Doc. no.: X3J16/95-0175
WG21/N0775
Date: 28 September 1995
Project: Programming Language C++
Reply to: Dag Brück
E-mail: dag@dynasim.se

Swedish comments on Committee Draft Ballot

Sweden has voted no with the following comments. However, we expect to change our vote to yes when the comments have been properly processed.

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.

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

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.

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,

- Building libraries that are distributed in binary form.
- Distributed development, involving the combination of several libraries that may reference other common libraries.
- Shared libraries containing templates.
- Support for large projects, for example, if template instantiation can be implemented with linear algorithms.
- Portability of the source model.
- Early diagnostics.

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.

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.

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.

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.

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.

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&`.

Values of bool type [basic.fundamental]

Footnote 26 is confusing and shall be removed. Footnotes are not normative.

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.

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;
```