Document Number: X3J16/92-0078
WG21/N0155
Date: 11 September, 1992
Project: Programming Language C++
Reply to: Dan Saks
dsaks@wittenberg.edu

X3J16 Meeting No. 9
WG21 Meeting No. 4
July 13-17, 1992

Toronto Hilton International
145 Richmond Street West
Toronto, Ontario M5H 2L2 Canada

1 Opening activities

Lenkov convened the meeting as chair at 9:04 (EDT) on Monday, July 13, 1992. Clamage was the vice-chair, and Saks was the secretary.

IBM, represented by Knuttila and Lajoie, hosted the meeting.

1.1 Opening comments

1.2 Introductions

1.3 Membership, voting rights, and procedures for the meeting

Clamage asked members to notify him of corrections to the membership list. Saks circulated an attendance list each day, which is attached as Appendix A of these minutes.

Lenkov reminded the attendees that this is a joint meeting of WG21 and X3J16. (The joint committee is denoted WG21+X3J16 in these minutes.) In straw votes, all WG21 technical experts may vote, even if this is the first meeting they've attended; however, X3J16 attendees can vote only if they are the voting representative of a member organization that has been represented at either of the previous two meetings. In WG21 formal votes, only the head of each national delegation may vote. In X3J16 formal votes, only one representative from each X3J16 member organization may vote if the organization meets the aforementioned attendance requirement.

1.4 Approval of the minutes from the previous meeting

Saks submitted the minutes from the previous meeting (92-0041 = N0118) for approval. He noted numerous minor corrections:

1. On page 1, change the date at the top center "1992".

2. On page 1, in the first sentence of item 1, delete "by Lenkov".

3. On page 11, in the first sentence of the paragraph beginning with "Koenig explained", change "than" to "that".

4. On page 12, in the last sentence in the paragraph beginning "Plauger thought", change "determined prior" to "determined by prior".

5. On page 15, add *: public X* to the declaration of *Y* in the code fragment.

6. On page 18, in the first sentence of the third paragraph from the end, change "produce a safe" to "produce safe".

7. On page 21, in the third sentence of the second paragraph, change "in style" to "in a style".

8. In Appendix B, add the motion by Plum/O'Riordan (from page 25) to accept the Working Paper as item 4 and renumber all subsequent motions.

Motion by Shopiro/Bruck: "Move we approve 92-0041 = N0118 with these corrections as the minutes of the previous meeting."

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 5 yes, 0 no.

1.5 Distribution of position papers, WG deliverables, and other documents not distributed before the meeting

1.6 Agenda review and approval

Lenkov submitted the proposed agenda (92-0063 = N0140) for approval. He proposed adding a technical session on strings on Monday at 19:30. Lajoie requested an additional item 1.7.1 to report on the previous evening's WG21 meeting.

Motion by Saks/O'Riordan: "Move that we accept the proposed agenda with these additions."

Motion passed X3J16: lots yes, 0 no, 1 abstain.
Motion passed WG21: 5 yes, 0 no, 0 abstain.

1.7 Conversion to type I - X3 Approval

Lenkov reported that X3 approved converting C++ standards development to a type I project (92-0065 = N0142).

1.7.1 Report on the previous evening's WG21 meeting

Lajoie and Saks explained that WG21 selects a drafting committee for each meeting. Each motion must be presented in writing to the drafting committee for approval and possible rewording before the committee can vote on it. This insures that members whose native language is not

English (and even those whose native language is English) have an opportunity to read the motion and understand it before voting. The drafting committee for this meeting was Saks and Rafter.

Lajoie and Saks asked that each working group designate a scribe to bring the WG's motion to the drafting committee. They wanted the scribes to be members of the drafting committee as well. The drafting committee will meet at the end of Thursday's session to finalize the wording of formal motions to be presented Friday. No one objected to the request.

Lajoie asked Lenkov to call for a volunteer to act as liaison to SC21/WG3 (SQL) committee. Schwarz thought we decided last time not to send a liaison. Lajoie disagreed, explaining that several members at the last meeting thought sending a liaison was a good idea, but no one volunteered. Beech (a member of the ANSI SQL committee X3H2) favored sending a liaison, adding that it would aid the discussions of a C++ binding for SQL.

Lenkov asked for a volunteer to be liaison to SC21/WG3. No one volunteered.

Lajoie also asked for a volunteer to be liaison to WG20. She added that Carter has already asked WG20 to add Steinmuller and Vilot to its mailing list, and Carter will ask WG20 to also add Plauger and Shopiro as well.

Lajoie asked the Libraries WG to consider Plum's *localedef* proposal (N0126 = 92-0049). She also asked them to consider the AFNOR C++ Experts group's concerns about synchronizing the ISO C library with the C++ library. Plauger explained that the AFNOR C++ group is concerned because WG14 (ISO C) expects to vote out the normative addendum to ISO C for DIS balloting in December. This addendum includes a major addition to large character set support.

1.8 Liaison reports

=== X3J11 (ANSI C) ===

Plauger explained that X3J11 is still working on acceptance of its type I project. It is also waiting to hear from ISO about how to handle interpretations of the standard.

=== WG14 (ISO C) ===

Plauger explained that the WG14 normative addendum contains three parts:
1. UK's clarifications of ambiguities in the ISO C standard
2. Japanese extensions for multibyte character support
3. Danish alternate to trigraphs

He said that although it's very likely that the first two parts will be approved, it's possible that the third part might not, which would put C out of synch with C++.

=== X3J11.1 (NCEG) ===

Swan said he hasn't been attending NCEG meetings, so he shouldn't be liaison anymore.

Clamage noted that some NCEG papers were distributed in recent WG21+X3J16 mailings.

Plauger explained that NCEG is not producing a standard, but rather a technical report that recommends ways for implementors to experiment with numerical extensions.

2 WG reports

Lenkov opened the committee of whole.

2.1 Core Language WG

Lenkov said that Koenig was absent due to illness. Lajoie offered to head the WG at this meeting.

Plum suggested that the Core WG needed two chairs -- a program management chair and a technical chair. Others agreed with Plum. Schwarz suggested forming another working grouping to handle the lesser core issues that are essentially editorial.

Lajoie offered to take care of administrative issues within the Core Language WG and insure that tasks are delegated properly.

Lajoie summarized the foremost open issues:
1. Name lookup
   -- friend classes and friend functions
   -- class types first referenced in a parameter list
   -- template type arguments
   -- incomplete types
2. Lifetime of temporaries
3. Interaction of function overloading and argument matching

Shopiro suggested that the Core group should look at Kendall's paper (92-0053 = N0130) on unifying lvalue and references. Schwarz said it contains no changes; it's all editorial. Kendall said he proposed some minor changes. Lenkov asked the Core group to look at the paper and decide.

2.2 Extensions WG

In Stroustrup's absence, Knuttila presented the WG's report. They had two top concerns:
-- run-time type identification
-- name space control.
They also planned to consider:
-- *operator new* and *operator delete* for arrays
-- *~const* - declaring members "never const" so you don't have to cast away constness when implementing logically const objects

-- overloading operator.()

Kendall asked if there was a procedure for rejecting, more-or-less finally, a proposed extension. Several members said no.

The committee discussed policies on sending rejection letters to people who have proposed extensions. Schwarz suggested that each such letter should be a polite "thanks for sharing" letter, with relevant technical papers attached, but no formal statement of the committee's action. He did not want to give proposers grounds for protesting the manner in which their proposals were handled. O'Riordan noted that each letter comes from member of Extensions group, not from the committee.

After further discussion, Lenkov said that when there are papers summarizing the WG's reasons for rejecting a proposed extension, the letter writer should send them along with the rejection letter whenever appropriate. No one objected.

2.3 Libraries WG

Vilot presented the Libraries WG report (92-0034 = N0111). He said the proposed Working Paper section on Language Support (92-0042 = N0120) might be ready for a vote at this meeting. The proposed *bits* and *bit_string* classes (92-0051 = N0128) might also be ready. He also asked members to read and comment on the revised input/output library (92-0059 = N0136) and the string library (92-0045 = N0122).

Vilot outlined the WG's open issues:

1. language support:
   - -- should *operator new* return a null or throw an *xalloc*?
   - -- *renew* function template
   - -- assertion support
2. input/output:
   - -- copy constructors
   - -- national language character, wide character and multibyte character support
   - -- file open modes
   - -- exceptions thrown by streambuf
   - -- string streams
   - -- exception specifications
   - -- *const* qualifiers
3. C library:
   - -- are *wchar_t*, *size_t*, etc. distinct for overloading? (depends on Core WG)
   - -- signals and exceptions
   - -- *localedef*
4. strings:
   - -- national language character set support, e.g., is ASCII 0x00 allowed at arbitrary positions?
   - -- function and operator overloading
   - -- temporaries and *char* * conversions (depends on Core WG)
   - -- regular expressions
   - -- string streams

-- exception specifications

5. containers:
   -- *DynArray<T>*, *PointerDynArray<T>*
   -- exception specifications
   -- should *operator[]* be overloaded despite the possibility of dangling references?
   -- *bits<n>*, *bit_string*

Vilot also said they will consider the AFNOR C++ Experts' concerns over synchronizing the C++ and ISO C libraries.

2.4 Environments WG

Stone presented the WG's report. He summarized the status of their open issues:

1. One Definition Rule (ODR) -- Turner, Johnson and Rabinov offered to write a paper, but did not. Chapin volunteered to update his earlier proposal (91-0073), especially with regard to template instantiation.

2. Translation Limits
   -- Stone consolidated email discussion by Stone, Kohlmiller, Edelson and Clamage
   -- Stone will broadcast a rough draft of a proposal with rationale to the *x3j16-all* reflector after this meeting

3. Mixed C/C++ Environments - Hartinger, Schwarz and Yaker offered to work on this, but made no progress.

4. Static Initialization
   -- Kearns replied via email to the discussion of his proposals at the last meeting. He submitted revised proposals via email (X3j16-env-293).
   -- The WG will discuss Kearns' proposals 1, 3 and 5.
   -- The WG may reconsider feasibility of dynamic initialization.

5. Accessing *argc* and *argv* from static constructors

Stone reported that traffic on their email reflector was light. They did not plan to present any formal proposals to entire committee at his meeting.

2.5 Formal Syntax WG

Roskind summarized the WG's open issues:

1. Binding of :: vs. *struct* for referring to hidden class names:
   -- *struct ::T* vs. *::struct T*
   -- *A::class B* vs. *class A::B*
   The latter arises in:

```
struct A {
    struct B {
        //...
        } B;
    };
```

The unadorned *A::B* refers to the member *B*, not the *struct B*. The question is how to refer to the *struct B*.

2. Qualified names as declarators. For example,

```
class A {
    int A::f(int);
};
```

is syntactically valid according the the June '92 Working Paper. Should this example be allowed syntactically and semantically?

3. When does an enumerator enter scope? The WG grouped noted differences in the corresponding wording in the C standard and the current Working Paper. For example, it might be that:

```
enum E { red = sizeof(red) };
              ^             ^
              |             red enters scope here in C?
             red enters scope here in C++
```

4. Ongoing correction of typographical errors in the grammar.

2.6 C Compatibility WG

Plum reported that the "impressionistic" list of C++ incompatibilities with C was distributed as email messages x3j16-compat-44 and -45. This is the first draft of the addendum requested by SC22 to be included in the eventual standard. He invited feedback from members and asked them to respond to the x3j16-compat reflector.

Plum said the committee voted to add the C lexical grammar to the Working Paper, but this was not done. He asked Shopiro to at least add a footnote to say that the C++'s lexical grammar is supposed to be the same as C.

Plum intends to propose that the draft use the following "spelled-out" names for the previously approved type categories:

FT -> function type
COT -> completely-defined object type
IOT -> incompletely-defined object type

Plum said the WG will consider introducing the notion of structural equivalence for C *structs*, unions and *enums*. Schwarz and Vilot said they opposed complicating the "one-definition" rule any further.

Plum said the WG will continue its investigation of the type system. The Working Paper says the type of a tagless class is a "made-up name". He said that if they get agreement on adding structural equivalence, maybe they can leave the Working Paper as is regarding tagless classes. Regarding the type of a pointer to array of unknown bound, Plum said that maybe Schwarz is right that we don't need variables or expressions of this type.

Plum listed other issues being handled by members of the WG:
-- use of terms "argument" vs. "parameter" (Koch)
-- null pointer constant (Kohlmiller)
-- conformance and diagnostics (Johnson)

3 Scheduling WG sessions

4 Working Paper for Draft Proposed Standard

Shopiro presented the editor's report for the June '92 Working Paper (92-0079 = N0156). He had promised to change the Working Paper to be more like an ISO document, but had not. He said he still intended to do so. He also apologized that a change in the footnoting macros used in formatting the document produced change bars at each footnoted line, even though there was no real change.

Shopiro explained that he added a new sentence 11.5p1.3 (section 11.5 paragraph 1 sentence 3) to close a loophole in protected access. Several members said they didn't understand the loophole. Kendall was confused by what it means "to access an address." Schwarz asked if the word "address" was defined anywhere. Saks suggested that the new sentence could be combined with the previous sentence by changing the "access" to "refer to" and by not enumerating all the ways that a member could be accessed. Shopiro said he'd take another crack at the paragraph.

Gibbons noted another problem with that new sentence. Citing 5.3p2, he said the new 11.5p1.3 does not account for the fact that the type of *&D::f*, where *f* is member inherited from base class *B*, is ``pointer to member of *B*'', not ``pointer to member of *D*''.

O'Riordan said he thought the sentiment of the change was correct, but the words needed work. He also noted that the new rule did not address access to a protected member of an indirect base class.

Saks asked if the statement in 12.4p8.2 that the delete operator "frees memory" is a requirement, or just wishful thinking. He also noted that 12.4p10.3 uses the word "illegal" when it should say "shall not" or "is undefined". Plauger said the draft must not say "illegal" because the standard does not have force of law. He also warned about the use of "only", because its meaning varies depending on its position in a sentence. Plum suggested new wording for the entire paragraph:

> When a destructor is invoked for an object, the object no longer exists; if the destructor is explicitly invoked again for the same object, the behavior is undefined. For example, if the destructor

for an auto object is explicitly invoked, and the block is subsequently left in a manner that would ordinarily invoke the implicit destruction, the behavior is undefined.

Terribile suggested using "In particular" instead of "For example".

Shopiro described his change to require all declarations and the definition of a function to have identical exception specifications (15.5p2.1). Scian noted that exception specifications can appear on pointers to functions. He observed that the change introduced an inconsistency -- the Working Paper requires consistency in the exception specification of all declarations and definitions of a function, but not in declarations of function parameters. For example, given

```
class C {
    void foo(void (*x)() throw (char)) throw (char);
};

void C::foo(void (*x)() throw (double)) throw (char) { ... }
```

the Working Paper now requires the exception specifications in the declaration and definition of *C::foo* to be the same; however, it does not require the exception-specifications to be the same for formal parameter *x*. O'Riordan said at the moment exception specifications are not part of the type. Even though the exception specifications are syntactically valid on pointers to functions, they have unspecified semantics in this context, and therefore are not allowed.

Schwarz asked if it's possible to change an exception specification when overriding a virtual function. Shopiro said he didn't know. O'Riordan reiterated that the exception specification is not part of the type. He explained that, years ago, when he, Stroustrup and Koenig were discussing the exception handling extensions, they considered whether exception specifications should be part of the type, and Stroustrup decided they should not. Hence, changing the exception specification when overriding a virtual function is legal. Schwarz suggested this is not an editorial issue -- it's a core issue.

O'Riordan pointed out that the change in 7.4p5.5 is substantive. The statement in the June '92 Working Paper is "Otherwise, a function or object declared in a linkage specification behaves as if it was explicitly declared extern." (Shopiro had added "or object".) O'Riordan said that treating an object declaration as if explicitly declared extern turns a definition into a declaration. Shopiro said he will reconsider the change. Scian pointed out that an annotation in the ARM makes it clear that

```
extern "C" { int a1; }
```

is a definition of *a1*, and

```
extern "C" int a2;
```

is a declaration of *a2*.

Ward noted that the changes in 10.2p1 weren't in Shopiro's report. (The changes reflect the relaxation of the rules for the return type of virtual functions approved at the last meeting.) She asked about the rule(s) when the return type of the overriding function is a pointer or reference to a class that is privately or protectedly derived from class *B* (where *B* is as described in 10.2p1.4.) Shopiro suggested that the Core WG should resolve the issue. Saks said it should go back to the Extensions WG -- vagaries in extensions are not core issues. Plum asked Saks what to do about problems in templates and exceptions? Saks said they should be handled by the Extensions WG also.

Stone questioned the meaning of 9.2.1p1 item number 3. O'Riordan said that when Core WG discussed that rule, they intended it to describe the effect of lexical reordering on the types of the declarators, but not the effect on run-time behavior implied by the order of declaration. But the words don't clearly reflect that intent.

The committee debated approving the Working Paper as is, or explicitly noting the errors in the motion to accept the document. Waggoner offered to draft a motion for a vote on Friday. Saks explained that the drafting committee meets Thurday evening to draft motions, to be copied and distributed before voting on Friday morning.

5 Rationale

No discussion.

6 WG sessions

Before recessing to working groups, Lenokv asked the groups to designate scribes. The volunteers were: Extensions: O'Riordan, Core: Pennello, Libraries: Schwarz, Environments: Ward, Compatibility: Koch, Syntax: Krohn

Lenkov closed the committee of the whole.

The committee recessed to working groups at 15:45 and reconvened at 08:35 on Wednesday.

7 WG progress reports

8 WG sessions

After brief status reports, the committee recessed to working groups at 08:40 and reconvened on Thursday at 08:35.

9 General Session

Lenkov opened the committee of the whole.

=== Core Language ===

Pennello presented the WG's proposal on friend declarations. The problem the group sought to solve is: If a name mentioned in a friend declaration is not declared, where does that name get declared? He added that if the name is already declared, there is no problem to solve.

Pennello said the aforementioned problem does not include declarations like:

```
class T *p;
```

Here, if *T* is already declared, this declaration uses that *T*. If not, this declaration inserts a declaration for *T* in the current scope. Schwarz disagreed that all compilers behave this way -- that some inject a declaration for *T* into the enclosing scope.

Pennello also explained that the problem addressed by the WG did not include declarations like:

```
class S;
```

which, if *S* is not declared previously, always injects *S* into the current scope.

Pennello enumerated the WG's guiding principles for friends:
-- Use the same name injection principles for both friend classes and friend functions.
-- Avoid injecting names into a class scope. (The WG didn't want to have a friend declaration inject a name into an enclosing class, especially if that name names a function. Would it introduce a member?)
-- Maintain compatibility with existing code at file scope level.
-- Use existing wording in the ARM whenever possible.

Pennello reviewed what the ARM says; in each case it is identical to the wording in the June '92 Working Paper.

[A] ARM section 9.1, page 168: "If a class mentioned as a friend has not been declared, its name is entered in the same scope as [that in which] the name of the class containing the friend declaration [was entered]."

Pennello said the wording is inaccurate and he added the bracketed text to clarify his presentation. In short, the rule says "A friend class introduces a name in the same scope as that containing the parent class."

ARM 11.4, page 250: "If a class or a function mentioned as a friend..."

Pennello said this sentence is identical to the rule on page 168, except it mentions functions as well.

[B] ARM 11.4, page 251: "A function first declared in a friend declaration is equivalent to an extern declaration."

[C] ARM 9.7, page 188 [annotation]: An example illustrates a function *g* declared two class levels deep referring to a global function. Thus, the function is moved out to the global level to make it extern, adhering to [B]. We might conclude from this example that "a friend function is declared in the nearest enclosing non-class scope".

[D] ARM 3.2, page 15: "A name first declared by a friend declaration belongs to the global scope."

Pennello said the WG suspected [D] was supposed to be the same as [A] but was inadvertently not edited.

Pennello said the WG spent several hours discussing various name injection methods. Most of the methods clashed with the above principles, and the WG dispatched them quickly. Pennello presented the two survivors, noting that the WG had a preference for one over the other:

[G] inject the name in the global scope

[N] inject the name in the nearest non-class scope

Without nested classes and local classes, these are the same.

Stroustrup asked why did the WG rejected injection one level up (into whatever scope). Pennello said he didn't want to inject a name into an unsuspecting class (as stated in the first principle, above). Stroustrup asked if the WG had considered injecting names into the enclosing scope, but with a constraint against injecting into a class. Pennello said doing so would break the example on page 188 of ARM.

Pennello gave the following example to show the difference between [G] and [N], above:

```
void f() {
    class A {
        friend class B;        //1
        static int x;
    };
    class B {
        void g() { A::x; }     //2
    };
}
class B {
    void g() { ??? }
};
```

Under [G], //1 injects class *B* into the global scope, and the injected declaration resolves to the global class *B*. But the global class *B* can't make any use of the members of *A*. Thus [G] makes little sense, and [N] is preferable. [N] makes the local class *B* a friend, and permits the reference to *A::x* in //2.

Pennello said the WG wondered that, if [G] was supposed to be the rule, why does the ARM say [A]? It would have been much simpler to just write what was written in [D] in the ARM.

Pennello gave the following example to illustrate ARM rule [B], above, which says that friend function declarations are equivalent to extern declarations:

```
void f() {
    extern g();       //3
    class A {
        friend g();   //4
    };
}
```

Here the *friend g* in //4 is clearly a reference of the *extern g* in //3. Furthermore, unlike in early and perhaps even current C compilers, *g* is not available in the global scope. Thus if you delete //3, the WG prefers that //4 inject a declaration of *g* into *f*'s scope, not into the global scope.

Pennello said the WG felt that [N] has several advantages over [G]:
-- [G] makes little sense when dealing with local classes
-- Why was the wording of [A] so involved when [G] can be stated simply?
-- [N] prevents injecting function declarations past functions and is consistent with having local extern functions.

Thus, the WG recommended [N].

O'Riordan presented the following examples to demonstrate his concerns about friend functions in local classes:

```
int f() {
    class X {
        friend int g();
    };
}
```

Here there's no way to define *g* outside *f*. Another example:

```
int x() {
    class T {
        friend int g() { T at; /* ... */ }
    };
}

int y() {
    class T {
        friend int g() { ? }
    };
}
```

Here, the *g*s are the same function. O'Riordan wondered that, if both *g*s are defined inline, how do you honor the one-definition rule while still making use of the friendship privileges? Yet another example:

```
int p() {
    class S {
        friend int p() { ... }
        };
    }
```

O'Riordan explained that the only function that can be a friend of a local class is the function enclosing the local class. But this is of limited utility.

O'Riordan's point was that you can't do anything with a friend function declared in a local class. Lajoie suggested continuing with the discussion of friend injection, and then reconsidering the utility of friend functions in local classes at a later time.

Pennello noted that the injection rules interact with the recently accepted name lookup rules. He added that the interaction is harmless, and there's always a way to get the desired effect. He gave this example:

```
class A {
    friend class B;   //1
    };
class B {             //2
    };
```

This is legal, and the *B* declared on //1 refers to the *B* defined on //2. When you put classes *A* and *B* inside a class *C*:

```
class C {
    class A {
        friend class B;   //1
        };
    class B {             //2
        };
    };
```

the name *B* is injected before the *C* (under either possible injection rule [G] or [N]). As such, *C::B* is not the friend, and the example is as if you had written:

```
class B;                    //1
class C {
    class A {               //2
        friend class B;     //3
    };
    class B {               //4
        };
    };
```

Pennello explained that now you run afoul of the new name lookup rules. The *B* on //3 first refers to the *B* on //1, then later refers to the *B* on //4 after reconsideration (rule 2 of name lookup). This is an error. But to get the desired effect, simply insert:

```
class B;
```

before //2.

Lajoie showed there is already precedent for using forward declarations to introduce names into the proper scope. She gave this example:

```
class Y;          //1
class X {
    class Y *p;   //2
};
class Y {
    class X *q;
};
```

Without //1, //2 declares *Y* as *X::Y*, which never gets completed.

Stroustrup said his intent was that friend functions and friend classes should be subject to the same injection rules. He also said his intent (not yet achieved) was that injection rules for friends should be the same as the rule for the name *Y* in

```
class Y *p;
```

He suggested this would allow Lajoie's example, even with //1 omitted.

Kendall asked Stroustrup what his intent was for:

```
void f(struct S *p); //1
...
struct S *q;         //2
```

Kendall added that it's not clear whether *S* in //2 refers to *S* in //1.

Plum explained that in C, //2 does not refer to //1. He added that this was not a conscious decision by the C committee, but fell out from other rules. C users complained when they discovered this. Plum said we might get similar flak if we take the same pedantic approach to Lajoie's example.

Lajoie asked for clarification of Stroustrup's intent that injection rules for friends should be the same as the rule for the name *Y* in

```
class Y *p;
```

She gave this example:

```
class C {
    class X {
        class Y *p;
    };
    class Y {
        class X *q;
    };
};
```

and asked where he intended *Y* to be introduced. He said it injects *Y* into the scope containing *C*. Pennello noted it becomes illegal on name lookup grounds, as in the friend example above.

Stroustrup said he preferred there be no injection rules. But, if you don't have them, you break every C and C++ program ever written. Gibbons suggested making injection an anachronism.

Straw vote: Who prefers rule [G] for friend declarations? 1. Who prefers [N]? lots.

Clamage presented his proposal for making *wchar_t* a distinct type for overloading, a modification of his and Schwarz's earlier proposal (92-0047 = N0124). He presented this sample code:

```
int i;
wchar_t w;

cout << i;          //1
cout << w;          //2

int foo(int);       //3
int foo(wchar_t);   //4
```

He explained that in C and in C++ as described by the June '92 Working Paper, *wchar_t* is a *typedef* defined to be one of the other integral types. He would like to be able to define the iostream library so that //1 and //2 invoke distinct functions on all implementations. He would also like the function declarations on //3 and //4 to be valid overloadings on all implementations. But, the Working Paper provides neither capability.

He proposed to solve the problem as follows: Make *wchar_t* a distinct type implemented the same as one of the standard integral types, i.e., with the same number of bits and signedness as one of these types. In contrast to his original proposal, he proposed making *wchar_t* a keyword, rather than a reserved word. (The other details of the proposal are listed in the motion presented on Friday. See agenda item 10.)

Gray asked what is result type of adding 1 to a *wchar_t* expression. Clamage said it's the promoted type. Gibbons asked if a *wchar_t* can have an explicit *signed* or *unsigned* modifier. Clamage said no.

Plum said this proposal doesn't say what happens what you convert from an integral type into *wchar_t*. O'Riordan replied that the intent was for conversion to *wchar_t* should be a standard conversion, not a promotion, so it's more "expensive".

Pennello asked Clamage about the following potential ambiguity:

```
f(short), f(int);
...
f(short-expr);
f(wchar-expr);
```

He noted that if *wchar_t* is represented as *int*, then *f(wchar-expr)* calls *f(int)*. Otherwise, it's ambiguous. Several members noted that this sort of implementation dependency in overload resolution is not new, and is documented in the ARM.

Gray asked if a bit-field can have type *wchar_t*. Clamage answered that if a bit-field can have an integral type, then a bit-field can be *wchar_t*.

Pennello reiterated his concern that *wchar_t* is either promoted to *int*, or converted to *int*, depending on the implementation.

Straw vote: Who favors Clamage's proposal: lots yes, 5 no, 3 abstain.

=== Extensions ===

Stroustrup reported that the WG worked on:
-- *~const*
-- *operator new[]()*
-- run-time type identification
-- default arguments everywhere
-- overloading based on enumerations

He also listed the backlog of proposals for consideration:
-- restricted pointers
-- name space control
-- *operator.()*
-- templates
-- extended character set
-- member initializers
-- return type of virtuals

Shopiro presented the *~const* proposal (from Thomas Ngo). The goals are:
-- to make "casting away const" unnecessary
-- to allow a more declarative programming style with respect to const
-- to allow a stronger definition of constness in the standard
-- to allow certain compiler optimization

Shopiro explained the difference between bitwise (concrete, physical) and meaningwise (abstract, logical) const using the following example of a class for complex numbers:

```
class complex {
    double re;
    double im;
    double abs;
public:
    double abs_val() const;
};

const complex c;
c.abs_val();
```

Suppose you would like to be able to compute the absolute value for const objects like *c*, so you declare *abs_val* as a const member function. But, suppose computing *abs_val* is expensive, so you write the function to compute the value and tuck it away (in the *abs* data member) for the next call. Inside *abs_val*, you must "cast away" the constness of the object referenced by *this* so you can alter the *abs* member. *c* is "meaningwise" const in that it appears to be const to the user, although its representation changes over time.

Shopiro added that bitwise constness does not imply meaningwise constness, and vice versa.

Shopiro listed the "hoped-for" advantages of bitwise constness:
-- only bitwise const objects are ROMable
-- bitwise constness enables certain compiler optimizations

Shopiro also listed the "hoped-for" advantages of meaningwise constness:
-- it supports declarative, high-level programming

Shopiro said the current Working Paper approach favors meaningwise constness, but tries to support both forms of constness using two kinds of classes -- those that are ROMable, and those that are not. If a class is ROMable, then declaring an object of that type const implies it's bitwise const. If a class is non-ROMable, then declaring an object of that type const implies it's meaningwise const, and so casting away const "works". That is, the object behaves exactly as if it were never declared const.

Shopiro explained that the Working Paper currently makes no distinction between casting away const inside or outside a member function. He said we might like to make this distinction because a meaningwise const member function often needs to change the object. But, outside a member function, casting away const is suspect.

Shopiro presented three alternative approaches to "casting away const":
1. leave things as they are now
2. "object const" -- The notion of constness depends on how an object is created. You can cast away const from a pointer (or reference) only if it really points (refers) to a non-const object.
3. "pointer const" -- Casting away constness is always undefined, no matter what the pointer points to or reference refers to. (This is needed for optimization.)

He said the WG preferred (2).

Although the WG has not decided on a syntax for *~const*, he used this example to illustrate the ideas:

```
class complex {
    double re;
    double im;
    double abs ~const;        // ~const data member
public:
    double abs_val() ~const; // ~const function member
};
```

He said the WG considered different semantics for *~const*:
1. *~const* data only -- you may declare class objects const, but even in a const object, a *~const* field is not const.
2. *~const* data and function "union" -- same a (1), plus a *~const* member function can change any data member, even if the object is const. (This implies non-ROMability.)
3. *~const* data and function "intersection" -- a data member must be *~const*, and it can only be changed by *~const* member function.

The WG did not have preference for any one of these three.

Shopiro emphasized that the WG was not up to deciding on syntax. They were still deciding on the semantics, or even to reject the extension altogether.

Clamage asked if the WG considered the impact of this extension on storage layout rules. Shopiro said no.

Stroustrup summarized the proposal to add *operator new* and *operator delete* for arrays (92-0055 = N0131). He summarized the reasons for the proposed extension:
-- users write expressions like *new x[7]*, i.e., they allocate arrays of class objects.
-- defining *::operator new* can cause trouble because it has to handle everything (not only different size objects, but arrays as well as scalars).

Stroustrup summarized the open issues:

-- The proposal suggests the notation *operator new[]()* and *operator delete[]()*, but the WG had not settled on the syntax.
-- Does *new x[20][35]* use the array *operator new* for *x* or for *x[20]*? The WG leaned toward treating *new x[20][35]* as *new x[20*35]*. (This means you can use *x::new[]()* for what appears to be a multi-dimensional array.)
-- Does *operator new[]()* need an argument specifying the number of array elements?

The primary reason the WG did not propose a vote at this meeting was that they were unable to decide on the last item. Stroustrup explained that some people needed more time to verify they didn't need this information in a call.

Stroustrup then reviewed the work on run-time type identification (92-0068 = N0145). The WG favored keeping *type_info* minimal. At the very least, *type_info* needs:
-- *operator ==*
-- *name()*: returns a name string
-- an ordering relationship
-- an operator to access extended type information (if extended type information exists)

Stroustrup said providing standard extended type information is still an open question:
-- can it support object i/o?
-- can it represent the declaration source? In other words, can you ask every question about a type?
-- anything else?

Stroustrup said the WG discussed templates with respect to run-time type identification. They agreed to allow *typeid(T)* where *T* is a non-polymorphic type (i.e. where *T* has no virtual functions). They also agreed there should be no special rules for type expressions that can be evaluated at compile time.

Stroustrup gave this example:

```
template <class T> void f(T a)
{
    if (typeid(T) == typeid(int)) { //1
        a++;                        //2
        // ...
    } else {
        // ...
    }
}
```

The expression *a++* on *//2* won't compile for an arbitrary type *T*. For this example to compile for all types *T*, the compiler must suppress further syntactic and semantic analysis once it determines that the condition on *//1* is false. Stroustrup said he thought the June '92

Working Paper says that the compiler must do syntactic and semantic analysis even if it knows the condition is false. The WG decided they don't want to change that rule.

Stroustrup presented alternative syntax considered for checked casts:

```
(?D *)p
(D *)p
ptr_cast(D *, p)
(virtual D *)p
?(D *)p
checked<D *>(p)
```

The WG favored the last one.

Stroustrup said some people strongly oppose "cryptic" character combinations, like (? ...). The most common mistake in using (? ...) is to leave out the '?', inadvertently writing an unconditional cast.

Stroustrup will continue exploring the use of function templates to support run-time type identification:

1. check<T>(v): examine *v*
   -- return 0 or throw something
   -- compile-time error if you can't examine *v*

2. coerce<T>(v): make a *T* out of a *v*
   -- *T* may be an incomplete type
   -- operation may reinterpret bits

3. convert<T>(v): apply a well-defined conversion
   -- *T* may not be an incomplete type
   -- compile-time error if conversion doesn't exist

Stroustrup suggested we might be able to deprecate the *(T)v* and *T(v)* notations for casts. Clamage said we can't deprecate *T(v)* because it's also the syntax for constructor calls.

Stroustrup summarized the WG members' work assignments:
-- *~const*: Knuttila, Bruck, and Schwarz
-- *operator new[]()*: Sloane, Gibbons
-- run-time type identification: Kiefer, Sloane
-- default arguments everywhere: Knuttila (write a rejection letter)
-- overloading based on enumerations: Bruck (coming right up)

Bruck presented his proposal to allow overloading of operators based on enumerations (91-0139 = N0072, with additional comments in 92-0070 = N0147). He explained the motivation for the proposal with the following example. Given:

```
enum status { bad, awful, hopeless };
f(int);
f(status);
```

then the call *f(bad)* calls *f(status)*, but *f(bad|awful)* calls *f(int)*, because *bad|awful* is an expression of type *int*. That is, you can overload function *f*, but calling an *f* with an expression involving enums is a "pitfall" because it always invokes *f(int)*.

Bruck also explained that the current Working Paper does not allow operator overloading as in:

```
enum season { winter, spring, summer, fall };

season operator++(season s)
    {
    switch (s)
        {
        case winter: return spring;
        case spring: return summer;
        case summer: return fall;
        case fall  : return winter;
        }
    }
```

Bruck explained that the proposal preserves an important objective of C++, namely, not to change operations on built-in types. He said that old C++ treated enums as a kind of *int*. Consequently, there was an implicit conversion from *int* to enum. As a consequence of that, if you allowed overloaded operations based on enums, it would be possible to redefine operations on *int*s, thus violating the objective. In current C++, each enum is a distinct type, so there's no implicit conversion from *int* to enum. Thus you can't define operations on *int*. Therefore, the WG proposed making overloading on enums legal.

Bruck said the essence of the proposal was to allow operator overloading on all user-defined types. He also said the proposal is not intended to make enums a "lightweight" class, i.e., not to allow any of the following for enums: derivation, constructors, member functions, assignment operators, or conversion operators.

Bruck provided an "impact statement" based on O'Riordan's experience in implementing this extension in Microsoft's C++ compiler:
-- it's trivial to implement
-- there's no interaction with the existing language
Also, the proposed feature has no direct impact on existing code -- you must decide to use the new feature.

Bruck said the WG considered alternative approaches:
1. Enums are useless, so we shouldn't care. Just leave things alone.
2. Use a class instead:
   -- it's more work
   -- it's less efficient
   -- you can't have constants (for use in case labels, etc.)
3. Fake enums with templates (as described in 92-0070 = N0147):
   -- it works (sort of)
   -- you can't have constants (for use in case labels, etc.)
   -- it's not efficient

-- it leaves the domain of enumerators undefined (you can create new symbolic values for enumerators at runtime)

Bruck noted a positive side-effect of proposal: You can make certain operations on enums illegal. For example, if you want *summer+winter* to be an error, then either:

1. Declare but don't define *operator+(season)*. This yields a link-time error if you try to call it.

2. Return a magic class, such as

```
class BadOp { ~BadOp(); } // private destructor
```

Returning an object with a private destructor is not possible, so

```
BadOp operator+(season, season);
```

causes a compile-time error as soon as it's used.

Bruck gave specific wording for the proposal. (See the wording of motion under agenda item 10.)

Bruck reported that the WG found a related problem with the existing specification for enums. Given:

```
enum openmode { in = 1, out = 2, ate = 4, bin = 8 };
openmode m = (openmode)(out | in);
```

then the value of *m* is not one of the declared enumerator values. Further, if *m* were initialized with

```
openmode m = (openmode)(ate | bin);
```

then its value is completely outside the range of enumerators. *m* is, however, within the "bit size" of type *openmode*.

Bruck said this coding technique is common in existing code. He suggested we should formalize this practice within the type system. The Extensions WG wanted to hand the issue to the C Compatibility and Core WGs for resolution.

Straw vote: Who favors this proposal: lots yes, 0 no, 5 abstain.

=== Libraries ===

Vilot presented the WG's latest thinking on the proposed language support library (92-0043 = N0120). He said the WG explored alternatives for what *::operator new* should do if it fails to allocate storage:
1. return null [0]
2. throw xalloc [13]
3. either or both of the above [3.5]
The numbers in brackets indicate the number of WG members who preferred that alternative.

The WG also considered whether the type of a new handler function should be *void (*)()* or *int(*)()*. They also considered omitting new handlers from the library entirely.

Vilot said the WG agreed that *::operator new(size_t, void *)* should be reserved, but that *::operator new(size_t)* and *::operator delete* should not. However, if defined by the user, they must have external linkage.

Vilot reported that WG spent much time on internationalization as it affects the string library. He also presented a rough classification of the types of strings in the C library and what to expect in the C++ library:

Representation	"C" facilities	Class facilities
individual bit	unsigned, operator	bits, bit_string
bytes	mem*()	\
chars	str*(), strn*()	} string
chars w/ locales	strcoll(), strxfrm()	/
wchar_t	wstr*(), wc*() (mb*())	wstring (encode/decode)

Vilot briefly summarized the WG's progress on i/o. Again, much the discussion focused on the impact of internationalization.

Vilot said the WG discussed Plum's localedef proposal (92-0049 = N0126). They agreed that for now, WG20 is handling the issue. Plauger will draft a letter from WG21 to WG20 expressing concern that the work of WG20 remain compatible with C and C++.

Vilot said the WG had a brief discussion on the dynamic array proposal (92-0046 = N0123). Some in the WG suggested the proposed class is too small to be useful, and should be dropped. The WG also discussed abandoning *operator[]* because it might return a dangling reference. They considered making *operator[]* return an object of a "helper class", as described by Coplien in his "Advanced C++ Programming Styles and Idioms". Vilot noted this was a trade between safety and efficiency.

Returning *::operator new*, Vilot explained that the WG leaned heavily toward requiring that it throw an *xalloc* exception when the allocation fails, and that it never return null. He asked for the committee's reaction.

O'Riordan said if you require the standard *::operator new* to throw an *xalloc*, you must either:
1. write everything to handle user-defined *new*'s that return nulls, or
2. require all user-defined *new*'s to throw *xalloc*.

He thought it would be difficult to specify the corresponding "contract" between the implementation and the programmer.

Saks replied that it's no easier to specify the requirements on the implementation and the programmer if you require that *new* return null than it is to specify the requirements when *new* throws an *xalloc*.

Koch questioned the restriction that user-defined *new* must be *extern*.

Vilot said that the WG didn't want to have to check everywhere that *new* might return null.

O'Riordan gave an example to defend returning null. Some users attempt to allocate a whole bunch of blocks they may not use immediately, then check if they got the block(s), when and if they actually need them. These users will need to write try blocks around each allocation, and map the *xalloc* exceptions into pointers whose values are null.

Bruck said he favored having *new* throw *xalloc*. If the constructor in a *new* expression throws an exception, users must write *try* blocks anyway.

Gibbons noted an efficiency consideration. Many implementations insert calls to *operator new* in constructors. If *operator new* might return 0, these constructors must check the return value. With inline constructors and low overhead objects, the amount of code space wasted on these checks might be large. He said it's a minor consideration, but worth noting.

Gray said he had a vague concern about *operator new* using exceptions. If *new* might throw *xalloc* and never returns null, users will be forced to have exception handling constructs in their programs. Clamage replied that there are alternatives that avoid exceptions:
-- supply your own *operator new* that doesn't throw an exception, or
-- use *malloc*, and *new* expressions with placement to avoid exceptions.

O'Riordan said the overhead of catching exceptions is much greater than testing for null, and would result in inefficient programs.

Straw vote: Who wants *new* to throw *xalloc* if it fails to allocate storage, and never return null? lots yes, 4 no, 8 abstain.

=== Syntax ===

Roskind presented the following grammar rules from the current Working Paper:

```
class-name:
    identifier                  //1
elaborated-type-specifier:
    class-key identifier        //2
    class-key class-name        //3
    enum enum-name
```

He explained that since rule //1 says a class-name is an identifier, rule //3 is sufficient and //2 is redundant. He asked the committee to consider removing the redundant rule.

Vilot pointed out that the identifier in //2 becomes a class-name after it has been declared as a class.

Straw vote: Who favors removing the redundant grammar rule? 17 yes, 9 no, 15 abstain.

Lenkov said the vote wasn't clearly in favor of the proposal because so many abstained. Bruck said he'd invoke the two-week rule if this proposal came to a formal vote.

Roskind alternatively proposed several additions to the grammar:

```
elaborate-type-specifier:
    enum identifier
    class-key enum-name
    enum class-name
    class-key typedef-name
    enum typedef-name
```

The idea is that after *enum* or *class-key*, you can have anything that looks like an identifier, including (from 18.1p1) class-name, enum-name, typedef-name, and of course, identifier itself. Therefore, if we don't take the short way out and assume people know that identifier in this context means all those identifier-like things, then we have to take the long way and explicitly mention all the combinations. The complete (sorted) list is:

```
elaborated-type-specifier:
    class-key class-name
    class-key enum-name          // new
    class-key identifier
    class-key typedef-name       // new
    enum class-name              // new
    enum enum-name
    enum identifier              // new
    enum typedef-name            // new
```

The WG also agreed to include

```
    class-key template-name
    enum template-name
```

though that requires adding template-name to the grammar.

During the ensuing discussing, someone suggested writing the grammar something like

```
elaborated-type-specifier:
    class-key identifier-thingie
    enum identifier-thingie
identifier-thingie:
    class-name
    enum-name
    identifier
    typedef-name
```

Several committee members expressed their desire to keep the grammar as small as possible.

Straw vote: Who favors adding the proposed new grammar rules: lots no.

Roskind explained that C++ has no notation for referring to the name of a nested class or struct hidden by a class member of the same name. He began with a simple example showing the need for elaborated names. Given

```
struct S { int a; } S;
```

since *S* refers to an object, you must use *struct S* to refer to the type-name.

Now, suppose you write

```
struct outer {
    struct inner {
        int a;
    } inner;
} outer;
```

How do you refer to the *inner* as a *typedef-name*? The problem is that the data member *inner* hides the type member *inner*. The WG presented the committee with three alternatives for accessing type *inner* within *outer*:

1. struct outer::inner
2. outer::struct inner
3. no new syntax

Roskind explained that (1) and (2) are new syntax, and not permitted by the June '92 Working Paper. Thus, if the committee chooses (3) there remains no way to refer to *inner* as a type member of *outer*. Roskind also noted that the Borland and MetaWare C++ compilers currently support notation (1).

Pennello (from MetaWare) explained that if you choose (2), you allow *struct x::struct y::struct z*. Choosing (1) means you only allow the one occurrence of *struct* at the beginning, and it applies to *z*.

Straw vote: Who favors no change: 5 yes, lots no, 5 abstain.
Straw Vote: Who favors (1)? lots yes, 0 no.

Krohn will write an analysis of this issue.

=== C Compatibility ===

Plum presented the following proposal: Treat C's "structure compatibility" rules as pragmatic layout requirements, but don't add any complications to the "one-definition rule" for type agreement.

He said the introduction to Chapter 9 of the Working Paper contains these words:

> The mechanisms for controlling the layout of class objects, for conforming to externally imposed formats, and for maintaining compatibility with C layouts (structs, unions, and bit-fields) are presented.

He suggested adding these words to 8.4.1p10 (Aggregates):

> A *C-struct* is an aggregate which is declared struct and contains no references, and contains no pointers to members. (Thus, its definition is valid in C.)
>
> A *C-union* is an aggregate which is declared union and contains no references, and contains no pointers to members. (Thus, its definition is valid in C.)

He noted these additions are similar to words in 17.4.1.5p4.

Several members suggested adding or removing features from "C-struct". Stroustrup suggested calling "C-struct" something else that did not convey the impression that the proposed struct layout rules were only for compatibility with C. Saks urged defining a C-struct "bottom up" through the type system. He also said that "C-struct" is the right name, because the purpose of Plum's proposal is to preserve compatibility with C. Vilot agreed with Saks about defining C-struct through the type system, but disagreed that we should limit our concerns to C compatibility. Stroustrup elaborated on his reason for wanting a name other than C-struct -- much of the email he receives is from people who want their C++ to communicate with Fortran.

Plum presented new words he proposed to add to the Working Paper in 9.2p7:

1. If two types *T1* and *T2* are the same type, then *T1* and *T2* are *layout-compatible* types.

2. Two C-struct types are layout-compatible if they have the same number of members, and corresponding members (in order) have layout-compatible types.

3. Two C-union types are layout-compatible if they have the same number of members, and corresponding members (in any order) have layout-compatible types.

4. Two enumeration types are layout-compatible if they have the same sets of enumerator values.

5. If a C-union contains several C-structs that share a common initial sequence, and if the C-union object currently contains one of these C-structs, it is permitted to inspect the common initial part of any of them. Two C-structs share a common initial sequence if corresponding members have layout-compatible types (and, for bit-fields, the same widths) for a sequence of one or more initial members.

Shopiro wanted greater control over storage layout. Plum reminded the committee that the WG's charter is simply to provide behavior for C source compiled as C++.

6. A pointer to a C-struct object, suitably converted, points to its initial member (or if that member is a bit-field, then to the unit in which it resides) and vice versa. There may therefore be unnamed padding within a C-struct object, but not at its beginning, as necessary to achieve appropriate alignment.

Schwarz said he'd like to see an analysis of whether this affects the legality of any C++ program.

Plum asked if anyone representing a vendor knew reasons why a C++ compiler would want to lay out C-structs and C-unions differently that they would in C? Shopiro said he didn't know of any.

Vilot asked if we can have a type system and structural equivalence in the same language. Stroustrup said no, but we can have layout requirements outside the type system.

Straw Vote:
Who thinks this a good proposal, except maybe for the terminology? 21.
Who thinks it needs more work? 6.
Who thinks it's not right at all? 0.
Who thinks it's none of the above? 11

Knuttila questioned the merits of Plum's rules 5 and 6. He said his compiler doesn't implement them because they hurt code generation. He wanted to see what class of C++ programs this supports.

Schwarz expressed concern over the impact of the proposal on the type system. He said rule 6 makes some previously invalid conversions valid. For example, rule 6 permits converting a pointer to a struct beginning with *int* into a pointer to *int*. Saks said he didn't see how layout compatibility hooks into the type system to permit conversions that previously were not allowed.

=== Editorial ===

Waggoner presented wording for a motion to approve the Working Paper. That motion specifically excluded approval for the phrase "or object" in 7.4p5.5. O'Riordan, who had noted the problem in that sentence on Monday, said he thought the problem was not just "or object" (which

Shopiro had added), but the whole sentence. He suggested just approving the entire Working Paper, and noting an action item for Shopiro to fix the sentence. No one objected.

=== Environments ===

Stone discussed the WG's latest ideas on the order of static initialization. The WG consolidated Kearns' proposals 1 through 4 (91-0137 = N0070) into the following "phases" of initialization for static objects within a single translation unit:

0. "Zero" -- All fundamental objects, pointers, and member pointers, including those inside class objects and aggregates, are set to zero cast to the appropriate type.
1. "Constant" -- All fundamental objects, pointers, member pointers, and references, including those inside class objects and aggregates with constant expression[*] initializers are initialized.
2. "Run-time" -- Within each translation unit all objects initialized with non-constant expressions[*] or with a constructor are initialized in definition order. The elements of an array are initialized in order of increasing subscripts, and members of a class are initialized in order of declaration.

[*] The June '92 Working Paper defines constant expressions to exclude floating constants and addresses.

These proposed phases leave the order of initialization between translation units unspecified.

Koch presented this example:

```
extern int x;
int y = x + 2;
int x = 10;
```

and explained that performing the phases as if they were distinct steps in order initializes *y* with 12.

Becker wanted to be sure the phases are described "as if" they occurred in this order, not that they actually happen in this order.

Rabinov wondered if a class with an inline constructor would be initialized in phase 1 or 2. O'Riordan said he thought the current definition of constant expression does not treat constructor calls as constant expressions even if inlined.

Koch wondered if these phases reflect existing practice. O'Riordan replied that current implementations typically perform phases 0 and 1 when loading the program, then phase 2 at the start of execution. Swan said the sections of the Working Paper (8.4 and 12.6) that describe static initialization don't say so explicitly.

Chapin said the phases were meant to describe what is commonly done in most implementations, as well as clarify the draft. O'Riordan agreed that these phases described existing practice.

Roskind encouraged Koch to poll the vendors in the room to see if these phases indeed reflect existing practice.

Shopiro suggested we had the option to declare a collection of programs that nobody ever writes to have undefined behavior, and by so doing, give vendors freedom to choose among a variety of implementation techniques. He cautioned that specifying defined behavior for all programs no matter how weird they are constrains vendors substantially.

Charney asked for rules that specify static initialization order across source files. Stone said the WG deferred that issue. He added that he saw no way to guarantee the ordering unless the programmer can specify it explicitly, but that requires a language extension.

Saks agreed with Roskind -- that the WG should take this opportunity to assess what existing practice really is. He added that defining heretofore undefined behavior may impose a burden on implementors, but leaving the initialization order unspecified places a greater burden on programmers trying to write portable code.

Yaker suggested that polling vendors about existing practice should be done over the e-mail reflector.

O'Riordan asked how demand load libraries fit into the proposed static initialization phases. Koch said the phases do not cover demand load libraries.

O'Riordan noted that phases 0 and 1 omit any mention of static objects appearing inside functions. Koch said the WG deferred this issue.

Straw vote: Who approves of the general direction taken by the Environments WG? lots yes.

Recessed at 6:10 Thursday and reconvened at 8:30 Friday.

10 General Session

=== Environments ===

Chapin said the WG was considering another strategy for static initialization within a translation unit:
-- Initialize each static object in order as it appears in the source.
-- The value of a static initialization expression that refers to an uninitialized object is undefined.

Chapin used Koch's example from Thursday:

```
extern int x;
int y = x + 2; // 1
int x = 10;
```

Under this alternative initialization strategy, *x* referenced on //1 is not initialized yet, so the value of *y* is undefined. Chapin said this strategy allows implementors the freedom to initialize constants first.

Chapin gave another illustrating the difference between the initialization phases and "in order" initialization:

```
int b[] = {b[1], 100, b[0]};
```

Using the initialization phases you get:

```
step 0: int b[] = {0, 0, 0};
step 1: int b[] = {0, 100, 0};
step 2: int b[] = {100, 100, 0};
        int b[] = {100, 100, 100};
```

With "in order" initialization, the result is undefined.

Chapin said the WG also discussed the order of static initialization among translation units. They considered three possibilities:
1. Make the compiler determine the order, via static analysis
2. Do initialization on demand, via dynamic analysis
3. Let the programmer specify the order

The WG thought (1) asked too much of compiler writers and (2) required too much run-time overhead. So the WG leaned toward an extension to allow programmers to specify the order of initialization across translation units.

Chapin presented this example of one possible language extension:

```
// header.h
after "mod1"
extern void f();

----------

// main.c - application code
#include "header.h"

// f() called in a static constructor somewhere here...

main() { ... }

----------
```

```
// mod1.c - library code
name "mod1"
#include "header.h"

T obj;   // obj must be initialized before calling f()

void f()
    {
    ...
    // use obj;
    ...
    }
```

Here, static initialization in the module containing

```
name "mod1"
```

must be done before the static initialization in the module containing

```
after "mod1"
```

Vilot noted this feature is similar to that being considered by the Extensions WG for name-space control. Chapin said the Environments WG should consider working with the Extensions WG on this.

Chapin said the proposal in 91-0073 about the one-definition rule (ODR) is "almost right". The document needs some statement to the effect that "two definitions in separate translation units are not the same unless they are token identical." He gave this example:

```
/* file1.c */

struct X {
    int member;
};

----------

/* file2.c */

typedef int T;
struct X {
    T member;
};
```

Chapin said this code violates the ODR because the structs are not token identical. However, this is perfectly legal C. Becker observed that this problem arises much more frequently in using templates.

Chapin briefly described some other issues in enforcing the ODR for templates. Shopiro suggested that the discussion of the ODR for templates should continue on both the Environments and Extensions email reflectors.

=== Drafting Committee ===

Saks presented the motions drafted by the drafting committee. He explained that the motion to make *wchar_t* a distinct type as presented to the drafting committee missed changes that should be made to the library section of the Working Paper. He thought it clearly indicated that no one had really examined the impact of this proposal on the library, but he doubted the omissions were grounds for defeating the motion.

Ward announced her intention to invoke the two-week rule because she was undecided about making *wchar_t* a keyword. She explained that the proposal distributed two weeks before the meeting (92-0047 = N0124) did not suggest making *wchar_t* a keyword. Roskind was also uneasy about making *wchar_t* a keyword. He didn't think the proposal captured the sentiments of the Core WG.

Vilot suggested approving the motion and assigning a work item for the Library WG to examine the proposal's effects on the library. Someone wanted the Extensions WG to look at the proposal. Bruck said he didn't know what the Extensions WG would do with it.

Plum didn't think this proposal had a serious negative impact on C compatibility. He explained that, several times during the development of the C standard, the Japanese requested making *wchar_t* a keyword.

Scian asked if *size_t* might also become a keyword designating a distinct type. Schwarz said he and Clamage looked at making *size_t* keyword, but decided against it.

Plauger said he didn't like the keyword, but this proposal is the right solution. He agreed that we didn't do all our homework, but we don't need to get it right before we vote it in.

Gray said this proposal was not discussed in the Core WG. The WG discussed the original paper (92-0047 = N0124) and found flaws, but they didn't see the proposal again until Clamage presented it to the full committee.

Lenkov, Plauger and Plum suggested that two-week rule could not be invoked here. However, Lenkov agreed to check with X3 for a better understanding of the rule.

Roskind wanted to put forward the original proposal from 92-0047 = N0124. Neither Clamage nor Pennello agreed.

Straw vote: Who thinks the proposal is ready for a formal vote? lots yes, 11 no.

About the motion to approve the Working Paper, Saks explained he and O'Riordan discussed the motion Thursday evening, and decided the problem in 7.4p5.5 was indeed the phrase "or object" added by Shopiro to the

June '92 Working Paper. O'Riordan said he was nonetheless willing to accept the Working Paper as is, as long as the minutes note that Shopiro has an action item to fix that sentence.

Straw vote: Who accepts the motion as is? lots yes.

Saks presented the wording for the motion regarding the scope of names first declared as friends. Yaker questioned the use of the phrase "the smallest non-class scope". Several suggested alternative wording, but the committee decided to make no change. Saks said that future motions should include definitions for terms that are part of the motion which should be added to the glossary of the Working Paper.

Lenkov closed the committee of the whole.

Motion by Druker/Waggoner: "Move we accept 92-0060 = N0137 as the current Working Paper."

Motion passed X3J16: 39 yes, 0 no.
Motion passed WG21: 5 yes, 0 no, 0 abstain.

Saks noted the following action items for Shopiro:

Clarification of the Working Paper:
- 7.4p5.5: fix it
- 9.2.1: item (3) regarding scope rules
- 11.5p1: regarding protected member access
- 12.4p4: regarding meaning of delete
- 12.4p10: regarding invocation of destructors

Refer to Extensions WG:
- 10.2p1: regarding accessibility and return type of virtual functions
- 15.5p2: regarding exception specifications, declarations and definitions

Motion by Bruck/Gibbons: "Move we change 13.4p6.1 from:

> An operator function must either be a non-static member function or take at least one argument of a class or a reference to a class.

to:

> An operator function must either be a non-static member function or take at least one argument of a class, a reference to a class, an enumeration, or a reference to an enumeration."

Motion passed X3J16: 39 yes, 0 no.
Motion passed WG21: 5 yes, 0 no, 0 abstain.

Motion by Pennello/Ward: "Move that we amend the Working Paper as follows:

3.2p1: Change the next-to-last sentence, which begins

> A name first declared ...

to

> A name first declared by a friend declaration belongs to either the global scope or a function scope; see 11.4. The name of a class first declared in a return or argument type belongs to the global scope.

9.1p2: Delete the last sentence, which begins

> If a class mentioned as a friend...

11.4p3: Change the whole paragraph to

> If a class or function mentioned as a friend has not been declared, its name is entered in the smallest non-class scope that encloses the friend declaration."

Pennello remarked that the Core WG recognizes that the last sentence in the change to 3.2p1 is probably wrong, but the WG should address it later. They haven't changed the draft in this regard. He also noted that the sentence prior to the sentence deleted from 9.1p2 points to 11.4p1, so there is no need to duplicate the sentence on friends.

Motion passed X3J16: 39 yes, 0 no.
Motion passed WG21: 5 yes, 0 no, 0 abstain.

Motion by Becker/Allison: "Move that we amend the Working Paper as follows:

2.8p1: Add *'wchar_t'* to the list of keywords.

2.9.2p4.2: Replace the entire sentence with

> A wide-character literal is of type *wchar_t*.

2.9.4p4.2: Replace the entire sentence with

> A wide-character string is of type array of *wchar_t*.

3.6.1p4: insert the following paragraph before 3.6.1p4:

> Type *wchar_t* is a type whose range of values can represent distinct codes for all members of the largest extended character set specified among the supported locales(_lib.locale). Type *wchar_t* has the same size, signedness, and alignment requirements as one of the other integral types.

4.1p1.1: add *'wchar_t, '* after *'A char, '*.

4.1p1.2: add *'wchar_t, '* after *', the char, '*.

4.1p1.4: replace 'If' with 'Except for type *wchar_t*, if'.

4.1p1.4: insert the following after 4.1p1.4:

> For *wchar_t*, if an *int* can represent all the values of the original type, the value is converted to an *int*; otherwise if an *unsigned int* can represent all the values, the value is converted to an *unsigned int*; otherwise, if a *long* can represent all the values, the value is converted to a *long*; otherwise it is converted to *unsigned long*.

17.4.10p1.1 change 'four' to 'three'.

17.4.10p2.1 change 'and *wchar_t* (both' to '('."

Ward/Scian: "Move that we table this motion by Becker/Allison."

Motion to table failed X3J16: 16 yes, 22 no, 0 abstain.
Motion to table failed WG21: 0 yes, 5 no, 0 abstain.

In response to concerns raised over having separate procedural votes for X3J16 and WG21, Lenkov said he will check with X3 on how to handle procedural issues.

Motion by Becker/Allison passed X3J16: 33 yes, 7 no.
Motion by Becker/Allison passed X3J16: 5 yes, 0 no, 0 abstain.

Lenkov opened the committee of the whole.

=== Core ===

Lajoie explained the WG's recent discussions on the lifetime of temporaries. They considered three possible timings for the destruction of temporary objects:
1. immediately
2. at end of statement (EOS)
3. at end of block (EOB)

However, they discounted (1) because it had previously been shot down by the full committee at the Dallas meeting.

Lajoie summarized the WG's goals for the lifetime of temporaries:
1. The following must work:

```
const char *p = s + t;
printf("%s\n", p);
```

   where *s* and *t* are string objects with implicit conversion to *char* *.
2. For expressions *e1*, *e2*, and *e3*

```
e1, e2; e3;
```

   must be functionally equivalent to

```
e1; e2; e3;
```

3. The solution must be consistent with other features of C++.
4. The solution must make programs written by "ordinary-non-expert" users work.
5. The solution must be simple for programmers to explain, i.e., it must be "teachable".
6. The solution must be efficient in terms of speed and space.
7. Programmers should have some control in destroying temporaries.

Winder said that some members of the Core WG felt they shouldn't try too hard to make goal 1 work. Schwarz suggested that goal 2 should be

2. For expressions *e1*, *e2*, and *e3*

```
e1, e2; e3;
```

must be functionally equivalent to

```
{ e1; } e2; e3;
```

Lajoie reported that the WG considered current practice, but found that every vendor handled temporaries differently, so that was no help.

Lajoie said the WG identified five control flow constructs under which managing temporaries is a problem: ?:, ||, &&, *goto-label*, and *switch*.

They also considered two possible policies for managing temporaries created in the branch of a control flow statement:
1. The compiler must "remember" if a temporary was created and destroy it only if necessary. (Run-time flags are one possible implementation of this policy.)
2. The compiler treats temporaries like variables. That is, if control flow jumps over the initialization of a temporary, it's a compile-time error.

The WG came up with four possible solutions by combining these two policies each with EOS and EOB:
1. "EOS with flags" -- the compiler must generate flags for ?:, ||, and && to insure temporaries are destroyed.
2. "EOB with flags" -- the compiler must generate flags for ?:, ||, &&, *goto* and *switch* to insure temporaries are destroyed.
3. "EOS with no flags" -- jumping over the creation of temporaries with destructors in ?:, || or && is a compile-time error.
4. "EOB with no flags" -- jumping past the creation of temporaries with destructors in ?:, ||, &&, *goto*, and *switch* is a compile-time error.

Plum noted that the Working Paper says (in 12.2) that "There are only two things that can be done with a temporary: fetch its value..., or bind a reference to it." He said he thought many WG members had the opinion that the statement is misleading. It should include as a third choice "take the address of part of a temporary".

Lajoie explained that the WG was concerned that the "EOB with flags" approach was difficult to implement and ran slowly, and that exception handling only made the concerns worse. They also had concerns about "EOB with no flags", which was supposed to be conceptually simple because it treats temporaries just like variables. But temporaries can be created in the middle of expressions, whereas variables can only be created in declarations. If the WG decided to treat expressions that create temporaries like declarations, then they would have to severely limit the use of jumps in C++ programs. On the other hand, if they created additional rules to permit jumping past some, but not all expressions, then the "EOB with no flags" approach loses its conceptual simplicity.

The "EOB with no flags" approach also raises portability concerns. If C++ translators must issue diagnostics for jumping past the creation of a temporary, the standard must specify exactly where temporaries are created and where they might be destroyed. A related question is: Should C++ allow jumping past the creation of a temporary that has no constructor and destructor. If so, does a C-style struct have a default constructor and destructor?

Lajoie said that for all these reasons, the WG is leaning toward EOS. But they have more work to do before they decide. She summarized the assignments for the WG members:

1. The impact of flags models on run-time performance of applications, including in presence of exception handling - O'Riordan
2. A summary of issues on lifetime of temporaries - O'Riordan and Pennello
3. The meaning of "implicit initialization" and the meaning of default constructors and destructors for C-style structs - Pennello
4. An analysis on different kinds of temporaries (e.g., dangerous vs. safe) - Kendall
5. The implicit scope of a case statement - Adamczyk

Redelmeier asked why the WG discounted immediate destruction. Pennello said the WG was persuaded against it by Koenig's killer example (the function passthru) described in 92-0020 = N0098.

Bruck asked if the WG had considered early creation of temporaries, as in

```
void f(string s, string t)
{ // creating temporary string for s + t here???
    if (...) {
        char *p = s + t;
        // ...
    } else {
        char *q = s + t;
        // ...
    }
}
```

Lajoie said they had not.

=== Libraries ===

Vilot resumed Thursday's discussion of whether *::operator new* should throw an *xalloc* exception rather than return null when the allocation fails. He said that the primary effect of the change would be:
-- existing "erroneous" programs (those that do not check that new returns null) would become safer
-- existing "correct" programs that test for a null return and abort, would continue to work, but the existing code that tests for null would become extraneous.
Plauger noted that the change only affects programs that check for null and DON'T shut down. They will have to be rewritten. The WG was generally of opinion that the change was worth the bother.

Arguing that *::operator new* should continue to return null on failure, O'Riordan gave the following example. Suppose you rewrite

```
bar() {
    char *p = 0;
    p = new char[128];
    foo();
    delete [] p;
}
```

as

```
bar() {
    char *p = 0;
    try {
        p = new char[128];
    } catch (xalloc) { }
    foo();
    delete [] p;
}
```

to accommodate the possibility that *new* might throw an *xalloc*. Now, you call *bar*, and the *new* expression in *bar* succeeds. *bar* proceeds to call *foo*. *foo*, whatever it is, throws an *xalloc*. But, *bar* doesn't catch it, because the call to *foo* is outside the try block. So *p* is discarded without destroying **p*.

Becker and Druker said they realized that introducing exceptions changes the architecture of programs. Druker added that Stroustrup and Koenig made that point in their original exception handling paper.

Vilot briefly summarized the WG's thinking on the string class. He said the WG was considering splitting the string class into more than one class with different comparison operators. When he asked the full committee how they felt about this, most said they agreed with splitting up the class.

Steinmuller said the WG was close to agreement on low-level memory management features for the string class. However, there were still questions about higher level functionality. Vilot said Becker had volunteered to work on a "text" class with higher level string functionality.

Straw vote: Who wants Becker to proceed? lots yes.

Plauger said that the normative addendum to the ISO C standard will be distributed in an upcoming mailing. He said we will need to consider the AFNOR C++ Expert's request to keep the C and C++ libraries synchronized.

Vilot listed the WG's deliverables for the next meeting:
-- 17 Library Introduction - Vilot
-- 17.1 Language Support - Vilot
-- 17.2 Strings - Steinmuller
-- 17.3 I/O - Schwarz
-- Letter to WG20 regarding *localedef* - Plauger
-- 17.5 Bits - Allison
-- class *text* - Becker

Saks asked what the WG did about the AFNOR C++ Expert's request. Vilot said he thought we were already doing what they asked, so there's nothing more to do. Plum said he thought they had asked SC22 to form an ad-hoc group to see that WG14 and WG21 are doing all they can to keep the libraries synchronized.

Lenkov closed the committee of the whole.

11 New business (if any)

None.

12 Review of the meeting

12.1 Review of decisions made and documents approved

See Appendix B.

12.2 Review of action items

Lenkov reviewed the action items.

13 Schedule of mailings and volunteers to help

Ward announced that Tektronix won't do the post-Portland mailing, so we need another volunteer.

Plum said he'd heard of problems with past mailings in Europe. Siemens Nixdorf offered to handle the mailing within Europe for the post-Toronto and pre-Boston mailings.

14 Plans for the future

14.1 Agenda items for the next meeting

Saks said he did not want one general session, as had been the practice. He wanted three distinct general sessions in this order:
1. for issues to be discussed in anticipation of a formal vote
2. for issues about which WGs would like to hold straw votes
3. for WG presentations for information purposes only

Schwarz asked for a separate agenda item on *iostreams*. Plum said we need scheduled time for national delegation caucuses.

14.2 Technical sessions for the next meeting

Schwarz requested that we hold no technical sessions featuring outside speakers.

Straw vote: Who wants to discontinue outside technical speakers? lots yes, 2 no.

14.3 Day Schedule for the Boston Meeting

Lenkov asked if we should change Friday's meeting time from the normal 08:30 to 12:30. No one had an opinion.

Gibbons wanted to be sure we did not schedule any votes past the scheduled adjournment time.

14.4 Confirming hosts for the next three meetings

Plauger summarized meeting dates for the next three meetings:
-- Nov 1-6, 1992 in Boston, MA, hosted by OSF
-- March 7-12, 1993 in Portland, OR, hosted by Tektronix, Mentor Graphics, and Sequent
-- July 11-16, 1993 in Munich, Germany, hosted by Siemens Nixdorf

15 Adjournment

The committee thanked Lajoie and Knuttila of IBM for hosting the meeting.

Motion by Plauger/Budge: "Move we adjourn."

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 5 yes, 0 no.

The committee adjourned at 12:33.

Appendix A - Attendance

Name	Affiliation	M	Tu	W	Th	F
Pieb, Wolfgang	Amdahl	V	V	V		
Coleman, Kim	Apple	A	A	A	A	A
Rabinov, Arkady	Apple Computer	V	V	V	V	
Shopiro, Jonathan	AT&T	V	V	V	V	V
Stroustrup, Bjarne	AT&T Bell Labs		A	A	A	
Krohn, Eric	Bellcore	V	V	V	V	V
Becker, Pete	Borland International	V	V	V	V	V
Swan, Randall	C-Team	V	V	V	V	V
Dovich, Steven J.	Cadence Design Systems	A	A	A	A	
Friedenbach, Ken	Cadence Design Systems	V	V	V	V	V
Kendall, Sam	CenterLine Software	A	A	A	A	A
Druker, Samuel	Cognex	V	V	V	V	V
Kohlmiller, Paul	Control Data Systems	V	V	V	V	V
Holly, Mike	Cray Research	V	V	V	V	V
Stump, Mike	Cygnus Support	A	A	A	A	A
Kelley, David	Data General	V	V	V	V	V
Allison, Chuck	DECUS	V	V	V	V	V
Winder, Wayne	Digital Equipment	V	V	V	V	V
Adamczyk, Steve	Edison Design Group	A	A	A	A	A
Anderson, Mike	Edison Design Group	A	A	A	A	A
Yamaryo, Masakazu	Fujitsu	V	V	V	V	V
Dewhurst, Steve	Glockenspiel			A	A	A
Lenkov, Dmitry	Hewlett-Packard	V	V	V	V	V
Greg Colvin	I. H. S.	A	A	A	A	A
Choi, Ernest	IBM		A			
Knuttila, Kim	IBM	V	V	V	V	V
Lajoie, Josee	IBM	A	A	A	A	A
Nelson, Clark	Intel	V	V	V	V	V
Roskind, Jim	James Roskind Computing	V	V	V	V	V
Munch, Max	Lex Hack & Associates	A			A	
Schwarz, Jerry	Lucid	V	V	V	V	V
Yaker, Laura	Mentor Graphics	V	V	V	V	V
Pennello, Tom	MetaWare	V	V	V	V	V
Gray, Jan	Microsoft	A	A	A	A	A
O'Riordan, Martin	Microsoft	V	V	V	V	V
Simonsen, Karl	Microsoft		A			
Redelmeier, D. Hugh	Mimosa Systems	A	A	A	A	A
McLay, Michael	NIST	V	V	V	V	
Boreham, Tim	Nutat Technologies	A				
Vilot, Mike	ObjectWare	V	V	V	V	V
Beech, David	Oracle	A	A	A	A	
Stone, Paul	Perennial	V	V	V	V	
Plum, Thomas	Plum Hall	V	V	V	V	V
Charney, Reg	Program Conversions	V	V	V	V	V
Wilcox, Tom	Rational	A	A	A	A	A
Saks, Dan	Saks & Associates	V	V	V	V	V
Budge, Kent G.	Sandia National Laboratory	V	V	V	V	V
Koch, Gavin	SAS Institute	V	V	V	V	V
Tooke, Simon	SCO Canada	V	V	V	V	V
Kiefer, Konrad	Siemens AG	V	V	V	V	V

Hartinger, Roland	Siemens Nixdorf	V	V	V	V	V
Steinmueller, Uwe	Siemens Nixdorf	A	A	A	A	A
Sloane, Alan	Sun Microsystems	V	V	V	V	V
Bruck, Dag	Swansea Agility Team	V	V	V	V	V
Gibbons, Bill	Taligent	V	V	V	V	V
Meyers, Richard	Taligent	A	A	A	A	A
Clamage, Steve	TauMetric	V	V	V	V	V
Ward, Cynthia	Tektronix	V	V	V	V	V
Traughber, Tom	Texas Instruments	V	V	V	V	V
Rafter, Mark	UK (Parallax)	V	V	V	V	V
Dodgson, David	Unisys	V	V	V	V	V
Houck, Christopher	Unisys	A	A	A	A	A
Waggoner, Susan	US West	V	V	V	V	V
Zeiger, Paul	US West	A	A	A	A	
Chapin, Peter	Vermont Technical College	V	V	V	V	V
Scian, Anthony	Watcom	V	V	V	V	V
Welch, James	Watcom	A	A	A	A	A
Dawes, Beman		A	A	A	A	A
Nikolaidis, Peter		A	A	A	A	A
Plauger, P. J.		V	V	V	V	V
Price, Philip		A	A	A	A	A
Terribile, Mark		A	A	A	A	A
Total Attendance		67	69	68	67	64
Total Voting Members		45	45	45	43	41

Mark: V = voting; A = attending (not voting)

Appendix B - Motions Passed

1. Motion by Shopiro/Bruck: "Move we approve 92-0041 = N0118 with these corrections as the minutes of the previous meeting."

   1. On page 1, change the date at the top center "1992".

   2. On page 1, in the first sentence of item 1, delete "by Lenkov".

   3. On page 11, in the first sentence of the paragraph beginning with "Koenig explained", change "than" to "that".

   4. On page 12, in the last sentence in the paragraph beginning "Plauger thought", change "determined prior" to "determined by prior".

   5. On page 15, add *: public X* to the declaration of *Y* in the code fragment.

   6. On page 18, in the first sentence of the third paragraph from the end, change "produce a safe" to "produce safe".

   7. On page 21, in the third sentence of the second paragraph, change "in style" to "in a style".

   8. In Appendix B, add the motion by Plum/O'Riordan (from page 25) to accept the Working Paper as item 4 and renumber all subsequent motions.

   Motion passed X3J16: lots yes, 0 no.
   Motion passed WG21: 5 yes, 0 no.

2. Motion by Druker/Waggoner: "Move we accept 92-0060 = N0137 as the current Working Paper."

   Motion passed X3J16: 39 yes, 0 no.
   Motion passed WG21: 5 yes, 0 no, 0 abstain.

3. Motion by Bruck/Gibbons: "Move we change 13.4p6.1 from:

   > An operator function must either be a non-static member function or take at least one argument of a class or a reference to a class.

   to:

   > An operator function must either be a non-static member function or take at least one argument of a class, a reference to a class, an enumeration, or a reference to an enumeration."

   Motion passed X3J16: 39 yes, 0 no.
   Motion passed WG21: 5 yes, 0 no, 0 abstain.

4. Motion by Pennello/Ward: "Move that we amend the Working Paper as follows:

3.2p1: Change the next-to-last sentence, which begins

> A name first declared ...

to

> A name first declared by a friend declaration belongs to either the global scope or a function scope; see 11.4. The name of a class first declared in a return or argument type belongs to the global scope.

9.1p2: Delete the last sentence, which begins

> If a class mentioned as a friend...

11.4p3: Change the whole paragraph to

> If a class or function mentioned as a friend has not been declared, its name is entered in the smallest non-class scope that encloses the friend declaration."

Motion passed X3J16: 39 yes, 0 no.
Motion passed WG21: 5 yes, 0 no, 0 abstain.

5. Motion by Becker/Allison: "Move that we amend the Working Paper as follows:

2.8p1: Add *'wchar_t'* to the list of keywords.

2.9.2p4.2: Replace the entire sentence with

> A wide-character literal is of type *wchar_t*.

2.9.4p4.2: Replace the entire sentence with

> A wide-character string is of type array of *wchar_t*.

3.6.1p4: insert the following paragraph before 3.6.1p4:

> Type *wchar_t* is a type whose range of values can represent distinct codes for all members of the largest extended character set specified among the supported locales(_lib.locale). Type *wchar_t* has the same size, signedness, and alignment requirements as one of the other integral types.

4.1p1.1: add *'wchar_t, '* after *'A char, '*.

4.1p1.2: add *'wchar_t, '* after *', the char, '*.

4.1p1.4: replace 'If' with 'Except for type *wchar_t*, if'.

4.1p1.4: insert the following after 4.1p1.4:

For *wchar_t*, if an *int* can represent all the values of the original type, the value is converted to an *int*; otherwise if an *unsigned int* can represent all the values, the value is converted to an *unsigned int*; otherwise, if a *long* can represent all the values, the value is converted to a *long*; otherwise it is converted to *unsigned long*.

17.4.10p1.1 change 'four' to 'three'.

17.4.10p2.1 change 'and *wchar_t* (both' to '('."

Motion by Becker/Allison passed X3J16: 33 yes, 7 no.
Motion by Becker/Allison passed X3J16: 5 yes, 0 no, 0 abstain.