

Doc No: X3J16/96-0084 WG21/N0902
Date: March 29th, 1996
Project: Programming Language C++
Ref Doc:
Reply to: Josee Lajoie
(josee@vnet.ibm.com)

+=====
| Core WG List of Issues |
+=====

The issues listed as closed in the version of the Core WG list of issues that appeared in the Pre-Santa Cruz mailing (96-0044/N0862) were removed from the Core WG list of issues and are therefore not listed in this version of the list.

The issues listed as editorial in this version of the list were categorized as editorial by the Core WG at the Santa Cruz meeting and will be handled as editorial by the editorial team helping the editor.

The issues listed as closed in this version of the list were resolved and voted on at the Santa Cruz meeting and the motions from the Santa Cruz meeting indicate the wording that will be added to the WP to resolve these issues.

+-----+
| C Compatibility |
+-----+

- 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?

+-----+
| 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

+-----+
| Core I |
+-----+

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

- 5.3.4 [expr.new]:
 - 453: Can operator new be called to allocate storage for temporaries, RTTI or exception handling?
 - 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
- 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)?
 - 626: What is the form of the implicitly-declared operator= if a base class has Base::operator=(B)?

```
+-----+
| Core II |
+-----+
```

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

- 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?

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

- 4.9 [conv.fpint]:
 - 617: Are floating point conversions unspecified or implementation-defined?
- 4.13 [conv.bool]:
 - 601: Should implicit conversion from int to bool be allowed?
- 5.9 [expr.rel]:
 - 493: Better description of the cv-qualification for the result of a relational operator needed

+-----+
| Core III |
+-----+

Exception Handling

- 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?
- 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?

+-----+
| Core Editorial |
+-----+

- 2.3 [lex.pptoken]:
 - 620: The non-terminal "header-name" is not defined
- 3 [basic]:
 - 460: Definition for the term "variable"
- 3.7.3 [basic.stc.dynamic]:
 - 546: What is the required behavior for a user allocator?
- 3.9 [basic.life]:
 - 608: Is an incompletely-defined object type an object type?
- 5.2.6 [expr.dynamic.cast]:
 - 549: Is a dynamic_cast from a private base allowed?
- 5.2.9 [expr.reinterpret.cast]:
 - 486: Can a value of enumeration type be converted to pointer type?
- 5.2.9 [expr.reinterpret.cast]:
 - 559: Are pointer-to-derived -> pointer-to-base conversions performed with

- a reinterpret_cast?
- 5.2.10 [expr.const.cast]:
 - 622: Definition for "multi-level pointers" needed
 - 577: Are there any requirements on the alignment of the pointer used with new with placement?
- 5.3.5 [expr.delete]:
 - 470: Deleting a pointer allocated by a new with placement
- 5.5 [expr.mptr.oper]:
 - 488: Can a pointer to a mutable member be used to modify a const class object?
- 5.18 [expr.comma]:
 - 609: Is "bitfield" an attribute remembered when used as the right of comma operator?
- 5.19 [expr.const]:
 - 610: Is a string literal considered a constant expression for the purpose of non-local static initialization?
- 7 [dcl.dcl]:
 - 213: Should vacuous type declarations be prohibited?
- 7.1.5 [dcl.type]:
 - 564: is 'void f(const a);' well-formed?
- 8.3.6 [dcl.fct.default] :
 - 530: Can default arguments appear in out-of-line member function definitions?
 - 586: When do access restrictions apply to default argument names?
- 9 [class]:
 - 627: What does it mean for the class name to be inserted as a public member name?
- 9.6 [class.bit]:
 - 267: What does "Nor are there any references to bitfields" mean?
 - 571: Is bitfield part of the type?
- 10[class.derived]:
 - 441: In which scope is the base class clause looked up access checked?
- 11 [class.access]:
 - 585: Is access checking performed on the qualified-id of a member declarator?
- 11.3 [class.access.dcl]:
 - 388: Access Declarations and qualified ids
- 11.4 [class.friend]:
 - 515: How can friend classes use private and protected names?
 - 532: Is a complete class definition allowed in a friend declaration?
 - 625: Can a friend function be declared "inline friend"?
- 12.4 [class.dtor]:
 - 293: Clarify the meaning of y.~Y
- 13.6 [over.built]:
 - 582: What are the cv-qualifiers for the parameters of a candidate function?
 - 583: For a candidate built-in operator, must cv-qualifiers of parameters of type pointer to member be the same?
- 15.1[except.throw]:
 - 628: Default argument on copy constructors & construction of exceptions
- 15.2 [except.ctor]:
 - 594: If a constructor throws an exception, in which cases is the storage for the object deallocated?
- 15.3 [except.handle]:
 - 540: How does name look up proceed in a function-try-block?
 - 590: With function try blocks, does the caller or callee catches exceptions from constructors/destructors called for parms?
 - 592: Can a type be defined in a catch handler?
- 15.4 [except.spec]:
 - 629: What does it mean for an exception-specification to be as restrictive as another exception-specification?
- 16.3 [cpp.replace]:
 - 632: Does redefining a macro make the program ill-formed or undefined behavior?

+-----+

- 2 [lex]:
 - 606: The description of the compilation model needs work
- 2.1 [lex.phases]:
 - 584: May a // comment end with an EOF instead of a newline?
- 2.9.3 [lex.fcon]:
 - 506: Is a program containing a non-representable floating point constant ill-formed?
- 3.9 [basic.types]:
 - 192: Should a typedef be defined for the type with strictest alignment?
- 4.12 [conv.class]:
 - 547: Semantics of standard conversion "derived to base" need better description
- 5.1 [expr.prim]:
 - 512: ambiguity when parsing destructors calls
 - 433: What is the syntax for explicit destructor calls?
 - 465: grammar needed to support template function call
 - 466: grammar needed to support ~int()
- 5.2.4 [expr.ref]:
 - 452a: How does name look up work after . or -> for namespace names or template names?
- 5.2.8 [expr.static.cast]:
 - 550b: Can a static_cast perform a conversion from an rvalue of base class type to an rvalue of derived class type?
- 5.2.9 [expr.reinterpret.cast]:
 - 538: Are user-defined conversions invoked as the result of a reinterpret_cast?
- 5.3 [expr.unary]:
 - 593: syntax for prefix ++ operator
- 5.16 [expr.cond]:
 - 496: The cv-qualification of the result of the conditional operator needs better description
- 5.18 [expr.comma]:
 - 618: syntax ambiguity between expression-list and comma expression
- 6.8 [stmt.ambig]:
 - 424: Must disambiguation update symbol tables?
- 7.1.5 [dcl.type]:
 - 116: Is "const class X { };" legal?
- 7.2 [dcl.enum]:
 - 503: Better semantics of bitfields of enumeration type needed
- 8.3.5 [dcl.fct]:
 - 567: Can a parameter have type 'T arr[]' where T is incomplete?
- 9 [class]:
 - 568: Can a POD class have a static member of type pointer-to-member, non-POD-struct or non-POD-union?
- 9.1 [class.name]:
 - 252: Can the definition of an incomplete class appear in an anonymous union?
- 9.5 [class.union]:
 - 266: Access specifiers in union member list
 - 105: How can static members which are anon unions be initialized?
 - 570: Name look up for anonymous union member names need to be better described.
- 9.6 [class.bit]:
 - 623: Representation of bitfields of bool type
 - 458: When is an enum bitfield signed / unsigned?
- 13.3 [over.match]:
 - 614: Is a complete type needed for function overload resolution?
- 13.3.3.2 [over.ics.rank]:
 - 599: Are user-defined conversion sequences always ambiguous when the user-defined conversions considered are different?
- 15.2 [except.ctor]:
 - 611: What happens when an exception is thrown from the destructor of a subobject?

15.3 [except.handle]:

539: Can one throw a pointer-to-member to a base class and catch it with a handler taking a pointer to a derived class?

16.8 [cpp.predefined]:

595: Is a macro `__STDC_plusplus__` needed?

=====
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++ implementation 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] exists 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: 606
Title: The description of the compilation model needs work
Section: 2.1 [lex.phases]
Status: closed
Description:
UK issues 19.
Interaction of templates with phases of translation needs to be specified.

Resolution:
See Santa Cruz motion 21).
Requestor: UK issues 19
Owner: Tom Plum (Lexical Conventions)
Emails:
Papers:

.....
Work Group: Core
Issue Number: 584
Title: May a // comment end with an EOF instead of a newline?
Section: 2.1 [lex.phases]
Status: closed
Description:
2.1 [lex.phases], 1st paragraph, third bullet, does not clearly answer this question.

Resolution:
No, a // comment must not end with an EOF instead of a newline.
See bullet 2.
Requestor: Mike Holly
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 definitional issues regarding 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
Emails:
Papers:

Work Group: Core
Issue Number: 620
Title: The non-terminal "header-name" is not defined
Section: 2.3 [lex.pptoken]
Status: editorial

Description:
The non-terminal "header-name" is not defined.

Requestor:
Owner: Tom Plum (Lexical Conventions)
Emails:
Papers:

Work Group: Core
Issue Number: 506
Title: Is a program containing a non-representable floating point constant ill-formed?
Section: 2.9.3 [lex.fcon]
Status: closed

Description:
2.9.1 [lex.icon] p3 says:
"A program is ill-formed if it contains an integer literal that cannot be represented by any of the allowed types."
For consistency with 2.9.1, shouldn't a program containing a non-representable floating point constant be ill-formed? (if the exponent is too large, for example?)

Resolution:
See Santa Cruz motion 22).

Requestor: Erwin Unruh
Owner: Tom Plum
Emails:
Papers:

Chapter 3 - Basic Concepts
Work Group: Core

Issue Number: 460
Title: Definition for the term "variable"
Section: 3 [basic] Basic concepts
Status: editorial
Description:

Editorial Box 5:
The definition for the term variable is needed.

Proposed Resolution:
"A variable is introduced by an object's declaration and the variable's name denotes the object."

Also UK issue 334.

Resolution:
Requestor:
Owner: Clark Nelson (Object Model)
Emails:
Papers:

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 diagnostic 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 desirable.

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?"

Note: Jim Welch will write a paper on this topic for the Scotts Valley meeting.

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.

Note: Jim Welch will write a paper on this topic for the Scotts Valley meeting.

Resolution:

Requestor: comment T25 (3.8)
Owner: Josee Lajoie (ODR)
Emails:
Papers:

95-0205/N0805

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).

At the Monterey meeting, Mike Anderson promised to present a paper at the Tokyo meeting with a proposed resolution.

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;

```

or:

```

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

```

Proposed Resolution:

Neal proposes that translation unit 1 (t1.cc) be made undefined by adding a rule to C++ analagous to the C rule quoted above. The C++ rule will have to take namespaces into account.

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: 546
 Title: What is the required behavior for a user allocator?
 Section: 3.7.3 [basic.stc.dynamic]
 Status: editorial
 Description:

3.7.3 [basic.stc.dynamic] para 3 says:
 "Any allocation and/or deallocation functions defined in a C++ program shall conform to the semantics specified in this subclause."
 3.7.3.1 [basic.stc.dynamic.allocation] para 2 says:
 "Each such allocation shall yield a pointer to storage (`_intro.memory_`) disjoint from any other currently allocated storage."

Does "currently" mean at the time of the call to the allocation function, or at the time it returns? If the latter, how can a user-defined allocation function return a pointer to storage that is

disjoint from any other currently allocated storage? Even if the former interpretation is correct, the above two rules would rule out all of the most useful ways of defining operator new - at least one of those rules must be changed.

Erwin Unruh suggests in core-6228 that this requirements belongs to the library clause that describes the requirements on the allocation functions provided by the standard library.

Resolution:

This will be handled in an editorial manner, along with Erwin Unruh's paper 96-0011/N0829.

Requestor: Fergus Henderson
Owner: Josee Lajoie (Memory Model)
Emails: core-6170
Papers:

Work Group: Core
Issue Number: 192
Title: Should a typedef be defined for the type with strictest alignment?
Section: 3.9 [basic.types] Types
Status: closed

Description:

It would be useful if <new.h> provided a typedef for a name such as `__strict_align_t`, to describe a type whose alignment is the strictest required in this environment. It is otherwise hard to write a portable overloaded new operator. Faking it, by defining a union of several "typical" types, is not really portable, and its quiet mode of failure might be extremely puzzling, because the program would run just fine most of the time in most environments, except that in some unusual environment the program would occasionally produce an alignment error.

As WG14 and X3J11 have found out, some compilers add an alignment requirement for structures embedded inside structures, one which is even more restrictive than the scalar types!
There are no real-world guarantees about alignment, unless the committee imposes them.

ALTERNATIVE: The committee could prescribe specific requirements for alignment. E.g., in any conforming environment, no object may have an alignment requirement more restrictive than this specific type:
`struct __strict_align_t { struct { long n; double d; }; };`

92/12/07 NOTE: To allow the writing of portable allocators, it may also be necessary to define an `__align_pointer(p)` function, which returns the nearest pointer (address) value which is aligned on the strictest boundary and is greater than or equal to the pointer value `p`.

Resolution:

This is a request for an extension.
We are too late in the standards process to be accepting extensions.

Requestor: Tom Plum / Dan Saks
Owner: Josee Lajoie (Memory Model)
Emails:
Papers:

Work Group: Core
Issue Number: 608
Title: Is an incompletely-defined object type an object type?
Section: 3.9 [basic.types]
Status: editorial

Description:

paragraph 6:
"The term incompletely-defined object type is a synonym for `imcomplete type`; the term completely-defined object type is a

synonym for complete type."

UK issue 400:

"In ISO 9899 an incomplete type is not an object type (Clause 6.1.2.5, first paragraph). Defining an "incompletely-defined object type" is a needless incompatibility with ISO 9899. Use another term.

Requestor: UK issue 400
Owner: Steve Adamczyk (Types)
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:

=====
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: active

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: 547
Title: Semantics of standard conversion derived to base need better description
Section: 4.12 [conv.class]
Status: closed

Description:
4.12 [conv.class] says:
"An rvalue of type "cv D", where D is a class type, can be converted to an rvalue of type "cv B", where B is a base class of D. If B is an inaccessible or ambiguous base class of D or if the conversion is implemented by calling a constructor and the constructor is not callable, a program that necessitate this conversion is ill-formed."

Isn't the copy constructor always called to convert an rvalue of a derived class type to an rvalue of base class type? If so, I don't

understand the phrase "..._if_ the conversion is implemented by calling a constructor...". Since all classes have a copy constructor (either user-declared or implicitly-declared), I would assume that, at least conceptually, a copy constructor is always used.

Also, the conversion is described as converting from "cv D" to "cv B". I don't believe it is accurate to say that the cv-qualifiers are always the same. Don't the cv-qualifiers on D depend on the cv-qualifiers acceptable for the copy constructor's 1st parameter and aren't the cv-qualifiers on B independent of the cv-qualifiers specified on the source type of the conversion?

Resolution:

The base to derived standard conversion was removed.
See Santa Cruz motion 14).

Requestor:

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: 512

Title: ambiguity when parsing destructors calls

Section: 5.1 [expr.prim] Primary expressions

Status: closed

Description:

5.1p7 says:
"A class-name prefix by ~ denotes a destructor."

There is a syntactic ambiguity on the usage of a destructor. The code '~X();' in the scope of a member function of class X can be interpreted as an explicit destructor call using the implicit this pointer. The other interpretation is the unary operator ~ applied to a function like cast.

Resolution:

See Santa Cruz motion 11).

Requestor: Erwin Unruh

Owner: Anthony Scian (Syntax)

Emails:

Papers:

.....

Work Group: Core
Issue Number: 433
Title: What is the syntax for explicit destructor calls?
Section: 5.1 [expr.prim] Primary expressions
12.4 [class.dtor] Destructors
Status: closed

Description:

Question 1:

p10 says:

The notation for explicit call of a destructor may be used for any simple type name. For example:

```
int* p;  
p->int::~~int();
```

Must the destructor name be a qualified-id or can it be written as:

```
p->~int();
```

?

Question 2:

Can a typedef name be used following the ~, and if so, what are the lookup rules?

```
struct A {  
    ~A(){}  
};
```

typedef class A B;

```
int main()  
{  
    A* ap;  
    ap->A::~~A(); // OK  
    ap->B::~~B(); // cfront/Borland OK, IBM/Microsoft/EDG error  
    ap->A::~~B(); // cfront OK, Borland/IBM/Microsoft/EDG error  
    ap->~B(); // OK?  
}
```

This issue concerns the lookup of explicit destructor calls for nonclass types as well.

```
typedef int I;  
typedef int I2;  
int* i;  
i->int::~~int();  
i->I::~~I();  
i->int::~~I();  
i->I::~~int();  
i->I::~~I2();
```

Which of these are well formed?

Resolution:

See text in 12.4 para 11 and Santa Cruz motion 3).

Requestor: John H. Spicer
Owner: Steve Adamczyk (Name Lookup)
Emails:
Papers:

.....

Work Group: Core
Issue Number: 465
Title: grammar needed to support template function call
Section: 5.1 [expr.prim] Primary expression
Status: closed

Description:

"id-expression" does not allow the syntax

f<arg>
needed for a call to a template function using explicit arguments.

Possible solution:

Add template-function-id (i.e. production for f<>) to the list of unqualified-ids:

```
unqualified-id:
    ...
    template-function-id
```

Resolution:

```
unqualified-id:
    template-id
```

where:

```
template-id:
    template-name < template-argument-list >
```

already allows the use of references to template functions using explicit arguments.

Requestor:

Owner: Anthony Scian (Syntax)
Emails:
Papers:

Work Group: Core

Issue Number: 466

Title: grammar needed to support ~int()

Section: 5.1 [expr.prim] Primary expression

Status: active

Description:

The grammar does not allow for explicit destructor calls for built-in types:

```
int* pi;
pi->~int();
```

Possible solution:

```
unqualified-id:
    ...
    ~enum-name
    ~typedef-name
    ~simple-type-specifier
```

Resolution:

See Santa Cruz motion 3).

Requestor:

Owner: Anthony Scian (Syntax)
Emails:
Papers:

Work Group: Core

Issue Number: 452a

Title: How does name look up work after . or -> for namespace names or template names?

Section: 5.2.4 [expr.ref] Class member access

Status: closed

Description:

5.2.4 says p3:

"If the nested-name-specifier of the qualified-id specifies a namespace name, the name is looked in the context in which the entire postfix-expression occurs."

This is backward. One doesn't know if the name is a namespace name until the name has been looked up. In which scope must the name following the . or -> operator be first looked up?

```

namespace N { }
struct S {
    class N { };
};
S s;

... s.N::b ...

```

The scope of the object-expression 's' or the scope in which the entire expression takes place?

Neal Gafter also asks:
 "In the syntax

```

    p->template T<args>::x

    in which scope(s) is T looked up?"

```

```

template <class X> class T { static X x; };

class C {
    template <class X> class T { static X x; };
};

C* p;
p->template T<args>::x

```

Resolution:

See Santa Cruz motion 2).

Requestor:

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

.....

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

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.

Resolution:

Requestor:

Owner: Bill Gibbons (RTTI)
 Emails:
 Papers:

.....

Work Group: Core
 Issue Number: 550b
 Title: Can a static_cast perform a conversion from an rvalue of

base class type to an rvalue of derived class type?
Section: 5.2.8[expr.static.cast]
Status: closed
Description:

paragraph 6 says:
"The inverse of any standard conversion, other than ... can be performed explicitly using a static_cast..."

The 'other than' list does not list the conversion from an rvalue of base class type to rvalue of derived class type.
It either should or the semantics of this cast should be described in 5.2.8, specially given that an implicit conversion from an rvalue of derived class type to an rvalue of base class type involves calling the base class copy constructor.

Resolution:
The base class rvalue conversion was removed from the WP.
See Santa Cruz motion 14).

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

Work Group: Core
Issue Number: 486
Title: Can a value of enumeration type be converted to pointer type?
Section: 5.2.9 [expr.reinterpret.cast]
Status: editorial

Description:
5.2.9 p5 says:
"A value of integral type can be explicitly converted to pointer type."
Can a value of enumeration type be explicitly converted to pointer type?

Resolution:
This is a substantive change to which the Core WG agreed to during the Thursday session of the Tokyo meeting.
Add to the sentence above:
"... of integral type or enumeration type..."

Requestor: Bill Gibbons
Owner: Steve Adamczyk (Type Conversions)
Emails:
Papers:

Work Group: Core
Issue Number: 538
Title: Are user-defined conversions invoked as the result of a reinterpret_cast?
Section: 5.2.9 [expr.reinterpret.cast]
Status: closed

Description:
struct A {
operator void* ();
} a;

main() {
int i = reinterpret_cast<int>(a);
}

Is A::operator void* invoked as the result of the reinterpret_cast?
Resolution:
The sentence that says:
"Implicit type conversions are done whenever necessary"
was removed from the WP.
See Santa Cruz motion 16).

Requestor: Jason Merrill
Owner: Steve Adamczyk (Type conversions)

Emails:
core-5913, core-5939 and following messages.

Papers:
.....

Work Group: Core
Issue Number: 559
Title: Are pointer-to-derived -> pointer-to-base conversions performed with a reinterpret_cast?
Section: 5.2.9 [expr.reinterpret.cast]
Status: editorial

Description:
paragraph 6 says:
"The operand of a pointer cast can be an rvalue of type 'pointer to incomplete class type'. The destination type of a pointer cast can be 'pointer to incomplete class type'. In such cases, if there is any inheritance relationship between the source and the destination classes, the behavior is undefined."

This paragraph should be deleted. It is misleading. With reinterpret_cast, there are never any pointer value adjustments, even when the pointers point to class types with an inheritance relationship. So there is nothing special when pointers to incomplete class types are operands of a reinterpret_cast.

Resolution:
At the Tokyo meeting, the core WG decided to handle this as an editorial matter.
Here is Steve Adamczyk's proposed resolution:
Move the paragraph to 5.4p4, as part of the description of the old-st cast, with a description something like "In such cases, if there is any inheritance relationship between the source and destination classes, it is unspecified whether the static_cast or reinterpret_cast interpretation is used." Also make it clear in 5.2.8 that at the point of a static_cast the class types must be complete.

Requestor:
Owner: Steve Adamczyk (Type conversions)
Emails:
Papers:
.....

Work Group: Core
Issue Number: 622
Title: Definition for "multi-level pointers" needed
Section: 5.2.10 [expr.const.cast]
Status: editorial

Description:
para 9 says:
"For multi-level pointers to data members, or multi-level mixed object and member pointers, ..."
These two terms are not defined in the WP.

Resolution:
Requestor:
Owner: Steve Adamczyk (Type conversions)
Emails:
Papers:
.....

Work Group: Core
Issue Number: 593
Title: syntax for prefix ++ operator
Section: 5.3 [expr.unary]
Status: closed

Description:
The grammar indicates:
unary-expression ::= ++ unary-expression
This seems to make things like ++(int&)x ill-formed.

Proposed Resolution:
unary-expression ::= ++ cast-expression

Resolution:

The proposed resolution was accepted.
See Santa Cruz motion 10).

Requestor: Jerry Schwarz
Owner: Anthony Scian
Emails:
core-6231

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 enclosing 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.

I will write a paper for the Santa Cruz meeting.

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: editorial

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:

This will be handled in an editorial manner, along with Erwin Unruh's
paper 96-0011/N0829.

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*);  
>     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: 470
 Title: deleting a pointer allocated by a new with placement
 Section: 5.3.5 [expr.delete] Delete
 Status: editorial

Description:

5.3.5 p2 says:
 "... in the first alternative (delete object), the value of the
 operand of delete shall be a pointer to a non-array object created
 by a new-expression without a new-placement specification, ..."

In some situations, it is well-defined what happens even when new
 with placement was called. Do we want to prohibit these cases?

Erwin Unruh also notes:

The deletion of a pointer gained by a placement new must be allowed.
 Using the default operator delete for a pointer gained by the library
 placement new is undefined. However, a user may write placement news
 that allocate storage in which case using delete on a pointer
 returned by such a placement new should be well-defined.

Proposed Resolution:

Replace 5.3.5[expr.delete] p2 to say:
 "... in the first alternative (delete object), the value of the
 operand of delete shall be a pointer to a non-array object created
 by a new-expression, ... In the second alternative (delete
 array), the value of the operand of delete shall be a pointer to
 an array created by a new-expression. If not, the behavior is
 undefined. In either alternative, if the operand of the delete
 expression is a pointer to an object created by a new expression
 with a new-placement specification, and if the library operator
 new with placement was used to allocate the storage, the behavior

of the delete expression is undefined."

Erwin Unruh will provide a paper for the Santa Cruz meeting (March 1996).

Resolution:

This will be handled in an editorial manner, along with Erwin Unruh's paper 96-0011/N0829.

Requestor: Jason Merrill
Owner: Josee Lajoie (Memory Model)
Emails: core-5569, core-6227

Papers:

Work Group: Core
Issue Number: 488
Title: Can a pointer to a mutable member be used to modify a const class object?
Section: 5.5 [expr.mptr.oper]
Status: editorial

Description:

5.5 p4 says:
"The restrictions on cv-qualification, and the manner in which cv-qualifiers of the operands are combined to produce the cv-qualifiers of the result, are the same as the rules for E1.E2..."

It should be noted that a pointer to member that refers to a mutable member cannot be used to modify a const class object.

```
struct S {
    mutable int i;
};
const S cs;
int S::* pm = &S::i;
cs.*pm = 88;
```

Proposed Resolution:

Add a note at the end of p4:
"Note: a pointer to member that refers to a mutable member cannot be used to modify a member of an object of const class type."

Resolution:

Requestor: Bill Gibbons
Owner: Bill Gibbons (pointer to member)
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 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:

Steve Adamczyk will write a paper on cv-qualifiers and operand types to be available for the Santa Cruz meeting (March 96).

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 implementation-defined."

Comparison of unrelated pointers should be unspecified or undefined. At present it reads implementation defined, but I doubt that the exact rules can be described by a compiler vendor.

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: 496
Title: The cv-qualification of the result of the conditional operator needs better description
Section: 5.16 [expr.cond] Conditional operator
Status: closed
Description:

5.16p3 says:
"...pointer conversions are performed on the pointer operands to bring them to a common type, which shall be a cv-qualified or cv-unqualified version of the type of either the second or the third expression.
...
if both the second and the third expressions are lvalues of related class types, they are converted to a common type (which shall be a cv-qualified or cv-unqualified version of the type of either the second or the third expression)..."

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

Also, does the phrase "same type" in paragraph 2 includes cv-qualifiers? That is, is the following well-formed?

```
const int i = 88;  
volatile int j = 99;  
const volatile *p = &((1) ? i : j);
```

Proposed Resolution:
See Santa Cruz motion 17).

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

.
Work Group: Core
Issue Number: 609
Title: Is "bitfield" an attribute remembered when used as the right operand of comma operator?
Section: 5.18 [expr.comma]
Status: editorial
Description:

```
Given:  
struct B {  
    unsigned bit:2;  
};  
B b;  
void f(int);  
void f(unsigned int);  
... f(((0, b.bit)+1)) ...
```

Is the bitfield attribute remembered when the type of the right hand expression becomes the resulting type of the comma expression? This will influence how the resulting type of the comma expression promotes.

Requestor:
See paper 96-0047/N0865 from the pre-Santa Cruz mailing.
Issue 571 was handled as an editorial issue.
The bitfield attribute is not part of the type.
If an lvalue-to-rvalue conversion is applied to an lvalue that refers to a bitfield, the result is a rvalue that is not a bitfield.
See 4.1 [_conv.lval_].

Owner: Steve Adamczyk (Type Conversions)

Emails:

Papers:

Work Group: Core

Issue Number: 618

Title: syntax ambiguity between expression-list and comma expression

Section: 5.18 [expr.comma]

Status: closed

Description:

The syntax given for expression-list (5.2) and the syntax given for the comma expression (5.18) are identical. A rule is needed to disambiguate the two cases.

Resolution:

This one is a completely incorrect statement. There is absolutely no ambiguity about what "kind" of comma a translator is dealing with given the current grammar. The grammar for an expression-list makes use of assignment-expression which cannot contain a non-nested comma-expression. Furthermore, the ISO C++ WP grammar is precisely identical in its treatment of commas as the ISO C grammar.

Requestor: UK issue 607

Owner: Anthony Scian (Syntax)

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:

Work Group: Core

Issue Number: 610

Title: Is a string literal considered a constant expression for the purpose of non-local static initialization?

Section: 5.19 [expr.const] Constant expressions

Status: editorial

Description:

In 5.19, paragraph 2 provides a list of expressions that can be used as constant expressions for the purpose of non-local static initialization (only). Should string literals be included in that list?

Or be in the list of expressions that can be used in an address constant expression (i.e. para 4)?

Resolution:

The WP will indicate that a string is a valid address constant

expression.

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

.....
=====

Chapter 6 - Statements

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?

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:

.....

Work Group: Core
Issue Number: 424
Title: Must disambiguation update symbol tables?
Section: 6.8 [stmt.ambig] Ambiguity resolution
Status: closed

Description:
The question is about the following sentence from 6.8p3 [stmt.ambig]

WP> The disambiguation is purely syntactic; that is, the meaning of
WP> the names, beyond whether they are type-ids or not, is not used
WP> in the disambiguation.

On the one hand, this would imply that a trial parser needn't update
a symbol table, since that would be processing that is not purely
syntactic.

On the other hand, some input would be disambiguated differently if
the symbol table were updated during trial parsing. Symbol table
updates would determine which names will be type-ids during the
actual parse.

To be more concrete and specific about the problem, consider the
statement in main() in the enclosed test case. Should this be
disambiguated as a declaration with a syntax error, or should it be
disambiguated as a well-formed expression?

```
struct T1
{
    T1 operator()(int x) { return T1(x); };
    int operator=(int x) { return x; };
    T1(int) {};
};
struct T2
{
    T2(int) {};
};
int a, (*(b)(T2))(int), c, d;
void main ()
{
    // Is the following a declaration with a syntax error?
    // Or is it a semantically valid expression?
    T1(a) = 3,
    T2(4),
    (*(b)(T2(c)))(int(d));
}
```

Resolution:

See Santa Cruz motion 12).

Requestor: Neal M Gafter <gafter@mri.com>

Owner: Anthony Scian (Syntax)

Emails:

Papers:

.....

=====
Chapter 7 - Declarations
=====

Work Group: Core

Issue Number: 213

Title: Should vacuous type declarations be prohibited?

Section: 7 [dcl.dcl] Declarations

Status: editorial

Description:

"A declaration introduces one or more names into a program and
specifies how those names are to be interpreted."

Is this intended to prohibit empty declarations like these?

```
enum { };
class { int i; };
class { };
typedef enum {};
```

In this case the WP should be clearer.

[Jerry Schwarz also notices:]

However, this can also be interpreted as prohibiting the following:
extern int i;

```
extern int i;
since the second declaration does not introduce anything (the name
has already been introduced in the program).
```

Resolution:

The first sentence in para 1, and para 3 will be reworked to make it clear that an empty declaration is ill-formed and that a declaration that redeclares an already existing name is well-formed.

Requestor: Tom Plum / Dan Saks
Owner: Steve Adamczyk (Types)
Emails:
Papers:

Work Group: Core
Issue Number: 116 (WMM.65)
Title: Is "const class X { };" legal?
Section: 7.1.5 [dcl.type] Type Specifiers
Status: closed

Description:

Is "const class X { };" legal, and, if so, what does it mean? i.e. if the declaration does not declare a declarator and a storage class specifier or a cv-qualifier is specified, are these simply ignored or is the declaration ill-formed?

Resolution:

See Santa Cruz motion 7).

Requestor: Mike Miller
Owner: Steve Adamczyk (Types)
Emails:
Papers:

Work Group: Core
Issue Number: 564
Title: is 'void f(const a);' well-formed?
Section: 7.1.5 [dcl.type] Type Specifiers
Status: editorial

Description:

The working paper says, in 7.1.5 para 3:

"At least on type-specifier is required in a function declaration unless it declares a constructor, destructor or type conversion operator.56)

56) There is no special provision for a decl-specifier-seq that lacks a type-specifier. The "implicit int" rule of C is no longer supported."

Annex C gives the following example:

"void f(const parm); // invalid C++"

A cv-qualifier (like const in the example above) is a type-specifier. So, according to the rule above, the example is valid, i.e. a declaration that has only cv-qualifiers in its type-specifier is valid according to 7.1.5.

Is the rule in 7.1.5 incorrect or is the example incorrect?

Resolution:

The WP will be updated to say:

"At least one type-specifier that is not a cv-qualifier is required in a typedef declaration. At least one type-specifier that is not a cv-qualifier is required in a function declaration unless it declares a constructor, destructor or type conversion operator.56)

56) There is no special provision for a decl-specifier-seq that lacks a type-specifier or that has a type-specifier that only specifies cv-qualifiers. The "implicit int" rule of C is no longer supported."

Requestor:
Owner: Steve Adamczyk (Types)
Emails:

Papers:

.....
Work Group: Core
Issue Number: 503
Title: Better semantics of bitfields of enumeration type needed
Section: 7.2 [dcl.enum] Enumeration declarations
Status: closed

Description:

7.2p5 describes the underlying type of enumeration types.
It should be made clear that this description does not apply to
the underlying type of enumeration bit-fields.

Also, something should be said about the signedness of enumeration
types. Bill Gibbons's suggested words:

"Even though the underlying type of an enumeration type will be
either signed or unsigned, enumerations themselves are neither
signed nor unsigned. [For example, a two-bit bit-field can hold an
enumeration with values {0,1,2,3}.]"

Resolution:

See Santa Cruz motion 8).

Requestor: Bill Gibbons
Owner: Steve Adamczyk (Types)
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.

I will present a paper for the Tokyo meeting to propose a possible resolution.

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 existence 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: 567
Title: Can a parameter have type 'T arr[]' where T is incomplete?
Section: 8.3.5 [dcl.fct] Functions
Status: closed

Description:
Is the following valid:
struct T;
void f(T arr[]); //1
?
8.3.4 says:
"As per 8.3.4, Arrays, paragraph 1, "In a declaration T D where D has the form "D1 [const-expr(opt)]" T shall not be a reference type, an incomplete type, ...".

Is //1 ill-formed because T is incomplete?

Proposed Resolution:

para 5 already properly covers this:
"If the type of a parameter includes a type of the form "pointer to array of unknown bound..."

Requestor: public comment T13.1
Owner: Steve Adamczyk (Declarators)
Emails:
Papers:

.....
Work Group: Core
Issue Number: 530
Title: Can default arguments appear in out-of-line member function definitions?
Section: 8.3.6 [dcl.fct.default] Default arguments
status: editorial

Description:
Issue 1):

```
-----  
For example  
struct X {  
    void f(int);    // no default argument here  
};
```

```
void X::f(int = 3) { } // is this allowed?
```

```
void g(X* xp) {  
    xp->f();    // uses default argument from definition  
}
```

This is particularly interesting when the function in question is a constructor. Adding default arguments outside of the class definition may add a default constructor to the class.

```
-----  
Issue 2):  
-----
```

Also, lijewski@roguewave.com notes:

Section 8.3.6 paragraph 4 contains the statement:

```
Declarations of a given function in different translation units  
shall specify the same default arguments (the accumulated sets of  
default arguments at the end of the translation units shall be  
the same).
```

Section 8.3.6 Paragraph 6 states contains the statement:

```
The default arguments in a member function definition that appears  
outs of the class definition are added to the set of default  
arguments provided by the member function declaration in the  
class definition.
```

Now consider the following example:

File x.h:

```
struct X { void f (int i); };
```

File x.cpp:

```
#include "x.h"  
  
void X::f (int i = 3) { }
```

File a.cpp:

```
#include "x.h"  
  
int main ()  
{  
    X x;  
    //  
    // Call X::f using default argument from x.cpp ???  
    //  
    // Is the DWP implying that an implementation must remember,  
    // across translation units, when a member function has some  
    // default arguments that aren't specified in its declaration in  
    // the class definition?  
    //  
    // I'd be mighty surprised if this were the intent :-) But then  
    // the ability to add default arguments in the definition of  
    // a member function outside of the class definition is  
    // practically guaranteed to contradict the statement from 8.3.6  
    // Paragraph 4 above.  
    //
```

```
// That is to say, adding default arguments in the definition of
// a member function outside of the class definition is
// guaranteed to contradict the statement in 8.3.6 Paragraph 4
// whenever the class definition and implementation are split
// between two files, and the class is used in any other
// translation unit.
//
return x.f();
}
```

Resolution:

Issue 1): closed: the WP indicates that default arguments can appear in a member function definition that appears outside of its class definition.

Issue 2): editorial: Change Section 8.3.6 paragraph 4 from:
"Declarations of a given function in different translation units..."

to:

"Definitions of a given function in different translation units..."

Requestor: Bill Gibbons / lijewski@roguewave.com

Owner: Steve Adamczyk (ODR)

Emails:

core-5855 and following messages

core-6342 and following messages

Papers:

95-0156=N0756 Default Arguments in Member Function Definition
by John Wilkinson

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);
};
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?

Resolution:

Requestor: Fergus Henderson

Owner: Steve Adamczyk (Default Arguments)

Emails:

core-5884

Papers:

Work Group: Core

Issue Number: 586

Title: When do access restrictions apply to default argument names?

Section: 8.3.6 [dcl.fct.default] Default arguments

status: editorial

Description:

```
class C {
    static int f() { return 0; }
public:
    C( int = f() ) { }
```

```
};
C c; // error? C::f accessible?

class D {
    static int f;
public:
    D( int = f ) { }
};
D d; // error? D::f accessible?
```

Does access checking take place when the default argument name is bound (at the point of the function declaration) or when the default argument name is implicitly used on the call?

Proposed resolution:

Access checking takes place when the default argument name is bound. That is, the example above is well-formed.

Resolution:

The proposed resolution will be incorporated into the WP.

Requestor: Neal M Gafter <gafter@mri.com>
Owner: Steve Adamczyk (Access Restrictions)
Emails:
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 or by way of redeclarations of the function? 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?

Resolution:

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

.....
=====

Chapter 9 - Classes

Work Group: Core
Issue Number: 568
Title: Can a POD class have a static member of type pointer-to-member, non-POD-struct or non-POD-union?
Section: 9 [class]
Status: closed

Description:

para 4 says:

"A POD-struct is an aggregate class that has no members of type pointer-to-member, non-POD-struct or non-POD-union (or arrays of such types) or reference, and has no user-defined copy assignment operator and no use-defined destructor."
And similar wording for POD-union.

An aggregate can have static members.
The wording above allows a POD class to have static members as well. However, it prohibits static members of type "pointer-to-member, non-POD-struct or non-POD-union (or arrays of such types) or

reference". Should it?

Proposed Resolution:

The sentence above should say:

"A POD-struct is an aggregate class that has no `_non-static_members`
...."

and similarly for POD-union.

Resolution:

See Santa Cruz motion 9).

Requestor:

Owner: Steve Adamczyk (Types)

Emails:

Papers:

Work Group: Core

Issue Number: 627

Title: What does it mean for the class name to be inserted as a
public member name?

Section: 9 [class]

Status: editorial

Description:

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* pbl;           // legal?
            A::B* pb2;       // illegal?
        };
    };
};
```

What does it mean for the class name to be inserted as a public
member name? Does this mean that C can refer to B which is a
private member of A? Refer to it as a qualified or unqualified
name?

Resolution:

This will be clarified as an editorial action item.

Requestor:

Owner: Steve Adamczyk (Name Look up)

Emails:

Papers:

Work Group: Core

Issue Number: 252

Title: Can the definition of an incomplete class appear in an
anonymous union?

Section: 9.1 [class.name] Class names

Status: closed

Description:

must an incomplete class object be completed in the same scope?
9.1p24 In C, a struct-or-union of incomplete type must be
completed in the same scope as the incomplete-type declaration, or it
remains an incomplete type.

[We believe the same is intended for incompletely-defined classes in
C++, but the document is not yet clear enough to tell.]

[Note JL:]

The resolution needs to clarify the following test case as well:

```
class C; //1
union {
    class C { ... }; //2
    ...
};
```

Does line //2 defines the class declared on line //1?

Resolution:

See Santa Cruz motion 4).

Requestor: Tom Plum / Dan Saks
Owner: Steve Adamczyk (Name look up)
Emails:
Papers:

Work Group: Core
Issue Number: 266
Title: Access specifiers in union member list
Section: 9.5 [class.union] Unions
Status: closed

Description:
9.5p3.2 - anonymous union may not have private or protected members.
This seems to imply that anonymous union may have public members;
and that non-anonymous union may have any access modifiers.
Is this wording really what is intended?

Resolution:

No action needed.

Requestor: Tom Plum / Dan Saks
Owner: Steve Adamczyk (Types)
Emails:
Papers:

Work Group: Core
Issue Number: 105 (WMM.27)
Title: How can static members which are anon unions be initialized?
Section: 9.5 [class.union] Unions
Status: closed

Description:
This is from Mike Miller's list of issues:
class C {
static union {
int i;
char * s;
};
union {
const int a, b;
};
};
int C::i = 3; // ? Is this syntax valid?
int C::a = 5; // ? Is this syntax valid?

Resolution:

See Santa Cruz motion 4).

Requestor: Mike Miller
Owner: Steve Adamczyk (Name Look up)
Emails:
Papers:

Work Group: Core
Issue Number: 570
Title: Name look up for anonymous union member names need to be better described.
Section: 9.5 [class.union] Unions
Status: closed

Description:
paragraph 2 says:
"The names of the members of an anonymous union shall be distinct
from other names in the scope in which the union is declared; ..."
Is this true?
How about:
int I;
static union {
class I { }; // error?
};
void f() {


```
        class I i; // is this OK?
    }
How about:
    class C;
    static union {
        class C { }; // does this complete the type of global
                    // class C?
    };
```

Resolution: See Santa Cruz motion 4).

Requestor:
Owner: Steve Adamczyk (Name Look up)
Emails:
Papers:

.....

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: 623
Title: Representation of bitfields of bool type
Section: 9.6 [class.bit] Bitfields
Status: closed

Description:
para 3 says:
"A bool type can be successfully stored in a bit-field of any nonzero size."
What does it mean "can be successfully stored"?

Resolution: See Santa Cruz motion 8).

Requestor:
Owner: Steve Adamczyk (Types)
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;    // allowed?
    ee bit:64;  // allowed?
    char bit:64; // allowed?
};
```

Resolution:
Requestor: ?
Owner: Steve Adamczyk (Types)
Emails:
core-1578

Papers:

.....

Work Group: Core
Issue Number: 267
Title: What does "Nor are there any references to bitfields" mean?
Section: 9.6 [class.bit] Bitfields
Status: editorial

Description:
9.6p3.5: "Nor are there references to bit-fields." Does this actually prohibit anything? A simple attempt to make a reference refer to a bit-field just creates a temporary:
union { int bitf:2; } u;
const int & r = u.bitf;
Or is this a syntactic restriction that prohibits something like
union { int (&rbitf):2 } u;
Or is it meant to prohibit the use of typedefs to attempt it, such as
union { typedef int bitf_t:2; bitf_t &rbitf; } u;
The intent needs clarifying.

Resolution:
Make it clear that it means:
A reference cannot be bound to a lvalue that refers to a bitfield.

Requestor: Tom Plum / Dan Saks
Owner: Steve Adamczyk (Types)
Emails:
Papers:

.....

Work Group: Core
Issue Number: 458
Title: When is an enum bitfield signed / unsigned?
Section: 9.6 [class.bit] Bitfields
Status: closed

Description:
enum Bool { false=0, true=1 };
struct A {
 Bool b:1;
};
A a;
a.b = true;
if (a.b == true) // if this is sign-extended, this fails.

Bill Gibbons proposed resolution:
Add after the sentence 9.7p5:
"It is implementation defined whether plain (neither explicitly signed or unsigned) int bitfield is signed or unsigned."
"...; enumeration bit-fields are neither signed nor unsigned."

Resolution:
See Santa Cruz motion 8).

Requestor: Sam Kendall
Owner: Steve Adamczyk (Types)
Emails:
Papers:

.....

Work Group: Core
Issue Number: 571
Title: Is bitfield part of the type?
Section: 9.6 [class.bit] Bitfields

Status: editorial

Description:

The description in 4.5 [conv.prom] para 3 seems to indicate that bitfield is part of the type. Is it?

If it is (as 4.5 seems to indicate) this subclause should be more explicit about it. If it isn't, bitfields should be discussed in lvalue/rvalue subclause [basic.lval] to describe how a bitfield lvalue is transformed into an rvalue.

Resolution:

See paper 96-0047/N0865 from the pre-Santa Cruz mailing. The bitfield attribute is not part of the type.

Requestor: Bill Gibbons

Owner: Steve Adamczyk (Types)

Emails:

Papers:

.....
=====

Chapter 10 - Derived classes

Work Group: Core

Issue Number: 441

Title: In which scope is the base class clause looked up?

Section: 10 [class.derived] Derived classes

Status: editorial

Description:

```
class C {
    class A { };
    class B : A { }; //1
};
```

Is A looked up in the scope of C or in the scope of B?
Is the declaration on line //1 ill-formed because the nested class B cannot refer to the private type A declared in C?
Or is it well-formed because the name A can be used in the scope of C?

Resolution:

This will be handled as an editorial issue.

Requestor:

Owner: Steve Adamczyk (Name Look up)

Emails:

Papers:

.....

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: 585
Title: Is access checking performed on the qualified-id of a member declarator?
Section: 11 [class.access]
Status: editorial

Description:

para 6 says:
"... access checking is not performed on the components of the qualified-id used to name the member in a declarator..."

Is this true if the qualified-id uses typedef names that are private?

```

class D { D f(); };
class C
{
    typedef D T;
};

```

D C::T::f() {} // Legal? T is a private typedef of C.

Proposed Resolution:

This issue will be handled as an editorial issue.

Resolution:

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

.....

Work Group: Core Language
Issue Number: 388
Title: Access Declarations and qualified ids
Section: 11.3 [class.access.dcl] Access Declarations
Status: editorial

Description:

The section says:
The base class member is given, in the derived class, the access in effect in the derived class declaration at the point of the access

declaration.

```
It isn't clear to me what this means for
class B { public: int i ; } ;
class D : private B {
    B::i ;
};

D* p ;
p->i ; // clearly legal
p->B::i ;
```

I don't care strongly about this, but I think it should be clarified.
(And added as an example).

Resolution:

This issue will be handled as an editorial issue.

Requestor: Jerry Schwarz
Owner: Steve Adamczyk (Access Specifications)
Emails:
Papers:

.....
Work Group: Core
Issue Number: 515
Title: How can friend classes use private and protected names?
Section: 11.4 [class.friend] Friends
Status: editorial
Description:

11.4 p2 says:
"Declaring a class to be a friend implies that private and protected names from the class granting friendship can be used in the class receiving it."

This is not very explicit.
Where can the private and protected names be used in the befriended class?
In the base classes of the befriended class?
In the nested classes of the befriended class?

Resolution:

This issue will be handled as an editorial issue.

Requestor: Erwin Unruh
Owner: Steve Adamczyk (Friends)
Emails:
Papers:

.....
Work Group: Core
Issue Number: 532
Title: Is a complete class definition allowed in a friend declaration?
Section: 11.4 [class.friend]
Status: editorial
Description:

Is this allowed:

```
class A {
    static int x;
    friend class B {
        int f() { return A::x; };
    };
};
```

If so, what is the scope of the class name B?

Resolution:

No, it is not allowed.
This will be handled as an editorial issue.

Requestor: Neal M Gafter <gafter@mri.com>
Owner: Steve Adamczyk (Friends)

Emails:

Papers:

.....

Work Group: Core
 Issue Number: 625
 Title: Can a friend function be declared "inline friend"?
 Section: 11.4 [class.friend]
 Status: editorial
 Description:

para 4 says:
 "No storage-class-specifier shall appear in the decl-specifier-seq
 of a friend declaration."
 Is the following allowed?

```
class C {
    inline friend void f();
};
void f() { }
```

Resolution:
 Yes it is allowed.
 The function has external linkage.
 This will be handled as an editorial issue.

Requestor:
 Owner: Steve Adamczyk (Friends)
 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: 293
 Title: Clarify the meaning of y.~Y
 Section: 12.4 [class.dtor] Destructors
 Status: editorial
 Description:

12.4p22 The notation y.~Y() is explicitly approved of by the example
 at bottom of ARM page 279), but nothing in the draft gives this
 explicit approval. Implementations differ. Committee should approve
 it or disapprove it.

Resolution:
 Yes it is allowed.
 This will be handled as an editorial issue.
 Requestor: Tom Plum / Dan Saks

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
```

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: active

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=(B)
```

?
para 10 does not clearly mention this.

Resolution:

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: 614
Title: Is a complete type needed for function overload resolution?
Section: 13.3 [over.match]
Status: closed

Description:

```
struct A;
struct B { };

struct D {
    D(const A&);
    D(const B&);
};

void foo(B& b) {
    D d(b); // must the implementation find the D(constB&) ctor
           // or must the types referred to be completed for
           // this program to be well-formed?
}
```

Resolution:

No action needed.
Yes, pointers and references may refer to incomplete types and overload resolution will still be successful.

Requestor:

Owner: Steve Adamczyk (function overload resolution)
Emails:
Papers:

.....

Work Group: Core
Issue Number: 599
Title: Are user-defined conversion sequences always ambiguous when the user-defined conversions considered are different?
Section: 13.3.3.2 [over.ics.rank]
Status: closed

Description:

para 3 second bullet:
"- User-defined conversion sequence U1 is a better conversion sequence than another user-defined conversion sequence U2 if they contain the same user-defined conversion operator or constructor and if the second standard conversion sequence of U1 is better than the second standard conversion sequence of U2."

Given the following code sample:

```
struct S {
    operator double();
    operator short();
};
```

```
};  
  
S s;  
... double(s) ...; // ambiguous?
```

There are two user-defined conversion sequences possible for this conversion:

```
S::operator double  
S::operator short -> standard conversion to double
```

and because the two user-defined conversion sequences use different user-defined conversions, the call is ambiguous.

This seems rather surprising.

Is this outcome really what the committee wanted?

Resolution:

The Core II WG decided that the current rules are acceptable as is.

Requestor:

Owner: Steve Adamczyk (function overload resolution)
Emails:
Papers:

Work Group: Core
Issue Number: 582
Title: What are the cv-qualifiers for the parameters of a candidate function?
Section: 13.6 [over.built]
Status: editorial

Description:

What are the cv-qualifiers for the parameters of a candidate function?

For example, given

```
class B {  
    operator const int **();  
};  
class D : B {  
    operator volatile int **();  
};  
B b;  
D d;  
... b == d ...
```

Is the builtin candidate function:

```
bool operator==(const volatile int**, const volatile int **);
```

or:

```
bool operator==(const int**, volatile int **);
```

?

Resolution:

The declarations for the built-in operators will be modified to indicate that the cv-qualifiers of the built-in operators is the union of the cv-qualifiers of the operands' types.

Requestor:

Owner: Steve Adamczyk (function overload resolution)
Emails:
Papers:

Work Group: Core
Issue Number: 583
Title: For a candidate built-in operator, must cv-qualifiers of parameters of type pointer to member be the same?
Section: 13.6 [over.built]
Status: editorial

Description:

The footnote associated with para 14, 15 and 16 says:

"When T is itself a pointer, the interior cv-qualifiers of the two parameter types need not be identical. The two pointer types are converted to a common type (which need

not be the same as either parameter type) by implicit pointer conversions."

This omits to take into account operands of type pointer to member with different cv-qualifiers on the pointer to member type.

Resolution:

The declarations for the built-in operators will be modified to indicate that the cv-qualifiers of the built-in operators is the union of the cv-qualifiers of the operands' types.

Requestor:

Owner: Steve Adamczyk (function overload resolution)

Emails:

Papers:

.....
=====

Chapter 15 - Exception Handling

Work Group: Core
Issue Number: 628
Title: Default argument on copy constructors & construction of exceptions
Section: 15.1[except.throw]
Status: editorial
Description:

```
struct A {  
    A(const A&, int i = expr) {  
        body;  
    }  
};
```

The following code

```
A a; throw a;
```

really is

```
A a;  
construct(exc_temp,a,default_expression);  
throw exc_temp;
```

Since the order of evaluation of function arguments is unspecified, it is unspecified whether a is evaluated before or after the default_expression. It is unspecified whether an expression in the default argument throws an exception and leads to terminate or not.

Proposed Resolution:

The "correct" repair to these problems would be to redefine the notion of constructor to disallow default arguments in a copy constructor. This would however have a big impact on existing code. So to repair the problem for the exception case only I would propose:

"When the copy constructor used to copy an exception object into the temporary or to copy the temporary into the named variable exits via an uncaught exception, it is implementation defined whether terminate is called. If terminate is not called, the old exception is abandoned (although the objects are destructed properly) and the new exception is used for a new exception lookup. This lookup either starts at point the abandoned exception was thrown or the point where the abandoned exception would have been caught. Which point is chosen implementation defined."

Resolution:

The answer is clear in 15.5.1 [except.terminate]:
"- when a exception handling mechanism, after completing evaluation of the object to be thrown but before completing the initialization of the exception-declaration in the matching handler,114) calls a user function that exits via an uncaught exception.

The evaluation of the default argument is part of the initialization of the internal temporary exception object, which occurs after the evaluation of the object to be thrown. So the answer is "terminate".

A note should be added to 15.1 to point to 15.5.1.

Requestor: Erwin Unruh
Owner: Bill Gibbons (exceptions)
Emails:
core-6346

Papers:

.....

Work Group: Core
Issue Number: 594
Title: If a constructor throws an exception, in which cases is the storage for the object deallocated?
Section: 15.2 [except.ctor]
Status: editorial

Description:

para 2 says:
"If the object or array was allocated in a new-expression, the storage occupied by that object is sometimes deleted also (5.3.4)."
Does this mean:
o deleted if an appropriate operator delete is present
or
o undefined behavior if delete must be called (runtime)

Resolution:

This is now described in detail in 5.3.4 [expr.new] paragraphs 17 & 18.

Requestor: public comment 7.12
Owner: Bill Gibbons (exceptions)
Emails:
Papers:

.....

Work Group: Core
Issue Number: 611
Title: What happens when an exception is thrown from the destructor of a subobject?
Section: 15.2 [except.ctor]
Status: closed

Description:

This section is not clear in describing what happens if an exception is thrown from the destructor of a subobject (i.e. for an array element or for a class member or base)?
Are the remaining elements/members/bases destroyed because of stack unwinding?
Is terminate called?

Resolution:

This is described in 15.2 [except.ctor]. The answer is:
Yes, remaining elements/members/bases are destroyed.
No, terminate is not called.

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

.....

Work Group: Core
Issue Number: 539
Title: Can one throw a pointer-to-member to a base class and catch it with a handler taking a pointer to a derived class?
Section: 15.3 [except.handle] Handling an exception
Status: closed

Description:

```
struct B { int i; };  
struct D : B { };  
int B::*pmb;
```

```

void f() {
    try {
        throw pmb;
    }
    catch (int D::*pmd) {
        // is the exception handled here?
    }
    catch(...) {
        // or here?
    }
}

```

Resolution:

No, since this case is not listed in 15.3 [except.handle].

Requestor:

Owner: Bill Gibbons (exceptions)
 Emails:
 Papers:

Work Group: Core

Issue Number: 540

Title: How does name look up proceed in a function-try-block?

Section: 15.3 [except.handle] Handling an exception

Status: Editorial

Description:

Can names of variables declared in the outermost block of the function be referred to?

If the function-try-block appears in a member function definition, are names declared in the scope of the class considered?

Resolution:

Function try-blocks and handlers follow the normal scoping rules apply, except that function parameters may not be redeclared at the outermost scope of function try-blocks, handlers, and catch clauses (editorial change). So the function try-block and any associated handlers are parallel scopes.

So names declared in the outermost block of the function may not be referred to in handlers; and names declared in scopes enclosing the function definition may be referred to just as an ordinary function body.

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: 590
Title: With function try blocks, does the caller or callee catches exceptions from constructors/destructors called for parms?
Section: 15.3 [except.handle] Handling an exception
Status: editorial

Description:

In the presence of function try blocks, if the constructor/desctructor for the function parameter throws an exception, who (caller/callee) is responsible for catching the exception?

```
class X {
public:
    ~X() { throw xx(); }
    // ...
};

class Y {
public:
    Y(int) { throw yy(); }
    // ...
};

class Z {
public:
    Z(const Z&) { throw zz(); }
    // ...
};

void f(X a, Y b, Z c) {
    // ...
}
catch (xx) {
    // will the xx thrown by ~X() be caught here?
}
catch (yy) {
    // will the yy thrown by Y(int) be caught here?
}
catch (zz) {
    // will the zz thrown by Z(const Z&) be caught here?
}

void g(X& x,Z& z)
{
    ff(x,1,z);
}
catch (xx) {
    // will the xx thrown by ~X() be caught here?
}
catch (yy) {
    // will the yy thrown by Y(int) be caught here?
}
catch (zz) {
    // will the zz thrown by Z(const Z&) be caught here?
}
}
```

Resolution:

Since we have now decided that the ctor and dtor calls for parameters are logically done at the call site (see 5.2.2), the answer is "callee".

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

ext-3402

Papers:

.....

Work Group: Core
Issue Number: 592
Title: Can a type be defined in a catch handler?
Section: 15.3 [except.handle] Handling an exception
Status: Editorial

Description:

Erwin Unruh in ext-3427:
"There are many places where 'types can not be defined'. The catch handler is one of the places where this is presently not the case.

I propose:

Add to [except.handle] 15.3:

"Types shall not be defined in an 'exception-declaration'."

Resolution:

The core III WG agreed at the Santa Cruz meeting that Erwin's proposed resolution is an editorial clarification.

Requestor: Erwin Unruh
Owner: Bill Gibbons (exceptions)
Emails: ext-3427

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: 629
Title: What does it mean for an exception-specification to be as restrictive as another exception-specification?
Section: 15.4 [except.spec]
Status: editorial

Description:

15.4 para 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."

Para 7 only defines what "to be as restrictive as" means for classes and pointers to classes. Something needs to be said about other types.

```
void fred() throw(int) {  
    throw 'a' ; // throw a char when an int is allowed?.  
}
```

```
void fred(int& i) throw(void*) {  
    throw &i ; // throw an int* when void* is allowed?.  
}
```

Resolution:

The core III WG agreed at the Santa Cruz meeting that the following clarification was editorial.

The phrase "function A is at least as restrictive as function B" means that all exceptions allowed by A are also allowed by B. The intent is that if a call which statically binds to B actually ends up at A, the called function (A) will not exit with an exception

which violates the promise made by the declaration in the static binding (B).

The WP should be revised to use the "allowed by" wording instead of the term "restrictive", which is not defined anywhere.

Requestor: Jerry Schwarz
Owner: Bill Gibbons (exceptions)
Emails:
core-6381

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 its input?

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

Papers:
.....
=====

Chapter 16 - Preprocessing Directives

Work Group: Core
Issue Number: 632
Title: Does redefining a macro make the program ill-formed or undefined behavior?
Section: 16.3 [cpp.replace]
Status: editorial
Description:
para 2 and 3:
"An identifier currently defined as a macro without use of lparen (an object-like macro) may be redefined by another #define preprocessing directive provided that the second definition is an object-like macro definition and the two replacement lists are identical.

An identifier currently defined as a macro using lparen (a function-like macro) may be redefined by another #define preprocessing directive provided that the second definition is a function-like macro definition that has the same number and spelling of parameters, and the two replacement lists are identical."

Does this mean that the program is ill-formed if the macro is redefined or does this mean the program has undefined behavior?

Resolution:
The WP will be modified to indicate that it is ill-formed.
Requestor:
Owner: Tom Plum (Preprocessor)
Emails:
Papers:
.....

Work Group: Core
Issue Number: 595
Title: Is a macro `__STDC_plusplus__` needed?
Section: 16.8 [cpp.predefined]
Status: closed
Description:
Resolution:

See Santa Cruz motion 23).

Requestor: ANSI public comment 8.5

Owner: Tom Plum (Preprocessor)

Emails:

Papers:

.....