What I provide here is a guide to using the language so that your software is comprehensible, maintainable, portable, extensible, efficient, and likely to behave as you expect.

The advice I proffer falls into two broad categories: general design strategies, and the nuts and bolts of specific language features. The design discussions concentrate on how to choose between different approaches to accomplishing something in C++. How do you choose between inheritance and templates? Between public and private inheritance? Between private inheritance and composition? Between member and non-member functions? Between pass-by-value and pass-by-reference? It's important to make these decisions correctly at the outset, because a poor choice may not become apparent until much later in the development process, at which point rectifying it is often difficult, time-consuming, and expensive.

Even when you know exactly what you want to do, getting things just right can be tricky. What's the proper return type for assignment operators? When should a destructor be virtual? How should `operator` `new` behave when it can't find enough memory? It's crucial to sweat details like these, because failure to do so almost always leads to unexpected, possibly mystifying program behavior. This book will help you avoid that.

This is not a comprehensive reference for C++. Rather, it's a collection of 55 specific suggestions (I call them *Items*) for how you can improve your programs and designs. Each Item stands more or less on its own, but most also contain references to other Items. One way to read the book, then, is to start with an Item of interest, then follow its references to see where they lead you.

The book isn't an introduction to C++, either. In Chapter 2, for example, I'm eager to tell you all about the proper implementations of constructors, destructors, and assignment operators, but I assume you already know or can go elsewhere to find out what these functions do and how they are declared. A number of C++ books contain information such as that.

The purpose of *this* book is to highlight those aspects of C++ programming that are often overlooked. Other books describe the different parts of the language. This book tells you how to combine those parts so you end up with effective programs. Other books tell you how to get your programs to compile. This book tells you how to avoid problems that compilers won't tell you about.

At the same time, this book limits itself to *standard* C++. Only features in the official language standard have been used here. Portability is a key concern in this book, so if you're looking for platform-dependent hacks and kludges, this is not the place to find them.

Another thing you won't find in this book is the C++ Gospel, the One True Path to perfect C++ software. Each of the Items in this book provides guidance on how to develop better designs, how to avoid common problems, or how to achieve greater efficiency, but none of the Items is universally applicable. Software design and implementation is a complex task, one colored by the constraints of the hardware, the operating system, and the application, so the best I can do is provide *guidelines* for creating better programs.

If you follow all the guidelines all the time, you are unlikely to fall into the most common traps surrounding C++, but guidelines, by their nature, have exceptions. That's why each Item has an explanation. The explanations are the most important part of the book. Only by understanding the rationale behind an Item can you determine whether it applies to the software you are developing and to the unique constraints under which you toil.

The best use of this book is to gain insight into how C++ behaves, why it behaves that way, and how to use its behavior to your advantage. Blind application of the Items in this book is clearly inappropriate, but at the same time, you probably shouldn't violate any of the guidelines without a good reason.

## Terminology

There is a small C++ vocabulary that every programmer should understand. The following terms are important enough that it is worth making sure we agree on what they mean.

A ***declaration*** tells compilers about the name and type of something, but it omits certain details. These are declarations:

```
extern int x;                                  // object declaration
std::size_t numDigits(int number);        // function declaration
class Widget;                                  // class declaration
template<typename T>                           // template declaration
class GraphNode;                               // (see Item 42 for info on
                                               // the use of "typename")
```

Note that I refer to the integer `x` as an "object," even though it's of built-in type. Some people reserve the name "object" for variables of user-defined type, but I'm not one of them. Also note that the function `numDigits'` return type is `std::size_t`, i.e., the type `size_t` in namespace `std`. That namespace is where virtually everything in C++'s standard library is located. However, because C's standard library (the one from C89, to be precise) can also be used in C++, symbols inherited from C (such as `size_t`) may exist at global scope, inside `std`, or both, depending on which headers have been `#included`. In this book, I assume that C++ headers have been `#included`, and that's why I refer to `std::size_t` instead of just `size_t`. When referring to components of the standard library in prose, I typically omit references to `std`, relying on you to recognize that things like `size_t`, `vector`, and `cout` are in `std`. In example code, I always include `std`, because real code won't compile without it.

`size_t`, by the way, is just a typedef for some unsigned type that C++ uses when counting things (e.g., the number of characters in a `char*`- based string, the number of elements in an STL container, etc.). It's also the type taken by the `operator[]` functions in `vector`, `deque`, and `string`, a convention we'll follow when defining our own `operator[]` functions in Item 3.

Each function's declaration reveals its ***signature***, i.e., its parameter and return types. A function's signature is the same as its type. In the