Doc No: X3J16/96-0110 WG21/N0928
Date: May 28th, 1996
Project: Programming Language C++
Ref Doc:
Reply to: Josee Lajoie
          (josee@vnet.ibm.com)

Core WG List of Issues

The issues listed as editorial or as closed in the version of the core list of issues that appeared in the Post-Santa Cruz mailing (96-0084/N0902) were resolved in the pre-Stockholm version of the working paper (WP) and are therefore not listed in this version of the core list of issues.

The issues listed as closed in this version of the core list of issues where opened issues in previous versions of the core list of issues and have been handled as editorial issues in the pre-Stockholm version of the WP.

C Compatibility

3.9.1 [basic.fundamental]:
  643: The term "integer code" needs to be defined
5.6 [expr.mul]:
  600: Should the value returned by integer division and remainder be defined by the standard?
5.19 [expr.const]:
  537: Can the implementation accept other constant expressions?
16.8 [cpp.predefined]:
  661: Should __DATE__ and __TIME__ be made locale aware?

Lexical Conventions

2.1 [lex.phases]:
  634: Do the phases of translation need to discuss shared libraries?
2.2 [lex.charset]:
  607: Definition needed for source character set

Core1

General

1.1 [intro.scope]:
  604: Should the C++ standard talk about features in C++ prior to 1985?
1.7 [intro.compliance]:
  602: Are ill-formed programs with non-required diagnostics really necessary?
  619: Is the definition of "resource limits" needed?

Linkage / ODR

3.2 [basic.def.odr]:
  427: When is a diagnostic required when a function used is not defined?
  556: What does "An object/function is used..." mean?
3.5 [basic.link]:
  526: What is the linkage of names declared in unnamed namespaces?

```
  615: Do conflicting linkages in different scopes cause undefined behavior?
7.5 [dcl.link]:
   78: Linkage specification and calling protocol
  420: Linkage of C++ entities declared within ‘extern "C"’
  616: Can the definition for an extern "C" function be provided in two
       different namespaces?
9.5 [class.union]:
  505: Must anonymous unions declared in unnamed namespaces also be static?

Memory Model
------------
3.7.3 [basic.stc.dynamic]:
  641: Which allocation/deallocation functions are predefined and which ones
       may be overriden in a program?
3.7.3.1 [basic.stc.dynamic.allocation]:
  642: Is the behavior of new(size_t, const std::nothrow&)
       implementation-defined?
5.3.1 [expr.unary.op]:
  645: Should &*(array+upperbound) be allowed?
5.3.4 [expr.new]:
  453: Can operator new be called to allocate storage for temporaries, RTTI
       or exception handling?
  577: Are there any requirements on the alignment of the pointer used with
       new with placement?
  637: How is operator delete looked up if the constructor from a new with
       placement throws an exception?
  638: Accesibility of ctor/dtor, operator new and operator delete
5.9 [expr.rel]:
  513: Are pointer conversions implementation-defined or unspecified?

Object Model
------------
3.6.2 [basic.start.init]:
  613: What is the order of destruction of objects statically initialized?
6.4 [stmt.select]:
  639: What is the lifetime of declarations in conditions?
6.7 [stmt.dcl]:
  635: local static variable initialization and recursive function calls
9.5 [class.union]:
  655: When is storing into another union member ill-formed?
10.1 [class.mi]:
  624: class with direct and indirect class of the same type: how can the
       base class members be referred to?
12.2 [class.temporary]:
  598: Should a diagnostic be required if an rvalue is used in a
       ctor-initializer or in a return stmt to initialize a reference?
12.8 [class.copy]:
  536: When can objects be eliminated (optimized away)?


+-------+
| Core2 |
+-------+

Sequence Points
---------------
1.8 [intro.execution]:
  603: Do the WP constraints prevent multi-threading implementations?
  605: The execution model wrt to sequence points and side-effects needs work
  633: Is there a sequence point after the operand of dynamic_cast is
       evaluated?

Name Look Up
------------
3.4.2.2[namespace.qual]:
  654: Qualified look up for names after global scope ::
```

```
7.3.3 [namespace.udecl]:
  646: Can a using declaration refer to a hidden base class member?
  650: How does name look up proceed for the name in a using declaration?
7.3.4 [namespace.udir]:
  612: name look up and unnamed namespaces
8.3 [dc.meaning]:
  636: Can a typedef-name be used to declare an operator function?
10.1 [class.mi]:
  446: Can explicit qualification be used for base class navigation?

Access
------
11.8[class.access.nest]:
  653: What does it mean for nested classes if a class-name is inserted into
       the scope of the class itself?
11.4[class.friend]:
  656: access of names used in base clauses

Types / Classes / Unions
------------------------
3.9 [basic.life]:
  621: The terms "same type" need to be defined
9.6 [class.bit]:
   47: enum bitfields - can they be declared with < or > bits than required?

Default Arguments
-----------------
8.3.6 [dcl.fct.default]:
  531: Is a default argument a context that requires a value?
  640: default arguments and using declarations
12.6 [class.init]:
  138: When are default ctor default args evaluated for array elements?

Type Conversions / Function Overload Resolution
-----------------------------------------------
4.13 [conv.bool]:
  601: Should implicit conversion from int to bool be allowed?
5.4 [expr.cast]:
  660: Conversions allowed by C style casts are too broad
5.9 [expr.rel]:
  493: Better description of the cv-qualification for the result of a
       relational operator needed
13.3.3.1 [over.best.ics]:
  652: Is a derived-to-base conversion required to be implemented by a copy
       constructor of the base class?
13.6 [over.built]:
  658: Should declarations for binary built-in operators only accept operands
       of the same type?
  659: Should the prototypes for built-in operators properly take into
       account arithmetic conversions?


+--------+
| Core 3 |
+--------+

Pointer to members
------------------
5.5 [expr.mptr.oper]:
  644: Must the operand of .* and ->* have a complete class type?

RTTI
----
5.2.6 [expr.dynamic.cast]:
  549: Is a dynamic_cast from a private base allowed?
```

Exception Handling

15.1 [except.throw]:

- 647: Is it implementation-defined or unspecified how the memory for the exception object is allocated?

15.3 [except.handle]:

- 541: Is a function-try-block allowed for the function main?
- 542: What exception can a reference to a pointer to base catch?
- 587: Can a pointer/reference to an incomplete type appear in a catch clause?
- 648: Is it implementation-defined or unspecified whether the stack is unwound before terminate is called?

15.4 [except.spec]:

- 588: How can exception specifications be checked at compile time if the class type is incomplete?
- 630: What is the exception specification of implicitly declared special member functions?
- 631: Must the exception specification on a function declaration match the exception specification on the function definition?
- 657: Must the exception-specification of a declaration be more or less restrictive than the exception-specification of the definition?

15.5.1 [except.terminate]:

- 649: Should it be mandated that terminate be called upon internal error?

15.5.2 [except.unexpected]:

- 651: Is unexpected called before the stack is partly unwound?

Closed Issues - issues resolved at the Tokyo meeting

4.9 [conv.fpint]:

- 617: Are floating point conversions unspecified or implementation-defined?

12.8 [class.copy]:

- 626: What is the form of the implicitly-declared operator= if a base class has Base::operator=(B)?

Chapter 1 - Introduction

Work Group: Core
Issue Number: 604
Title: Should the C++ standard talk about features in C++ prior to 1985?
Section: 1.1 [intro.scope]
Status: active
Description:

> UK issue 229:
> "Delete the last sentence of 1.1 and Annex C.1.2. This is the first standard for C++, what happened prior to 1985 is not relevant to this document."

Resolution:
Requestor: UK issue 229
Owner: Josee Lajoie (General)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 602
Title: Are ill-formed programs with non-required diagnostics really necessary?
Section: 1.7 [intro.compliance]
Status: active
Description:

> UK issue 9:

```
        "We believe that current technology now allows many of the
         non-required diagnostics to be diagnosed without excessive overhead.
         For example, the use of & on an object of incomplete type, when the
         complete type has a user-defined operator&(). We would like to see
         diagnostics for such cases."

        [note JL:]
        At the Tokyo meeting, we discussed this a bit and decided that this
        issue required more dicussions.

        Question: Do deprecated features render a program ill-formed but
        no diagnostic is required?

        See also UK issue 93.
Resolution:
Requestor:      UK issue 9
Owner:          Josee Lajoie (General)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   619
Title:          Is the definition of "resource limits" needed?
Section:        1.7 [intro.compliance]
Status:         active
Description:
        1.7 para 1 says:
          "Every conforming C++ implementation shall, within its resource
           limits, accept and correctly execute well-formed C++ programs..."
        The term resource limits is not defined anywhere.
        Is this definition really needed?
Resolution:
Requestor:      ANSI Public comment 7.12
Owner:          Josee Lajoie (General)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   603
Title:          Do the WP constraints prevent multi-threading
                implementations?
Section:        1.8 [intro.execution]
Status:         active
Description:
        UK issue 11:
        "No constraints should be put into the WP that preclude an
         implementation using multi-threading, where available and
         appropriate."

        Bill Gibbons notes:
        For example, do the requirements on order of destruction between
        sequence points preclude C++ implementations on multi-threading
        architectures?
Resolution:
Requestor:      UK issue 11
Owner:          Steve Adamczyk (sequence points)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   605
Title:          The execution model wrt to sequence points and side-effects
                needs work
Section:        1.8 [intro.execution]
Status:         active
Description:
        See UK issues 263, 264, 265, 266:
        1.8 para 9:
        "What is a "needed side-effect"? This paragraph, along with
         footnote 3 appears to be a definition of the C standard "as-if"
         rule.  This rule should be defined as such.  [Proposed definition
         of "needed": if the output of the program depends on it.]"
        1.8 para 10:
        "It is not true to say that values of objects at the previous
         sequence point may be relied on.  If an object has a new value
         assigned to it and is not of type sig_atomic_t the bytes making up
         that object may be individually assigned values at any point prior
         to the next sequence point.   So the value of any object that is
         modified between two sequence points is indeterminate between those
         two points.  This paragraph needs to be modified to reflect this
         state of affairs."

        Also, para 11:
        "Such an object [of automatic storage duration] exits and retains its
         last-stored value during the execution of the block and while the
         block is suspended ..."
        This is not quite correct, the object may not retain its last-stored
        value.

        Para 9, 10, 11 and 12 also contain some undefined terms.
Resolution:
Requestor:      UK issues 263, 264, 265, 266
Owner:          Steve Adamczyk (sequence points)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   633
Title:          Is there a sequence point after the operand of dynamic_cast
                is evaluated?
Section:        1.8 [intro.execution]
Status:         active
Description:
        Box 1 in 1.8 says:
        "The Working group is still discussing whether there is a sequence
         point after the operand of dynamic-cast is evaluated; this is a
         context from which an exception might be thrown, even though no
         function call is performed.  This has not yet been voted upon by the
         Working Group, and it may be redundant with the sequence point at
         function-exit.
Resolution:
Requestor:
Owner:          Steve Adamczyk (sequence points)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
==============================================================================
 Chapter 2 - Lexical Conventions
---------------------------------
Work Group:     Core
Issue Number:   634
Title:          Do the phases of translation need to discuss shared
                libraries?
Section:        2.1 [lex.phases]
Status:         active
Description:
        Box 3:
        Do the phase of translations need to discuss shared libraries?
Requestor:
Owner:          Tom Plum (Lexical Conventions)
Emails:
Papers:
```

```
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   607
Title:          Definition needed for character set(s)
Section:        2.1 [lex.charset]
Status:         active
Description:
        There are many issues regarding definitions of character sets.
        Here are the issues that were raised by the public comments:
        o In 1.4 [_intro.defs_]:
          Multibyte character.  This definition uses the terms "extended
          character set" which is not defined.
          Also, in the last sentence:  What is the basic character set?
          Is it the basic source character set or basic execution character
          set?
        o 2.11.2 [lex.ccon_]:
          Paragraph 1 uses the phrase "execution character set" which is not
          defined.
        o 3.6.1 [_basic.start.main_]:
          The description uses the phrase "null-terminated multibyte strings
          (NTMBSs)," but this is nowhere defined.
Resolution:
Requestor:      UK issue 288
Owner:          Tom Plum (Lexical Conventions)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
=============================================================================
```

Chapter 3 - Basic Concepts

```
----------------------------
Work Group:     Core
Issue Number:   427
Title:          When is a diagnostic required when a function/variable with
                static storage duration is used but not defined?
Section:        3.2 [basic.def.odr] One Definition Rule
Status:         active
Description:
        When is a diagnostic required if no definition is provided for a
        function or for variable with static storage duration?

        int main() {
                extern int x;
                extern int f();
                return 0 ? x+f() : 0;
        }

        Must a disgnostic be issued if x and f are never defined?

        The current WP contains this sentence: "If a non-virtual function is
        not defined, a diagnostic is required only if an attempt is actually
        made to call that function." This seems to be hinting that, for
        cases such as the one above, a diagnostic is not required.

        [Jerry Schwarz, core-6173:]
         I think we should be talking about undefined behaviors, not required
         diagnostics. That is, if a program references (calls it or takes its
         address) an undefined non-virtual function then the program has
         undefined behavior.

        [Fergus Henderson, core-6175, on Jerry’s proposal:]
         I think that would be a step backwards.  If a variable or function
         is used but not defined, all existing implementations will report a
         diagnostic.  What is to be gained by allowing implementations to
         do something else (e.g. delete all the users files, etc.) instead?

        [Mike Ball, core-6183:]
```

Then you had better not put the function definition in a shared library, since this isn’t loaded until runtime. Sometimes linkers will detect this at link time and sometimes they won’t.

[Sean Corfield, core-6182:]
I’d like it worded so that an implementation can still issue a diagnostic here (example above) AND REFUSE TO EXECUTE THE PROGRAM. If ’x’ and ’f’ were not mentioned in the program (except in their declarations) I would be quite happy that no definition is required. But unless an implementation can refuse to execute the program, you are REQUIRING implementations to make the optimisation and that is definitely a Bad Thing(tm), IMO. It seems the only way to allow that is to make the program ill-formed (under the ODR) but say no diagnostic is required.

[Fergus Henderson, core-6174:]
ObjectCenter reports a diagnostic only if an attempt is actually made to use the function or variable; in other words, link errors are not reported until runtime. In an interpreted environment, this is quite desireable.

See also UK issues 335, 336, 337.

Joe Coha also mentioned in private email:
"Do I really need to have one definition of the static data member in the program? Even if it’s unused? 9.4.2 says yes. However, this seems contradictory to the rules in 3.2. If a program is not required to define a non-local variable with static storage duration if the variable is not used, why is the WP requiring that the static data member be defined if it is not used?"

Resolution:
Requestor: Josee Lajoie
Owner: Josee Lajoie (ODR)
Emails:
core-6172
Papers:
95-0205/N0805

. . . .

Work Group: Core
Issue Number: 556
Title: What does "An object/function is used..." mean?
Section: 3.2 [basic.def.odr] One Definition Rule
Status: active
Description:
This is from public comment T25:
"It is not clear what object ’use’ and ’reuse’ is."

Neal Gafter also notes:
"When must a class destructor be defined?

According to a strict interpretation of 3.2 [basic.def.odr] paragraph 2, the destructor for class A in the program below needn’t be defined.

```
struct A {
        ~A();
};
void f() throw (A*)
{
        A *a = new A;
        throw a;
}
main()
{
        return 0;
}
```

The same question applies to many other contexts in which destructors are implicitly used. For example, the expression

```
new A[20]
```

generates code to call the destructor A::~A() when the constructor throws an exception. Does this mean the destructor must be defined in order to new an array?"

Also see UK issue 364.

Resolution:
Requestor: comment T25 (3.8)
Owner: Josee Lajoie (ODR)
Emails:
Papers:
95-0205/N0805

. . . .

Work Group: Core
Issue Number: 654
Title: Qualified look up for names after global scope ::
Section: 3.4.2.2[namespace.qual]
Status: active
Description:

The description in this clause indicates that

```
A::m
```

the name m is looked up in the scope of A and, if not found in A, in the scopes named by using directives in A, and if not found in these scopes, ...

This subclause omits to mention what happens if the name is qualified by the :: global scope resolution operator.

There are two options:

1. such a name is looked up just as unqualified-ids are, in which case a transitive closure of all active using directives in global scope is used.
2. such a name is looked up just as qualified-ids are, in which case the global scope is searched first, and if the name is not found in that scope, the scopes named by using directives in the global scope are searched, ...

I prefer option 2.

Resolution:
Requestor:
Owner: Steve Adamczyk (Name Look Up)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 526
Title: What is the linkage of names declared in unnamed namespaces?
Section: 3.5 [basic.link] Program and linkage
Status: active
Description:

What is the linkage of names declared in an unnamed namespace?
Internal linkage?
Internal linkage applies to variables and functions.
What would the status of a type definition be in an unnamed namespace? No linkage?
Can it be used to declare a function with external linkage?
Can it be used to instantiate a template?

```
namespace {
  class A { /* ... */ };
}
extern void f(A&);                           // error?
```

```
template <class T> class X { /* ... */ };
X<A> x;                                       // error?
```

If A does not have external linkage, then the two declarations are probably errors. If it does have external linkage, then the two declarations are legal (and the implementation probably has to worry about name mangling).

Resolution:
Requestor: Mike Anderson
Owner: Josee Lajoie (Linkage)
Emails:
core-5905 and following messages.
Papers:

. . . .

Work Group: Core
Issue Number: 615
Title: Do conflicting linkages in different scopes cause undefined behavior?
Section: 3.5 [basic.link] Program and linkage
Status: active
Description:

Is the following program, consisting of two translation units, well-formed? What should it print?
In C, this program would be undefined because "If, within a translation unit, the same identifier appears with both internal and external linkage, the behavior is undefined"
[ANSI C section 3.1.2.2]

```
// t1.cc
        #include <stdio.h>
        int main(void) {
                extern int *const pia ; // external linkage
                printf("%d\n", !pia);
                return( 0) ;
        }
        int ia = 0 ;
        static int *const pia =&ia ;    // internal linkage

// t2.cc
        extern int *const pia = 0;
```

Another example, using namespaces:

```
namespace N {
        static int i; //1
        int f(int j) {
                int i = 5; //2
                if (j > 0) return i;
                else
                {
                        extern int i; //3
                        return i;
                }
        }
}
```

7.3.1.2[namespace.memdef] para 4 says:
"When an entity declared with a block scope extern declaration is not found to refer to some other declaration, then that entity is a member of the innermost enclosing namespace."

3.5[basic.link] para 6 says:
"If the block scope declaration matches a previous declaration of the same object, the name introduced by the block scope declaration receives the linkage of the previous declaration; otherwise, it receives external linkage."

The declaration on line //3 refers to N::i.
However, the declaration of N::i on line //1 is hidden by the declaration of block scope i on line //2.
So the variable N::i introduced by the declaration on line //3 has external linkage, which does not match the linkage specified by the hidden declaration of N::i on line //1.

Proposed Resolution:

Add a rule to the C++ WP (probably in 3.5[basic.link] at the end of para 6) that says basically what the rule in the C standard says:

"If, within a translation unit, an extern block scope declaration gives an object external linkage and, a hidden declaration or a declaration of the same object that appears later on in the translation unit gives the object internal linkage, the behavior is undefined."

Resolution:
Requestor: Neal M Gafter <Neal.Gafter@Eng.Sun.Com>
Owner: Josee Lajoie (Linkage)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 613
Title: What is the order of destruction of objects statically initialized?
Section: 3.6.2 [basic.start.init]
Status: active
Description:

Given:

```
        struct A { int i; ~A(); };
        A a = { 1 };
```

If an implementation decides to initialize a.i "statically", when must the implementation destroy a.i? i.e. what does it mean in such cases to destroy a.i "in reverse order of construction"?

Resolution:
Requestor: Erwin Unruh
Owner: Josee Lajoie (Object Model)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 641
Title: Which allocation/deallocation functions are predefined and which ones may be overriden in a program?
Section: 3.7.3 [basic.stc.dynamic]
Status: active
Description:

Para 2 should be made clearer to indicate:
o which one of the allocation/deallocation functions are predefined, and
o which one a program may override.
I believe the answer to these two questions is not the same.

```
::operator new(size_t)
::operator new(size_t, void*)
::operator new(size_t, const std::nothrow&)
::operator new[](size_t)
::operator new[](size_t, void*)
::operator new[](size_t, const std::nothrow&)
::operator delete(void*)
::operator delete(void*, void*)
::operator delete(void*, const std::nothrow&)
::operator delete[](void*)
::operator delete[](void*, void*)
```

Resolution:
Requestor: Erwin Unruh

```
Owner:          Josee Lajoie (Memory Model)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   642
Title:          Is the behavior of new(size_t, const std::nothrow&)
                implementation-defined?
Section:        3.7.3.1 [basic.stc.dynamic.allocation]
Status:         active
Description:
        para 4 says:
        "If the allocation function returns the null pointer the result is
         implementation-defined."

        This means that any use of new(size_t, const std::nothrow&) directly
        depends on implementation-defined behavior.
Proposed Resolution:
        If the allocation function returns the null pointer, the new
        expression should yield null.
Resolution:
Requestor:      Erwin Unruh
Owner:          Josee Lajoie (Memory Model)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   621
Title:          The terms "same type" need to be defined
Section:        3.9 [basic.types]
Status:         active
Description:
        The WP needs to define what it means for two objects/expressions
        to have the same type. The phrase is used a lot throughout the WP.
Requestor:
Owner:          Steve Adamczyk (Types)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   643
Title:          The term "integer code" needs to be defined
Section:        3.9.1[basic.fundamental]
Status:         active
Description:
        para 1 says:
        "Objects declared as characters (char) shall be large enough to store
         any member of the implementation’s basic character set.  If a
         character from this set is stored in a character object, its value
         shall be equivalent to the integer code of that character."

        What does "integer code" mean?
        Maybe the same wording as the one used in C should be used.
Requestor:      UK issue 407
Owner:          Tom Plum (C compatibility)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
=============================================================================
 Chapter 4 - Standard Conversions
----------------------------------
Work Group:     Core
Issue Number:   617
Title:          Are floating point conversions unspecified or
                implementation-defined?
Section:        4.9 [conv.fpint]
```

Status: closed
Description:

para 2 says:
"Otherwise, it is an unspecified choice of either the next lower or higher representable value."
ISO C says:
"Otherwise, it is an implementation-defined choice of either the nearest lower or higher representable value."

Should this be "unspecified" or "implementation-defined"?

Resolution:
Requestor: UK issue 543
Owner: Steve Adamczyk (Type Conversions)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 601
Title: Should implicit conversion from int to bool be allowed?
Section: 4.13 [conv.bool]
Status: active
Description:

ISO Swedish comment R-28:
Strengthening of bool datatype [conv.bool] The original proposal for a Boolean datatype (called bool) provided some additional type-safety at little cost. SC22/WG21 changed the proposal to allow implicit conversion from int to bool, thereby reducing type-safety and error detectability.

The implicit conversion from int to bool shall be deprecated, as described in document 93- 0143/N0350. As a future work-item, the implicit conversion should be removed.

Also see UK issue 479 and 489.
(Disallow operands of bool type with operators ++, --).

Resolution:
Requestor: Swedish Delegation
Owner: Steve Adamczyk (Type Conversions)
Emails:
Papers:

. . . .

===

Chapter 5 - Expressions

Work Group: Core
Issue Number: 549
Title: Is a dynamic_cast from a private base allowed?
Section: 5.2.6 [expr.dynamic.cast]
Status: active
Description:

paragraph 8 says:
"...if the type of the complete object has an unambiguous public base class of type T, the result is a pointer (reference) to the T sub-object of the complete object. Otherwise, the runtime check fails."

This contradicts the example that follows:

```
class A { };
class B { };
class D : public virtual A, private B { };
...
D d;
B* bp = (B*) &d;
D& dr = dynamic_cast<D&>(*bp); // succeeds
```

According to the wording in paragraph 8, the cast above should fail.

Bill Gibbons noted the following:

First, the access restrictions on dynamic_casts appear to come from the access restrictions on static_cast, where neither upcasting nor downcasting across private derivation is allowed.

Yet dynamic_cast does not apply these restrictions consistently, even for simple downcasts:

```
struct A { virtual void f() { } };
struct B : private A { };
struct C : public  B { };
void f() {
    A *a = (A*) new C;
    B *b = static_cast<B*>(a);  // ill-formed
    B *b = dynamic_cast<B*>(a); // OK under 1st "otherwise"
}
```

I see several ways to clean this up:

(1) Change the first "otherwise" clause to also require that "v points (refers) to a public base class sub-object of the most derived object". This seems closest to the intent of the current wording. It would make the above example ill-formed.

This is equivalent to saying that a dynamic cast is OK if it can be done with a static cast to the most derived type followed by a static cast to the final type, ignoring the uniqueness and virtual inheritance restrictions on static downcasts.

(2) Say something like:

A dynamic cast is well-formed if there exists a class X within the most derived object hierarchy (including the most derived class) such that:

-- "v" refers to X or a public base class of X; and

-- T is X or a public base class of X.

That is, a dynamic cast is OK if it can be done with any combination of two static casts, ignoring the uniqueness and virtual inheritance restrictions on static downcasts. This would also make the above example ill-formed.

(3) Change both dynamic_cast and static_cast; see below.

I had also forgotten (and was somewhat dismayed to rediscover) that static_cast cannot be used to break protection. For example:

```
struct A { };
struct B : private A { };
void f() {
    B *b = new B;
    A *a1 = (A*) b;              // OK
    A *a2 = static_cast<A*>(b);  // ill-formed
    A *a3 = dynamic_cast<A*>(b); // well-formed,
                                 // but "a3" not usable
}
```

Did we really intend to do this, or was it an accidental side effect of defining static_cast in terms of the inverse of an implicit cast?

Also, I see no reason to restrict downcasting across private inheritance. If static_cast were changed to allow it, I would consider the "across private inheritance" part to be implicit, and the "downcasting" part to be the one that required an explicit cast.

In that light, I would propose one of these changes to dynamic_cast:

(1) Remove the first "public" from paragraph 8 and also allow downcasting to the most derived class, regardless of access.

(2) The equivalent of (2) above:

A dynamic cast is well-formed if there exists a class X within the most derived object hierarchy (including the most derived class) such that:

-- "v" refers to X or a base class of X; and

-- T is X or a public base class of X.

That is, a dynamic cast is OK if it can be done with a combination of two static casts, ignoring the uniqueness and virtual inheritance restrictions on static downcasts. This would also make the above example ill-formed.

____

Similarly, should upcasting of pointers to members across private inheritance be restricted more than upcasting of pointers to members across public inheritance?

____

Resolution:
Requestor:
Owner: Bill Gibbons (RTTI)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 645
Title: Should &*(array+upperbound) be allowed?
Section: 5.3.1 [expr.unary.op]
Status: active
Description:

para 1:
"The unary * operator performs indirection: the expression to which it is applied shall be a pointer to an object type or a pointer to function type and the result is an lvalue referring to the object or function to which the expression points."

```
int a[4];
... *(a+4) ...
```

The problem is that a+4 does not point to an object.
Is it ill-formed to apply the * operator to such an expression?

Resolution:
Requestor: Mike Miller
Owner: Josee Lajoie (Memory Model)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 453
Title: Can operator new be called to allocate storage for temporaries, RTTI or exception handling?
Section: 5.3.4 [expr.new] New

Status: active

Description:

Is it permitted for an implementation to create temporaries on the heap rather than on the stack? If so, does that require that operator new() be accessible in the context in which such a temporary is created?

Is an implementation allowed to call a replaced operator new whenever it likes (storage for RTTI, exception handling, initializing static in a library)?

Core 1 discussed this issue in Monterey.
This is the resolution the WG seemed to converge towards:

> The storage for variables with static storage duration, for data structures used for RTTI and exception handling cannot be acquired with operator new.
>
> global operator new/delete (either the user-defined ones or the implementation-supplied ones) will only be called from new/delete expressions and by the functions in the library.

Proposed Resolution:

The C standard says the following:
See 6.1.2.4 (storage durations of objects):

- o For objects of static storage duration:
  "For such an object, the storage is reserved ... prior to program start up.
  The C++ standard should probably say something like this in section 3.7.1 [basic.stc.stc].

- o For objects of automatic storage duration:
  "Storage is guaranteed to be reserved for a new instance of such an object on each normal entry into a block with which it is associated, or on a jump from outside the block to a labeled statement in the block or in an enclosed block. Storage for the object is no longer guaranteed to be reserved when execution of the block ends in any way. (Entering an enclosed block suspends but does not end execution of the exclosing block. Calling a function suspends but does not end execution of the block containing the call."
  The C++ standard should probably say something like this in section 3.7.2 [basic.stc.auto].

The C++ standard should also indicate the following restrictions:

> 12.2 [class.temporary] should probably indicate that the storage for temporaries is not allocated by operator new.
>
> 5.2.6[expr.dynamic.cast], 5.2.7[expr.typeid] and 15[except] should probably indicate that the storage for the data structures required for RTTI and exception handling is not allocated by operator new.

Resolution:
Requestor: Mike Miller
Owner: Josee Lajoie (Memory Model)
Emails:
core-5068
Papers:

. . . .

Work Group: Core
Issue Number: 577
Title: Are there any requirements on the alignment of the pointer used with new with placement?
Section: 5.3.4 [expr.new] New
Status: active

Description:

For example, 12.4 para 10 gives examples of placement new used with

```
        a buffer created as follows:
                class X { };
                static char buf[sizeof(X)];
        Is the alignment of a static array of char guaranteed to satisfy the
        alignment requirements of an arbitrary class X?
Resolution:
Requestor:      public comment T26
Owner:          Josee Lajoie (Memory Model)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   637
Title:          How is operator delete looked up if the constructor from a
                new with placement throws an exception?
Section:        5.3.4 [expr.new] New
Status:         active
Description:
        paragraph 18 says:
        "If the constructor exits using an exception and the new-expression
         contains a new-placement, a name lookup is performed on the name
         of operator delete in the scope of this new-expression."

        Jerry Schwarz says:
        > That doesn’t seem right.  I think I should be able to write
        >     struct X {
        >         void* operator new(size_t, void*);
        >         void operator delete(void*);
        >         void operator delete(void*, void*);
        >         X();
        >     };
        >     X* p;
        >     ... new(p)X; // uses X::operator new
        >                  // if X::X() throws an exception, storage should
        >                  // be deallocated by X::operator delete.
Resolution:
Requestor:      Jerry Schwarz
Owner:          Josee Lajoie (Memory Model)
Emails:
        core-6418
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   638
Title:          Accesibility of ctor/dtor, operator new and operator delete
Section:        5.3.4 [expr.new] New
Status:         active
Description:
        struct A {
                void * operator new(size_t);
                void operator delete(void *);
                virtual ~A();
        };
        struct B {
                void * operator new(size_t);
                void operator delete(void *);
                virtual ~B();
        };
        struct D : public A, public B {
                void *operator new(size_t);
                virtual ~D();
        };
        main() {
                A *pa = new D;
                delete pa; // A::operator delete() or B::operator delete()?
        }
```

When is it detected that operator delete is ambiguous?
When struct D is defined?
When the new expression is encountered?
Is the behavior undefined if new happens to throw an exception?

Similar questions for the accessibility of the destructor / operator delete.

Does it make a difference if a new with placement is used?
Does it make a difference if a new nothrow is used?
If new[] is used?

Resolution:
Requestor: Mike Anderson
Owner: Josee Lajoie (Memory Model)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 660
Title: Conversions allowed by C style casts are too broad
Section: 5.4 [expr.cast]
Status: active
Description:

Para 5:

"The conversions performed by static_cast (_expr.static.cast_), reinterpret_cast (_expr.reinterpret.cast_), const_cast (_expr.const.cast_), or any sequence thereof, can be performed using the cast notation of explicit type conversion."

I think this is too broad, as it makes this code well-formed:

```
struct A {
  operator int ();
};

const A a;

void f () {
  (void*)a; /* reinterpret_cast <void *>
               (static_cast <int> (const_cast <A&> (a))) */
}
```

Do people think that compilers should be required to handle this case?
How about the case where ’a’ is non-const (requiring only the first two new casts), or where the cast is to ’int’ (requiring only the latter two new casts)?

Resolution:
Requestor: Jason Merrill
Owner: Steve Adamczyk
Emails:
        core-6753
Papers:

. . . .

Work Group: Core
Issue Number: 644
Title: Must the operand of .* and ->* have a complete class type?
Section: 5.5 [expr.mptr.oper]
Status: active
Description:

Para 2:

"The binary operator .* binds its second operand, which shall be of type ‘‘pointer to member of T ’’ to its first operand, which shall be of class T or of a class of which T is an unambiguous and accessible base class."

And something similar in para 3 for the ->* operator.
Must T be a complete class type?
Can the pointer to member be of an incomplete class type?

Resolution:
Requestor: Jerry Schwarz
Owner: Bill Gibbons (Pointer to members)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 600
Title: Should the value returned by integer division and remainder be defined by the standard?
Section: 5.6 [expr.mul]
Status: active
Description:

ISO Swedish comment R-26:
Division of negative integers [expr.mul] Paragraph 4: The value returned by the integer division and remainder operations shall be defined by the standard, and not be implementation defined. The rounding should be towards minus infinity. E.g., the value of the C expression (-7)/2 should be defined to be -4, not implementation defined. This way the following useful equalities hold (when there is no overflow, nor "division by zero "):

(i+m*n)/n == (i/n) + m for all integer values m

(i+m*n)%n == (i%n) for all integer values m

These useful equalities do not hold when rounding is towards zero. If towards 0 is desired, it can easily be defined in terms of the round towards minus infinity variety, whereas the other way around is trickier and much more error-prone.

Resolution:
Requestor: Swedish Delegation
Owner: Tom Plum (C Compatibility)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 493
Title: Better description of the cv-qualification of the result of a relational operator needed
Section: 5.9 [expr.rel] Relational Operators
Status: active
Description:

5.9p2 says:
"Pointer conversions are performed on the pointer operands to bring them to the same type, which shall be a cv-qualified or cv-unqualified version of the type of one of the operands."

This seems to imply that the result has exactly the type of one of the operands, or an unqualified version of that type. In fact, the common type may have more qualifiers than either operand type.

[Note JL:
for example the following is allowed in C:

```
  const int* pci;
  const volatile* pvi;
  if (pci == pvi) { }
```

]

Proposed Resolution:
Resolution:
Requestor: Bill Gibbons
Owner: Steve Adamczyk (Type Conversions)

```
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   513
Title:          Are pointer conversions implementation-defined or
                unspecified?
Section:        5.9 [expr.rel] Relational Operators
Status:         active
Description:
        5.9p2 last '--' says:
        "Other pointer comparisons are unspecified."

        Andrew Koenig notes the following:
         Saying it is unspecified is a tremendous difference from C.  The
         point is that in C on, say, the Intel 386 in 16-bit mode, when doing
         an ordering comparison it is sufficient for the compiler to generate
         code to compare only the low-order 16 bits of the pointers because
         the comparison is defined only for two elements of the same array.
         If C++ is required to compare the whole address, that puts it at a
         significant performance disadvantage with respect to C.
Resolution:
Requestor:      Erwin Unruh
Owner:          Josee Lajoie (Memory Model)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   537
Title:          Can the implementation accept other constant expressions?
Section:        5.19 [expr.const] Constant expressions
Status:         active
Description:
        The C standard says, in its section on constant expressions:
        "An implementation may accept other forms of constant expressions."
        Should C++ say the same thing?

        In particular, implementations often accept extended forms of
        constant expressions in order to support 'offsetof', defined as
        returning an 'integral constant expression'. Are implementations
        prohibited to accept other forms of 'integral constant expressions',
        expressions which the WP does not describe as constant expressions?

        If, in C++, implementations are not allowed to extend the set of
        constant expressions, then the C compatibility appendix should list
        this as an incompatibility.
Resolution:
Requestor:      Dave Hendricksen
Owner:          Tom Plum (C Compatibility)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
==============================================================================
 Chapter 6 - Statements
------------------------
Work Group:     Core
Issue Number:   645
Title:          When is the result of an expression statement converted to an
                rvalue?
Section:        6.2 [stmt.expr]
Status:         active
Description:
        class C;
        extern C& f();
        void foo() {
                f(); //1
        }
```

Is line //1 ill-formed because the return value of f() is converted to an rvalue and C is an incomplete class type?

Resolution:
Requestor:
Owner: Steve Adamczyk (Type Conversions)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 639
Title: What is the lifetime of declarations in conditions
Section: 6.4 [stmt.select]
Status: active
Description:

```
        > struct T { T(int); ~T(); operator bool() const; /*...*/ };
        >
        > void f(int i)
        >    {
        >    while (T t = i) { /* do something with ’t’ */ }
        >    }
        >
        > How often is t constructed/destroyed?

        Another example:
          for ( T *p = first;
                T *next = p->next();
                p = next )
              { p->val = 1; }

        Solution 1:
          each time the loop is entered/exited.
        Solution 2:
          only once, making the loop equivalent to:
          {
          T t = i;
          while (t) { /* do something with ’t’ */ }
          }
```

Resolution:
Requestor: Jerry Schwarz
Owner: Josee Lajoie (Object Model)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 635
Title: local static variable initialization and recursive function calls
Section: 6.7 [stmt.dcl]
Status: active
Description:

```
        int foo(int i) {
                if (i == 0) return i;
                static int x ( foo (i-1) );
                return x;
        }
        ... foo (10) ...
```

What is the value of x after it has been initialized?

Resolution:
Requestor: Neal M Gafter
Owner: Josee Lajoie (Initialization)
Emails:
Papers:

. . . .

===

```
Chapter 7 - Declarations
--------------------------
Work Group:     Core
Issue Number:   646
Title:          Can a using declaration refer to a hidden base class member?
Section:        7.3.3 [namespace.udecl]
Status:         active
Description:
        struct A {
                typedef int T;
        };
        struct B : A {
        protected:
                typedef double T;
        };
        struct C : B {
                using A::T;
        };
        Is the using declaration above well-formed?
Resolution:
Requestor:
Owner:          Steve Adamczyk (Name look up)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   650
Title:          How does name look up proceed for the name in a using
                declaration?
Section:        7.3.3 [namespace.udecl]
Status:         active
Description:
        namespace A {
          class X { };
          void X();
        }

        void func() {
          using A::X; //1
          X();        // calls function A::X
          struct X x; // declares x to have type A::X ???
        }

        Are the class name A::X and the function name A::X both made visible
        by the using declaration on line //1?
Resolution:
Requestor:      Mike Miller
Owner:          Steve Adamczyk (Name look up)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   612
Title:          name look up and unnamed namespace members
Section:        7.3.4 [namespace.udir]
Status:         active
Description:
        paragraph 5 says:
        "If name look up finds a declaration for a name in two different
         namespaces, and the declarations do not declare the same entity
         and do not declare functions, the use of the name is ill-formed."

        Consider the program:

           struct S { };
           static int S;
```

```
int foo() { return sizeof(S); }
```

The sizeof will resolve to the static int S, because nontypes are favored.

The standard says that unnamed namespaces will deprecate the use of static so we should be able to rewrite the program as:

```
struct S { };
namespace {
   int S;
}
int foo() { return sizeof(S); }
```

However, the sizeof becomes ambiguous according to 7.3.4 para 5 because the two S are from different namespaces. Is this right? Doesn’t this mean that static should not be deprecated?

Resolution:

Requestor:

Owner: Steve Adamczyk (Name Look up)

Emails:

Papers:

. . . .

Work Group: Core

Issue Number: 78 (also WMM.38)

Title: Linkage specification and calling protocol

Section: 7.5 [dcl.link] Linkage Specifications

Status: active

Description:

```
extern "C" {
        // Typedef defined in extern "C" blocks:
        // What is the linkage of the function pointed at by ’fp’?
        typedef int (*fp)(int);

        // Type of a function parameter:
        // What is the linkage of the function pointed at by ’fp2’?
        int f(int (*fp2) (int));

        // Can function with C linkage be defined in extern "C"
        // blocks?
        int f2(int i) { return i; }

        // Can static function with C linkage be defined in
        // extern "C" blocks?
        static int f3(int i) { return i; }
}
```

If function declarations/definitions placed inside the extern "C" block have different properties from the ones placed outside these blocks, many areas of the C++ language will have to be aware of difference.

i.e.

a. function overloading resolution

b. casting

one will need to be able to cast from a pointer to a function with linkage "X" to a pointer to a function with linkage "Y".

In short, it needs to be determined to what extent the linkage is part of the type system.

[JL:]

The standard should not force implementations to accept the following code:

```
extern "SomeLinkage" int (*ptr)();
int (*ptr_CXX)();
ptr_CXX = ptr; // 1
```

i.e. an implementation should be able to issue an error for line (// 1).

See 95-0122/N0722 for a proposed resolution.

Core 1 discussed this issue in Monterey. The consensus the group seemed to converge towards was to leave it implementation defined whether or not the linkage specification is part of the type.

Resolution:
Requestor: John Armstrong (johna@kurz-ai.com)
Owner: Josee Lajoie (Linkage)
Emails:
core-1583, core-1584, core-1585, core-1586, core-1587, core-1589
core-1590, core-1591, core-1594, core-1595, core-1597, core-1598
core-1599, core-1608, core-1609, core-1612
core-920 (Hansen),core-985 (O'Riordan),core-1064 (Miller)
Papers: 94-0034/N0421

. . . .

Work Group: Core Language
Issue Number: 420
Title: Linkage of C++ entities declared within 'extern "C"'.
Section: 7.5 [dcl.link] Linkage Specification
Status: active
Description:

Given a declaration or definition of some C++ entity (e.g. a data member, a function member, and overloaded operator, an anonymous union object, etc) whose existance within an otherwise standard conforming program written in ANSI/ISO C would be a violation of the language rules, what is the effect of the linkage specification on the declarations/definitions of the C++ specific entities?

Example:

```
extern "C" {
        struct S {
                int data_member;
        };
        int operator+ (S&, int);
}
```

Resolution:
Requestor: Ron Guilmette
Owner: Josee Lajoie (Linkage)
Emails:
Papers:

. . . .

Work Group: Core Language
Issue Number: 616
Title: Can the definition for an extern "C" function be provided in two different namespaces?
Section: 7.5 [dcl.link] Linkage Specification
Status: active
Description:

Is the following compilation unit valid?

```
   namespace A { extern "C" int f() { return 1; } }
   namespace B { extern "C" int f() { return 2; } }
```

In other words, have I defined two different functions with the signature "f()" (valid), or have I provided two definitions for the same function (invalid)?

I don't find an answer to the question in the draft.

[...]

From the library implementation viewpoint, it would be nice if a non-C++ linkage specification meant that the namespace name was in some sense an "optional" part of the function's name:

```
  extern "C" void f() { } // A::f() and B::f() refer to this function
```

But we still want this property:

```
   namespace A { extern "C" void f(); }
   void foo() {
     f(); // error, f undeclared
   }
   void bar() {
     using A::f;
     f(); // ok
   }
```

The extern "C" function f can be defined in any namespace or outside all namespaces; there can be only one definition.

That is, the extern "C" affects the linkage of the name in such a way as to ignore the namespace name, but does not affect the scope of the name in the C++ source program.

Also:
That solution leaves open the problem of global variables in the C library. A typical implementation of errno is to make it a global int:

```
         namespace std { extern int errno; }
```

How can this be the same object as the errno in the C library? (An add-on C++ implementation does not have the option of replacing the C library.)

I suggest we give extern "C" for data the same effect on the name as for functions. We would then write

```
         namespace std { extern "C" int errno; }
         ...
         std::errno = 0; // sets the errno in the C library
```

Resolution:
Requestor: Steve Clamage
Owner: Josee Lajoie (Linkage)
Emails:
        core-6303
Papers:

. . . .

==

Chapter 8 - Declarators

Work Group: Core
Issue Number: 636
Title: Can a typedef-name be used to declare an operator function?
Section: 8.3 [dc.meaning]
Status: active
Description:

```
        typedef int I;
        struct S {
                operator I(); // Is this allowed?
        };
```

Resolution:
Requestor:
Owner: Steve Adamczyk (Name Look Up)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 531
Title: Is a default argument a context that requires a value?
Section: 8.3.6 [dcl.fct.default] Default arguments
status: active
Description:

```
        extern struct A a_default;
        extern struct B b_default;
        struct A {
                void f(B = b_default); //1
        };
        struct B {
                void f(A = a_default);
        };
        A a_default;
        B b_default;
        inline void A::f(B b) { /* ... */ }
        inline void B::f(A a) { /* ... */ }
```

Is this valid code?
Is the default value only needed if and when the function is called with less than the full number of arguments?

Proposed Resolution:

para 9 says:
"Default arguments are evaluated at each point of call before entry into a function."

The lvalue-to-rvalue conversion happens when a default argument expression is evaluated. Therefore, the type of a default argument expression does not have to be complete until the lvalue-to-rvalue conversion takes place, that is until the function is called.
So the declaration of A::f on line //1 above is well-formed.

To make this clear, the following could be added to the WP:
"The lvalue to rvalue conversion on a default argument expression takes place at the point of call."

Resolution
Requestor: Fergus Henderson
Owner: Steve Adamczyk (Default Arguments)
Emails:
        core-5884
Papers:

. . . .

Work Group: Core
Issue Number: 640
Title: default arguments and using declarations
Section: 8.3.6 [dcl.fct.default] Default arguments
status: active

Description:

para 9:
"When a declaration of a function is introduced by way of a using declaration (7.3.3), any default argument information associated with the declaration is imported as well."

Box 17:
Can additional default arguments be added to the function thereafter by way of redeclarations of the function?

```
        namespace N {
                void f(int, int);
        }
        using N::f;

        extern int a;
        void f(int, int = a); // Is this well-formed?

        // Where is the default argument useable?
        void g() {
                f(16); //1: ok?
        }

        namespace N {
                void g() {
                        f(16); //2: ok?
                }
```

```
}
```

Can the function be redeclared in the namespace with added default arguments, and if so, are those added arguments visible to those who have imported the function via using?

```
namespace N {
        void f(int, int);
}
using N::f;

namespace N {
        int a;
        void f(int, int = a);
}

// Where is the default argument useable?
void g() {
        f(16); //3 ok?
}
```

Proposed Resolution:

A using declaration is a declaration.
When a function is introduced by a using declaration, the accumulated set of default arguments associated with the function in the original namespace is imported into the scope where the using declaration appears. After this, the two declarations are treated as separate declarations.

Default arguments added to the function by way of redeclarations in the scope of the using declaration are not reflected into the declaration in the original namespace.
That is, line //1 above is ok.
Line //2 is ill-formed because the declarations for f in namespace N do not specify any default arguments.

Default arguments added to the function by way of redeclarations in the original namespace are not reflected into the using declarations for that function.
That is, line //3 is ill-formed because the declarations for f in global scope do not specify any default arguments.

This seems to follow the model already in the WP for additional declarations in the original namespace following a using declaration, see 7.3.3[namespace.udecl] para 8.

Resolution:
Requestor:
Owner: Steve Adamczyk (Default Arguments)
Emails:
Papers:

Chapter 9 - Classes

Work Group: Core
Issue Number: 505
Title: Must anonymous unions declared in unnamed namespaces also be declared static?
Section: 9.5 [class.union] Unions
Status: active
Description:

9.5p3 says:
"Anonymous unions declared at namespace scope shall be declared static."
Must anonymous unions declared in unnamed namespaces also be declared static?
If the use of static is deprecated, this doesn’t make much sense.

Proposal:
Replace the sentence above with the following:
"Anonymous unions declared in a named namespace or in the global namespace shall be declared static."

This is related to issue 526.

Resolution:
Requestor: Bill Gibbons
Owner: Josee Lajoie (linkage)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 655
Title: When is storing into another union member ill-formed?
Section: 9.5 [class.union] Unions
Status: active
Description:

Here is a program which is ill-formed in ISO C, but I cannot find any wording in the C++ working paper which would make it ill-formed in C++:

```
union {
        struct A {
                double w;
                long double x;
        } a;
        struct B {
                long double y;
                double z;
        } b;
} u;

int main() {
        u.b.y = 0.0;
        u.a.x = u.b.y;
}
```

ISO C disallows this because of the overlap. Since the lvalue => rvalue conversion of u.b.y occurs before u.a.x is modified, this code would appear to be valid C++.

If the members were aggregate instead of scalar types, this would be implicitly ill-formed. For example:

```
struct tag { int x[1000]; int y[1000] };

union {
        struct A {
                struct tag w;
                long double x;
        } a;
        struct B {
                long double y;
                struct tag z;
        } b;
} u;
```

Once the first array element is copied, the entire union member from which it came becomes invalid - because something has been stored into another union member. So the usage is already ill-formed for aggregates.

But what about scalars? In the original example the source and destination overlap, but does the execution model say that an entire

```
        scalar is fetched from memory before the store begins?
        Or should C++ have the same restriction on overlap as ISO C?
Resolution:
Requestor:      Bill Gibbons
Owner:          Josee Lajoie (Object Model)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   47
Title:          enum bitfields - can they be declared with < or > bits than
                required
Section:        9.6 [class.bit] Bitfields
Status:         active
Description:
        enum ee { one, two, three, four };
        struct S {
                ee bit:1;    //1: allowed?
                ee bit:64;   //2: allowed?
                char bit:64; //3: allowed?
        };

        ANSI C says the following:
        "The expression that specifies the width of a bit-field shall ...
         not exceed the number of bits in an object of compatible type."

        Shouldn’t C++ say something similar?
Proposed Resolution:
        Possible Solutions:

        1) minimum length:
        ------------------
          o solution 1:
            Impose a minimum length.
            "The width of a bit-field shall be sufficient to hold all of the
             values of the bit-field’s type."
            This makes line //1 above ill-formed.
          o solution 2:
            Impose no minimum length.
            In C, a bit-field can be declared with fewer bits than what is
            necessary to hold the values of an object of compatible type.

          o proposed resolution:
            -------------------
            solution 2.
            This is common practice.

        2) maximum length:
        ------------------
          o solution 1:
            Impose a maximum length.
            "The width of a bit-field shall not exceed the number of bits in
             an object of the same type."
            This makes lines //2 and //3 above ill-formed.
          o solution 2:
            Impose no maximum length.

          o proposed resolution:
            -------------------
            At the Santa Cruz meeting, folks preferred solution 2.
            Folks believed that imposing a limit on the width of a bit-field
            was not necessary. Yes, if the width of a bit-field is greater
            than the width of an object of the same type, the value stored in
            the bit-field will be truncated when it is fetched out of the
            bit-field. Folks believed this was something users should be
            aware of. Folks believed that the language should not prevent
            users from declaring a bit-field with a width greater than the
            width of an object of the same type if they wanted to.
Resolution:
Requestor:      ?
Owner:          Steve Adamczyk (Types)
Emails:
        core-1578
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
=============================================================================
 Chapter 10 - Derived classes
------------------------------
Work Group:     Core
Issue Number:   624
Title:          class with direct and indirect class of the same type: how
                can the base class members be referred to?
Sections:       10.1 [class.mi] Multiple base classes
Status:         active
Description:
        para 3 says:
        "[Note: a class can be an indirect base class more than once and can
         be a direct and indirect base class.]"
        The WP should describe how base class members can be referred to,
        how conversion to the base class type is performed, how
        initialization of these base class subobjects takes place.
Resolution:
Requestor:
Owner:          Josee Lajoie (Object Model)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   446
Title:          Can explicit qualification be used for base class navigation?
Sections:       10.1 [class.mi] Multiple base classes
Status:         active
Description:
        Can explicit qualification be used for base class sublattice
        navigation?

        class A {
        public:
          int i;
        };
        class B : public A { };
        class C : public B { };
        class D {
        public:
          int i;
        };
        class E : public D { };
        class F : public E { };
        class Z : public C, public F { };
        Z z;
        ... z.F::E::D::i; // is qualification allowed here to navigate the
                          // base class sublattice?
Resolution:
Requestor:      Bill Gibbons
Owner:          Steve Adamczyk (Name Look up)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
=============================================================================
 Chapter 11 - Member Access Control
------------------------------------
Work Group:     Core
```

Issue Number: 656
Title: access of names used in base clauses
Section: 11.4[class.friend]
Status: active
Description:

```
        class A;
        class T1 {
                friend class A;
                class T2 { };
        };

        class A : T1::T2 {           //1: can T1::T2 be used here?
                class B : T1::T2 {  //2: how about here?
                };
        };
```

Proposed Resolution:

Either //1 or //2 is ill-formed:
either:
//1 is ill-formed:

Since the base-clause of class A (i.e., the befriended class) is not part of the declarations for the members of A, the private members of the class granting friendship cannot be used in the base-clause of A.

or:
//2 is ill-formed:

Access for names in the base-clause of a class is checked in the same way as access for names referred to in the member functions of the class. In this case, since A::B is not a friend of class T1, the base clause for A::B cannot access T1::T2, a private member of T1.

I prefer solution 1).

Resolution:
Requestor:
Owner: Steve Adamczyk (Access Specifications)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 653
Title: What does it mean for nested classes if a class-name is inserted into the scope of the class itself?
Section: 11.8[class.access.nest]
Status: active
Description:

9[class] para 2 says:
"The class-name is also inserted into the scope of the class itself. For purposes of access checking, the inserted class name is treated as if it were a public member name."

Given:

```
        class A {
                class B {
                        class C {
                                B* pb1;        //1 legal?
                                A::B pb2;      //2 legal?
                        };
                };
        };
```

Because class name B is inserted as a public member name in the scope of its class, does this mean that C can refer to B even though B is a private member of A? Is the answer different if B is referred to as A::B?

Proposed Resolution:

Because B is inserted in its own class scope as a public member, accessing B from the scope of a nested class is well-formed eventhough B is a private member of its enclosing class.

I believe the answer should be the same whether B is referenced just as "B" or whether it is referenced as a qualified name "A::B".

11.8[class.access.nest] should probably say something like this:
"Because a class name is inserted in its own class scope as a public member (_class_), accessing the class-name from the scope of a nested class is well-formed even if the class is a private member of its enclosing class."

Resolution:
Requestor:
Owner: Steve Adamczyk (Access Specifications)
Emails:
Papers:

. . . .

==

Chapter 12 - Special Member functions

Work Group: Core
Issue Number: 598
Title: Should a diagnostic be required if an rvalue is used in a ctor-initializer or in a return stmt to initialize a reference?
Section: 12.2 [class.temporary]
Status: active
Description:

12.2p5:
"A temporary bound to a reference in a constructor’s ctor-initializer (12.6.2) persists until the constructor exits. ...
A temporary bound in a function retrun statement (6.6.3) persits until the function exits."

This actually means that there is no reliable way to initialize a reference member or a return value of reference type with an rvalue expression. Given that, a diagnostic should be required.

Resolution:
Requestor: Tom Plum
Owner: Josee Lajoie (Object Model)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 138 (WMM.89)
Title: When are default ctor default args evaluated for array elements?
Section: 12.6 [class.init] Initialization
Status: active
Description:

From Mike Miller’s list of issues.
WMM.89. Are default constructor arguments evaluated for each element of an array or just once for the entire array?

```
        int count = 0;
        class T {
                int i;
        public:
                T ( int j = count++ ) : i ( j ) {}
                ~T () { printf ( "%d,%d\n", i, count ); }
        };
        T arrayOfTs[ 4 ];
```

Should this produce the output :-

```
        0,4
        1,4
        2,4
        3,4
        or should it produce :-
                0,1
                0,1
                0,1
                0,1
Proposed Resolution:
        8.3.6[dcl.fct.default] para 9 says:
        "Default arguments are evaluated at each point of call before the
         entry into a function."
        This should also be true if the function call is implicit.
        That is, the test case above should produce the first output
        suggested above.

        Para 9 should be clarified to say that it also applies to functions
        that are implicitly called.
Resolution:
Requestor:      Mike Miller / Martin O'Riordan
Owner:          Steve Adamczyk (Declarators)
Emails:
        core-668
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   626
Title:          What is the form of the implicitly-declared operator= if a
                base class has Base::operator=(B)?
Section:        12.8 [class.copy]
Status:         closed
Description:
        What is the form of the implicitly-declared operator= if the class
        has a base class that has a copy assignment operator that does not
        take a reference parameter, i.e.
                Base::operator=(Base)
        ?
        para 10 does not clearly mention this.
Resolution:
        This was handled editorially in the pre-Stockholm version of the WP.
        Such class gets a copy assignment operator of the form:
                Derived::operator=(const Derived &)
Requestor:
Owner:          Josee Lajoie (Object Model)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   536
Title:          When can objects be eliminated (optimized away)?
Section:        12.8 [class.copy]
Status:         active
Description:
        Paragraph 15 indicates that an implementation is allowed to eliminate
        an object if it is created with the copy of another.

        ISSUE 1:
        --------
        However, this is in clear contradiction with other WP text:
        3.7.1[basic.stc.static] says:
          "If an object of static storage duration has initialization or a
           destructor with side effects; it shall not be eliminated even if
           it appears to be unused."

        3.7.2[basic.stc.automatic] says:
          "If a named automatic objects has initialization or a destructor
           with side effects; it shall not be destroyed before the end of its
           block, nor shall it be eliminated as an optimization even if
           appears to be unused."
```

So which is right?

Many have suggested different ways to resolve this difference:

Andrew Koenig [core-5975]:

> The correct way to resolve the contradiction is to say that copy optimization applies only to local objects.

Patrick Smith [core-6083]:

1) Just weaken 3.7.1 and 3.7.2 so they can be overridden by the copy constructor optimization.

2) Restrict the copy constructor optimization to only eliminate temporaries representing function return values.

3) Require the programmer to explicitly mark the classes for which the copy constructor optimization is permitted even though it would violate 3.7.1 or 3.7.2.

4) Require the programmer to explicitly mark the classes for which the copy constructor optimization is not permitted when it would violate 3.7.1 or 3.7.2.

ISSUE 2:

Jerry Schwarz in core-5993:

> What may be of concern is not side effects in general, but resource allocation. E.g. if Thing is intended to obtain a lock that is held until it is destroyed, then you do indeed have to be careful about the semantics you give to the copy constructor.

```
    {
        Thing outer ; // get the lock
        {
            Thing inner = outer ; // copy constructor increments
                                  // count on lock.

            // do stuff that requires the lock
            inner.release() ;  // decrement count
            // do stuff that doesn’t require the lock
        }

        // do stuff that still requires the lock.
    }
```

> The optimization allows outer and inner to be aliased, and the explicit release in inner may cause the lock to be released too early.

Is Jerry’s concern worth worrying about?

Two possible resolutions were proposed:

Jerry suggested the following:

> When we introduced the "explicit" keyword I remember considering what it would mean on copy constructors and thinking about the possibility that it would suppress this optimization.

Jason Merrill proposed in c++std-core-5978:

> Perhaps the language in class.copy should be modified so that it only applies when the end of one object’s lifetime coincide with the beginning of its copy’s lifetime.

Resolution:
Requestor: John Skaller

Owner: Josee Lajoie (Object Model)
Emails:
Papers:

. . . .

===

Chapter 13 - Overloading

Work Group: Core
Issue Number: 652
Title: Is a derived-to-base conversion required to be implemented by a copy constructor of the base class?
Section: 13.3.3.1 [over.best.ics]
Status: active
Description:

Is a derived-to-base conversion required to be implemented by a copy constructor of the base class? Or is it always the best constructor of the base class that’s used?
i.e., which constructor is called in the following example:

```
class B;
class D;

class B {
public:
        B( const B& );    // #1 - a copy constructor
        B( const D& );    // #2 - a different constructor
};

class D: public B { };

class Q {
public:
        operator D ();
};

void func1(B);

void func2() {
        D d;
        Q q;

        B b( d );         // case 1: #1 or #2?
        B b2 = d;         // case 2: #1 or #2?

        func1( d );       // case 3: #1 or #2?
}
```

Case 1 is direct initialization, so presumably all constructors are considered, thus #2 is the one that is used.

For case 2, 8.5[dcl.init] paragraph 12, 4th bullet, 2nd sub-bullet would appear to apply, in which case both #1 & #2 are considered, so #2 is used.

Case 3 should be the same as case 2, but 13.3.3.1.2 [over.ics.user] paragraph 4 says:

> "A conversion of an expression of class type to the same class type is given Exact Match rank, and a conversion of an expression to a base class of that type is given Coversion rank in spite of the fact that a copy constructor (i.e., a user-defined conversion function) is called for those cases."

This paragraph makes the assumption that the only way to perform such a conversion is by copy constructor, but constructor #2 can also perform this conversion.

Proposed Resolution:

1) Require that in all cases where a class is being initialized by a

derived class, the copy-constructors are the only ones considered, i.e. in the example above, all cases would resolve to #1.

2) In all places where a copy-constructor is called for, all constructors of the target class are actually considered, i.e. change the phrase "a copy-constructor is called" to "a constructor is called to copy ...". The one selected by overload resolution is the one used, even if that use does not include calling it (eg. in cases of elimination of temporaries). In the above example, this would resolve all cases to #2. The special status of ’copy-constructor’ then only affects whether one is implicitly generated (and what its signature is).

Ben has a slight preference for option #2.

Resolution:

Requestor: Ben Schreiber

Owner: Steve Adamczyk (Type Conversions)

Emails:

core-6667

Papers:

. . . .

Work Group: Core

Issue Number: 658

Title: Should declarations for binary built-in operators only accept operands of the same type?

Section: 13.6 [over.built]

Status: active

Description:

Currently prototypes for the built-in operators accept operands of different types. For example, the signed integral subset of the "operator+" prototypes is:

```
int operator+(int, int);
long operator+(int, long);
long operator+(long, int);
long operator+(long, long);
```

Some examples argue strongly for another model:

* The operators only take operands of the same type (so the conversions are implied as part of calling the operators).

Consider:

```
 struct A {
     operator int();
     operator long();
 };

 void f(A a) {
     a + 0;   // ill-formed
}
```

This is ambiguous: the two builtin functions

```
int operator+(int, int);
long operator+(long, int);
```

are both equally good matches, and so overload resolution fails.

Somewhat more surprisingly:

```
struct A {
    operator int();
    operator long();
};
```

```
    void f(A a) {
        int x = a;   // ill-formed
   }
```

This is also ambiguous; the relevant prototypes are:

```
    int& operator=(int&, int);
    int& operator=(int&, long)
```

Proposed Resolution:

There are several options here:

(1) Do nothing. This leads to very surprising ambiguity errors, especially with assignment.

(2) Change the prototypes for assignment so that they require the operands to have the same type. This makes assignment well-behaved at the cost of inconsistency with the other operators; and the first example remains counter-intuitive.

(3) Change all the prototypes. This makes both examples intuitive. It is also more consistent with the rules in clause 5 (by one interpretation).

Bill Strongly favors (3).

Resolution:
Requestor: Bill Gibbons
Owner: Steve Adamczyk (Type Conversions)
Emails:
        core-6704
Papers:

. . . .

Work Group: Core
Issue Number: 659
Title: Should the prototypes for built-in operators properly take into account arithmetic conversions?
Section: 13.6 [over.built]
Status: active
Description:

Consider:

```
int f(int, int);
long f(long, long);
void g() {
   f(3, 4L);  // ambiguous - an existing problem
}

int operator+(int, int);  // proposed prototypes
int operator+(long, long);
void g() {
   3 + 4L;    // ambiguous under existing overloading rules
}
```

This problem occurs because arithmetic conversions break a key design principle of conversions:

The inverse of a standard conversion is normally *not* a standard conversion.

This is true for everything except the arithmetic conversions. And that exception pretty much breaks overloading for arithmetic parameters.

In the first example above, the fact that "long" => "int" is a standard conversion makes the first function callable, which leads to the ambiguity.

Proposed Resolution:

Several possible ways to improve the current rules:

- * Change the prototypes of all the operators (issue 658), and change the overloading rules so that when calling builtin arithmetic operators, conversions which go forwards in the sequence (long double, double, float, unsigned long, long, unsigned int, int) are not considered, plus the special case that "unsigned int" => "long" is only considered if it is value-preserving.

- * Change the prototypes of all the operators (issue 658), and change the overloading rules so that if any call is found to be ambiguous, it is reconsidered with the above restrictions.

- * Deprecate the "long" => "int" and related standard conversions, so that there is some hope of fixing this in the next revision of the standard.

Bill likes the second option best.

Resolution:
Requestor: Bill Gibbons
Owner: Steve Adamczyk (Type Conversions)
Emails:
        core-6710
Papers:

. . . .

===

Chapter 15 - Exception Handling

Work Group: Core
Issue Number: 647
Title: Is it implementation-defined or unspecified how the memory for the exception object is allocated?
Section: 15.1 [except.throw]
Status: active
Description:

para 4:
"The memory for the temporary copy of the exception being thrown is allocated in an implementation-defined way."

Shouldn’t this say "unspecified".
Must implementations document how memory is allocated?

Resolution:
Requestor:
Owner: Bill Gibbons (exceptions)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 541
Title: Is a function-try-block allowed for the function main?
Section: 15.3 [except.handle] Handling an exception
Status: active
Description:

I assume the new syntax that allows for function-try-block is also allowed if the function is main:

```
        main()
        try {
        }
        catch (...) { }
```

What is the effect of the catch(...) in main if the constructor for an object with static storage duration throws an exception (and the constructor does not catch the exception)?

Because the WP does not dictate a precise moment for the construction of objects with static storage duration (these objects can be constructed at any time before the first statement in main or...), is it implementation-defined whether the handler in main catch an exception thrown from a constructor for a global static object? Or is the catch in main guaranteed to catch (or guaranteed not to catch) such an exception?

Resolution:

This following tentative resolution was adopted by the Core III WG at the Santa Cruz meeting and it will be presented to the committee for a vote at the Stockholm meeting:

Function try-blocks are allowed on main(). But static ctors & dtors are logically executed before main() is entered and after main() exits, so exceptions thrown by static ctors/dtors are not caught. This implies a slight wording change in the description of static ctors/dtors.

Requestor:
Owner: Bill Gibbons (exceptions)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 542
Title: What exception can a reference to a pointer to base catch?
Section: 15.3 [except.handle] Handling an exception
Status: active
Description:

15.3 says:

> A handler with type T, const T, T&, or const T& is a match for a throw-expression with an object of type E if
> ...
> [3] T is a pointer type and E is a pointer type that can be converted to T by a standard conversion.

This allows code like this:

```
struct A { };
struct B { };
struct D : A, B { };
D d;

try {
        D* pd = new D;
        throw pd;
}
catch (B*& pb) {// OK, B*& is a valid handler
                // for a throw of type D*
}
```

However, code equivalent to this outside of the exception handling try/catch mechanism is disallowed, i.e.

```
        B*& pb = new D; // error
```

The current language rules (8.5.3) require that the reference be of const type for this initialization to be valid. i.e.

```
        B* const & pb = new D; // OK
```

preventing the pointer referred to by the reference from being modified with the value of a pointer of a different type.

Going back to the original example with EH, 15.3 allows someone to write code as follows in the handler, code which modifies the original exception thrown:

```
        catch (B*& pb) {
                pb = new B;
        }
```

Allowing this doesn’t seem to make much sense to me because if the program ever tries to refer to the original exception thrown as a D* after the assignment to pb has taken place (using a rethrow, for example) undefined behavior is almost guaranteed to take place i.e. the exception of type D* has become an object of type B* and the type system has been completely bypassed.

I believe 15.3 should say that a handler with type T& is _not_ a match for a throw-expression with an object of type E if T and E are pointer types that are not of the same types.

There may be other adjustments needed as well to make 15.3 mimic more closely the rules on reference initialization.

Resolution:
Core III agreed with the proposed resolution at the Santa Cruz meeting. This will be presented for a vote at the Stockholm meeting.

Requestor:
Owner: Bill Gibbons (exceptions)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 587
Title: Can a pointer/reference to an incomplete type appear in a catch clause?
Section: 15.3 [except.handle] Handling an exception
Status: active

Description:
15.3/1 says:
"The exception-declaration [in a catch clause] shall not denote an incomplete type."

This comes from 92-120/N0197 issue 3.3:
"No, an incomplete type can not appear in a catch clause.

A pointer or reference to an incomplete type may appear in a catch clause, however."

Should pointers and references to incomplete types also be disallowed in catch clauses?

The resolution of issue 3.3 (and the related requirement that incomplete types be allowed in exception specifications) place unreasonable constraints on implementations.

In particular, they force implementations to handle exceptions by matching the *names* of classes. This is because it is not possible to generate type information for an incomplete class. Since the class need not ever be complete, an implementation may not rely on type information generated in another translation unit; rather, it must associate the incomplete type with the appropriate type information by searching for the type name.

Is the need for pointers/references to incomplete types in catch clauses sufficient to justify these kinds of restrictions on the implementations? And similarly, is the need for incomplete types in exception specifications of function definitions sufficient to justify these restrictions?

Resolution:
Core III is leaning towards requiring complete types.
This will be brought up for a vote at the Stockholm meeting.

Requestor: Bill Gibbons

```
Owner:          Bill Gibbons (exceptions)
Emails:
        ext-3367
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   648
Title:          Is it implementation-defined or unspecified whether the stack
                is unwound before terminate is called?
Section:        15.3 [except.handle] Handling an exception
Status:         active
Description:
        para 8:
        "Whether or not the stack is unwound before calling terminate() is
         implementation-defined."

        Shouldn’t this say "unspecified".
        Must implementations document which one happens first?
Resolution:
Requestor:
Owner:          Bill Gibbons (exceptions)
Emails:
Papers:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Work Group:     Core
Issue Number:   588
Title:          How can exception specifications be checked at compile time
                if the class type is incomplete?
Section:        15.4 [except.spec]
Status:         active
Description:
        Issue 1:
        --------
        struct A;
        struct B;
        void f() throw(A);
        void g() throw(B) { f(); }

        Because A and B have incomplete type, static checking isn’t possible
        because it can’t be determined if B is derived from A.

        [Mike Ball, ext-3386]:
        "Having these types incomplete here essentially obviates strong
         signature checking, which some of our customers have stated very
         strongly that they want.

         I think that requiring complete types in a throw specification will
         not produce the dependencies people are assuming.  From what I have
         seen, types thrown tend to be from a rather small set of classes
         especially designed to be thrown as exceptions.  This means that
         requiring that they be complete would probably not have cascading
         effects.  That is, it might pull in the headers defining the
         exception class hierarchy, but probably not a whole lot else."

        [Andrew Koenig, ext-3387]:
        "As with function argument types, I think it should be OK to use an
         incomplete type in an exception specification:

            struct A;
            void f() throw(A);

         as long as you complete it

            struct A { };

         before calling or defining the function:
```

```
    void g() { f(); }
```

Issue 2:
----
paragraph 2 says:
"If a virtual function has an exception-specification, all declarations, including the definition, of any function that overrides that virtual function in any derived class shall have an exception-specification at least as restrictive as that in the base class."

What does "shall" mean if incomplete types are used?
Incomplete types make it impossible to determine if the clause is adhered to.

[John Skaller, ext-3379]:
"A reasonable interpretation is that an incomplete type B ’is not as restrictive as’ a type A and so this ought to require a diagnostic. My argument -- you can complete B later to be anything you want, so the throw spec of B doesn’t exhibit a restriction, as required.

[Mike Ball, ext-3380]:
"One could also argue that it could also be checked at the definition point of the overriding function, at which point it would certainly be no burden on the programmer to require that the type be complete."

Resolution:
Requestor: John Skaller
Owner: Bill Gibbons (exceptions)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 630
Title: What is the exception specification of implicitly declared special member functions?
Section: 15.4 [except.spec]
Status: active
Description:

The following program is ill-formed with the present WP:

```
    class exception {
    public:
            virtual ~exception() throw();
    };
    class logic_error : public exception {
    };
```

Unfortunately it occurs in the WP itself.

The reason for it being ill-formed is that class logic_error gets an implicitly declared destructor. This destructor gets the usual exception specification, namely none, which may throw anything. This violates the constrain that a virtual function in the derived class must have an exception specification at least as restrictive as that of the base class.

Proposed Resolution:

The possibilities I see at the moment are:

1. always "throw anything"
2. union of exception specification of base functions
3. intersection of exception specification of base functions
4. union of exception specification of base and member functions
5. intersection of exception specification of base and member functions

The simplest solution is 1. This means any user having a virtual destructor with an exception specification must add a destructor declaration in each derived class (this includes the std library).

A more relaxed and save solution would be 4. Then the exception specification of the generated function would never be violated, but it would be convenient when being in single inheritance. This would also match the usual rules for inheriting. When you do not declare an overriding function in a derived class, the exception specification of the base function will be kept. With option 4 this would also (almost) hold for the implicitly declared functions.

The versions 2, 3 and 5 would lead to situations, where the exception specification of a generated function is violated. I would see this as not acceptable.

Resolution:
Mike Anderson will prepare a paper for the pre-Stockholm mailing.
Requestor: Erwin Unruh
Owner: Bill Gibbons (exceptions)
Emails:
core-6398
Papers:

. . . .

Work Group: Core
Issue Number: 631
Title: Must the exception specification on a function declaration match the exception specification on the function definition?
Section: 15.4 [except.spec]
Status: active
Description:
para 2 says:
"If any declaration in any translation unit of a program of a function has an exception-specification, all declarations including the definition, of that function shall have an exception specification with the same set of type-ids."

para 5 says:
"Calling a function through a declaration whose exception specification is less restrictive than that of the function's definition is ill-formed."

First, this is contradictory. Must the declarations be the same or can some declarations be less restrictive than the definition?

Second, shouldn't the behaviour be undefined, not ill-formed with no diagnostic required (para5)? I don't understand how runtime behaviour can cause the program to become ill-formed. How can a program be either ill-formed or well-formed depending on its input?

Resolution:
Requestor: Fergus Henderson
Owner: Bill Gibbons (exceptions)
Emails:
core-6391, core-6401
Papers:

. . . .

Work Group: Core
Issue Number: 657
Title: Must the exception-specification of a declaration be more or less restrictive than the exception-specification of the definition?
Section: 15.4 [except.spec]
Status: active
Description:
paragraph 5 says:

"Calling a function through a declaration whose exception-specification allows other exceptions than those allowed by the exception-specification of the function’s definition is ill-formed. No diagnostic is required."

This seems inconsistent with the rules for virtual functions and assignment to function pointers where such situations would make the program ill-formed.

Proposed Resolution:

Change the wording above to:

"Calling a function that has a definition specifying an exception-specification that allows other exceptions than those allowed by the exception-specification of the function declaration visible at the point of call is ill-formed. No diagnostic is required."

Resolution:
Requestor: Patrick Smith
Owner: Bill Gibbons (exceptions)
Emails:
core-6521
Papers:

. . . .

Work Group: Core
Issue Number: 649
Title: Should it be mandated that terminate be called upon internal error?
Section: 15.5.1 [except.terminate]
Description:

The WP states that one of the situations in which terminate() is called is:

- when the implementation’s exception handling mechanism encounters some internal error

Should this requirement be removed?

This was discussed briefly at a Core-3 session in Santa Cruz, and general opinion was that this requirement should be removed, since an internal error condition already implies undefined behavior. Most implementations would chose to call abort() in this situation rather than terminate(), since there’s no guarantee that terminate() will be able to do anything useful without running into the same internal error condition.

The ARM’s original wording for this situation was

- when the exception handling mechanism finds the stack corrupted

which suggests trying to deal with a user-caused error rather than an implementation error, but it’s still undefined behavior.

The ARM wording stayed in the WP until the April 95 version, when it changed to its current form. The change doesn’t seem to be traceable to anything in the pre- or post-Austin mailings, but the fact that it was changed rather than removed suggests that someone thought it was worthwhile. Is there a rationale for keeping it?

Resolution:
Requestor: Jonathan Schilling
Owner: Bill Gibbons (exceptions)
Emails:
Papers:

. . . .

Work Group: Core
Issue Number: 651
Title: Is unexpected called before the stack is partly unwound?
Section: 15.5.2 [except.unexpected]

Description:

```
        int i = 0;

        void my_unexpected(void)
        {
                i = 1;
                throw char('a');
        }

        class A {
                ~A() { i = 2; }
        };

        void f(void) throw (char)
        {
                std::set_unexpected(my_unexpected);
                A a;
                throw int(1);
        }
```

The question is: in which order are a.~A() and my_unexpected called. The answer will effect whether i has the value 1 or 2 after calling f.

Proposed Resolution:

Possible Solutions:

- the stack is not unwound, so i becomes 2. This would mean that the search for a handler which includes the checks for exception specifications must precede the stack unwinding. Core III has avoided to make such an asumption to allow an implementation to fold handler-search with stack-unwinding. This option is not viable.
- the stack is unwound, so i becomes 1. For this option, the exact place of where the stack unwinding stops must be specified. A rule of thumb would be:
  The destructors whose exception would be caught by the exception specification are executed.
- it is implementation defined, but the result must be either 1 or 2. This means the implementation must choose one of the solutions above.
- it is unspecified or undefined. I don't like this solution since a call to unexpected can be solved accurately. Having a part of undefined behaviour would make this completely unreliable. We should avoid unspecified behaviour in this case.

Erwin prefers (and proposes) that the stack be unwound, but can live with it being implementation-defined.

Resolution:
Requestor: Erwin Unruh
Owner: Bill Gibbons (exceptions)
Emails:
core-6485
Papers:

Chapter 16 - Preprocessing Directives

Work Group: Core Language
Issue Number: 661
Title: Should __DATE__ and __TIME__ be made locale aware?
Section: 16.8 [cpp.predefined]
Status: active
Description:

The description for the __DATE__ and __TIME__ macros indicate that their values use the English format for date and time.
Should the value of the macros be made locale specific?

Resolution:

Requestor:
Owner: Tom Plum (C Compatibility)
Emails:
Papers:
. . . .