Document Number: WG21/N0661
X3J16/95-0061
Date: 30 May 1995
Project: Programming Language C++
Reply to: Dan Saks
dsaks@wittenberg.edu

X3J16 Meeting No. 17
WG21 Meeting No. 12
5 - 10 March 1995

Radisson Hotel
Austin, TX 78701 USA

1 Opening activities

Lenkov convened the meeting as chair at 09:00 (CST) on Monday, 6 March 1995. Lajoie was the vice-chair, and Saks was the secretary.

Motorola (represented by Wood) hosted the meeting.

1.1 Opening comments

1.2 Introductions

Saks circulated an attendance list each day, which is attached as Appendix A of these minutes. Lajoie circulated a copy of the membership list (SD-2 = 95-0001R0) for members to make corrections.

1.3 Membership, voting rights, and procedures for the meeting

Lenkov reminded the attendees that this is a co-located meeting of WG21 and X3J16. (The joint membership is denoted WG21+X3J16 in these minutes.)

Lenkov explained the voting rules:

-- In straw votes, all WG21 technical experts may vote, even those who haven't attended previous WG21 meetings. An X3J16 attendee may vote only if he/she is the voting representative of a member organization that has met X3's attendance requirements (N0609 = 95-0009). (The voting representative is the principal member, or an alternate if the principal is not present.) A WG21 technical expert who is also an X3J16 voting member still casts only one vote in a straw vote.

-- 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, and only then if the organization meets X3's attendance requirements.

1.4 Distribution of documents not distributed before the meeting

1.5 Approval of the minutes from the previous meeting

Saks submitted the minutes from the previous meeting (N0597 = 94-0210) for approval.

Motion by Saks/Dawes:

> Move we approve N0597 = 94-0210 as the minutes of the previous meeting.

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

1.6 Agenda review and approval

Lenkov submitted the proposed agenda (N0610 = 95-0010) for approval with this addition:

4.2 Call for Volunteers to Help the Project Editor

Motion by Saks/Bruck:

> Move we accept N0610 = 95-0010 as amended as the agenda for this meeting.

Motion passed X3J16: lots yes, 0 no, 0 abstain.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

1.7 Report on the WG21 Sunday meeting

Harbison summarized the highlights of Sunday's WG21 meeting. Six delegations attended: Canada, Germany, Japan, Sweden, the UK, and the US. He explained that WG21 discussed the comments on the CD registration ballots that came from national bodies (NBs) that normally don't attend. The WG21 members who were present generally agreed that:
-- Although some NBs suggested forming a separate ad hoc WG to clarify fundamental language concepts, the Core WG(s) are already handling the issues properly.
-- WG21 should not split the C++ library from the C++ language as a separate work item.
-- WG21 should not restructure the library (all parts should remain normative), although the library might need more explicit support for freestanding environments.

Harbison conveyed Stroustrup's suggestion to disband the Extensions WG at end of this meeting and distribute the lingering extensions work to the Core WG(s). He also explained that the editing schedule following this meeting is very tight. Any resolutions drafted at this meeting must provide detailed wording for changes to the draft.

Harbison said the CD will be available for public review via ftp. It will be available to WG21+X3J16 even before it's available to the public.

Plauger explained the administrative tasks that the WG21 convener (Harbison) must complete before the end of the meeting. He must:
-- write a "Disposition of Comments on CD Registration Ballot" to circulate within SC22, and
-- get permission from WG21 to submit the draft for CD ballot.

Lenkov appealed for volunteers to offer to help the project editor (Koenig) edit the draft.

1.8 Liaison reports

==== WG14 (ISO C) ====

Plum explained that WG14 met in Plano, TX USA in December 1994. He said the Technical Report (including proposals drafted by the former Numerical C Extensions Group) is complete and will be available from ANSI shortly. The C standard is now open for revision; the revised language (to be) is known unofficially as "C9x". Plum said the only non-controversial proposal for change is to add // comments.

1.9 New business

Lenkov said he sent out a letter ballot to X3J16 requesting authorization for X3 to bill members for mailing service. The ballots came back with 39 yes and 1 no. He said the ballot needed 46 yes votes (2/3 of the membership) to pass. In the hope of approving the ballot, he asked those members who had not replied to the ballot to reply in writing at this meeting.

X3J16 members discussed the ramifications of the vote. Lenkov explained the approving the fee doesn't you must pay it -- you pay the fee only if you want to receive paper copies of the mailings.

1.10 Drafting Committee

Lenkov explained the role of the drafting committee: to prepare a written statement of the formal motions so that voting members (particularly those whose native language is not English) have an opportunity to read and understand what they are voting on.

Lenkov said that Saks (as secretary) is always on the drafting committee. For this meeting, Corfield, Hartinger, and Unruh will also participate. Also, each ad hoc working group (WG) chair is responsible to bring that WG's motions to, and participate in, the drafting committee. Lenkov added that each chair may delegate the job to another WG member, but the chair remains responsible.

Gafter also volunteered for the drafting committee.

The committee recessed to WGs at 10:20 on Monday.

2 WG sessions

3 WG sessions

The committee reconvened at 08:35 on Wednesday.

4 Project Editor's report

Lenkov opened the committee of the whole.

Koenig reported that the current Working Paper (WP) (N0629 = 95-0029) incorporates all the resolutions passed in November, 1994 (at Valley Forge, PA), modulo mistakes and oversights. He explained that small parts of the library clauses are missing from the draft due to typesetting glitches. He said he corrected the errors in his online copy and he wrote an audit program to catch similar problems in the future. He asked committee to approve draft despite the minor glitches.

Koenig explained that he exercised his executive authority by making a substantive change to the description of bad_typeid (5.2.7) and its interaction with subscripting. In effect, the draft now says that:

-- If p is a pointer to a polymorphic type and p has value 0, then typeid(*p) throws bad_typeid even though the result of evaluating *p by itself would be undefined.
-- Similarly, typeid(p[i]) does not evaluate i.

Koenig acknowledged editing help from Adamczyk, Lajoie, Stroustrup, Plum, and Vilot. Vilot did all the editing in the library clauses. Koenig added that Appendix C still contains some items that should be moved to normative clauses; he has already moved them in his online copy of the WP.

Koenig announced that members of the public who wish to be notified about the US public review (when information is available) should send a note to c++std-notify@research.att.com. The content of the note is unimportant.

In response to a request for a list of the known errors in the WP as distributed in the mailing, Koenig explained that he doesn't have the tools to produce the list quickly. He offered to provide anyone with an updated Postscript file of the WP incorporating all the corrections.

Straw Vote: Who approves of N0629 = 95-0029 as the current WP? lots yes, 0 no, 1 abstain.

4.2 Call for Volunteers

Harbison explained that WG21+X3J16 should review the WP twice:
1. A "big" review during the week of 27-31 March, and
2. A "final" review during the week of 10-14 April

Harbison said Koenig will submit the document to SC22 on 14 April.

Adamczyk, Becker, Gibbons, Holaday, Lajoie, Myers, Scian, and Stroustrup volunteered to review the WP.

Koenig asked that review comments should be as specific as possible. He preferred that reviewers submit a diff patch of the troff version of the WP along with any comments. If that's not possible, comments should specify the replacement text and the clauses to which it applies. Bruck suggested that Koenig post a note to the -edit reflector explaining the form he would like for the patches.

5 General Session I

5.1 Core Language WG

==== Core (Adamczyk) ====

Adamczyk presented a proposal to relax restrictions on function overloading (N0631 = 95-0031). Specifically, the WG proposed eliminating the restriction that a program can't overload functions that differ only in that one has a parameter of type T and the other has a corresponding parameter of type T &. For example,

```
    void f(int);
    void f(int &);  // error according to WP; propose to allow
```

He explained that the restriction was intended to prevent declarations of functions that are indistinguishable and therefore always cause an ambiguity when called. However, this restriction doesn't cover all cases, and it causes grief when using templates and overloaded sets formed by using namespaces. Compilers will still catch any ambiguities at the call points. (The formal wording for the proposal appears as motion 2 later in these minutes.)

Straw Vote: Who favors this proposal? lots yes, 0 no, 0 abstain.

Adamczyk presented the WG's proposal to allow default arguments on overloaded operator functions. He explained the status quo:
-- The WP is unclear whether operator new allows default arguments
-- Some compilers allow default arguments on operator()
The WG recommended:
-- Any operator function may have default arguments
-- Operator functions may have any number of arguments

For example, the proposal would allow:

```
    struct A {
        int operator()(int, float, int = 5);
        A operator++(int = 0);
        A operator=(const A&, bool = false);
    };
```

Adamczyk explained that operator expressions can still call only those functions that can be called with one or two arguments. Other "weird" operator functions can only be called through an explicit call.

Adamczyk said there's a more conservative fall-back position that WG21+X3J16 might prefer, namely, to allow defaults only on operator new and operator().

Clamage asked if this proposal allows declarations for an operator with additional non-default arguments. Unruh asked if the requirement that at least one operand must have a class type will remain in effect. Adamczyk said yes to both Clamage and Unruh.

Pennello said his compiler (MetaWare) already behaves according to this proposal. Vilot thought this proposal was too much change this late in the standards process; he preferred the fall-back position. Saks and Shopiro were concerned that

```
    class X {
        operator+(X, X);    // missing friend
        ...
```

won't be caught at the declaration. Shopiro said it might not even get diagnosed at the call(s) if conversions allow the calls to match another binary function.

Straw Vote: Who favors this proposal? 20 yes, 25 no, 9 abstain.

Adamczyk explained the "fall-back" proposal again:
-- Clarify that operator new and operator() may have default arguments
-- Clarify that operators can't have extra arguments
(See motion 3.)

Straw Vote: Who favors this "fall-back" proposal? lots yes, 1 no, 10 abstain.

Adamczyk explained that the new bool type introduces potential ambiguities in overload resolution. For example, given:

```
    void f(void *);
    void f(bool);
    int *p;
    ...
    f(p);   // Newly ambiguous
```

The problem is not bool itself, but that the conversion from T * to bool is a standard conversion, so it applies in many places. Adamczyk said there's a similar problem with:

```
    class D : public B { ... };
    void f(B *);
    void f(bool);
    D *p;
    ...
    f(p);   // Newly ambiguous
```

The WG considered two possible solutions:
1. "Put the genie back in the bottle." Allow the conversion from T * to bool only where it was allowed previously. That is, don't consider conversion to bool as a general conversion, but just enumerate those places, such as if conditions, where conversion to bool may occur.
2. Change the cost of this conversion in overload resolution. Specifically, make conversion of anything to bool worse than any other standard conversion.

The WG recommended (2). Adamczyk said this is intuitive because it's the standard conversion that throws away the most information.

Koenig asked who'd ever overload f(bool) and f(T *). Schwarz said iostream does, for example, in operator<<.

Bruck suggested changing "anything" in (2) to "pointer or pointer to member". Adamczyk asked the WG members if they'd accept Bruck's suggestion. No one objected. Adamczyk changed the wording of proposal accordingly (see motion 4).

Straw Vote: Who favors this proposal as amended? lots yes, 2 no, 8 abstain.

Adamczyk proposed a "quasi-editorial" change to clause 12.2.3.2 on ranking standard conversion sequences. He said this change is probably what we meant to say, but didn't get the words right. See motion 5 for the details of the proposed change.

Straw Vote: Who favors this proposal? lots yes, 0 no, 3 abstain.

Adamczyk suggested removing clause 4.12 (regarding the standard conversion from a derived class type to its base class type). He said this section was added as an editorial change without committee vote. He explained that it was intended to support casts, but now that casts are defined in terms of initialization, this clause might not be necessary anymore. Adamczyk said he suggested removing the derived-to-base conversion to his Core WG, but the WG disagreed.

Adamczyk explained that if we removed the conversion, then the last line of this example becomes ill-formed:

```
    struct B { };
    struct D : B { };
    struct Z {
        operator D();
    } z;

    const D& dr1 = z;   // OK
    const B& br1 = z;   // OK
          D  vd  = z;   // OK
          B  vb  = z;   // OK with standard conversion; not OK without
```

If the standard conversion exists, the last line calls the constructor. If the constructor is not callable, the program is ill-formed.

[Note:
    After the meeting, Welch pointed out, and Adamczyk agreed, that the example above should have used () initialization instead of = initialization:

```
    const D& dr1(z);    // OK
    const B& br1(z);    // OK
          D  vd(z);     // OK
          B  vb(z);     // OK with standard conversion; not OK without
```

    The "=" form doesn't really have the problem under discussion.
End Note.]

Shopiro said he thought the last line in the example would be surprising to users because it creates an object and then "slices" it. He said he'd rather make users "slice" objects explicitly. Therefore, we should eliminate the conversion.

Straw Vote: Who wants to remove the derived-to-base conversion (clause 4.12) from the WP? 18 yes, 18 no, 13 abstain.

Adamczyk presented some problems regarding class name injection (N0444 = 94-0057):

Problem 1: It's inconsistent. The WP employs name injection, but only in some contexts:

```
    x.T::a
    p->T::a
    x.operator T
    p->operator T
    x.operator T::U
    p->operator T::U
    A::operator T
    A::operator T::U
    A() : T(x)
    Template classes
```

Problem 2: We won't know how to resolve a lot of core issues until we know if we have name injection. Name injection makes the answers easy.

Problem 3: Without name injection, names can be "hijacked", as in:

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

    struct B : public A {
        B* next;            // Oops, wrong B
        B(const B&);        // Oops, not a copy constructor
    };
```

Without name injection, B inside struct B (above) refers to A::B.

Adamcyzk recommended adopting uniform name injection as proposed in N0663R1 = 95-0063R1 (which includes N0444 = 94-0057, section 6, first option). The resulting simplification deletes about a half-dozen special cases from the WP.

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

Gafter explained another name lookup issue (from N0663R1 = 95-0063R1 section 2) using this example:

```
    struct T {
        enum E { e = 3 };
        static int x[e];
    };

    int T::x[e]; // Is e found?
```

He recommended that the answer is "yes"; names (such as e) following the qualified-id (T::x) are looked up in the scope of the class (T).

Straw Vote: Who favors this proposal? lots yes, 0 no, 7 abstain.

Gafter explained another name lookup issue (from N0663R1 = 95-0063R1 section 3) using this example:

```
    class X { };
    int X;
    class Y : X { }; // which X?
```

He recommended adding words to Clause 10 of the WP to say the base-class specifier is evaluated as a type.

Straw Vote: Who favors this proposal? lots yes, 0 no, 4 abstain.

Gafter presented yet another name lookup issue, this one regarding ambiguities in operator delete occurring in multiple inheritance lattices (N0663R1 = 95-0063R1, section 8). He presented words to resolve the problem (from the same paper). Scian noted that the proposed words account for

```
    operator delete(void *)
```

but not for

```
    operator delete(void *, size_t).
```

Gafter said the omission was not intentional and he would fix the words accordingly.

Straw Vote: Who favors this proposal? lots yes, 0 no, 7 abstain.

(Motion 6 incorporates the proposals approved in the previous five straw votes.)

==== Core (Lajoie) ====

Lajoie presented several proposal regarding C++'s memory model (from issues in N0568 = 94-0181).

Lajoie began by presenting wording to clarify fundamental concepts of the memory model, as suggested in items 1.1 through 1.5 of N0568 = 94-0181. This wording included definitions for the terms "object representation" and "value representation". See Motion 7 (which refers to N0670 = 95-0070) for details.

Regarding the proposed representation of signed and unsigned integer types, Koenig said there's a problem with the representation of ones-complement zero. Plum replied that the proposal describes the same behavior as in C. Lajoie explained that these words come from 3.1.2.5 of ANSI C, plus terms used in reply to defect reports against that section.

Straw Vote: Who favors this proposal? lots yes, 0 no, 8 abstain.

Lajoie presented the WG's recommendation regarding object lifetime issues (from N0655 = 95-0055). The first issue is "What can be done with a pointer to an object that has been destroyed?" The WG recommended

    A pointer to an object of type T that has been destroyed still points at valid memory.

For example:

```
    T* pt = new T();
    pt->~T();
    &pt;               // ok: pt points to valid memory
    void* q = pt;      // ok: pt points to valid memory
    pt->f();           // undefined: f is a non-static member function
```

Shopiro asked if referring to *pb is ok. Lajoie said it's an open issue.

Lajoie gave another example:

```
    class C {
        void f();
        void destroy();
    };

    void C::destroy ()
    {
        this->~C();
        f();            // undefined: f is a non-static member function
        void *q = this; // ok
    }
```

Lajoie presented the following proposed rule for pointer manipulations:

> When converting a T* to a void*, the void* is guaranteed to point at the start of the storage holding an object of type T.

Also:

> An object of type T that is a base subobject is not guaranteed to have the same size, layout and polymorphic behavior as a complete object of type T.

For example:

```
    class A { };
    class B : public virtual A { };
    class C : public virtual A { };
    class D : public B, public C { };
```

The size and layout for the D's base subobject C might be different from the size and layout of a complete C object.

Lajoie presented another proposal regarding object lifetimes:

> If the pointer in a placement new expression [using the library's placement new] has type T1* and the new expression creates an object of type T2, referring to the original object using the T1* pointer has the same effect as using the pointer after the object has been destroyed.

For example:

```
    class B {
        void mutate();
        virtual ~B();
    };

    class D1 : public B { };
    class D2 : public B { };

    void mutate(B** pb2, void *p) {
        (*pb2)->~B();
        new (p) D2;
    }

    void* p = malloc(...);
    B* pb1 = new (p) D1;
    mutate(&pb1, p);
    pb1->f();                   // undefined
    &pb1;                       // ok: valid storage
```

Another example:

```
    void B::mutate() {
        this->~B();
        new (this) D2;
        f();                    // undefined
        void *q = this;         // OK, valid memory
    }

    void* p = malloc(...);
    B* pb = new (p) D1;
    pb->mutate();
    pb->f();                    // undefined
```

Lajoie presented yet another proposal regarding object lifetimes:

A new expression creates an object of type T which has the size and the layout of a complete object of type T.

If the pointer in a new-placement expression has type T* and the pointer points to a complete object of (dynamic) type T and the new expression creates an object of type T, referring to the object T after the new expression has completed is well-defined.

Straw Vote: Who favors everything proposed since the last straw vote?
lots yes, 0 no, 5 abstain.

Lajoie presented another recommendation from the Core WG:

An object need not be destroyed before its memory is reused or released.

For example:

```
    {
        T t;
        new (&t) T;
    }                       // OK: destructor for T not called
```

Another example:

```
    {
        T t;
        new (&t) X;         // OK if t has enough storage to hold an X
        &t;                 // OK
        t.f();              // undefined: refers to a member of a T
    }                       // undefined: destroyed twice
```

Yet another example:

```
    {
        T t;
        new (&t) T;     // if this throws an exception, then...
    }                   // ... undefined behavior on block exit
```

And one more example:

```
{
    T t;
    new (&t) X;         // OK if t has enough storage to hold an X
    new (&t) T;
}                       // OK
```

Straw Vote: Who favors this proposal? lots yes, 1 no, 6 abstain.

(Motion 8 incorporates the proposals approved in the previous two straw votes.)

Lajoie presented the WG recommendation on initialization issues (from N0651 = 95-0051). Regarding initialization of objects with static storage duration, they proposed to specify that default initialization to zero:
-- initializes all scalar members of an object of class type to zero converted to the appropriate type (applied recursively for the scalar members of base and nested class members).
-- initializes pointers to members with the null member pointer value of that type.
Dynamic initialization occurs after default initialization to zero.

For example:

```
    class A {
        int i;
        float f;
    public:
        A() : i(88) { }
    };

    class C {
        int j;
        A a;
    };

    A a1;   // a1.f = 0.0
    C c;    // c.j = 0, c.a.i = 0 and c.a.f = 0.0
```

In this example above, c.j and c.a.i are initialized to 0, and a1.f and c.a.f are initialized to 0.0.

Lajoie presented the WG's recommendations on the meaning of an initial-izer of the form T(). She said the WG agreed that saying it has the same semantics as default initialization for a static object of type T is too expensive. The WG's compromise proposal is:
-- if T is a class type with a user-declared constructor then call it
-- otherwise, initialize the object as if it had static storage duration.

Gibbons wondered that, if initialization to zero is a good thing, why not do it always. Shopiro replied that initialization to zero can be expensive, and users might want to avoid it. This proposal gives users a way to turn off initialization to zero by writing an explicit default constructor.

Lajoie explained that the WP is already clear about initialization of scalars and classes with explicitly-declared constructors, but it's not clear about classes with implicitly-declared constructor. Anderson said this proposal also covers arrays, which are not already covered by other rules in the WP.

Lajoie gave this example:

```
    class X {
        float f;
    public:
        virtual void f();
    };

    X x;
```

Here, x.f is initialized to 0.0 and the X's implicitly-declared default constructor executes.

Lajoie then discussed the initialization in new T(). The WG recommended it should have the same semantics as T() for both class and non-class types. Thus, new T() may have different semantics than new T.

For example:

```
    new int();          // object initialized to 0

    typedef int I[3];
    int* p = new I();   // all array elements initialized to 0
```

Lajoie then discussed the meaning of t() in a mem-initializer, as in:

```
    template <class T> struct X {
        T t;
        X<T>() : t() { }
    };
```

She said the WG recommended that t() in a mem-initializer should have the same initialization semantics as T(). For example:

```
    struct A {
        int i;
    };
    struct B : A {
        B() : A() { }   // A::i initialized to 0
    };
```

The WG intends this to work for non-class types as well:

```
    struct Xi {
        int i;
        Xi() : i() {}    // i() is well-formed and initializes i to 0
    };
```

Winder asked about the behavior of a derived class with an explicit constructor where its base has only an implicit constructor. Gafter said it may zero some memory more than once. Welch said that initialization code might set an test flags to avoid duplicate initialization.

Straw Vote: Who favors this proposal? lots yes, 6 no, 6 abstain.

Lajoie presented new words to describe the initialization of const objects:

> Unless explicitly declared extern, a const object does not have external linkage and shall be initialized. That is, for a const object of type T, if T is a type with a user-declared default constructor, the T's default constructor is called; otherwise the const object of type T must be initialized with an explicit initializer.

Straw Vote: Who favors this proposal? lots yes, 0 no, 6 abstain.

Lajoie presented the WG's recommendation to allow brace elision for all aggregate initializations. Specifically,

> If the assignment-expression [in the aggregate initializer] can initialize a member (considering all type conversions), that member is initialized. Otherwise, if the member is itself an aggregate, then the initializer initializes the first member of the subaggregate member.

She gave this example:

```
    struct A {
        int i;
        operator int();
    };
    struct B { A a1; A a2; };
    A a;
    B b1 = { 1, a };        // initialializes b1.a2 with a
```

Straw Vote: Who favors this proposal? lots yes, 1 no, 6 abstain.

Lajoie presented.the WG's recommendation to allow redundant braces around scalar initializers, as in:

```
    int j = i;
    int j = { i };  // allowed in C and allowed in C++
```

The proposal stipulates that the expression initializing an object of scalar type can be optionally enclosed in braces only if the object is a complete object (i.e. not itself an aggregate member) of scalar type.

Straw Vote: Who favors this proposal? lots yes, 0 no, 3 abstain.

Lajoie presented yet another proposal regarding initialization:

> When an aggregate is initialized with an initializer clause, if some members are initialized with constant expressions and other members are initialized with dynamic initialization, an implementation may initialize the entire object during the dynamic phase of construc-

tion.

Straw Vote: Who favors this proposal? lots yes, 0 no, 4 abstain.

Lajoie presented the WG's final proposal regarding initialization:

> If there are fewer initializers in the list than there are members in the aggregate, then the remaining aggregate members are initialized with zero.

Straw Vote: Who favors this proposal? lots yes, 0 no, 3 abstain.

(Motion 9 incorporates the proposals accepted in the previous six straw votes.)

Lajoie presented the WG's recommendations regarding class copy operations (from issues listed in N0580R1 = 94-0193R1). She began with the WG's proposed definition for "copy assignment":

> A user-declared copy assignment operator, operator=, is a non-static member function of class X with exactly one parameter of type X, X& or const X&. If there is no user-declared copy assignment operator, one with exactly one parameter of type X& or const X& is implicitly declared.

For example:

```
class B { };
class C : public B {
public:
    C& operator=(const B&);  // not a copy assignment
};
```

Straw Vote: Who favors this proposal? lots yes, 0 no, 4 abstain.

Lajoie then recommended:

> The implicitly-declared copy assignment operator for class X calls the copy assignment operator for X's direct base classes and members. It is unspecified whether the subobjects representing virtual base classes are assigned more than once by an implicitly-declared copy assignment.

Straw Vote: Who favors this proposal? lots yes, 1 no, 7 abstain.

Schwarz asked if there are situations that inhibit generating a copy assignment, such as when a base class has a private copy assignment. Gafter said such situation inhibit generating a definition, but not a declaration. The copy assignment is always declared.

Lajoie also presented the following, which the WG considered as an editorial clarification:

> Because a copy assignment operator= is implicitly declared for a class if not declared by the user, a base class copy assignment operator= is always hidden by the copy assignment operator of a derived class.

She gave this example:

```
    struct B {
        virtual int operator=(int);
        virtual B& operator=(B&);
    };
```

```
    struct D : public B {
        int operator=(int);
        D& operator=(B&);
    };
    D obj1, obj2;
    B* ptr = &obj1;
    void f() {
        ptr->operator=(99);      // calls D::operator=(int)
        *ptr = 99;               // ditto
        ptr->operator=(obj2);    // calls D::operator=(B&)
        *ptr = obj2;             // ditto
    }
```

(Motion 10 incorporates the recommendations of previous two straw votes, plus the editorial clarification.)

Lajoie said the WG also recommended adopting the copy optimization proposal from N0641 = 95-0041. She gave this example from the paper:

```
    string s1 = "hello";
    string s2 = s1;
```

If s1 is not used again, the implementatin may treat s2 as an alias for s1. Similarly, if s2 is not used again, the implementation may eliminate the copy. In general, if the implementation can detect that either the original object or the copy won't be used again until it's destroyed, the implementation can instead create a reference to the original object. See motion 11.

Straw Vote: Who favors this proposal? lots yes, 0 no, 4 abstain.

5.2 Library WG

Vilot said the following people are responsible for collecting issues on the library clauses:

Clause 17:	McKenna	chrism@cadence.com
Clause 18:	Henricson	mats.henricson@eua.eric
Clause 19:	Vilot	mjv@objects.mv.com
Clause 20:	Myers	myersn@roguewave.com
Clause 21:	Wilhelm	rkw@chi.andersen.com
Clause 22:	Schwarz	jss@declarative.com
Clause 23:	Podmolik	jlp@chi.andersen.com
Clause 24:	Dodgson	dsd@tredysvr.tredydev.unisys.com
Clause 25:	Becker	pete@borland.com
Clause 26:	Ward	ward@roguewave.com
Clause 27:	Hinke	hinke@roguewave.com

He said problems in a particular library clause should be reported directly to the appropriate person.

Vilot said the Library WG discussed one proposal for handling exceptions in the library (N0606 = 95-0006) and decided not to support it. (The vote in the WG was 0 yes, lots no.) The WG also discussed exception specifications in the library (N0620 = 95-0020) and it remains open.

Vilot said the WG also considered a proposal to define the effect of exceptions thrown from destructors (N0623 = 95-0023) and approved it with the following revised wording:

> The effect of an exception propogating from the destructor, default constructor, or copy constructor of an object called by any standard library function is undefined. In standard containers and strings, the effect of the exception is not defined, but after such an exception, the effect of any operation on the container or string (including destruction) is undefined.

Steinmuller asked Vilot to add copy assignment to the list of special members functions in the first sentence. Vilot agreed.

Straw Vote: Who favors adding the above paragraph to subclause 17.3.3?
lots yes, 1 no, 6 abstain.

Vilot presented a proposal to integrate non-converting constructors into the library (N0604 = 95-0024, with minor changes). See motion 12.

Straw Vote: Who favors this proposal? lots yes, 0 no, 0 abstain.

Vilot proposed closing several open issues from Clause 18 [language support] as per N0649 = 95-0049, with minor changes. See motion 13.

Straw Vote: Who favors this proposal? lots yes, 0 no, 4 abstain.

Vilot said the WG deferred discussion on revising the language support exceptions (N0566R1 = 94-0179R1), and decided against changes in the organization of the headers (N0576 = 94-0178).

Vilot recommended changes in the memory allocation facilities of the standard library (N0647 = 95-0047). This proposal:
-- prohibits operator new from returning 0
-- adds another operator new with an exception specification that throws nothing
See motion 14 for details.

Vilot gave these examples:

```
    T *p = new T;           // never returns null
    T *q = new (nothrow) T; // never throws an exception
```

Straw Vote: Who favors this proposal? lots yes, 1 no, 11 abstain.

Vilot proposed numerous small changes to the STL clauses (N0614R1 = 95-0014R1, with minor changes). See motion 15.

Straw Vote: Who favors this proposal? lots yes, 0 no, 8 abstain.

Vilot said the WG discussed implementing STL allocators using partial specialization (N0619 = 95-0019), but most WG members abstained from voting on it. The WG decided against eliminating certain global function templates from clause 20 [general utilities] (N0639 = 95-0039).

Vilot presented a proposal to modify class auto_ptr and add counted_ptr as per N0589R1 = 94-0202R1 with the following changes:
-- Note that the data member 'px' is mutable.
-- Note that the copy constructor and assignment operator modify the argument 'r', even though it is declared as a reference to a const auto_ptr.

Stroustrup asked to split the discussion on changing auto_ptr from the discussion of adding counted_ptr.

Straw Vote: Who favors adding counted_ptr?
    WG21+X3J16: 8 yes, 18 no, 15 abstain.
    WG21 only: 2 yes, 3 no, 1 abstain.

Straw Vote: Who favors the changes to auto_ptr?
    X3J16 only: 34 yes, 3 no.
    WG21 only: 5 yes, 1 no.

See motion 16 for the final form of this proposal.

Vilot proposed closing a few open issues from Clause 21 [strings] as per N0616R1 = 95-0016R1 with minor changes. See motion 17.

Straw Vote: Who favors this proposal? lots yes, 0 no, 4 abstain.

Vilot presented a proposal to modify basic_string to make it usable as an STL sequence (N0628R1 = 95-0028R1). He explained that the paper recommended removing the existing interface and adding a new one. The WG agreed with adding the new interface, but advised leaving the existng one intact. See motion 18.

Straw Vote: Who favors the WG's proposal (adding a new interface and leaving the existing one intact)? lots yes, 4 no, 4 abstain.

Vilot presented a pair of proposals for minor repairs to Clause 22 [localization] as per N0601R1 = 94-0214R1 and N0625 = 95-0025 with a minor change. See motion 19.

Straw Vote: Who favors this proposal? lots yes, 0 no, 9 abstain.

Vilot proposed minor changes to Clause 23 [containers] to close a few open issues as per N0613R3 = 95-0013R3. See motion 20.

Straw Vote: Who favors this proposal? lots yes, 0 no, 1 abstain.

Vilot said the Library WG rejected a proposal to change container adapters (N0612 = 95-0012) by a vote of 6 yes, 15 no. They also rejected proposals to add hash tables to STL (N0652 = 94-0175 and N0605 = 945-0218) without any technical discussion.

Vilot presented a pair of proposals to modify Clause 24 [iterators] by relaxing requirements on the result of operator++(int) for iterators (N0621 = 95-0021) and fixing the streambuf iterator (N0622 = 95-0022). See motion 21.

Straw Vote: Who favors this proposal? lots yes, 0 no, 1 abstain.

Vilot presented a proposal to resolve a few open issues in Clause 26 [numerics] as per N0637 = 95-0037. See motion 22.

Straw Vote: Who favors this proposal? lots yes, 0 no, 6 abstain.

Vilot presented a proposal to remove basic_convbuf from the library as per N0664 = 95-0064 with changes. See motion 23.

Vilot explained that, with this change, you can no longer transfer a wchar_t in memory directly to or from a file. Schwarz said he thinks you can still do it, but only with extra work. Schwarz added that the advantage of this change is that it eliminates a complicated class along with the requirement that filebuf use that class. Plauger agreed with Schwarz -- eliminating basic_convbuf is a step in the right direction.

Straw Vote: Who favors this proposal? lots yes, 1 no, 2 abstain.

Vilot said the WG plans to investigate a way to specify basic_filebuf multibyte conversions in a way that allows more efficient implementation of overflow() and underflow().

Vilot presented a proposal to close numerous open issues in Clause 27 [input/output] as per N0634 = 95-0034. See motion 24.

Straw Vote: Who favors this proposal? lots yes, 0 no, 3 abstain.

Lenkov closed the committee of the whole.

The committees recessed at 17:50 on Wednesday and reconvened at 08:30 on Thursday.

5.3 Extensions WG

Stroustrup presented the proposals from the Extensions WG. He said only one of the following issues was controversial in the WG. The WG was nearly unanimous on the others.

Stroustrup described a proposal to add placement delete as a way to prevent memory leaks from placement new expressions that throw an exception from a constructor (N0642 = 95-0042). See motion 25.

Stroustrup gave this example:

```
z p;
...
new (p) x;
```

Under this proposal, an exception thrown from the new expression invokes delete p if and only if there's an operator delete(void *, z) in scope that matches operator new(size_t, z).

Adamczyk observed that the WP currently prohibits overloading operator delete, but this proposal allows it. He asked if that's the intent. Stroustrup said yes. Pennello asked if you can handle such exceptions using try and catch explicitly. Stroustrup said no, not if the placement argument is an allocator.

Straw Vote: Who favors this proposal? lots yes, 1 no, 3 abstain.

Stroustrup presented a proposal to clarify and simplify the rules for throwing objects whose types have private or ambiguous base classes. With this change, a program can can throw an object with a private or ambiguous base class, but it can't catch the exception by such a base. See motion 26 for details.

Stroustrup explained that these rules are context independent. That is, where the throw and catch occur doesn't affect whether the handler matches the throw.

Straw Vote: Who favors this proposal? lots yes, 1 no, 1 abstain.

Stroustrup presented a proposal to allow

```
    throw X;
```

to mean

```
    throw X();
```

(from N0632 = 95-0032). If X names both a type and an object, throw X still throws the object. See motion 27.

Several members were confused about whether motion 9 (on initializers of the form T()) affects this motion. Stroustrup explained that, as worded, this motion assumes motion 9 has not [yet] passed.

Straw Vote: Who favors this proposal?
    WG21+X3J16: 27 yes, 17 no, 5 abstain.
    WG21 only: 4 yes, 1 no, 1 abstain.

Stroustrup presented a proposal to add function-try-blocks as a way to catch exceptions thrown by ctor-initializers. He gave this example:

```
    X::X()
    try                 // this is new
        : a(1), b(2)
    {
    ...
    }
    catch (e) {         // so it this
        ...
    }
```

By this proposal, any function can use a function-try-block. For constructors and destructors, it's essential. For others, it's just a convenience. See motion 28 for details.

Straw Vote: Who favors this proposal? lots yes, 4 no, 4 abstain.

Stroustrup proposed to clarify the effect of using-declarations on overload resolution (from N0643 = 95-0043 and from edit box 38 in clause 7.3.3). He gave this example:

```
    struct B {
        virtual void f(int);
    };
    struct D : B {
        virtual void f(double);
        using B::f;
    };
    void g(D* p)
    {
        p->f(1);    // call B::f
    }
```

Under this proposal, the using-declaration introduces B::f into D as if it were a member of D. See motion 29 for precise wording.

Straw Vote: Who favors this proposal? lots yes, 0 no, 2 abstain.

Stroustrup presented a proposal to clarify operator lookup in the presence of namespaces (N0645 = 95-0045). He explained the problem and the proposed solution using an example very similar to the one on the first page of that paper. See motion 30 for details.

Straw Vote: Who favors this proposal? lots yes, 2 no, 5 abstain.

Stroustrup proposed to ban specialization of member template types. For example:

```
    template <class T> class X {
        template <class U> class M {
            ...
        };
    };
```

```
    template <class T> class X<T>::M<char> { // error
       ...
    };
    class X<int>::M<char> { // ok
        ...
    };
```

See motion 31 for details.

Straw Vote: Who favors this proposal? lots yes, 0 no, 2 abstain.

Stroustrup presented a proposal to specify the overloading and linkage of template functions (from discussion in N0578 = 94-0191 and item 3.21 of N0607 = 95-0007). He explained the problem with an example. See motion 32 for the example as well as precise wording for the resolution.

Straw Vote: Who favors this proposal? lots yes, 1 no, 5 abstain.

Regarding the example from the previous discussion (see the text of motion 32), Shopiro asked what happens the if the templates are in the same file. Stroustrup said the next proposal answers that question.

Stroustrup explained a proposal to clarify template function overload resolution (based in discussion in N0578 = 94-0191 and item 3.22 of N0607 = 95-0007). He gave this example to illustrate the proposal's effect:

```
    template <class T>
    void swap(T&, T&);                  // 1

    template <class T>
    void swap(vector<T>&, vector<T>&);  // 2

    int i, j;
    vector<int> v1, v2;

    swap(i, j);     // uses 1

    swap(v1, v2);   // uses 2
```

See motion 33 for details.

Straw Vote: Who favors this proposal? lots yes, 1 no, 2 abstain.

Stroustrup described a proposal to clarify partial specialization of class templates (from discussion in N0578 = 94-0191). He used this example to illustrate the proposal's effect:

```
    template <class T>
    class list { ... };         // 1
```

```
    template <class T>
    class list<T*> { ... };     // 2

    class list<void*> { ... };  // 3

    list<int> li;       // uses 1
    list<int*> lip;     // uses 2
    list<void*> lvp;    // uses 3
```

See motion 34 for details.

Straw Vote: Who favors this proposal? lots yes, 2 no, 3 abstain.

Stroustrup presented resolutions to three open issues from N0607 = 95-0007. See motion 35.

Straw Vote: Who favors this proposal? lots yes, 0 no, 2 abstain.

Stroustrup presented a proposal to allow explicit qualification of template names in certain contexts (from issue 6.21 of N0607 = 95-0007 and edit box 63 from clause 14.9.1 of the WP). Stroustrup explained the proposal using this example:

```
    class X {
        template<size_t> X* alloc();
    };

    void f(X* p)
    {
        p->alloc<200>();            // error
        p->template alloc<200>();   // ok (this is new)
    };
```

See motion 36 for details.

Stroustrup said this is the only proposal that was somewhat controversial in the WG. The WG approved it 9 yes, 3 no.

Straw Vote: Who favors this proposal? lots yes, 6 no, 8 abstain.

5.4 Environments WG

None.

5.5 C Compatibility WG

Plum presented a proposal to ban "implicit int" from C++ (N0633 = 95-0033). He explained that Bruck presented this proposal last year. Plum (as liaison) took the issue to WG14. WG14's members fell into two camps: (1) ban it, or (2) deprecate it. Plum said the WG21+X3J16 C Compatibility WG recommended banning it. See motion 37.

Koenig said he'd vote against the ban unless you could still write

```
    main()
    {
        ...
    }
```

Straw Vote: Who favors this proposal? lots yes, 3 no, 0 abstain.

5.6 Formal Syntax WG

None.

5.2 Library WG (revisited)

Vilot reopened the discussion on the effect of exceptions thrown from destructors (N0623 = 95-0023). He presented revised wording for the proposal. See motion 41.

Much discussion followed. Colvin thought the revised proposal was too general -- it's not limited to containers and strings as was the earlier version. Vilot said we need some statement of constraints on programs. Corfield and Koenig agreed strongly. Stroustrup thought the revised proposal was a definite improvement.

Straw Vote: Who favors this proposal?
    WG21+X3J16: lots yes, 4 no, 4 abstain.
    WG21 only: 4 yes, 0 no, 2 abstain.

Vilot introduced two proposals on standard exceptions: (1) N0566R1 = 94-0179R1, and (2) N0665 = 95-0065, a compromise written the night before by Schwarz, Myers, Clamage and Colvin (the "Gang of Four"). Dawes favored (2), as did Clamage. Clamage added that all the issues were on the table before the paper was written, and all four authors held different positions going in. Clamage said he thought it represents a good compromise.

Straw Vote: Who favors the proposal in N0665R1 = 95-0065R1? lots yes, 2 no, 6 abstain.

Henricson suggested changing the name [XUNEXPECTED] to bad_exception. No one objected. See motion 38.

Lenkov closed the committee of the whole.

The committees recessed at 11:55 on Thursday and reconvened at 16:00 on Thursday.

6 WG Sessions

7 Distribute formal motions

8 General Session II

8.1 Library WG

Vilot presented revised wording for motions 23 and 41. No one objected to the revisions.

Vilot presented a proposal to specify the library for freestanding implementations (N0654 = 95-0054). Several members objected to the omission of <cstdarg> from the list of minimum library components. Vilot amended the proposal to include <cstdarg>. See motion 42. No one objected to the proposal.

Vilot introduced a proposal to overload the math functions in the library. See motion 43. No one objected to the proposal. Unruh noted that the proposal makes sqrt(2) ambiguous.

Vilot introduced a proposal to restore the basic_string getline function.

Straw Vote: Who favors this proposal? lots yes, 1 no, 0 abstain.

Schwarz presented details of the "Gang of Four" compromise proposal on class exception (N0665 = 95-0065). Unruh noted that the proposal does not declare a copy constructor for the class. Schwarz said it should.

Vilot said the proposal doesn't change the placement of names in standard headers. Schwarz said the authors did not intend to make any changes. Stroustrup said he thought a single exception hierarchy is not a good idea, but he'd go along. He also said that bad_unexpected is poorly named.

Vilot said there's still a problem with a declaration for string in a header where it doesn't belong. Plum suggested making certain signatures optional under certain circumstances to cure the problem. Vilot said the WG would consider that.

8.2 Core WG

Lajoie reintroduced the proposal to resolve object initialization issues. She explained that the words that went into the final form of the resolutions were not what she presented to the committee yesterday. She explained the new wording. (See motion 9, which refers to N0675 = 95-0075.) No one objected to submitting this for a formal vote.

8.3 Extensions WG

Stroustrup summarized the WG's most recent deliberations. The WG considered:

-- adding typedef templates to C++. The WG decided they would not be that useful.

-- allowing the keyword explicit on operator functions. The WG agreed it could be done, but wouldn't be all that useful.
-- allowing initialization of member aggregates. They decided it was a rare problem that's easy to work around.

Stroustrup said the template compilation model is still an open issue. Plum requested a technical session on this topic for the next meeting. Hartinger said the German delegation is very concerned about the model, particularly because of implicit context merging. Others expressed interest in a technical session, which Spicer agreed to do.

Stroustrup said the WG discussed a proposal to allow N::m to refer to something that isn't declared in N, but is accessible because of a using-directive (N0635 = 95-0035). They made no progress.

8.4 Environments WG

Nothing.

8.5 C Compatibility WG

Nothing.

8.6 Formal Syntax WG

Nothing.

9 WG sessions (if any time left)

Lenkov closed the committee of the whole.

The committee recessed at 17:30 on Thursday and reconvened at 08:35 on Friday.

10 Review of the meeting

WG21+X3J16 reviewed the wording of the formal motions in preparation for voting.

10.1 Formal motions

Lenkov counted 38 X3J16 members and 7 WG21 delegations.

1) Motion (to accept the WP) by Dawes/Rumsby:

Move we accept N0629 = 95-0029 as the current WP.

Motion passed X3J16: lots yes, 0 no, 1 abstain.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

2) Motion (to relax restrictions on overloading) by Adamczyk/Lajoie:

Move we amend the WP as proposed in N0631 = 95-0031, and change the first sentence of 13.1 [over.load] paragraph 2 to:

Certain function declarations cannot be overloaded:

Motion passed X3J16: lots yes, 1 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

3) Motion (to allow default arguments on operator new and operator()) by Adamczyk/Lajoie:

Move we amend the WP as follows:

-- Delete 8.3.6 [dcl.fct.default] paragraph 9.

-- Add to the end of 3.7.3.1 [basic.stc.dynamic.allocation] paragraph 1:

Parameters other than the first can have associated default arguments (_dcl.fct.default_).

-- Add to the end of 13.4 [over.oper] paragraph 8:

, except where explicitly stated below. Operator functions cannot have more or fewer parameters than the number required for the corresponding operator, as described in the rest of this section.

-- Add after the first sentence of 13.4.4 [over.call] paragraph 1:

It can have default arguments.

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

4) Motion (to make conversions of pointer types to bool worse than other standard conversions in overload resolution) by Adamczyk/Lajoie:

Move we amend 13.2.3.2 [over.ics.rank] paragraph 4 by adding an additional bullet before the first bullet:

-- A conversion that is not a conversion of a pointer, or pointer to member, to bool is better than another conversion that is such a conversion.

Motion passed X3J16: lots yes, 1 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

5) Motion (to add additional subsequence cases in the ordering of standard conversions) by Adamczyk/Wilcox:

Move we amend the WP as follows:

-- In 13.2.3.2 [over.ics.rank] paragraph 4, add additional bullets at the end:

-- Conversion of C to B is better than conversion of C to A.

-- Conversion of B* to A* is better than conversion of C* to A*.

-- Binding an expression of type B to a reference of type A& is better than binding an expression of type C to a reference of type A&.

-- Conversion of B::* to C::* is better than conversion of A::* to C::*.

-- Conversion of B to A is better than conversion of C to A.

-- Add to the end of the first bullet of the same paragraph:

, and conversion of A* to void* is better than conversion of B* to void*.

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

6) Motion (to add class name injection and resolve some name lookup issues) by Pennello/Adamczyk:

Move we amend the WP in accordance with the substantive changes (indicated in bold face) of N0663R1 = 95-0063R1, with the recommendation in section 8 of that paper revised to read:

Add to the WP subclause 12.5 [class.free] after paragraph 10:

For a virtual destructor (whether user-defined or implementation-generated), the deallocation function to be called is determined by looking up the name of operator delete in the context of the outermost block of that destructor's definition (ignoring any names defined in that block). If the result of the lookup is ambiguous, inaccessible or not unique, the program is ill-formed.

With the footnote:

Note that this applies to destructor definitions, not mere declarations. A similar restriction is not needed for the array version of the delete operator because 5.3.5 [expr.delete] requires the static type to be the same as the dynamic type.

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

7) Motion (to incorporate a memory model compatible with C) by Lajoie/Bruns:

Move we adopt the changes described in N0670 = 95-0070.

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

8) Motion (to define the interaction of the memory and object models) by Lajoie/Bruns:

Move we adopt the changes proposed in N0673 = 95-0073.

    Motion passed X3J16: lots yes, 2 no.
    Motion passed WG21: 5 yes, 0 no, 1 abstain.

9) Motion (to resolve core issues of initialization) by Anderson/Lajoie:

    Move we adopt the changes from N0675 = 95-0075.

Winder asked if this consistent with the upcoming motion (motion 27) to make throw X the same as throw X()? Anderson said the motions are not, and should not be, consistent with each other.

Motion passed X3J16: lots yes, 4 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

10) Motion (to define behavior of class copy) by Lajoie/Wilkinson:

    Move we adopt the proposed changes from N0671 = 95-0071.

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

11) Motion (to enable implementations to optimize away copy operations) by Lajoie/Wilkinson:

    Move we adopt the resolution proposed in N0641 = 95-0041.

Bruck said the meaning of the word "use" in the cited paper needs editorial clarification. Gibbons said the proposed change isn't needed under the execution model's "as if" rule, except for copy operations involving the execution of one constructor and one destructor.

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

12) Motion (to integrate non-converting constructors into the library) by Myers/Henricson:

    Move we amend the WP as per N0624 = 95-0024, except that:
    -- in clause 23 [lib.containers], do not make bits(unsigned long) explicit
    -- in clause 26 [lib.numerics], do not make valarray constructors explicit

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

13) Motion (to close clause 18 [lib.language.support] issues) by Henricson/Rumsby:

    Move we amend the WP as follows:
    -- Adopt the proposed resolutions for issues 1, 2 and 4 from N0649 = 95-0049.
    -- Remove the header <stddef>
    -- In subclause 23.2.1.1 [lib.cons.bits], change the bits constructor to:

```
        bits(const string& str, size_t pos = 0, size_t n = -1);
```

-- Delete subclause 18.1.1 [lib.stddef.values].
-- Add a subclause to 21.1.1.3 [lib.basic.string] describing the template class basic_string member:

```
static const size_type npos = -1;
```

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

14) Motion (to require that operator new throw exceptions) by Dawes/ Myers:

Move we amend the WP as described in N0647 = 95-0047.

Motion passed X3J16: lots yes, 5 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

15) Motion (to make numerous small STL changes) by Podmolik/Dawes:

Move we amend clauses 20 [lib.utilities], 23 [lib.containers], 24 [lib.iterators] and 25 [lib.algorithms] as described in N0614R1 = 95-0014R1, except that:

-- In section 1.2 of the paper, change the box in tables 49 and 50 describing the return type of iterator expressions from:

iterator type pointing to X::reference

to:

when operator*() is applied, yields type X::iterator

-- similarly for const_iterator

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

16) Motion (to modify auto_ptr) by Dawes/Henricson:

Move we amend the WP as per N0589R1 = 94-0202R1, page 1, with an additional note that:

The template constructor and template assignment operator modify the argument r, even though it is declared as a reference to a const auto_ptr.

Motion passed X3J16: lots yes, 2 no.
Motion passed WG21: 5 yes, 1 no, 0 abstain.

17) Motion (to close clause 21 [lib.strings] issues) by Wilhelm/Rumsby:

Move we amend the WP by adopting the recommendations for the following issues from N0616R1 = 95-0016R1:
-- issue 6: change order of template arguments
-- issue 9: remove const qualifier on charT argument
-- issue 10: change const_pointer back to const charT *

Motion passed X3J16: lots yes, 0 no.

Motion passed WG21: 6 yes, 0 no, 0 abstain.

18) Motion (to add an iterator interface to basic_string without removing the indexing interface) by Wilhelm/Dawes:

    Move we amend the WP as per N0628R1 = 95-0028R1.

    Steinmuller said this change has consequences for implementations, not just for the interface.

Motion passed X3J16: lots yes, 2 no.
Motion passed WG21: 4 yes, 1 no, 1 abstain.

19) Motion (to make minor changes to clause 22 [lib.localization]) by Myers/Rumsby:

    Move we amend the WP as per N0601R1 = 94-0214R1 and N0625 = 95-0025, with the following changes:

    -- The enumerator values will not be specified, but left to be implementation-defined.

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

20) Motion (to make minor changes to clause 23 [lib.containers]) by Podmolik/Dawes:

    Move we amend the WP as per the recommendations for the following issues from N0613R2 = 95-0013R2:
    -- issue 6:
        -- rename bits member function toggle() to flip()
        -- change "bits" to "bitset"
        -- change the header name from <bits> to <bitset>
    -- issue 7: add vector<bool>::flip()
    -- issue 8: add nested class bitset::reference

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

21) Motion (to make minor changes to clause 24 [lib.iterators]) by Myers/Dawes:

    Move we amend the WP as recommended in N0621 = 95-0021 (relax requirements on Iterator++ result) and N0622 = 95-0022 (fix streambuf iterator).

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

22) Motion (to close clause 26 [lib.numerics] issues) by Holly/Myers:

    Move we amend the WP as per N0637 = 95-0037 from the following issues:
    -- issue 002: change the definition of operator>> for complex
    -- issue 006: change valarray
    -- issue 008: add/specify numeric_limits specializations in clause 18

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

23) Motion (to remove basic_convbuf) by Schwarz/Rumsby:

Move we amend the WP as per N0664 = 95-0064, sections 2 and 3, with the changes described in N0674 = 95-0074.

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

24) Motion (to close clause 27 issues) by Schwarz/Myers:

Move we amend the WP as per N0634 = 95-0034 from the following issues:

- -- issue 004: remove static functions eof() and newline()
- -- issue 011: define stossc() in Annex D (Deprecated)
- -- issue 012: change sputc(int_type) to sputc(char_type) and document the intended behavior of sputc(-1)
- -- issue 015: add tellg, seekg, tellp, seekp to streams
- -- issue 016: change operator>>(streambuf&) and operator<<(streambuf &) to (streambuf*), and document that calling them with a null pointer results in undefined behavior
- -- issue 023: change operator<<(ostream&, void*) to operator<<(ostream &, const void*)

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

25) Motion (to add placement delete semantics) by Corfield/Gibbons:

Move we amend the WP as described in N0642 = 95-0042.

Motion passed X3J16: lots yes, 2 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

26) Motion (to clarify the rules for throwing objects whose types have private or ambiguous base classes) by Gibbons/Spicer:

Move we amend the WP as follows:

-- replace subclause 15.3 [except.handle] paragraph 2 with:

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

[1] T and E are the same type, or

[2] T is a public base class of E, or

[3] T is a pointer type and E is a pointer type that can be converted to T by a standard pointer conversion (_conv.ptr_) not involving conversions to pointers to private or protected base classes.

-- delete subclause 15.1 [except.throw] paragraph 3.

Motion passed X3J16: lots yes, 1 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

27) Motion (to allow throw simple-type-specifier) by Koenig/Bruck:

    Move we amend the WP as follows:

    -- add the following after clause 15 [except] paragraph 1:

        If throw is followed by a simple-type-specifier, instead of an assignment-expression, the implementation creates and throws an object of the specified type. If the type is a class with a default constructor (_class.ctor_), that constructor will be called; otherwise the result is the default value given to a static object of the specified type.

    -- in clause A.4 [gram.expr] and clause 15 [except] replace:

```
throw-expression:
    throw assignment-expression-opt
```

        with:

```
throw-expression:
    throw assignment-expression-opt
    throw simple-type-specifier
```

Bruns said the Core WG spent a lot of time trying to make T distinct from T(). He advised against this change.

Koenig said he mulled this proposal over, and still stands by it. Anderson said this creates a necessary inconsistency.

Shopiro and others suggested this is change is gratuitous.

Motion failed X3J16: 14 yes, 29 no.
Motion failed WG21: 0 yes, 4 no, 2 abstain.

28) Motion (to allow handling of exceptions in mem-initializers) by Bruck/Corfield:

    Move that we amend the WP as follows:

    -- replace the grammar in clause 8.4 [dcl.fct.def] and clause A.7 [gram.dcl.decl]:

```
function-definition:
    decl-specifier-seq-opt declarator ctor-initializer-opt
            function-body
```

        with:

```
function-definition:
    decl-specifier-seq-opt declarator ctor-initializer-opt
            function-body
    decl-specifier-seq-opt declarator function-try-block
```

    -- add a new grammar production in clause 15 [except] and clause A.13 [gram.except]:

```
function-try-block:
    try ctor-initializer-opt function-body handler-seq
```

-- add the following paragraph after clause 15 [except] paragraph 2:

A function-try-block associates a handler-seq with a constructor initializer and the function body. An exception thrown in the constructor initializer or the function body transfers control to a handler in a function-try-block in the same way as an exception thrown in a try-block transfers control to other handlers.

-- add the following paragraphs after subclause 15.3 [except.handler] paragraph 7:

Referring to any non-static member or base class of the object in a handler of a function-try-block of a constructor or destructor for that object results in undefined behavior.

The fully constructed base classes and members of an object shall be destroyed before entering the handler of a function-try-block of a constructor or destructor for that object.

The scope and lifetime of the parameters of a function or constructor extend into the handlers of a function-try-block.

If the handlers of a function-try-block contain a jump into the body of a constructor or destructor, the program is ill-formed.

If a return statement appears in a handler of a function-try-block of a constructor, the program is ill-formed.

The exception being handled shall be rethrown if control reaches the end of a handler of the function-try-block of a constructor or destructor. Otherwise, the function shall return when control reaches the end of a handler of the function-try-block (_stmt.return_).

Myers said this introduces a new and unnecessary instance of undefined behavior (namely, when execution runs off the end of a catch clause). Bruck said this is the same as the undefined behavior that existed previously.

Motion passed X3J16: lots yes, 3 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

29) Motion (to clarify the effect of using-declarations on overload resolution) by Ball/Gibbons:

Move we amend the WP by adding the following after 7.3.3 [namespace.udecl] paragraph 12:

For the purpose of overload resolution, the functions which are

introduced by a using-declaration into a derived class will be
treated as though they were members of the derived class. In particular, the implicit "this" parameter shall be treated as if it
were a pointer to the derived class rather than to the base class.
This has no effect on the type of the function, and in all other
respects the function remains a member of the base class.

Motion passed X3J16: lots yes, 1 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

30) Motion (to describe operator lookup in the presence of namespaces)
by Corfield/Holaday:

Move we amend the WP by replacing 13.2.1.2 [over.match.oper]
paragraph 3 with:

For a fully-qualified type ::N1::...::Nn::C1::...::Cm::T where Ni is
a namespace name and Ci is a class name, the fully- qualified namespace name ::N1::...::Nn is called the "namespace of the type T".
To lookup X in the "context of the namespace of the type T" means to
perform the qualified name lookup of ::N1::...::Nn::X
(_over.call.func_).

For a unary operator @ and operand e1 of type T1, or binary operator
@ and operands e1 and e2 of types T1 and T2 respectively, three sets
of candidate functions (member candidates, non-member candidates and
built-in candidates) are constructed as follows:

-- If T1 is a class type, the set of member candidates is the
result of the qualified lookup of T1::operator@ (_over.call.
func_); otherwise, the set of member candidates is empty.

-- The set of non-member candidates is the union of the following
name lookups:

    -- The unqualified lookup of operator@ in the context of the
    expression according to the usual rules for name lookup
    except that all member functions are ignored.

    -- For each Ti of class type and each of its direct and indirect base class types, operator @ is looked up in the
    context of the namespace of each of those types according to
    the usual rules for name lookup.

    -- For each Ti of enumeration type, operator @ is looked up in
    the context of the namespace of the type according to the
    usual rules for name lookup.

-- For the binary operator , or the unary operator &, the set of
built-in candidates is empty. For all other operators, the set
of built-in candidates includes all of the built-in operators

defined in _over.built_ that, compared to the given operator,

    -- have the same operator name, and

    -- accept the same number of operands, and

    -- accept operand types to which the given operand or operands
    can be converted according to _over.best.ics_.

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

31) Motion (to restrict specialization of member types) by Spicer/Ball:

Move we amend the WP by replacing the first sentence of 14.5 [temp.spec] paragraph 1 with:

Except for a type member or template class member of a non-specialized template class, the following can be declared by a declaration where the declared name is a template-id: a specialized template function, a template class, or a static member of a template; that is: ...

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

32) Motion (to resolve overloading and linkage of template functions) by Spicer/Corfield:

Move we amend the WP by adding the following new section as 14.9.x before 14.9.4 [temp.over.spec]:

Overloading and linkage [temp.over.link]

It is possible to overload template functions so that specializations of two different template functions have the same type. For example:

```
// file1.c                      // file2.c
template<class T>               template<class T>
void f(T*);                     void f(T);
void g(int* p) {                void h(int* p) {
    f(p); // call f_PT_pi           f(p); // call f_T_pi
}                               }
```

Such specializations are distinct functions and do not violate the ODR.

The signature of a specialization of a template function consists of the actual template arguments (whether explicitly specified or deduced) and the signature of the function template.

The signature of a function template consists of its function signature and its return type and template parameter list. The names of the template parameters are significant only for establishing the relationship between the template parameters and the rest of the signature.

Motion passed X3J16: lots yes, 1 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

33) Motion (to clarify overloading of function templates) by Gibbons/Spicer:

Move we amend the WP according to N0668 = 95-0068, section 1.

Motion passed X3J16: lots yes, 1 no.

Motion passed WG21: 6 yes, 0 no, 0 abstain.

34) Motion (to describe partial specialisation of template classes) by Gibbons/Spicer:

    Move we amend the WP according to N0668 = 95-0068, section 2.

Motion passed X3J16: lots yes, 3 no.
Motion passed WG21: 5 yes, 0 no, 1 abstain.

35) Motion (to resolve various template issues) by Spicer/Corfield:

    Move we amend the WP according to N0667 = 95-0067.

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

36) Motion (to allow disambiguation of template names in certain contexts) by Spicer/Gibbons:

    Move we amend the WP as follows:

    -- replace BOX 63 (at the end of 14.9.1 [temp.arg.explicit]) with:

    For a template function name to be explicitly qualified by template arguments, the name must be known to refer to a template. When the name appears after . or -> in a postfix-expression, or after :: in a qualified-id where the nested-name-specifier depends on a template argument, the member template name must be prefixed by the keyword template. Otherwise the name is assumed to name a non- template. For example:

```
class X {
public:
    template<size_t> X* alloc();
};
```

```
void f(X* p)
{
    X* p1 = p->alloc<200>();          // error < means less than
    X* p2 = p->template alloc<200>(); // < starts explicit
                                      // qualification
}
```

    If a name prefixed by the keyword template in this way is not the name of a member template function, the program is ill-formed.

    -- in 5.1 [expr.prim] paragraph 7 and A.4 [gram.expr] replace the grammar for qualified-id with:

```
qualified-id:
    nested-name-specifier template-opt unqualified-id
```

    -- replace the first clause of 5.1 [expr.prim] paragraph 8 with:

    A nested-name-specifier that names a class (_dcl.type_) followed by ::, optionally followed by the keyword template (_temp.arg. explicit_), and then followed by the name of a member of either that class (_class.mem_) or one of its bases (_class.derived_), is a qualified-id;

-- 5.2 [expr.post] paragraph 1 and A.4 [gram.expr] add the following productions to the grammar for postfix- expression:

```
postfix-expression . template id-expression
postfix-expression -> template id-expression
```

-- replace the first sentence of 5.2.4 [expr.ref] paragraph 1 with:

A postfix expression followed by a dot . or an arrow -> , optionally followed by the keyword template (_temp.arg.explicit_), and then followed by an id-expression, is a postfix expression.

Motion passed X3J16: lots yes, 6 no.
Motion passed WG21: 4 yes, 0 no, 2 abstain.

37) Motion (to ban implicit int in C++) by Micco/Plum:

Move we amend subclause 7.1.5 [dcl.type] paragraph 3 and subclause 7.1.3 [dcl.typedef] paragraph 1, as described on page 2 of N0633 = 95-0033.

Koch said this is the greatest thing we're going to do today.

Motion passed X3J16: lots yes, 1 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

38) Motion (to decouple class exception) by Schwarz/Clamage:

Move we amend the WP as proposed in N0665R1 = 95-0065R1, and in addition, change all uses of the placeholder name XUNEXPECTED to bad_exception.

Motion passed X3J16: lots yes, 1 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

39) Motion (to authorize convener to submit disposition of comments for CD registration ballot) by Harbison/Lajoie:

Move we authorize the WG21 convener to submit document N0669 = 95-0069 to SC22 as the Disposition of Comments for the CD Registration Ballot.

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

40) Motion (to authorize the convener to submit the CD) by Harbison/Lajoie:

Move we authorize the WG21 convener to submit the Working Paper, as modified by resolutions at this meeting, to SC22 for registration as a Committee Draft and for subsequent CD ballot.

Motion passed X3J16: lots yes, 2 no, 2 abstain.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

41) Motion (to note constraints on programs) by Steinmuller/Dawes:

Move we amend the WP as per N0623 = 95-0023, but instead add the following paragraphs to subclause 17.3.3 (Constraints on Programs):

In certain cases (replacement functions, handler functions, operations on types used to instantiate standard library template components), the C++ Standard Library depends on components supplied by a C++ program. If these components do not meet their requirements, the Standard places no requirements on the implementation.

In particular, the effects are undefined in the following cases:

-- for replacement functions (18.4.1), if the replacement function does not implement the semantics of the applicable "Required behavior" paragraph
-- for handler functions (18.4.2.2, 18.6.1.1, 18.6.2.2), if the installed handler function does not implement the semantics of the applicable "Required behavior" paragraph
-- for types used as template arguments when instantiating a template components, if the operations on the type do not implement the semantics of the applicable "Requirements" subclause (20.1, 23.1, 24.1,

-- if any of these functions or operations throws an exception, unless specifically allowed in the applicable "Required behavior" paragraph.

Myers said this omits a restriction on specializing templates.

Motion passed X3J16: lots yes, 3 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

42) Motion (to define library for freestanding implementations) by Spicer/Becker:

Move we amend the WP as per N0654 = 95-0054, page 2, adding the following line to Table 25:

Subclause	Header(s)
18.7 Other runtime support	<cstdarg>

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

43) Motion (to overload math functions) by Gibbons/Ball:

Move we amend the WP as per N0666 = 95-0066.

Motion passed X3J16: lots yes, 0 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

44) Motion (to restore basic_string getline) by Becker/Dawes:

Move we amend the WP as per N0611 = 95-0011, changing the function signature to:

```
template<class charT, class IS_traits, class STR_traits,
        class STR_Alloc>
basic_istream<charT, IS_traits> &
```

```
        getline(basic_istream<charT, IS_traits> &is,
            basic_string<charT, STR_traits, STR_Alloc> &str,
            charT delim = IS_traits::newline());
```

Motion passed X3J16: lots yes, 2 no.
Motion passed WG21: 6 yes, 0 no, 0 abstain.

10.2 Review of action items

Lenkov opened the committee of the whole.

The WG chairs submitted the following action items.

Core WG (Lajoie):

- -- Lajoie will assist Koenig in incorporating the Core WG resolutions into the WP.
- -- Lajoie will investigate the issues regarding the ODR.

Core WG (Adamczyk):

- -- Eager will write a paper on volatile semantics and volatile copy constructors.
- -- Welch will write a paper on implied accessibility.
- -- Gafter will analyze name lookup (no promise of paper).
- -- Gafter will write a paper on defining operators like "|" directly for bool.
- -- Micco will write a paper on the requirements on a parameter passed to va_start.
- -- Schreiber will write a paper on references in unions.
- -- Adamczyk will reconcile chapters 13 and 14 with respect to over-loading and templates, and add examples to chapter 13.

Extensions WG:

- -- None

Library WG:

- -- Schwarz will write a paper describing how to do direct wide character output to files.

C Compatibility WG:

- -- Plum will add items to C Compatibility Annex on: copying volatile PODs, class declarations that erroneously have a storage class, and banning implicit int.

Other:

- -- Harbison, Plum and Lajoie will segregate the examples and notes from the normative text in the draft.

11 Plans for the future

11.1 Next meeting

Murphy advised that the host(s) for future meetings reserve a block of rooms for the weekend prior to the meeting (for those who arrive early).

Lenkov said that the following people will meet on July 7 and 8 to handle the US public comments: Stroustrup, Spicer, Gibbons, Adamczyk, Pennello, Holaday, Wilcox, Schwarz, Plum, Vilot, Podmolik, Wilhelm, Dawes, Becker, Clamage, Scian.

11.2 Mailings

Hartinger announced that, since Email is now widely available, Siemens-Nixdorf will not handle any mailings outside Germany after the upcoming post-meeting mailing.

Lajoie offered to send postal mail directly to each head of delegation for the pre-Monterey meeting mailing. Harbison said he'd figure out what to do about later mailings to WG21.

Lajoie said she must receive all items for the post-Austin mailing by March 24, 1995 and for the pre-Monterey mailing by May 30.

11.3 Following meetings

Harbison listed the meeting dates for the upcoming meetings:
-- July 9-14, 1995 in Monterey, CA, hosted by Sun Microsystems
-- November 5-10, 1995 in Tokyo, Japan
-- March 10-15, 1996 in Santa Cruz, USA hosted by Borland
-- July 7-12, 1996 in Stockholm, Sweden, hosted by Ericsson
-- November 10-15, 1996, Kona, Hawaii, USA hosted by Plum Hall

Harbison asked how many members were planning to go to Tokyo. He counted about 23 X3J16 voting members, and 26 all together.

Corfield offered to host the July, 1997 meeting in the UK. Harbison asked for volunteers to host other 1997 meetings in the USA.

12 National delegation caucuses

13 Adjournment

The committees thanked Wood and Motorola for hosting the meeting. Applause.

Lenkov closed the committee of the whole.

Motion by Dawes/Harbison:

> Move we adjourn.

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

The committee adjourned at 10:30 on Friday.

Appendix A - Attendance

Name	Affiliation	Stat	M	Tu	W	Th	F
Goldberg, Jody	Algorithmics	P	A	A	A	A	A
Motamedrasa, Saeed	AMD	P	V	V	V	V	V

Podmolik, Larry	Andersen Consulting	P	V	V	V	V	V
Wilhelm, Richard	Andersen Consulting	A	A	A	A	A	A

Winder, Wayne	Asymetrix	P	V	V	V	V	V
Koenig, Andrew	AT&T Bell Labs	A	V	V	V	V	V
Stroustrup, Bjarne	AT&T Bell Labs	A	A	A	A	A	A
Becker, Pete	Borland	P	V	V	V	V	V
Swan, Randall	C-Team	P	V	V	V	V	V
Burleson, Kate	Centerline Software	A	V	V	V	V	V
Holly, Mike	Cray Research	P	V	V	V	V	V
Druker, Samuel	Cygnus Support	A	V	V	V	V	
Yurkovsky, Victor	Data In Formation	P	A	A	A	A	
Schwarz, Jerry	Declarative Systems	P	V	V	V	V	V
Kimmel, Cathy	Digital Equipment	O	A	A	A	A	A
Meyers, Randy	Digital Equipment	P	V	V	V	V	V
Bruck, Dag	Dynasim AB	P	V	V	V	V	V
Andrews, Graham	Edinburgh Port. Compilers	P	A	A	A	A	A
Adamczyk, Steve	Edison Design Group	P	V	V	V	V	V
Anderson, Mike	Edison Design Group	A	A	A	A	A	A
Spicer, John	Edison Design Group	S	A	A	A	A	A
Henricson, Mats	Ellemtel	P	V	V	V	V	V
Coha, Joseph	Hewlett-Packard	A	V	V	V	V	V
Lenkov, Dmitry	Hewlett-Packard	P	A	A	A	A	A
Lajoie, Josee	IBM	P	V	V	V	V	V
Murphy, Michael	IBM	A	A	A	A	A	A
Colvin, Greg	IMR	P	V	V	V	V	V
Roskind, Jim	Info Seek	P	V	V	V	V	V
Auld, Will	Intel	A	V	V	V		
Andersson, Per	Ipso Object Software	P	V	V	V	V	V
Stuessel, Marc	IST GmBH	P	V	V	V	V	V
Ichiro Koshida	Japan	P	A	A	A	A	A
Tsutomu Kamimura	Japan	P	V	V	V	V	
Munch, Max	Lex Hack & Associates	O	A	A	A	A	A
Dum, Stephen	Mentor Graphics	P	V	V	V	V	V
Shopiro, Jonathan	Merrill Lynch	P	V	V	V	V	V
Pennello, Tom	MetaWare	P	V	V	V	V	V
Marcotty, Michael	Metrowerks	A	A	A	A	A	
Caves, Jonathan	Microsoft	A	A	A	A	A	A
Schreiber, Ben	Microsoft	P	V	V	V	V	V
Eager, Michael	Microtec Research	A	V	V	V	V	V
Saini, Atul	Modena Software	P	V	V	V		
Bruns, John	Nations Banc-CRT	P	V	V	V	V	V
Vilot, Mike	Object Craft	P	V	V	V	V	
Benito, John	Perennial	P	V	V	V	V	V
Plum, Tom	Plum Hall	P	V	V	V	V	V
Corfield, Sean	Programming Research	P	V	V	V	V	V
D'Antonio, Larry	Ramapo College	P	A	A	A	A	A
Wilcox, Thomas R.	Rational	P	V	V	V	V	V
Eckel, Bruce	Revolution2	P	V	V	V		
Guilmette, Ron	RG Consulting	P	V	V	V	V	
Hinke, John	Rogue Wave Software	A	A	A	A	A	A
Myers, Nathan	Rogue Wave Software	P	V	V	V	V	V
Saks, Dan	Saks & Associates	P	V	V	V	V	V
Koch, Gavin	SAS Institute	P	V	V	V	V	V
Tooke, Simon	SCO Canada	P	V	V	V	V	V

Hartinger, Roland	Siemens Nixdorf	P	V	V	V	V	V
Langer, Angelika	Siemens Nixdorf	A	A	A	A	A	A
Steinmuller, Uwe	Siemens Nixdorf	A	A	A	A	A	A
Unruh, Erwin	Siemens Nixdorf	O	A	A	A	A	A
Mehta, Michey	Silicon Graphics	A		A	A	A	
Wilkinson, John	Silicon Graphics	P	V	V	V	V	V
Ball, Mike	Sun Microsystems	A	V	V	V	V	V
Clamage, Steve	Sun Microsystems	A	A	A	A	A	A
Gafter, Neal	Sun Microsystems	A	A	A	A	A	
Landauer, Doug	Sun Microsystems	P	A	A	A	A	A
Micco, John	Symantec	P	V	V	V	V	V
Gibbons, Bill	Taligent	P	V	V	V	V	V
Yinsun Feng	Taligent	O	A	A	A	A	A
Harbison, Sam	Tartan	P	V	V	V	V	V
Sullivan, Larry	TechVisions	P	A	A	A	A	A
Tydeman, Fred	Tydeman Consulting	O	A	A	A	A	A
Rumsby, Steve	UK	P	V	V	V	V	V
Bixler, Don	Unisys	A	V	V	V	V	V
Mollitor, Robert	Visix Software	A	V	V	V	V	V
Scian, Anthony	Watcom	P	V	V	V	V	V
Welch, Jim	Watcom	A	A	A	A	A	A
Plauger, P. J.	WG14	O	A	A	A	A	
Crowfoot, Norm	Xerox	P	A	A	A	A	A
Dawes, Beman		P	V	V	V	V	V
Holaday, Thomas		P	V	V	V	V	V
Total Attendance			79	80	80	77	68
Total Votes			52	52	52	49	45