Disposition of Comments from the C++ CD Ballot

Thomas Plum

Convener, ISO/IEC JTC1/SC22/WG21

Draft of 01/29/97 8:00 AM

This is an edited version of WG21/N0799, the collected comments from the CD ballot. I have added the late Canadian comments, and I have inserted after all comment the actions taken by consensus within WG21 to resolve the issues. This document accompanies WG21/N1037, a revised Working Draft which we request be issued by the SC22 secretariat for a second CD ballot. We acknowledge the extensive prior work by Sam Harbison, Convener, ISO/IEC JTC1/SC22/WG21 until September 1996.

General comments

Some comments were echoed by several national bodies. We report our consensus here to avoid repetition:

1. Separation of library and language, creating a multipart standard.

   We recognize that the library has been less stable than the language, and the possible separation of the library from the language has been raised within WG21. However, there has been no support from the national bodies attending meetings to make this separation, because it was felt that issuing both language and libraries simultaneously was important.

2. Dealing with WG21's internal issue lists.

   WG21 maintains several "to do" lists that include various issues that have arisen from national body comments, internal discussions, public comments, etc. Some of the issues on these lists have been substantive, but others are clearly less important for the standard. We believe we have addressed the important issues.

   We would like to discourage national bodies from blanket references to these internal documents in their official comments, since such comments do not acknowledge the obvious difference in importance between issues. We encourage national bodies to explicitly list any issues about which they are concerned. (Cross-referencing such comments with our issues list is helpful but not necessary.)

   Nevertheless, we believe the current working draft eliminates all major open issues from WG21's issue lists.

3. Extensions.

   WG21 recognizes that several language extensions were made immediately prior to issuance of the first CD. Most of these had been under discussion for a long time. In accordance with the usual needs for stability, WG21's policy has been to resist adding any extensions not specifically necessary to resolve national body comments.

Australia

Approved without comments.

Belgium

Did not vote.

Brazil

Approved without comments.

Canada

[Canada did not submit an official vote before the end of balloting, but they did submit a late vote to SC22, which is included here.]

Canada voted “Yes, with comments.” The comments are:

- The issue lists of the various working groups must be addressed.

  See “General Comments” at the start of this document

- No further extensions may be accepted.

  See “General Comments” at the start of this document

The “issue lists” referred to in point one are described in the following documents:

- N0689 IOStreams Issues List by John Hinke
- N0698 Clause 22 (Localization Library) Issues (V.2) by Nathan Myers
- N0699 Clause 20 (Utilities Library) Issues (V.2) by Nathan Myers
- N0702 Clause 24 (Iterators) Issues List by David Dodgson
- N0705 Clause 18 (Language support Library) Issues List - Version 2 by Mats Henricson
- N0758 Template Issues and Proposed Resolutions (Revision 13) by John Spicer
- N0759 Clause 21 (Strings Library) Issues List: Revision 7 by Rick Wilhelm
- N0760 Clause 23 (Containers Library) Issues List - Revision 5 by Larry Podmolik
- N0761 Open Issues for Numeric Libraries (Revision 3) by Judy Ward
- N0765 Core WG list of issues by Josée Lajoie

China

Approved without comments

Czech Republic

Approved without comments.

Denmark

Approved without comments.

Finland

Approved without comments.

France

Disapproved with comments. AFNOR's comments already made at the registration stage are still relevant. In particular AFNOR proposal on character sets (in the spirit of Ada standard) has not been taken in account, this is at least one major reason to not change our vote to approval."

[We therefore repeat the original comments from AFNOR on the CD Registration Ballot.]

First, France supports the standardization of the C++ programming language and wants this standardization to take effect AS SOON AS POSSIBLE.

The C++ reference manual is not enough stable and is still a moving target; as an example, document WG21 N0582 summarizes a non exhaustive list of unresolved issues concerning templates.

R–4 France considers that it is time not to **STOP defining additional extensions** that cannot realistically be reviewed correctly and in a timely planning; the process is too much delayed.

A lot of such extensions have been defined during this process and France considers that the very first objective of the standardization IS NOT MET in document N0545, that is, a clear, non-ambiguous, mature and simple definition of the language.

> WG21 believes that all major extensions were completed before the CD ballot, and no additional extensions were planned. Some small extensions have been made to address CD ballot comments.

R–5 The document is not conformant to ISO directives of **drafting and presentation** of International Standards. A non exhaustive list of examples is:

R–5.a a **table of content** is missing,

> A Table of Contents has been added.

R–5.b even though not required by ISO, a **glossary** shall be included given the context, which is, a complex language with a lot of concepts,

> Although desirable, a proper glossary seems beyond our ability to produce and review within the required schedule time frames.

R–5.c **ISO terminology** (and definitions) presented by ISO/SC1 shall be used instead of ANSI/X3-TR-1-82:1982

This change has been made.

R–5.d Though not required by ISO, a **Rationale** is required at this stage of the process. It is unrealistic to make effective review of the proposed standard without a Rationale.

We agree on the value of a Rationale, but the Working Group does not have the extensive resources necessary to produce one.

R–6 The structure of the proposed reference manual shall be revised. We think that:

- The **core language shall be separated from the libraries**. The core language is what is handled by the compiler.

- Language support libraries should stay in the core language.

- Other libraries shall be in required (non-optional) normative annexes.

See “General Comments” at the start of this document

R–7 The general structure of the libraries shall be revised. In particular, it is necessary to **DECOUPLE libraries**. In the current proposal, there are unacceptable cross and forward references between libraries (and sections). The order of introduction of libraries is not adequate.

We have taken some steps in this direction, particularly by separating exceptions and strings.

R–8 The **character set** defined by the reference manual is not acceptable. In fact, programmers from France (and other nations) cannot, given the current definition, use local characters in identifiers, strings and comments. Modern ISO standard programming languages (as ISO8652-95 Ada) provide definition for characters. We recommend that such a definition shall be used in C++. In particular, use of ISO 10646 shall be accepted (see attached document).

We have added support for multinational character sets and their use in identifiers and literals. The solution chosen is somewhat more general than the one adopted by Ada, and includes support for ISO 10646.

R–9 We support the proposal in document SC22 N0582 concerning templates. That is, we think that a **model of compilation** shall be included in the language.

A new template compilation model is specified.

R–10 In the same way, we support the introduction of keyword “**typename**” instead of overloading “typedef”.

This change has been incorporated.

Other important technical comments (on lvalues, void, relation with environment, templates, exceptions, temporaries,...) are not introduced in this list of objections because we think they can be handled during the next stage and also because we need a C++ Rationale.

[Here was attached in the original CD Registration Ballot comments pages 9 through 14 of RM9X:5.0, an Ada95 draft (1 June 1994)]

Germany

The German DIN NI-22 has voted on CD 14882 as follows:

x Disapproval of the draft for reasons below

x Acceptance of these reasons and appropriate changes in the text will change our vote to approval.

R–11 **One Definition Rule (ODR)** The ODR is one of the important requirements for the CD-Ballot already mentioned as R-31 in the Disposition of comments on CD Registration (Doc No: ISO/IEC JTC1/SC22/WG21/N0669) by the German delegation. The ODR is not described in the Draft which is base for this CD-Ballot (although we know that it was defined and voted in for the next issue of the Draft).

We think, that this issue has sufficiently been addressed within the latest Draft Proposed Working Paper WG/N0996, X3J16/96-178. The issue is closed.

R–12 **Template Compilation Model** We doubt that the template model as specified is defined clear enough so that it can be used in a portable way. It is also not clear whether context merging problems keep the user from determining the cause of errors since the context for some specialization has been synthesized by the compiler in a non-obvious and complex way.

The context merge might also force implementors to store/retrieve lots of information (persistent symbol table) which can lead to a less efficient implementation.

Extensive work in this area has been done.

R–13 **Locale Library** The locale's functionality shall be in accordance with the C locale defined by POSIX and X/OPEN. It is not clear whether this has in principle been accepted for the C++ locale design. The C++ locale should be efficiently implementable on top of the C locale.

Locale support has changed significantly, taking into account these comments.

R–14 **Input/Output Library** This library seems far from being complete; there is still a lot of open work. We think unless more effort is spent on this part of the library it will fail the schedule.

We believe all significant issues in the I/O library have been resolved.

R–15 **Other issues** The are a lot of outstanding issues listed in the German public review comments (X3J16/95-0132, WG21/N0732) which are not addressed by this Draft. (Table 1 on page 5.)

Table 1: Germany's "other issues"

R-no., DIN no.	Comment	WP chap.	Related doc(s)	Recommendation for ISO/ANSI
R–1	Predicate throwing	15	X3J16/95-0093 WG21/N0693	yes
R–2	Rethrowing pending exceptions	15	X3J16/95-0092 WG21/N0692	yes
R–3	Incorporate the long long integral data type	2,3,4,5,7,17,18,22,27	X3J16/95-0115, WG21/N0715	yes, ext-reflector

	in C++			
R–4	The default destructor should be virtual automatically	12.4	Email: 16.6.95	no, need a proposal before
R–5	There should be a provision in the language to inquire an objects identity	9,10	Email 16.6.95	no, use dynamic_cast<void*>
R–6	const Pointers to non-const Objects	8	Proposal 19.6.95	no
R–7	Pointers to const Objects	8	Proposal 19.6.95	no already allowed
R–8	Construction and Destruction of const or volatile Objects	12	Proposal 19.6.95	no
R–9	Derived Classes comments	10	Email: 26.6.95, X3J16: edit-561	yes, 10.3 last sentence is editorial
R–10	Numerics Library	26	Email:26.6.95, X3J16: edit-564	yes, editorial
R–11	Missing preprocessing numbers in Chapter 2.3	2	--, X3J16: edit-562	yes, already a core issue
R–12	The return-type of default operator= should be const T&	12.8	Email: 28.6.95	no
R–13	Avoid copy ctor when binding an exception object by reference to catch declaration	15	via phone	no, bug in some compilers
R–14	Why does C++ this archaic header file mechanism?	16	Email: 29.6.95, DIN #502	no, a module system was requested
R–15	class auto_ptr: get->m; what is m?	20.4.5, page 20-17	Email: 21.5.95, X3J16: edit-563	yes, edit-reflector
R–16	class auto_ptr: the copy ctor should be private to forbid parameter passing of auto_ptr per value.	20.4.5	Email: 21.5.95, X3J16: edit-566	yes, but need discussion on lib-reflector
R–17	The operational semantic for back() is wrong.	23.1, Table 53, p. 23-5	Email: 21.5.95, X3J16: edit-565	yes, edit-reflector
R–18	The meaning of comparing things is not consistent.	23.1, , p. 23-6	Email: 21.5.95, X3J16: edit-567	yes, edit-reflector
R–19	What is „m“ ?	23.3.1, p. 23-32	Email: 21.5.95, X3J16: edit-568	yes, edit-reflector
R–20	Logic error: replace !(*i	25.3.2, p.	Email:21.5.95,	yes, lib-reflector

	> *j) with !(*i < *j)	25-20	X3J16:lib-3822	
R–21	Just titles no contents	23,24,25	Email: 21.5.95, X3J16: edit-570	yes, edit-reflector
R–22	Findings in chapter 20,23,24,25	20	Fax: 30.6.95, X3J16: edit-579, lib-3829	yes, edit-reflector, lib-reflector
R–24	POD structures	9	X3J16: edit-575	yes, edit-reflector
R–25	Layout compatible types	3.9	X3J16: edit-576	yes, edit-reflector
R–26	Explicit initialization of globals before main	8	X3J16/95-0093, WG21/N0693	yes, ext-reflector

Status Revision 1:

The list mentioned above has been updated (X3J16/96-0051, WG21/N0869) and is therefore added to this document. From that list, the following items are solved by the current Draft (X3J16/96-0178, WG21/N0996): 19, 20, 21. All other issues must be checked against the CD-2 WP and will be kept open until CD-2 Ballot. With CD-2 Ballot, the German delegation expects to provide an update of this table. However, the German Delegation reports that they have no objections to shipping CD-2 with these open issues.

2. German public review comments

No.	Comment	Author	incoming Date	WP chapter	DIN WG Topic leader	related Document	German WG vote must,shall,nice ,opposed, abstain	Re-commen-dation for ISO/ANSI [status]
1	Derived Classes comments	Ulrich Eisenecker	26.6.1995	10	Eisenecker	Email: 26.6.95 X3J16: edit-561	not voted on	**yes**, [open]
2	Numerics Library	Ulrich Eisenecker	26.6.1995	26	Eisenecker	Email:26.6.95 X3J16: edit-564	not voted on	**yes**, [open]
3	class auto_ptr: the copy ctor should be private to forbid parameter passing of auto_ptr per value.	Uli Breymann	30.6.1995	20.4.5	Bormann, Breymann	Email: 21.5.95 X3J16: edit-566	not voted on	**yes**, [open]
4	What is „m“ ?	Uli Breymann	30.6.1995	23.3.1 p. 23-32	Bormann, Breymann	Email: 21.5.95 X3J16: edit-568	not voted on	**yes**, [open]
5	Logic error: replace !(*i > *j) with !(*i < *j)	Uli Breymann	30.6.1995	25.3.2 p. 25-20	Bormann, Breymann	Email:21.5.95 X3J16:lib-3822	not voted on	**yes**, [open]
6	Findings in chapter 20,23,24,25	Carsten Bormann	30.6.95	20	Bormann, Breymann	Fax: 30.6.95 X3J16: edit-579 lib-3829	not voted on	**yes**, [open]
7	Layout compatible types	Manfred Lichtmannegger	30.6.95	3.9	Unruh	X3J16: edit-576	not voted on	**yes**, [arrays still open]
8	pop() should return a value	Nicolai Josuttis	6.12.95	23.2.4.1 23.2.4.2 23.2.4.3	Kiefer	X3J16: lib-4496	pop_value()	**yes** [open]
9	Mandate for bad_alloc	Ulrich Eisenecker	7.12.95	26.3	Kiefer	X3J16: lib-?	not voted on	**yes** [open]
10	Can the class instantiated by the user ?	Ulrich Eisenecker	7.12.95	26.3.2.4 footnote 219	Kiefer	X3J16: lib-?	not voted on	**yes** [open]
11	Design of basic_istream and basic_ostream	Udo Mueller	8.12.95	27	Kiefer, Lichtmann-egger	X3J16: lib-?	note that op << with int is not possible	**yes** [open]
12	Editorial issues	Udo Mueller	8.12.95	20.3.6.1 20.4.1.1 20.4.1.3	Kiefer	X3J16: lib-?	not voted on	**yes** [open]
13	Editorial issues	Udo Mueller	8.12.95	27	Kiefer	X3J16: lib-?	not voted on	**yes** [open]
14	Wrong Specifications for Logical Iterator Conditions	Ulrich Breymann	6.1.96	25.3.3.1	Kiefer	X3J16: lib-?	not voted on	**yes** [open]
15	Missing iterator parameter	Ulrich Breymann	24.1.96	24.3.2.6.5	Kiefer	X3J16: edit-?	not voted on	**yes** [open]
16	Instantiation of virtual functions in templates	Udo Müller	13.2.96	14.3.2, §6	Unruh		**opposed** (4:1)	**no**
17	Minor Typos	Nicolai Josuttis	8.12.95	21.1.1.8.3 21.1.1.9.1 23.2.1 23.2.1.2	Kiefer	X3J16: edit-624	not voted on	**yes** [open]
18	Remove find_first, rfind and find_last	Nicolai Josuttis	8.12.95	21.1.1.9.3 21.1.1.9.4	Kiefer	X3J16: edit-624	WG recommends an informative vote	**yes** [open]
19	Introduce const & bitset::operator[] () const	Nicolai Josuttis	8.12.95	23.2.1	Kiefer	X3J16: lib-4496	not voted on	**yes** [open]
20	Specify #include <iterator>	Nicolai Josuttis	24.1.96	21.1	Kiefer	X3J16: lib-4496	not voted on	**yes**

	for containers			23.2 23.3				[open]
21	capacity() and reserve()	Nicolai Josuttis	24.1.96	21.1.1.6 23.2.5.4	Kiefer	X3J16: lib-4496	not voted on	**yes** [open]
22	Move min(), max() swap() into utility part	Nicolai Josuttis	24.1.96	25 -> utility	Kiefer	X3J16: lib-4496	not voted on	**yes** [open]
23	Move bitset section	Nicolai Josuttis	19.2.96	23.4	Kiefer	X3J16: edit-624	not voted on	**yes** [open]
24	Distance n=0	Ulrich Breymann	19.2.96	24.1.6	Kiefer	X3J16: edit-?	not voted on	**yes** [open]
25	Access to Exception Object's Type Information	Roland Hartinger	6.3.96	5.2.7 15.5.4 18.5.3	Hartinger	X3J16/96-0052 WG21/N0870 core-6404	not voted on	**yes** [open]

[The rest of this page left blank intentionally.]

Japan

[Disapproved with the following comments.]

In addition to technical comments, we would like to raise a procedural issue on the current CD ballot. After the draft for voting was sent to each national body, significant amount of modifications to the draft were discussed and agreed on before the voting deadline, i.e., in the Monterey meeting held in July. Since official voting is on the April draft, we find the situation unnatural. Such a situation should be avoided for future voting.

We agree to avoid this situation in the future.

In any case, our vote is on April draft, and our comments are on its contents; progress made in Monterey meeting is not reflected in our comments at all. We conducted a review by experts that consists of current working group members, people with experience with other standard activities such as internationalization, and recognized experts in C++ in academic and industrial communities. We are very much aware that the standard as proposed has considerable amount of complexity and the quality of the document remains much to be improved. The national body, however, felt that the current situation reflects the complexity and diversity of programming for modern system development, large and growing popularity of the language, and the effort that majority of people could afford to make under various constraints. When we consider practical alternatives, we believe that the current draft is a reasonable step toward a potential standard, and more effort is needed to stabilize and refine the draft before it is considered for a standard.

Among the various comments and issues discussed, we present the following three with the condition mentioned in the first paragraph. Most of other comments are of editorial nature, and they will be provided directly to the editor or to each subcommittee.

R–16 A) **Implementation Dependent Extensions** The current C standard permits implementation dependent extension with the introduction of the notion of conforming implementation as follows.

"A confirming implementation may have extensions (including additional library functions), provided they do not alter behavior of any strictly conforming program." (In section 1.7 compliance)

This is a very useful rule as one can build an implementation that permits extended set of characters such as Kanji for identifiers within the current framework of the C standard. We propose that the C++ standard should permit this.

We have revised clause 1.3 paragraph 7 as suggested here.

R–17 B) **String** We have provided a large number of comments on the draft on the subject. Taka Adachi has already communicated with other members of library subcommittee; some of issues raised are being in agreement. Among them, there are two outstanding issues due to potential of large changes.

1. The dependence of each member function on the traits should be made explicit.

The dependence of iostreams and string's member functions on the **traits** is made explicit by **21.1 Character traits [lib.char.traits]**, paragraph two which says:

"Most classes specified in clauses _lib.string.classes_ and _lib.input.output_ need a set of related types and functions to complete the definition of their semantics. These types and functions are provided as a set of member typedefs and functions in

the template parameter `traits' used by each such template. This subclause defines the semantics guaranteed by these members."

2. Possible elimination of use of charT() as default argument in seven members.

The only place where a string's function uses charT() as default argument is in:

iterator insert(iterator p, charT c = charT());

- potential of having eos() different from char() and explicit indication of dependence on traits

eos() as been removed, and is replaced by charT(). Therefore it is directly dependent on the character type.

Taka will communicate with library members and work actively to try to reach agreement.

R–18 C) **IO streams** A number of issues and comments were collected and most of them being communicated separately. We will describe one outstanding issue here. The problem is the lack of dependence of type definitions such as int_type, pos_type, off_type and state_type on charT in the definition of templated ios_traits.

(See comments at the end of the example.)

For example, considering 'char' specialization, we might define the following.

```
template <class charT> struct ios_traits {
   . . . .
   typedef charT char_type;
   typedef int   int_type;
   . . . .
   };
```

We would have to accept `int_type` as a constant definition in all of the specialized traits, not only in `ios_trait<char>`, but in `ios_traits<wchar_t>` and in `ios_traits<ultrachar>` as well. It would lead to the restriction upon implementations in that all of the `charT` have to be converted in 'int' range. This may be too restrictive for future wide character types and user-defined character types and user-defined character types.

Therefore, consider adopting

```
namespace std(
   template <class charT> struct ios_traits<charT> {}

struct ios_traits<char> {
   typedef char char_type;
   typedef int int_type;
   typedef streampos pos_type;
   typedef streamoff off_type;
   typedef mbstate_t state_type;

// 27.4.2.2 values:
   static char_type eos();
   static int_type  eof();
   static int_type  not_eof(char_type c);
   static char_type newline();
   static size_t    length(const char_type's);
   // 27.4.2.3 tests:
```

```
    static bool eq_char_type (char_type, char_type):
    static bool eq_int_type(int_type, int_type);
    static bool is_eof(int_type);
    static bool is whitespace (const ctype<char_type> ctype&, char_type);
  // 27.4.2.4 conversions:
    static char_type to_char_type(int_type);
    static int_type   to_int_type(char_type);
    static char_type* copy(char_type*dst, const carr* src, size_t n);
    static state_type get_state(pos_type);
    static pos_type   get_pos(streampos fpos, state_type state);
    };
    struct ios_traits<wchar_t> {
    typedef wchar_t char_type;
    typedef wint_t- int_type;
    typedef wstreampos pos_type;
    typedef wstreamoff off_type;
    typedef mbstate_t state_type;
  // 27.4.2.2 values:
    static char_type eos();
    static int_type  eof();
    static char_type not_eof(char_type C);
    static char_type newline();
    static size_t    length(const char_type* s);
  // 27.4.2.3 tests:
    static bool eq_char_type(char_type, char_type);
    static bool eq_int_type(int_type, int_type);
    static bool is_eof(int_type);
    static bool is_whitespace(const ctype<char_type> ctype&, char_type);
  // 27.4.2.4 conversions:
    static char_type  to_char_type(int_type);
    static int_type   to_int_type(char_type);
    static char_type* copy(char_type* dst, const char* src, size_t n);
    static state_type get_state(pos_type);
    static pos_type   get_pos(streampos fpos, state_type state);
    };
  }
```

As a result of the separation of the two specializations, we have to change the descriptions in [lib.streams.types], as follows; **27.4.1 Types**

```
typedef OFF_T streamoff;
```

The type streamoff is an implementation-defined type that satisfies the requirements of type OFF_T.

```
typedef WOFF_T wstreamoff;
```

The type wstreamoff is an implementation-defined type that satisfies the requirements of type WOFF_T.

```
typedef POS_T streampos;
```

The type streampos is an implementation-defined type that satisfies the requirements of type POS_T.

```
typedef WPOS_T wstreampos;
```

The type wstreampos is an implementation-defined type that satisfies the requirements of type WOPS_T.

```
typedef SIZE_T streamsize;
```

The type streamsize is a synonym for one of the signed basic integral types. It is used to represent the number of characters transferred in an I/O operations, or the size of I/O buffers.

The above approach can be found in "defining nothing in the template version of traits and defining everything in each specializations" by N. Kumagai (X3J16/94-0083). We regret that mistakes in the document for Austin (X1J16/95-0064) caused to introduce such inappropriate definitions to the current WP.

We should not put any definitions (static member functions or typedefs) related to int_type, off_type, pos_type and/or state_type in the template definition of the traits. The reason is that these three types depend on the template parameter class 'charT' for variety of environments (ASCII, stateless encoding for double byte characters, and UniCode). For example,

```
charT          char           wchar_t
int_type       int            wint_t
off_type       streamoff      wstreamoff
pos_type       streampos      wstreampos
state_type     mbstate_t      mbstate_t
```

Note that the two of the above types, 'wint_t' 'mbstate_t' are defined in C Amendment 1 (or MSE).

We cannot assume that two implementation-defined types, streampos and wstreampos have the same definitions because under some shift encoding, wstreampos has to keep an additional information, the shift state, as well as the file position. We should represent them with two different symbols. POS_T and WPOS_T so as to give a chance to provide separate definitions for these two specifications.

For pos_type in both of specialized traits, 'mbstate_t' is introduced from C Amendment 1 (or former MSE) and is an implementation-defined type to represent any of shift states in file encoding.

The type, INT_T is not suitable for the definition of streamsize because INT_T represents another character type whose meaning is to specify the definitions of streampos.

> A proposition equivalent in principle as been accepted; see 1.1 Character traits [lib.char.traits].

Netherlands

[Disapproved with the following comments.]

R–19 1. The document has a normative **reference to the C standard** (ISO/IEC 9899). However, C++ deviates from C in many places. Still, nowhere in the C++ standard these deviations are made explicit (other than in an informative annex which strictly speaking is no part of the standard). The compliance section should make clear what happens when a program makes use of these deviating features. As it stands, it is not written anywhere that the ‘new’ (C++) specification takes precedence over the ‘old’ (C) specification.

Except for explicit cross-references (see clause 1.2), an 'old' C specification is not normative for C++; in this respect, the C++ standard is self-contained.

R–20 2. The document is virtually **un-readable for non-experts**. Obviously, the document is not meant to teach a person C++. However, for the document to be useful, it should be reasonably accessible (i.e., readable) for any person with a reasonable experience in C++. It is clearly possible to write a standard that way (e.g. the C standard). Furthermore, it is clearly possible to write a definition of C++ that way (e.g., the books of B. Stroustrup).

We hope the current draft improves this situation.

R–21 3. The document is not very clear about **what can be expected of a C++ implementation**. A more precise definition of what is required of an implementation and what freedom the implementor has is needed. As an example, the C standard can be looked at, with sections on constraints (strict) and semantics (less strict).

We believe that our document is more strict in this regard than the C standard; in our document ***all*** negative requirements mandate diagnostics except where explicitly noted. In the C standard, there are numerous perfectly-diagnosable negative requirements that do not actually require diagnosis, for no reason other than that the requirement happens to appear in a semantics section rather than in a constraints section.

R–22 4. The standard library contains far **too many classes and features**. It is clear that C++ (like C) needs a standard library. The reason for this is that the language itself has no method for communication with the system (I/O, date/time and other system dependencies). A standard library for these system-dependent features is necessary to allow portable code to be written. The C standard was focussed upon these issues, however the proposed C++ standard library introduces many classes and features (in particular standard data structures and algorithms) that do not add to the portability of the language.

This is not to say that it would not be nice to have standard definitions for these features, but it should not be part of the C++ language standard. Adding these features to the C++ standard complicates both the implementation (e.g. for embedded systems) and the learning of the language. Furthermore, it is not clear at the moment whether the features included in the library are really what is needed. In fact, if one considers the currently available commercial libraries as an indication, there is much more need for a standard library for handling GUI's. This is understandable, since GUI interfaces are generally system dependent and a standard library for that would greatly improve portability. This cannot be said about e.g. a list data structure. A telling sign about the need for the features of the current library is that currently it is not supported by any of the major C++ development environments.

The proposed C++ library has now been implemented and commercially released by several vendors. The national bodies represented in the working group meetings felt strongly that such a standard library was necessary.

R–23 5. NNI objects to the fact that the complete **library has been defined in terms of templates**. This makes it more difficult to use the library without a complete and in-depth knowledge of the language. A grow-path from C to C++ is also not facilitated by the use of templates in the library. Furthermore, it makes it impossible to implement the library on older systems (where templates might not be

supported yet) and on smaller (e.g. embedded) systems (where templates are not implemented to save space).

We feel that the use of templates is essential to the production of a library which is both generic and efficient.

R–24 6. NNI requires that all **outstanding issues** on open issue lists (like the one in SC22/N1885) are dealt with in a satisfactory manner.

The new draft addresses what we feel are the important outstanding issues. WG21's internal issue lists include many items that are not essential to the standard, and the fact that some issue lists may still have unresolved issues is acceptable unless national bodies have comments on specific unresolved issues.

Minimal changes needed to change our vote to a (reluctant) yes are:

- Solving the issues mentioned in item 6
- Deleting the library from the standard (except for the language support libraries and the C standard library). (This would solve items 4+5).

The possibility of separating the language and the library has been raised several times, but all national bodies represented at the meetings have firmly resisted the separation, in favor of a single standard.

New Zealand

Did not vote.

Romania

Approved without comments.

Slovenia

Approved without comments.

Sweden

[Disapproved with the following comments.]

Note: document numbers of the form 95-0151/N0751 refer to SC22/WG21 documents.

R–25 **Progress of SC22/WG21** We are generally pleased with the current rate of progress of SC22/WG21. However, as a large number of minor issues remain to be considered and resolved, we favour submitting a second Committee Draft for balloting. We are convinced that the document will be ready for the DIS stage after the second CD ballot.

We agree.

The delay caused by submitting a second CD is unfortunate, but we nevertheless support Schedule Scenario #2 of document 95-0151/N0751.

Before the CD ballot closed and these comments were received, WG21 agreed on a slightly different (and longer) schedule.

R–26 **Issues lists** The most important work in the near future is to resolve all known outstanding problems, as documented by several “issues lists ” distributed in SC22/WG21.

See “General Comments” at the start of this document

R–27 **Clarification of templates [temp]** We believe that the template source model, the template compilation model, and the practical implications of both, need to be further clarified. In particular, there are several issues of great practical importance that cannot be discussed and analyzed because the necessary framework is missing. For example,

R–27.a Building libraries that are distributed in binary form.

R–27.b Distributed development, involving the combination of several libraries that may reference other common libraries.

R–27.c Shared libraries containing templates.

R–27.d Support for large projects, for example, if template instantiation can be implemented with linear algorithms.

R–27.e Portability of the source model.

R–27.f Early diagnostics.

(In response to R-27a-f) The description of the template compilation model has been substantially clarified, in particular with respect to name leakage and separate compilation of templates. The phases of translation have been similarly enhanced to cover template compilation.

R–28 **Copying semantics of auto_ptr [lib.auto.ptr]** The current semantics of the copy-constructor and the assignment operator of auto_ptr are error-prone, without providing great utility. We would instead prefer a distinct member function for transferring ownership from one auto_ptr object to another.

In essence we prefer the semantics described in document 94-0202/N0589 over the semantics described in the revised version 94-0202R1/N0589R1.

SC22/WG21 considered the issue but left it without action.

R–29 **Division of negative integers [expr.mul]** Paragraph 4: The value returned by the integer division and remainder operations shall be defined by the standard, and not be implementation defined. The rounding should be towards minus infinity. E.g., the value of the C expression (-7)/2 should be defined to be -4, not implementation defined. This way the following useful equalities hold (when there is no overflow, nor “division by zero ”):

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

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

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

> Division of negative integers has been extensively discussed in SC22/WG14 (C), proposing the desired specification in the next release of the ISO C standard. SC22/WG21 has not changed the corresponding normative text because the change in C is not yet effective, but added a footnote describing the intention of following ISO C.

R–30 **ISO Latin-1 in identifiers [lex.name]** We support the French CDR ballot comment suggesting that C++ allow letters of ISO 8859-1 (Latin-1) in identifiers. [See R–8.]

> We have added a method for representing and accepting a program that contains identifiers written in Latin-1, as well as other extended character sets. (See clauses 2.1, 2.2, and 2.10.) See also ballot comment R-8.

R–31 **Strengthening of bool datatype [conv.bool]** The original proposal for a Boolean datatype (called bool) provided some additional type-safety at little cost. SC22/WG21 changed the proposal to allow implicit conversion from int to bool, thereby reducing type-safety and error detectability.

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

> We feel that the current conversion rules are required for compatibility with existing C++ code.

R–32 **Definition of inline functions [dcl.fct.spec]** In paragraph 3 it is stated that "A call to an inline function shall not precede its definition. " One consequence of this restriction is that adding an inline keyword (which is only a recommendation, just like register) can make a program ill-formed. We suggest that this restriction is removed.

> The addition of the inline keyword is not merely a 'recommendation'; it invokes normatively different behavior.

R–33 **Declaration of ios_base::iword [lib.ios.base.storage]** Function iword() returns a long& while the text seems to imply that the array is an array of type int. We believe the return value shall be int&.

> We have corrected the specification of iword to use type long everywhere.

R–34 **Values of bool type [basic.fundamental]** Footnote 26 is confusing and shall be removed. Footnotes are not normative.

> A rigorous-checking implementation could use an "uninitialized boolean" value which is neither true nor false. The footnote just clarifies this possibility.

R–35 **Language Independent Arithmetic** We generally support a binding to LIA-1 (ISO/IEC 10967-1) in C and C++. Future work should track this standard and subsequent standards, e.g., LIA-2. If the current wording of the standard meets LIA-1, in particular the specification of the numeric_limits class, the standard should explicitly reference document ISO/IEC 10967-1.

> We explicitly reference ISO/IEC 10967-1.

R–36 **Unique type introduced by typedef** One of the Swedish comments on the CD suggested a new typedef that would introduce a distinct type. We think this issue should be further investigated at some point in the future. The proposed syntax is:

```
typedef explicit int Height;
typedef explicit int Length;
```

The issue has been deferred to a future revision of ISO C++.

Switzerland

Approved without comments.

Ukraine

Approved without comments.

United Kingdom

The UK votes NO to the CD ballot. The technical issues that the UK wants to see addressed are in the accompanying document 95/0013. The UK feels that so many substantive changes need to be made to the Draft Standard that it would be impossible for the vote to be changed to YES without first seeing the revised document, even if all our technical issues were accepted unchanged.

It should also be noted that it is our opinion that the document submitted was not stable enough to warrant a CD ballot, and did not include the editorial boxes, some of which dealt with contentious issues. WG21 has been actively pursuing changes, some of which are contrary to the CD document, during the ballot process. ...

R–37 ... The UK submits that it is necessary to have a **period of stability** prior to any future ballots: standards have a long term impact and rushing them into print for short term needs often results in very costly post-publication maintenance and public confusion.

> WG21 has endeavored to meet this requirement from the UK. The WD that was approved at the July 1996 Stockholm meeting was held stable through the November 1996 Kona meeting to the greatest extent possible, allowing only the correction of generally-agreed bugs and clarification of technical issues. We believe that this was accomplished in accordance with the UK requirements.

R–38 [**Document 95-0013** contains on the order of 300 issues, some of which are editorial.]

> WG21 believes that the current status is in accordance with the opinion of the UK delegation; that is, the great majority of these issues have been successfully resolved, and those that remain are not considered sufficiently major to delay publication of the second CD.

United States

Abstain. [The US TAG was unable to develop an official position that they could transmit to the ANSI.]

[End of collected comments]