C++

The Programming Language

Waylon Warren

Larsen & Keller

C++: The Programming Language

C++ is a computer language or program language. It provides low-level memory manipulation and is also used for object-oriented, imperative and generic programming features. C++ has helped develop many other languages like Java, C#, and D. This book presents the complex subject of C++ in the most comprehensible and easy to understand language. The topics included in it are of utmost significance and are bound to provide incredible insights to students. Some of the diverse topics covered in this text address the varied branches that fall under this category. Those in search of information to further their knowledge will be greatly assisted by this textbook.

Waylon Warren pursued his PhD. in Programming Language from Cornell University, United States. He has been the recipient of two awards for his research works in the field of programming languages especially C++. Warren has authored and edited more than 57 articles, journal papers and book chapters in this field. He is a renowned lecturer of undergraduates programs and travels extensively for educating students across the globe.

ISBN 978-1-63549-158-6

Larsen & Keller
www.larsen-keller.com

Warren C++: The Programming Language Larsen & Keller

C++: The Programming Language

Edited by
Waylon Warren

Larsen & Keller
www.larsen-keller.com

C++: The Programming Language
Edited by Waylon Warren
ISBN: 978-1-63549-158-6 (Hardback)

Larsen & Keller
Published by Larsen and Keller Education,
5 Penn Plaza,
19th Floor,
New York, NY 10001, USA

Cataloging-in-Publication Data

C++ : the programming language / edited by Waylon Warren.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-63549-158-6
1. C++ (Computer program language).
I. Warren, Waylon.
QA76.73.C153 C15 2017
005.133--dc23

Printed and bound in China.

For more information regarding Larsen and Keller Education and its products, please visit the publisher's website www.larsen-keller.com

C++: The Programming Language

Preface

C++ is a computer language or program language. It provides low-level memory manipulation and is also used for object-oriented, imperative and generic programming features. C++ has helped develop many other languages like Java, C#, and D. This book presents the complex subject of C++ in the most comprehensible and easy to understand language. The topics included in it are of utmost significance and are bound to provide incredible insights to students. Some of the diverse topics covered in this text address the varied branches that fall under this category. Those in search of information to further their knowledge will be greatly assisted by this textbook.

A detailed account of the significant topics covered in this book is provided below:

Chapter 1- Programming language is a computer language aimed at communicating instructions to a device, particularly a computer. C++ is one of these programming languages, which has imperative and object oriented programing features. This text guides the reader in having an in depth understanding of C++.

Chapter 2- There are a number of programming languages which are related to C++. Some of these are C, Ada, CLU and ALGOL 68. All these languages are well structured, imperative and object oriented. This section strategically encompasses and incorporates the major components and key concepts of all the programming languages that are related to C++.

Chapter 3- C++ has a number of programming languages; some of these languages are C++03, C++11, C++14 and C++17. C++03 is the standard version of C++ globally whereas C++11 is the standard version of C++. The following chapter helps the reader in understanding all the types of C++.

Chapter 4- CLI is a programing language, which was created by Microsoft. It was created with the intention of replacing the managed extensions for C++. Some other extensions of C++, like Cilk was created with the purpose of programming languages for multithreaded parallel computing. This section also focuses on some aspects of C++, aspect C++, CLI.

Chapter 5- Computer programming has a feature called copy elision. Copy elision is a method which is essential to eliminate the unnecessary coping of objects. Templates (C++) and decltypes are other prominent techniques used in C++. The text elucidates all the tools and methods of C++.

Chapter 6- Subroutine is a sequence of programming which is assigned to perform precise tasks, packaged as a unit. Exception safety on the other hand is a set of guidelines that can be used by clients when handling safety in any programming language, specifically C++. This chapter strategically encompasses and incorporates the methods and tools of C++, providing a complete understanding.

Chapter 7- Generic programming has software such as Ada, Delphi, Eiffel, Java and C# whereas metaprogramming writes programs with the skill to treat programs as their data. Which means if a data is being analyzed it can simultaneously be modified also. C++ programming is best understood in confluence with the major topics listed in following chapter.

Chapter 8- Generic programming is a style of computer programming where the algorithms written in generic programming are written in terms of types to-be-specified-later. The other diverse aspects of C++ are metaprogramming, compatibility of C and C++, criticism of C++ and Sieve C++ Parallel programming system. This topic will provide an integrated understanding of C++ programming language.

Chapter 9- C++ Standard Library is a collection of classes and functions. This collection is written in the core language. The topics discussed in this text are C++ string handling, functional (C++), sequence container (C++) and standard template library. The diverse aspects of C++ Standard Library have been carefully analyzed in this chapter.

It gives me an immense pleasure to thank our entire team for their efforts. Finally in the end, I would like to thank my family and colleagues who have been a great source of inspiration and support.

Editor

Table of Contents

Introduction to C++

Programming language is a computer language aimed at communicating instructions to a device, particularly a computer. C++ is one of these programming languages, which has imperative and object oriented programing features. This text guides the reader in having an in depth understanding of C++.

C++

C++ is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation.

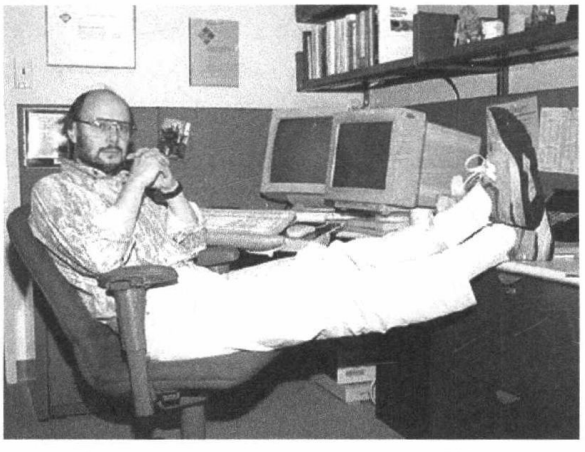
It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights. C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications, including desktop applications, servers (e.g. e-commerce, web search or SQL servers), and performance-critical applications (e.g. telephone switches or space probes). C++ is a compiled language, with implementations of it available on many platforms and provided by various organizations, including the Free Software Foundation (FSF's GCC), LLVM, Microsoft, Intel and IBM.

C++ is standardized by the International Organization for Standardization (ISO), with the latest standard version ratified and published by ISO in December 2014 as *ISO/IEC 14882:2014* (informally known as C++14). The C++ programming language was initially standardized in 1998 as *ISO/IEC 14882:1998*, which was then amended by the C++03, *ISO/IEC 14882:2003*, standard. The current C++14 standard supersedes these and C++11, with new features and an enlarged standard library. Before the initial standardization in 1998, C++ was developed by Bjarne Stroustrup at Bell Labs since 1979, as an extension of the C language as he wanted an efficient and flexible language similar to C, which also provided high-level features for program organization.

Many other programming languages have been influenced by C++, including C#, D, Java, and newer versions of C (after 1998).

History

In 1979, Bjarne Stroustrup, a Danish computer scientist, began work on the predecessor to C++, "C with Classes". The motivation for creating a new language originated from Stroustrup's experience in programming for his Ph.D. thesis. Stroustrup found that Simula had features that were very helpful for large software development, but the language was too slow for practical use, while BCPL was fast but too low-level to be suitable for large software development. When Stroustrup started working in AT&T Bell Labs, he had the problem of analyzing the UNIX kernel with respect to distributed computing. Remembering his Ph.D. experience, Stroustrup set out to enhance the C language with Simula-like features. C was chosen because it was general-purpose, fast, portable and widely used. As well as C and Simula's influences, other languages also influenced C++, including ALGOL 68, Ada, CLU and ML.



Bjarne Stroustrup, the creator of C++

Initially, Stroustrup's "C with Classes" added features to the C compiler, Cpre, including classes, derived classes, strong typing, inlining and default arguments.

In 1983, *C with Classes* was renamed to *C++* ("++" being the increment operator in C), adding new features that included virtual functions, function name and operator overloading, references, constants, type-safe free-store memory allocation (new/delete), improved type checking, and BCPL style single-line comments with two forward slashes (//). Furthermore, it included the development of a standalone compiler for C++, Cfront.

In 1985, the first edition of *The C++ Programming Language* was released, which became the definitive reference for the language, as there was not yet an official standard. The first commercial implementation of C++ was released in October of the same year.

In 1989, C++ 2.0 was released, followed by the updated second edition of *The C++ Programming Language* in 1991. New features in 2.0 included multiple inheritance, abstract classes, static member functions, const member functions, and protected members. In 1990, *The Annotated C++ Reference Manual* was published. This work

became the basis for the future standard. Later feature additions included templates, exceptions, namespaces, new casts, and a boolean type.

After the 2.0 update, C++ evolved relatively slowly until, in 2011, the C++11 standard was released, adding numerous new features, enlarging the standard library further, and providing more facilities to C++ programmers. After a minor C++14 update released in December 2014, various new additions are planned for 2017 and 2020.

Etymology

According to Stroustrup: "the name signifies the evolutionary nature of the changes from C". This name is credited to Rick Mascitti (mid-1983) and was first used in December 1983. When Mascitti was questioned informally in 1992 about the naming, he indicated that it was given in a tongue-in-cheek spirit. The name comes from C's "++" operator (which increments the value of a variable) and a common naming convention of using "+" to indicate an enhanced computer program.

During C++'s development period, the language had been referred to as "new C" and "C with Classes" before acquiring its final name.

Philosophy

Throughout C++'s life, its development and evolution has been informally governed by a set of rules that its evolution should follow:

- It must be driven by actual problems and its features should be useful immediately in real world programs.
- Every feature should be implementable (with a reasonably obvious way to do so).
- Programmers should be free to pick their own programming style, and that style should be fully supported by C++.
- Allowing a useful feature is more important than preventing every possible misuse of C++.
- It should provide facilities for organising programs into well-defined separate parts, and provide facilities for combining separately developed parts.
- No implicit violations of the type system (but allow explicit violations; that is, those explicitly requested by the programmer).
- User-created types need to have the same support and performance as built-in types.
- Unused features should not negatively impact created executables (e.g. in lower performance).

- There should be no language beneath C++ (except assembly language).
- C++ should work alongside other existing programming languages, rather than fostering its own separate and incompatible programming environment.
- If the programmer's intent is unknown, allow the programmer to specify it by providing manual control.

Standardization

C++ is standardized by an ISO working group known as JTC1/SC22/WG21. So far, it has published four revisions of the C++ standard and is currently working on the next revision, C++17.

Year	C++ Standard	Informal name
1998	ISO/IEC 14882:1998	C++98
2003	ISO/IEC 14882:2003	C++03
2011	ISO/IEC 14882:2011	C++11
2014	ISO/IEC 14882:2014	C++14
2017	to be determined	C++17
2020	to be determined	C++20

In 1998, the ISO working group standardized C++ for the first time as *ISO/IEC 14882:1998*, which is informally known as *C++98*. In 2003, it published a new version of the C++ standard called *ISO/IEC 14882:2003*, which fixed problems identified in C++98.

The next major revision of the standard was informally referred to as "C++0x", but it was not released until 2011. C++11 (14882:2011) included many additions to both the core language and the standard library.

In 2014, C++14 (also known as C++1y) was released as a small extension to C++11, featuring mainly bug fixes and small improvements. The Draft International Standard ballot procedures completed in mid-August 2014.

After C++14, a major revision, informally known as C++17 or C++1z, is planned for 2017, which is almost feature-complete.

As part of the standardization process, ISO also publishes technical reports and specifications:

- ISO/IEC TR 18015:2006 on the use of C++ in embedded systems and on performance implications of C++ language and library features,
- ISO/IEC TR 19768:2007 (also known as the C++ Technical Report 1) on library extensions mostly integrated into C++11,
- ISO/IEC TR 29124:2010 on special mathematical functions,

- ISO/IEC TR 24733:2011 on decimal floating point arithmetic,
- ISO/IEC TS 18822:2015 on the standard filesystem library,
- ISO/IEC TS 19570:2015 on parallel versions of the standard library algorithms,
- ISO/IEC TS 19841:2015 on software transactional memory,
- ISO/IEC TS 19568:2015 on a new set of library extensions, some of which are already integrated into C++17,
- ISO/IEC TS 19217:2015 on the C++ Concepts

More technical specifications are in development and pending approval, including concurrency library extensions, a networking standard library, ranges, and modules.

Language

The C++ language has two main components: a direct mapping of hardware features provided primarily by the C subset, and zero-overhead abstractions based on those mappings. Stroustrup describes C++ as "a light-weight abstraction programming language [designed] for building and using efficient and elegant abstractions"; and "offering both hardware access and abstraction is the basis of C++. Doing it efficiently is what distinguishes it from other languages".

C++ inherits most of C's syntax. The following is Bjarne Stroustrup's version of the Hello world program that uses the C++ Standard Library stream facility to write a message to standard output:

```
#include <iostream>

int main()

{

        std::cout << "Hello, world!\n";

}
```

Within functions that define a non-void return type, failure to return a value before control reaches the end of the function results in undefined behaviour (compilers typically provide the means to issue a diagnostic in such a case). The sole exception to this rule is the main function, which implicitly returns a value of zero.

Object Storage

As in C, C++ supports four types of memory management: static storage duration objects, thread storage duration objects, automatic storage duration objects, and dynamic storage duration objects.

Static Storage Duration Objects

Static storage duration objects are created before main() is entered and destroyed in reverse order of creation after main() exits. The exact order of creation is not specified by the standard (though there are some rules defined below) to allow implementations some freedom in how to organize their implementation. More formally, objects of this type have a lifespan that "shall last for the duration of the program".

Static storage duration objects are initialized in two phases. First, "static initialization" is performed, and only *after* all static initialization is performed, "dynamic initialization" is performed. In static initialization, all objects are first initialized with zeros; after that, all objects that have a constant initialization phase are initialized with the constant expression (i.e. variables initialized with a literal or constexpr). Though it is not specified in the standard, the static initialization phase can be completed at compile time and saved in the data partition of the executable. Dynamic initialization involves all object initialization done via a constructor or function call (unless the function is marked with constexpr, in C++11). The dynamic initialization order is defined as the order of declaration within the compilation unit (i.e. the same file). No guarantees are provided about the order of initialization between compilation units.

Thread Storage Duration Objects

Variables of this type are very similar to static storage duration objects. The main difference is the creation time is just prior to thread creation and destruction is done after the thread has been joined.

Automatic Storage Duration Objects

The most common variable types in C++ are local variables inside a function or block, and temporary variables. The common feature about automatic variables is that they have a lifetime that is limited to the scope of the variable. They are created and potentially initialized at the point of declaration and destroyed in the *reverse* order of creation when the scope is left.

Local variables are created as the point of execution passes the declaration point. If the variable has a constructor or initializer this is used to define the initial state of the object. Local variables are destroyed when the local block or function that they are declared in is closed. C++ destructors for local variables are called at the end of the object lifetime, allowing a discipline for automatic resource management termed RAII, which is widely used in C++.

Member variables are created when the parent object is created. Array members are initialized from 0 to the last member of the array in order. Member variables are destroyed when the parent object is destroyed in the reverse order of creation. i.e. If the