THE EXPERT'S VOICE® IN C++

C++ Standard Library Quick Reference

Peter Van Weert
Marc Gregoire

Apress®

C++ Standard Library Quick Reference

Peter Van Weert
Marc Gregoire

Apress®

C++ Standard Library Quick Reference

Peter Van Weert
Kessel-Lo, Belgium

Marc Gregoire
Meldert, Belgium

ISBN-13 (pbk): 978-1-4842-1875-4
ISBN-13 (electronic): 978-1-4842-1876-1
DOI 10.1007/978-1-4842-1876-1

Library of Congress Control Number: 2016941348

Managing Director: Welmoed Spahr
Lead Editor: Steve Anglin
Technical Reviewer: Bart Vandewoestyne
Editorial Board: Steve Anglin, Pramila Balan, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Michelle Lowman, James Markham, Susan McDermott, Matthew Moodie, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing
Coordinating Editor: Mark Powers
Copy Editor: Tiffany Taylor
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com/9781484218754. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Printed on acid-free paper

To my parents and my brother and his wife.
Their support and patience helped me in finishing this book.
—Marc Gregoire

In loving memory of Jeroen. Your enthusiasm and courage
will forever remain an inspiration to us all.
—Peter Van Weert

Contents at a Glance

Contents

About the Authors

Peter Van Weert is a Belgian software engineer whose main interest and expertise are programming languages, algorithms, and data structures.

He received his master's of science in computer science summa cum laude with congratulations of the Board of Examiners from the University of Leuven. In 2010, he completed his PhD thesis on the design and efficient compilation of rule-based programming languages at the research group for declarative programming languages and artificial intelligence of the same university. During his doctoral studies, he was a teaching assistant for object-oriented programming (Java), software analysis and design, and declarative programming.

After graduating, Peter joined Nikon Metrology to work on large-scale, industrial application software in the area of 3D laser scanning and point cloud inspection. At Nikon, he has mastered C++ and refactoring and debugging of very large code bases and has gained further proficiency in all aspects of the software development process, including the analysis of functional and technical requirements, and agile and scrum-based project and team management.

In his spare time, he has co-authored two award-winning Windows 8 apps, and he is a regular speaker at and board member of the Belgian C++ Users Group.

Marc Gregoire is a software engineer from Belgium. He graduated from the University of Leuven, Belgium, with a degree in "Burgerlijk ingenieur in de computer wetenschappen" (equivalent to a master's of science in engineering in computer science). The year after, he received the cum laude degree of master's in artificial intelligence at the same university. After his studies, Marc started working for a software consultancy company called Ordina Belgium. As a consultant, he worked for Siemens and Nokia Siemens Networks on critical 2G and 3G software running on Solaris for telecom operators. This required working in international teams stretching from South America and USA to EMEA and Asia. Now, Marc is working for Nikon Metrology on industrial 3D laser scanning software.

His main expertise is C/C++, specifically Microsoft VC++ and the MFC framework. He has experience in developing C++ programs running 24/7 on Windows and Linux platforms: for example, KNX/EIB home automation software. In addition to C/C++, Marc also likes C# and uses PHP for creating web pages.

Since April 2007, he has received the yearly Microsoft MVP (Most Valuable Professional) award for his Visual C++ expertise.

Marc is the founder of the Belgian C++ Users Group (www.becpp.org), author of *Professional C++*, and a member on the CodeGuru forum (as Marc G). He maintains a blog at www.nuonsoft.com/blog/.

About the Technical Reviewer

Bart Vandewoestyne is an enthusiastic, solo-parenting software engineer living in Belgium. After obtaining his master's degree from the Computer Science department at the University of Leuven, he worked as a researcher in the numerical analysis and applied mathematics section of that same university. He successfully completed his PhD in 2008. Three years of postdoctoral work later, Bart left the academic world for industry. He now works as a senior development software engineer for Esterline Belgium, where he develops and maintains software for professional flight-simulator alignment.

Bart enjoys reading about and exploring all aspects of C++ software development. In his spare time, and when he's away from his keyboard, he enjoys running, swimming, paragliding, and spending quality time with his now 6-year-old son Jenne. He wants the world to know how much he cares about Jenne, and he hopes that his child will also transform his passion into his profession.

Introduction

The C++ Standard Library

The C++ Standard Library is a collection of essential classes and functions used by millions of C++ programmers on a daily basis. Being part of the ISO Standard of the C++ Programming Language, an implementation is distributed with virtually every C++ compiler. Code written with the C++ Standard Library is therefore portable across compilers and target platforms.

The Library is more than 20 years old. Its initial versions were heavily inspired by a (then proprietary) C++ library called the *Standard Template Library (STL)*, so much so that many still incorrectly refer to the Standard Library as "the STL." The STL library pioneered generic programming with templated data structures called *containers* and *algorithms*, glued together with the concept of *iterators*. Most of this work was adapted by the C++ standardization committee, but nevertheless neither library is a true superset of the other.

The C++ Standard Library today is much more than the STL containers and algorithms. For decades, it has featured STL-like string classes, extensive localization facilities, and a stream-based I/O library, as well as all headers of the C Standard Library. In recent years, the C++11 and C++14 editions of the ISO standard have added, among other things, hash map containers, generic smart pointers, a versatile random-number-generation framework, a powerful regular expression library, more expressive utilities for function-style programming, type traits for template metaprogramming, and a portable concurrency library featuring threads, mutexes, condition variables, and atomic variables. Many of these libraries are based on Boost, a collection of open-source C++ libraries.

And this is just the beginning: the C++ community has rarely been as active and alive as in the past few years. The next version of the Standard, tentatively called C++17, is expected to add even more essential classes and functions.

Why This Book?

Needless to say, it is hard to know and remember all the possibilities, details, and intricacies of the vast and growing C++ Standard Library. This handy reference guide offers a condensed, well-structured summary of all essential aspects of the C++ Standard Library and is therefore indispensable to any C++ programmer.

You could consult the Standard itself, but it is written in a very detailed, technical style and is primarily targeted at Library implementors. Moreover, it is very long: the C++ Standard Library chapters alone are over 750 pages in length, and those on the

C Standard Library encompass another 250 pages. Other reference guides exist but are often outdated, limited (most cover little more than the STL containers and algorithms), or not much shorter than the Standard itself.

This book covers all important aspects of the C++14 and C11 Standard Libraries, some in more detail than others, and always driven by their practical usefulness. You will not find page-long, repetitive examples; obscure, rarely used features; or bloated, lengthy explanations that could be summarized in just a few bullets. Instead, this book strives to be exactly that: a summary. Everything you need to know and watch out for in practice is outlined in a compact, to-the-point style, interspersed with practical tips and short, well-chosen, clarifying examples.

Who Should Read This Book?

The book is targeted at all C++ programmers, regardless of their proficiency with the language or the Standard Library. If you are new to C++, its tutorial aspects will quickly bring you up to speed with the C++ Standard Library. Even the most experienced C++ programmer, however, will learn a thing or two from the book and find it an indispensable reference and memory aid. The book does not explain the C++ language or syntax itself, but is accessible to anyone with basic C++ knowledge or programming experience.

What You Will Learn

- How to use the powerful random-number-generation facilities
- How to work with dates and times
- What smart pointers are and how to use them to prevent memory leaks
- How to use containers to efficiently store and retrieve your data
- How to use algorithms to inspect and manipulate your data
- How lambda expressions allow for elegant use of algorithms
- What functionality the library provides for file and stream-based I/O
- How to work with characters and strings in C++
- How to write localized applications
- How to write safe and efficient multithreaded code using the C++11 concurrency library
- How to correctly handle error conditions and exceptions
- And more!

General Remarks

- All types, classes, functions, and constants of the C++ Standard Library are defined in the `std` namespace (short for *standard*).
- All C++ Standard Library headers must be included using `#include <`*header*`>` (note: no `.h` suffix!).
- All C Standard Library headers are available to C++ programmers in a slightly modified form by including `<c`*header*`>` (note the `c` prefix).[1] The most notable difference between the C++ headers and their original C counterparts is that all functionality is defined in the `std` namespace. Whether it is also provided in the global namespace is up to the implementation: portable code should therefore use the `std` namespace at all times.
- This book generally only covers the C headers if there are no more modern, C++-style alternatives provided by the C++ Standard Library.
- More advanced, rarely used topics such as custom memory allocators are not covered.

Code Examples

To compile and execute the code examples given throughout the book, all you need is a sufficiently recent C++ compiler. We leave the choice of compiler entirely up to you, and we further assume you can compile and execute basic C++ programs. All examples contain standard, portable C++ code only and should compile with any C++ compiler that is compliant with the C++14 version of the Standard. We occasionaly indicate known limitations of major compilers, but this is not a real goal of this book. In case of problems, please consult your implementation's documentation.

Unless otherwise noted, code examples can be copied as is and put inside the `main()` function of a basic command-line application. Generally, only two headers have to be included to make a code snippet compile: the one being discussed in the context where the example is given, and `<iostream>` for the command-line output statements (explained shortly). If any other header is required, we try to indicate so in the text. Should we forget, the appendix provides a brief overview of all headers of the Standard Library and their contents. Additionally, you can download compilable source code files for all code snippets from this book from the Apress website (`www.apress.com/9781484218754`).

[1] The original C headers may still be included with `<`*header*`.h>`, but their use is deprecated.

Following is the obligatory first example (for once, we show the full program):

```
#include <iostream>
int main() {
   std::cout << "Hello world!" << std::endl;
}
```

Many code samples, including those in earlier chapters, write to the standard output console using `std::cout` and the `<<` stream operator, even though these facilities of the C++ I/O library are only discussed in detail in Chapter 5. The stream operator can be used to output virtually all fundamental C++ types, and multiple values can be written on a single line. The I/O manipulator `std::endl` outputs the newline character (`\n`) and flushes the output for `std::cout` to the standard console. Straightforward usage of the `std::string` class defined in `<string>` may occur in earlier examples as well. For instance:

```
std::string piString = "PI";
double piValue = 3.14159;
std::cout << piString << " = " << piValue << std::endl;  // PI = 3.14159
```

More details regarding streams and strings are found in Chapters 5 and 6, respectively, but this should suffice to get you through the examples in earlier chapters.

Common Class

The following `Person` class is used in code examples throughout the book to illustrate the use of user-defined classes together with the Standard Library:

```
#include <string>
#include <ostream>

class Person {
public:
   Person() = default;
   explicit Person(const std::string& first,
                   const std::string& last = "", bool isVIP = false)
      : m_first(first), m_last(last), m_isVIP(isVIP) {}

   const std::string& GetFirstName() const { return m_first; }
   void SetFirstName(const std::string& first) { m_first = first; }

   const std::string& GetLastName() const { return m_last; }
   void SetLastName(const std::string& last) { m_last = last; }

   bool IsVIP() const { return m_isVIP; }
```

```
private:
   friend bool operator<(const Person&, const Person&);
   std::string m_first;
   std::string m_last;
   bool m_isVIP = false;
};

// Comparison operator.
bool operator<(const Person& lhs, const Person& rhs) {
   if (lhs.IsVIP() != rhs.IsVIP()) return rhs.IsVIP();
   if (lhs.GetLastName() != rhs.GetLastName())
      return lhs.GetLastName() < rhs.GetLastName();
   return lhs.GetFirstName() < rhs.GetFirstName();
}

// Equality operator.
bool operator==(const Person& lhs, const Person& rhs) {
   return lhs.IsVIP() == rhs.IsVIP() &&
      lhs.GetFirstName() == rhs.GetFirstName() &&
      lhs.GetLastName() == rhs.GetLastName();
}

// operator<< to support output to C++ streams.
// Details of this streaming operator can be found in Chapter 5.
std::ostream& operator<<(std::ostream& os, const Person& person) {
   os << person.GetFirstName() << ' ' << person.GetLastName();
   return os;
}
```