**Portability**

Mature languages attract international standards.  These standards promote uniformity across host platforms.  Committees of compiler writers, academics, applications developers and representatives from various national organizations review, discuss, negotiate and reach consensus on which elements should be included in the standard definition.  Elements that are excluded are called *extension*s.  Compiler writers include their extensions in their own implementations of the agreed standard.

A standard definition enables developers to write programs with the expectation that they will compile and run on a wide variety of platforms that support the standard.  By avoiding the use of extensions, we increase the portability of our software.  We call standard-compliant software *highly portable*.

This chapter reviews the C standards that have been defined throughout the language's history, introduces some coding guidelines and describes utilities available for analyzing the degree of portability.

The C Standards

Dennis Ritchie created the C language between 1969 and 1973, while working at AT&T Bell Labs.  Brian Kernighan and Dennis Ritchie published their original description of the language in the classic book entitled "The C Programming Language" in 1978.  They published a second edition in 1988.

Since then, the C language has undergone three 'standard' definitions: one in 1989, one in 1999 and the latest in 2011.



C89

In 1982, the American National Standards Institute formed committee X3J11 to propose the first standard for C.  An international group converted this standard with minor modifications into the ISO/IEC 9899:1990 standard, which ANSI subsequently adopted.  This was called Standard C to distinguish it from Kernighan and Ritchie C (K & R C, in short).

Standard C introduced:

* function prototypes
* a standard library
* whitespace before #
* new keywords

In 1995, ANSI introduced some amendments to Standard C, called C95, which included new conversion specifiers for printf() and scanf() and more library functions.

C99

ISO/IEC approved the second international standard in 1999: ISO/IEC 9899:1999 (C99).

C99 introduced:

* long long data type
* \_Bool data type
* \_Complex data type
* complex arithmetic
* variable-length arrays
* // C++ style comments
* better support for floating-point types, including math functions for all types

C11

ISO/IEC approved the third international standard in 2011: ISO/IEC 9899:2011 (C11).

C11 introduced:

* multithreading support
* improved Unicode support
* bounds checking interfaces
* a create and open mode ("x") for fopen()
* removal of gets()
* optional support for complex arithmetic
* optional support for variable-length arrays

The optional support applies to features that had been mandatory under the C99 standard.

The C++ Standards

Object-oriented languages have evolved since the concept of objects was formally incorporated into the Simula language by Ole-Johan Dahl and Kristen Nygaard in the 1960s.  International standards have documented their evolution.  The most recent standard for the C++ language is formally known as ISO/IEC 14882:2014 and extends over 1300 pages.  ISO stands for the International Organization for Standardization.  IEC stands for the International Electrotechnical Commission.

This chapter reviews the milestones in the evolution of C++, highlights some of the features introduced during its evolution with respect to the original version of the language, and briefly discusses a few of the topics that have evolved with the C++ standards.

Milestones

C++ was originally designed as a synthesis of C and object-orientation Simula-style.  C had and still has no object-oriented capabilities.  Simula introduced the terms class, object, inheritance, virtual methods and subclasses (derived classes) formally to the programming community.

[Bjarne Stroustrup](http://www.research.att.com/~bs/homepage.html) created C++ at Bell Labs (AT&T Research Labs) by augmenting C with the object-oriented features of Simula.  He released C++ officially in October 1985.  His web site includes a quite useful and up-to-date [glossary of technical terms](http://www.research.att.com/~bs/glossary.html).

The ISO/IEC Standards

At the time of printing, three standard definitions have been approved by the international programming community.

* C++98
* C++11
* C++14

C++98

The first official standard that defined the C++ language is formally known as ISO/IEC 14882:1998 and less formally as C++98.  The international programming community ratified this definition in 1998 and published it in a document that contains about 800 pages.  The definition is based in part on the ISO/IEC 9899:1990 standard for the C language (informally known as C89).

C++98 augmented pre-standard C++ with

"additional data types, classes, templates, exceptions, namespaces, inline functions, operator overloading, function name overloading, references, freestore management operators, and additional library facilities."

The library facilities included a newly re-written iostream library and the string class.

C++11

The second official standard that re-defined the language is formally known as ISO/IEC 14882:2011 and less formally as C++11.  The international programming community ratified this definition on August 12 2011.  The definition is based on C++98 and C99 and includes several major additions to the core language as well as several major extensions of the standard library.

The objectives of the C++11 committee had included:

* making C++ easier to teach and to learn through increased uniformity
* making C++ better for systems programming and library construction
* improving the type safety of the language

The features that C++11 added to C++98 included (amongst others):

* the nullptr keyword replacing the NULL macro
* the auto keyword inferring the type of a left operand implicitly from the type of the right operand in an assignment expression
* inherited constructors
* features covered in the next volume of this series of notes
  + move constructors and assignment operators
  + lambda expressions (anonymous functions)
  + library support for multi-threading classes
  + range based for loops
  + strongly typed enumerations
  + uniformity amongst initializers
  + initializers for class members

C++14

The third official standard that re-defined the language is formally known as ISO/IEC 14882:2014 and less formally as C++14.  The international programming community ratified this definition on August 18 2014.  The definition expanded the application of the auto keyword to return types, the application of templates to variables.  The changes made the language safer and more convenient.

These notes align with this standard. See [C++14](https://en.wikipedia.org/wiki/C%2B%2B14).

C++17

The ISO/IEC 14882 standards committee is working on the next iteration scheduled for ratification in July 2017.  The committee intended this revision to be a major amendment to the C++14 standard, but several new features did not make the cut. See [C++17](https://en.wikipedia.org/wiki/C%2B%2B17).

C++20

C++20 adds more new major features than C++14 or C++17. Changes that have been accepted into C++20 include three-way comparisons, [=, this] as a lambda capture, template parameter lists on lambdas, etc... See [C++20](https://en.wikipedia.org/wiki/C%2B%2B20).

Compiler Support Status

A language standard is a specification for compiler writers.  Different writers introduce different features adopted in a standard at different times.  The support status for the features approved in C++11 and C++14 is tabulated at <http://en.cppreference.com/w/cpp/compiler_support>.  Links to the individual compiler web sites are included there.