Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat
CA 1		12.2	p5	te	Resolve Core Issue 1376. The proposed resolution from February 2012 seems stalled and does not address other ways by which an non-prvalue may designate a temporary.	In subclause 12.2 [class.temporary] paragraph 5, Replace: The second context is when a reference is bound to a ~~temporary~~<ins>prvalue</ins>. In subclause 8.5.3 [dcl.init.ref] paragraph 5, Insert: Otherwise, a <ins>prvalue </ins>temporary of type "*cv1* T1" is created and initialized from the initializer expression using the rules for a non-reference copy-initialization. In subclause 8.5.3 [dcl.init.ref] paragraph 6, Insert: [*Note:* 12.2 describes the lifetime of temporaries bound to references. —*end note*]	
CA 2		3.2	p6	te	**ODR and closure types with vague linkage** Although this may be a C++ ABI Issue, we would like an indication from CWG as to the intent of the committee. Does the committee intend for the ODR to imply that closure types need to have an	Various options from discussion at http://sourcerytools.com/pipermail/cxx-abi-dev/2013-January/002544.html: 1. Guarantee the layout of lambdas in functions with weak linkage. We'd still be able to optimize all other lamb-	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

					ABI-specified layout?
					
					Consider the following with source, with one object compiled with `-DMAIN` and another without and then linked together:
					
					- for `f` in `foo()`, the layout needs to be compatible between different implementations since the static local is shared between translation units
					
					- for `ff` in `foo()`, the layout needs to be compatible between different implementations in order to satisfy the ODR requirement that the program behave as if there was only one definition of the inline function
					
					`extern "C" int printf(const char *, ...);`
					`extern long gz;`
					
					`inline void foo() {`
					`  long x = 0, q = 0, &z = gz;`
					`  static auto f = [=, &z]() mutable`
					`  { q += ++x; gz = q + x; };`
					
					`  long a, b;`
					`  auto ff = [=]`
					`  { sizeof(a /*not an odr-use*/),`
					`  printf("%u\n", &b < &a); };`
					`  f();`
					`  ff();`
					`}`
					
					`void bar();`
						das, so this isn't really that bad; it's just a bit disappointing for us compiler hackers and (a subset of) our users.
						
						2. Ban lambdas in functions with weak linkage, similar to how C bans static variables in (C's definition of) inline functions.
						
						Of course, "weak linkage" is not a concept in the standard, and you'd have to formalize that quite carefully to avoid sweeping up a ton of interesting cases involving anonymous namespaces. And, of course, this would mean banning a bunch of code that doesn't actually run afoul of this.
						
						3. Give lambdas internal linkage by fiat and hack the ODR to make that work out.
						
						I imagine this rule would come across like "lambdas in inline functions will behave like they have different types in different translation units, and that's not a formal ODR violation, but if it affects the semantics of your program, tough cookies."	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

CA 3		5.1.2		te	Resolve CWG issue 1607.
					
					#if ! MAIN
					void bar() { foo(); }
					#else
					long gz;
					int main() {
					foo();
					bar();
					foo();
					return gz;
					}
					#endif
						Options 1 and 2, respectively, take away optimization opportunities and functionality for obscure reasons.
						
						While we recognize that Option 3 introduces additional undefined behavior in the language, we feel that it resolves the issue neatly since it merely describes the natural result of implementation freedom in forming closure types.
						The options listed for resolution of the issue in April 2013 do not appear to address the case originally presented with the partial specializations.
						
						Of the four, #2 may be the best balance with the overriding aim of making SFINAE work both for function and class templates.	

CA 4		14.8.2.5	p10	te	**Deduction with P/A length mismatch for function parameter lists**
					
					Consider the following:
					
					template <typename U>
					struct A {
					template <typename V> operator A<V>();
					};
					
					template <typename T>
					void foo(A<void (T)>);
					
					void foo();
						Specify the cases where deduction fails because list contexts do not match in length.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

					`int main() {`
					`  A<void (int, char)> a;`
					`  foo<int>(a);`
					`  foo(a);  // deduces T to be int`
					`}`
					
					In subclause 14.8.2.5 [temp.deduct.type] paragraph 10 of N3690, deduction from a function type considers P/A pairs from the parameter-type-list only where the "P" function type has a parameter.
					
					Deduction is not specified to fail if there are additional parameters in the corresponding "A" function type.
					
					Is the above example intended to be well-formed?		

CA 5		13.3.3.1	p4	ed	Resolve CWG issue 1673.	The following proposed resolution merges the one in CWG issues list R85 with the one sent to the CWG chair last year in February 2012. **Proposed wording:** However, user-defined conversion sequences are not considered if: • the target is the first parameter of a constructor of a class X, or • the target is the implicit object parameter	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

						of a user-defined conversion function, and the constructor or user-defined conversion function is a candidate by:
						• 13.3.1.3 [over.match.ctor], when the argument is the temporary acting as the source in the second step of a class copy-initialization.
						• 13.3.1.4 [over.match.copy], 13.3.1.5 [over.match.conv], or 13.3.1.6 [over.match.ref] (in all cases).
						• the second phase of 13.3.1.7 [over.match.list] when the initializer list has exactly one element, and the conversion is to a class X or reference to (possibly cv-qualified) X.
						
						[*Example:*
						`struct X { };`
						`struct B { operator X&(); };`
						`B b;`
						`X x({b}); // error: B::operator X&() is not a candidate`
						—end example]	

CA 6		10.2	p3, 4, 7	te	**Hiding of base member named by using-declaration unsupported by class member lookup** N3690 subclause 3.3.10 [basic.scope.hiding] has various cases of name hiding which re-	Add wording to eliminate hidden declarations from the declaration set.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

					fer to derived classes. All such cases point to 10.2.
					
					N3690 subclause 7.3.3 [namespace.udecl] paragraph 15 says:
					
					When a *using-declaration* brings names from a base class into a derived class scope, member functions and member function templates in the derived class override and/or hide member functions and member function templates with the same name, parameter-type-list (8.3.5), cv-qualification, and *ref-qualifier* (if any) in a base class (rather than conflicting).
					
					Notwithstanding the example in that paragraph, the effect of such “hiding” on class member name lookup is unclear.
					
					Consider:
					
					`struct B { void h(int); };`
					`struct D : B`
					`{ using B::h; void h(int); };`
					`void foo()`
					`{ void (D::*memfp)(int) = &D::h; }`
					
					C++03 wording; subclause 10.2 [class.member.lookup] paragraph 2:
					
					A member name `f` in one subobject `B` *hides*		

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat
					a member name `f` in a sub-object `A` if `A` is a base class sub-object of `B`. Any declarations that are so hidden are eliminated from consideration. Each of these declarations that was introduced by a *using-declaration* is considered to be from each sub-object of `C` that is of the type containing the declaration designated by the *using-declaration*. Under this wording, the declaration that was introduced by the using-declaration is hidden and would not be found by class member lookup. This is not the case with the C++11 wording. By N3690 subclause 10.2 [class.member.lookup] paragraphs 3, 4 and 7, the result of the name lookup is the declaration set of *S*(*f*,*C*). The declaration set contains both the member named by the using-declaration as well as the member declared in the derived class using a declarator since both are declarations present in the derived class. The resolution for taking the address of an overloaded function would then fail since there is more than one selected function.		
CA 7		7.2	p5	te	**Representability within an enumeration**	Add a note to clarify the behavior of the case	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

					type
					The type of a enumerator before the enumeration is complete is the type of its initializer. This initializer may be of enumeration type.
					A enumerator with no initializer has a value that is one greater than the previous enumerator. Its type before the enumeration is complete is the type of the previous enumerator unless if the incremented value is not representable in that type. An unspecified integral type sufficient to contain the incremented value is used instead (and the program is ill-formed if no such type exists).
					The values of an enumeration are defined by [dcl.enum]. It can be taken that a value is not representable in an enumeration type if it is not within the range of enumeration values of the enumeration type.
					For the case below, there is implementation divergence between compilers with almost a 50/50 split between the ones surveyed:
					`template <typename T>`
					`struct Hack;`
					`enum E { F, T };`
						presented.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

					`template <>`
					`struct Hack<E>`
					`{ enum { val = 42 }; };`
					
					`namespace A {`
					`  enum { A = F, B, C =`
					`Hack<decltype(B)>::val };  // #1 --`
					`// Hack<E>::val`
					`}`
					
					`namespace B {`
					`  enum { A = T, B, C =`
					`Hack<decltype(B)>::val };  // #2 --`
					`// Hack<?>::val`
					`}`
					
					The compilers surveyed produce either an error for both #1 and #2 or produces an error for neither of the two.
					
					In short, we are not aware of any implementation that behaves consistently with what appears to be the obvious interpretation of the wording.		

CA 8		6.4.2	p2	te	**Undefined behavior for members of scoped enumerations in a switch** Using a scoped enumeration as the controlling condition of a switch does not seem to work well. The current (N3690) wording of [stmt.switch]	In 6.4.2 [stmt.switch] paragraph 2: <ins>For a type that is subject to the integral promotions (4.5), integral</ins>~~Integral~~ promotions are performed.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

CA 9		14.8.3		te	paragraph 2 does not mention scoped enumerations and says that "[i]ntegral promotions are performed"; however, it is not defined by [conv.prom] what integral promotion on a scoped enumeration is.
					Partial ordering while ignoring cv-qualification on top of function types
					Should one template be considered more specialized than the other below?
					This is similar to CWG issue 1221 that we opened for reference collapsing.
					`// C++03 behaviour is different from C++11 behaviour.`
					`template <class T>`
					`int foo(const T &t);  // not a candidate under C++03`
					`// SFINAE C++03 14.8.3 [temp.ovr]    paragraph 1`
					`//        C++03 14.8.2 [temp.deduct] paragraph 4`
					`//        C++03 14.8.2 [temp.deduct] paragraph 2 (last point)`
					`// There is a wording problem in C++03 as to what happens with argument deduction itself forms an invalid type.`
					`// Substitution is only said to occur either from explicitly specified template arguments or into non-deduced contexts.`
					`template <class T>`
						Take a unified approach to resolving CWG issue 1221 and this additional case.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

CA 10		5.16	p2	te	`void foo(T &t);  // less specialized template under N3690`
					`   // N3690 14.8.2.4 [temp.deduct.partial] paragraphs 6, 9.`
					
					`void bar() { }`
					`int main() { return foo(bar); }`
					Expanding conditional expression special case for throw expressions
					
					The Standard (N3690) in 5.16 [expr.cond] paragraph 2, makes a special case for when one of the result (second or third) expressions is a (possibly parenthesized) throw expression. Unfortunately, the wording does not handle certain natural continuations:
					
					The parenthesized case was added in CWG issue 1550.
					
					`// last operand of comma expression should be considered the form of the comma-expression`
					`    1 ? 0 : (0, throw 0);`
					
					`// both branches of a conditional are throw expressions`
					`    1 ? 0 : (1 ? throw 0 : throw 0);`
						Preferably, exceptions manually specified using the form of the syntactic construct would not be used; however, lacking a general mechanism to specify properties such as an expression being not returning, the specification should either not have any special rules for semantic reasons which are formulated by syntax, or should have as many rules as necessary to cover obvious and easily described cases.	

CA 11		12.3.1	p1	te	**Unnecessary change in definition of converting constructors** 12.3.1 [class.conv.ctor] has removed the requirement that a *converting constructor*	Reverse the change.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat
					must be callable with a single parameter. This change does not appear to be necessary, as all overload resolution contexts for which converting constructors are specifically considered to be candidates (13.3.1.3 [over.match.ctor], 13.3.1.4 [over.match.copy]) are initialization contexts outside of list initialization. Any constructors which are now converting constructors which are not callable with only a single parameter cannot be viable candidates in the aforementioned contexts.		
CA 12		18.8.5	p8	te	**std::current_exception and odr-use, accessibility or existence of a copying constructor** `std::current_exception` is intended to be implementable such that copying the exception object is a viable strategy; however, there seems to be no indication that when an exception is thrown, that a copying constructor is required to be accessible and considered odr-used. In particular, the candidate used to construct the exception object is allowed to be a move constructor or otherwise unsuitable for performing the copy because the exception ob-	Either specify the appropriate odr-use, candidate selection and access checking at the throw site (rendering move-only types non-throwable) or modify the library specification to allow copying only for very limited cases.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Sub-clause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

ject is an lvalue.

Consider the following:

a.cc:

```
struct A {
   friend void foo();
private:
   A() { }
   A(const A &) = delete;
   template <typename T> A(T &);
   A(A &&) { }
};

void foo() {
   throw A();
}
```

b.cc:

```
#include <exception>
struct A;
void foo();

bool bar() {
   try { foo(); }
   catch (...) { return
std::current_exception() ==
std::current_exception(); }
}
```

How is an implementation expected to perform the copying?

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

CA 13		15.3	p16	te	Should the best candidate for the copy operation have been considered odr-used at the point of the throw-expression?
					
					Similarly, is the access checking for the copying constructor performed in the context of the throw-expression?
					
					Converting constructor candidate surprise for copy initialization of catch-clause parameter
					
					N3690 subclause 15.3 [except.handle] paragraph 16 states that the catch-clause parameter is copy-initialized from the exception object.
					
					It does not say that, for catching-by-value (as opposed to by pointer or by reference), it is copy-initialized from the exception object cast to be an lvalue of the declared type of the catch-clause parameter.
					
					Consider the following:
					
					ehDerivedToBaseConvertingA.cc:
					
					`extern int ret;`
					`struct B;`
					
					`struct A {`
					`  A() { }`
					`  A(B &) { ret = 1; }`
					`};`
						Specify the initialization such that the constructor used is predictable in the context of the catch-clause.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

					struct B : A { };
					
					void foo() {
					  throw B();
					}
					
					ehDerivedToBaseConvertingB.cc:
					extern int ret;
					struct B;
					
					struct A {
					  A() { }
					  A(B &) { ret = 1; }
					};
					
					void foo();
					
					int ret = 0;
					
					int main() {
					  try { foo(); }
					  catch (A a) { if (ret == 0) throw; }
					}
					
					It seems that a good number of implementations have no problem performing copy-initialization with converting constructors which convert from a derived type to a base type except that they do not consider such candidates when initializing a catch-clause		

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

CA 14		4.1	p2	te	parameter.
					Seeing as derived-to-base converting constructors are most likely bad form, perhaps overload resolution should not consider them viable candidates at all?
					Alternatively, the initialization of catch-clause parameters can be specified such that the type of source expression is adjusted and not necessarily an lvalue of the most derived type of the exception object.
					Definedness of out-of-lifetime lvalue-to-rvalue conversion of block-scope constants
					Certain contexts, such as lambda expressions and local class definitions, allow name expressions which name block-scope constants even though the evaluation of the name expression may (technically) occur after the storage for the "constant" has been released.
					Consider:
					`struct Base {`
					`  virtual int call() = 0;`
					`};`
					`Base *foo() {`
					`  constexpr int x = 0;`
						Remove this source of undefined behavior.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

					`struct Local : Base {`
					`  virtual int call() { return x; }`
					`};`
					`static Local local;`
					`return &local;`
					`}`
					
					`int main() {`
					`return foo()->call();`
					`}`
					
					While the likely intention is that the lvalue-to-rvalue conversion of the block-scope constant is implemented by using the value of the constant expression in place of reading from storage, it seems that the wording does not prevent the program above from being subject to undefined behaviour caused by lifetime violation.
					
					In particular, it seems that a name expression that appears in a potentially-evaluated expression such that the object named is not odr-used (by that instance of the name) may still be evaluated, in theory, as an lvalue through which the object named or a subobject thereof is accessed (see N3690 subclause 4.1 [conv.lval] paragraph 2).		

CA 15		14.8.2.5	p17	te	**Applicability of non-type template parameter type agreement on deduction from**	Consider modifying the restriction to instead treat the instance of the template parameter	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

					return types
					
					In N3690 subclause 14.8.2.5 [temp.deduct.type] paragraph 17, there is a requirement regarding the type of a non-type template parameter and the type of the deduced template argument (if any).
					
					Namely, the two types are required to match "exactly" if the template argument is deduced from a use of the template parameter in the parameter-list of the function.
					
					The requirement as worded seems oddly restricted. It does not cover return types for example.
					
					Consider:
					
					`template <int N> struct A;`
					
					`template <short N>`
					`A<N> *foo();`
					
					`void bar() {`
					`  A<1> *(*fp)(void) = &foo;`
					`}`
					
					GCC and Clang does not accept the above. We are not sure if there is separate wording which supports their behavior.
					
					The result of the wording is also odd in that
						as being in a non-deduced context.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

					adding the explicit cast to the use in the function type would cause the use to be in an non-deduced context, and the behaviour with the explicit cast would be different from that without if the argument could be otherwise deduced.
					Consider:
					`template <int N> struct A;`
					`template <short N>`
					`void foo(A<N> *, char (*)[N]);`
					`template <short N>`
					`void bar(A<int(N)> *, char (*)[N]);`
					`void (*fp1)(A<1> *, char (*)[1]) = foo;`
					`void (*fp2)(A<1> *, char (*)[1]) = bar;`
					Perhaps some rationale would be useful as to why failing the deduction for foo() is superior to treating it as non-deduced.
					Related:
					http://llvm.org/bugs/show_bug.cgi?id=16279, "Deduction succeeds despite type mismatch of non-type template parameter and deduced argument"
					http://gcc.gnu.org/bugzilla/show_bug.cgi?id=57570, "Deduction succeeds despite type		

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat
					mismatch of non-type template parameter and deduced argument"		
CA 16		5.1.2	p17	te	Resolve CWG issue 1613.		
CA 17		5.1.2	p12	te	Resolve CWG issue 1612.		
CA 18		14.7.1	p13	te	Resolve CWG issue 1664.		
CA 19		5.1.2	p6	te	Resolve CWG issue 1663.		
CA 20		5.1.2	p10	te	Resolve CWG issue 1662.		
CA 21		3.4.3	p1	te	**Name lookup in nested-name-specifiers versus lookup-dependent grammar productions** According to subclause 3.4.3 [basic.lookup.qual] paragraph 1: If a `::` scope resolution operator in a *nested-name-specifier* is not preceded by a *decltype-specifier*, lookup of the name preceding that `::` considers only namespaces, types, and templates whose specializations are types.		

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

					Consider the following source:
					`template <typename T> struct B { };`
					
					`namespace N {`
					`  namespace L {`
					`  template <int> void A();`
					`  }`
					
					`  namespace M {`
					`  template <int> struct A { typedef int y; };`
					`  }`
					
					`  using namespace L;`
					`  using namespace M;`
					`}`
					
					`B<N::/*template */A<0>::y> (x); //`
					`  // basic.lookup.qual applies?`
					
					The interpretation of the < token after `N::A` depends on name lookup (14.2 [temp.names] paragraph 3).
					
					If this lookup occurs prior to determining that the quoted portion of 3.4.3 paragraph 1 applies, then it will find that the name is ambiguous.
					
					When should the name lookup for the potential *template-name* occur in relation to determining the applicability of the wording in		

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat
					question? Is the last line equivalent to `B<N::M::A<0>::y> x;` or not? Of interest is that if the optional `template` keyword was uncommented then lookup for A is not necessary to determine the interpretation of the < token after it. Is the behavior for the modified case (with the `template` keyword present) expected to be the same as the case where `template` remains commented out?		

CA 22		5.1.2	p7	te	**Lookup for __func__ in lambda bodies**
					Is it intended that the following can be a translation unit in a well-formed program?
					`namespace K {`
					`  auto ff = [] { return __func__; };`
					`}`
					When we are told by N3690 subclause 5.1.2 [expr.prim.lambda] paragraph 7 that the compound-statement "yields" the function-body of the function call operator, it is understood that it means that a function-body is produced from the compound-statement
						If lookup failure is expected for the case presented, clarify by adding an example.
						In the alternative, add wording for the scope of a predefined variable, `__func__`, within the compound-statement of a lambda expression.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat
					and that the latter is not an actual function-body. We are also told that for the purposes of name lookup, the compound-statement is considered in the context of the lambda-expression. We find that, in the absence of a function-body, `__func__` is not specified to be a predefined variable (8.4.1 [dcl.fct.def.general] paragraphs 7-8). We also note that the form of a function definition (8.4.1 paragraph 1) is not present in the above program and that the wording in subclauses 3.3.2 [basic.scope.pdecl] and 3.3.3 [basic.scope.block] (paragraphs 8 and 2, respectively) covers only function-local predefined variables in function definitions. The conclusion is that `__func__` in a lambda body is bound using the context of the lambda-expression, and not bound later to be the function-local predefined variable which would exist in the context of the function-call operator's compiler-generated definition. In the case of the above code, it means that the lookup for `__func__` fails and renders the program ill-formed.		

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat
					Implementations such as GCC behave in a manner inconsistent with this interpretation.		
CA 23		12.8	p31	te	Resolve CWG issue 1493. An exception object is an lvalue whose lifetime is not necessarily ending soon—consider rethrows and exception_ptr.	Remove initialization of exception-declarations as a context for copy elision.	
CA 24		15.2	p2	te	Revisit CWG issue 1424. It is rather conspicuous that an array new requires the odr-use of the destructor (of the most derived object) even if the constructor used for the initialization is a noexcept-specification compatible with noexcept(true). It may also make sense if the destruction of completely constructed subobjects (and the corresponding case with delegating constructors) is omitted when the constructor for the most derived object is a noexcept-specification compatible with noexcept(true). Note that the current treatment of base and member subobject destruction and general stack unwinding is not consistent for noexcept-specifications—for general stack unwinding, destructors might not be called before terminate() is. Related discussion:	Identify cases where the destructor would not be called because of interaction with noexcept and exempt them from causing odr-use. Also, harmonize the treatment of subobject destruction for partially constructed objects with general stack unwinding.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat
					https://groups.google.com/a/isocpp.org/forum/#!topic/std-discussion/DL75bqa8f2g		
CA 25		8.5	p17	te	Resolve CWG issue 670. It seems that certain initialization contexts are expected to be slicing/copy operations but the overload resolution is not straightforward (see CA 13). Related bug reports: http://llvm.org/bugs/show_bug.cgi?format=multiple&id=16773 http://gcc.gnu.org/bugzilla/show_bug.cgi?format=multiple&id=58052	Modify the overload resolution such that the first parameter is considered to be converted to a lvalue/xvalue of the target type and the ellipsis conversion is not considered. Also, in 12.3 [class.conv], modify the following to be a note: At most one user-defined conversion (constructor or conversion function) is implicitly applied to a single value.	
CA 26		2.2	p2	te	In N3690 subclause 2.2 [lex.phases] paragraph 2, the formation of a character sequence which matches the syntax of a UCN via line-splicing causes undefined behavior. It is unclear whether a program which has such a line-splicing in a context where it will be reverted (as part of a raw-string literal) will be subject to undefined behavior because of UCN formation.	The undefined behavior is undesired.	
CA 27		8.2		te	Resolve CWG issue 1740.		
CA		3.2	p3	te	N3690 subclause 3.2 [basic.def.odr] paragraph 3, in describing cases where odr-use	Clarify and also resolve CWG issue 1741.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat
28					does not occur says in part: ... unless x satisfies the requirements for appearing in a constant expression (5.19) ... By this, a reader might assume that x is such that an lvalue-to-rvalue conversion may be applied within a constant expression to a glvalue that refers to x; however, the wording seems rather unclear.		

CA 29		12.8	p2	te	According to N3690 subclause 12.8 [class.copy] paragraph 2:
					A non-template constructor for class X is a copy constructor if its first parameter is of type `X&`, `const X&`, `volatile X&` or `const volatile X&`, and either there are no other parameters or else all other parameters have default arguments (8.3.6).
					On the topic of providing default arguments later in the translation unit, paragraph 7 has this to say:
					Thus, for the class definition
					`struct X {`
					`  X(const X&, int);`
					`};`
					a copy constructor is implicitly-declared. If the user-declared constructor is later defined
						Resolve CWG issue 1344 by specifying that the formation of special-member functions by adding default arguments to later declarations is ill-formed.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

as

`X::X(const X& x, int i =0) { /* ... */ }` then any use of X's copy constructor is ill-formed because of the ambiguity; no diagnostic is required.

It does not say that the user-declared constructor is not a copy constructor.

The "no diagnostic required" probably needs clarification if it is not meant to say that an actual use which hits the ambiguity does not need a diagnostic.

This begs the question: is the same class type trivially copyable in some places and not in others (depending on which default arguments are known in that context)?

Consider the following two files:

```
#include <type_traits>
struct A
{ A() = default;
  A(const A &) = default;
  A(A &, int); };

A::A(A&, int = 0) { }
inline constexpr bool isTrivialA() { re-
turn std::is_trivial<A>::value; }

static_assert(!isTrivialA(), "");
// GCC asserts, Clang and ICC does not
```

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

					`#include <type_traits>`
					`struct A`
					`{ A() = default;`
					`  A(const A &) = default;`
					`  A(A &, int); };`
					
					`inline constexpr bool isTrivialA() { return std::is_trivial<A>::value; }`
					
					`static_assert(isTrivialA(), ""); // none of GCC, Clang or ICC assert`
					
					It would seem that
					• whether the user-defined constructor is a copy constructor is not agreed upon across implementations, and
					• the definition of isTrivialA() does not violate the ODR rules nor does it violate the constraints for std::is_trivial, yet it ends up with behavior that is not consistent across instances from different translation units.
					This probably affects at least some of the other special member functions as well.		

CA 30		8.5.3	p5	te	Resolve CWG issue 1604.	If the target that is the destination of the non-reference copy-initialization is specified to be the reference (and not the temporary invent-	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Canadian C++14 Comments

Authors: Hubert Tong, Michael Wong

Document number: N3771

Date: 2013-09-22

Project: Programming Language C++, CWG

Reply-to: Michael Wong, michaelw@ca.ibm.com

Date: 2013-09-22	Document:	Project: Programming Languages — C++

MB[1] NC	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of[2] comment	Comments	Proposed change	Observations of the secretariat

						ed in [dcl.init.ref]), the direct initialization step would naturally not require a constructor. That is, the user-defined conversion function is called and the result is used to direct-initialize the reference.
						
						This may also address the concerns over slicing and performance mentioned in CWG issue 1650.
						
						In addition to wording to specify the above, add a note:
						A conversion function returning a cv-qualified type may produce a value for which the binding is ill-formed.
						
						Example:
						`struct B { };`
						`struct A { operator const B(); };`
						
						`void foo() {`
						`  typedef const B ConstB;`
						`  B &&b1 = ConstB(); // ill-formed`
						`  B &&b2 = A(); // ill-formed`
						`}`	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial