2

C++11 Support in Compilers

The following table lists new C++11 features that are mentioned in this book, and whether or not a specific compiler supports that feature. The compilers are those that have been used to test the sample code in this book.

C++11 FEATURE	GCC 4.6	MS VC++ 2010
`[[caries_dependency]]` attribute	–	–
`[[noreturn]]` attribute	–	–
`__func__`	•	–
`<ccomplex>`	•	–
`<cfenv>`	•	–
`<cinttypes>`	•	–
`<codecvt>`	–	•
`<cstdalign>`	–	–
`<cstdbool>`	•	–
`<cstdint>`	•	•
`<ctgmath>`	•	–
`<cuchar>`	–	–
`<system_error>`	•	•
`<typeindex>`	•	•
`std::all_of()`	•	•

continues

(continued)

C++11 FEATURE	GCC 4.6	MS VC++ 2010
Alternative function syntax (trailing return type)	•	•
`std::any_of()`	•	•
`std::array`	•	•
`std::async()`	•	–
Atomic types and operations `<atomic>`	•	–
`auto` keyword	•	•
`std::bind()`	•	•
Bitwise function objects (`std::bit_and`, `std::bit_or`, and `std::bit_xor`)	•	•
`std::call_once()`	•	–
`cbegin()/cend()/crbegin()/crend()`	•	•
`char16_t`	•	•
`char32_t`	•	•
Chrono library `<chrono>`	•	–
Compile-time rational arithmetic library `<ratio>`	•	–
Condition variables `<condition_variable>`	•	–
`constexpr` keyword	•	–
`std::copy_if()`	•	•
`std::copy_n()`	•	•
`std::current_exception()`	•	•
`decltype` keyword	•	•
Delegating constructors	–	–
Double right angled brackets without spaces for nested templates	•	•
Emplace operations on containers	•	•
Explicit conversion operators	•	–
Explicitly defaulted/deleted member functions	•	–
`final` keyword on classes	–	–
`final` keyword on methods	–	–
`std::find_if_not()`	•	•

C++11 FEATURE	GCC 4.6	MS VC++ 2010
`std::forward_list`	•	•
`std::function`	•	•
Futures and promises `<future>`	•	-
`std::get_money()`	•	•
`std::get_time()`	–	•
In-class member initializers	–	–
Inherited constructors	–	–
`std::initializer_list`	•	–
`std::iota()`	•	•
`std::is_sorted()`	•	•
`std::is_sorted_until()`	•	•
Lambda expressions	•	•
Locally (in function) defined class as template argument	•	•
`long long`	•	•
`std::make_exception_ptr()`	•	–
`std::make_move_iterator()`	•	•
`std::max()` with more than 2 arguments	•	–
`std::mem_fn()`	•	•
`std::min()` with more than 2 arguments	•	–
`std::minmax()`	•	•
`std::minmax_element()`	•	•
Move semantics (using rvalue references)	•	•
`std::move()`	•	•
`std::move_backward()`	•	•
`std::move_iterator`	•	•
Mutual exclusion `<mutex>`	•	–
Nested exceptions (`std::nested_exception`)	•	–
New string prefixes (`u8`, `u` and `U`)	•	–

continues

(continued)

C++11 FEATURE	GCC 4.6	MS VC++ 2010
`noexcept` keyword	•	–
`std::none_of()`	•	•
`nullptr`	•	•
Numeric conversions (`std::to_string()`, `std::to_wstring()`, `std::stoi()`, `std::stol()`, ...)	•	•
`override` keyword	–	–
`std::packaged_task`	•	–
`std::partial_sort_copy()`	•	•
`std::partition_copy()`	•	•
`std::partition_point()`	•	•
`std::put_money()`	•	•
`std::put_time()`	–	•
Random numbers library `<random>`	•	•
Range-based `for` loop	•	–
Raw string literals	•	–
Regular expressions `<regex>`	•	•
`std::rethrow_exception()`	•	•
Rvalue references `&&`	•	•
Rvalue references for `*this`	–	–
`std::shared_ptr`	•	•
`static_assert()`	•	•
Strongly typed enumerations (`enum class`)	•	–
Threads `<thread>`	•	–
Tuples `<tuple>`	•	•
Type traits `<type_traits>`	•	•
Type/template aliases (`using` keyword)	–	–
Uniform initialization	•	–
`std::unique_ptr`	•	•

C++11 FEATURE	GCC 4.6	MS VC++ 2010
`std::unordered_map`	•	•
`std::unordered_multimap`	•	•
`std::unordered_multiset`	•	•
`std::unordered_set`	•	•
User defined literals	–	–
Variadic templates	•	–
`std::weak_ptr`	•	•

As of this writing, to enable the C++11 features with GCC 4.6, you have to specify the `-std=c++0x` *command line parameter. C++0x was the working name during the standardization process of the new C++ standard.*