Doc No: X3J16/96-0191 WG21/N1009
Date: November 7, 1996
Project: Programming Language C++
Ref Doc: X3J16/96-0185 WG21/N1003
Reply To: William M. Miller
wmm@world.std.com

REVISIONS TO PUBLIC REVIEW COMMENTS

Neal Gafter posted the following document to the c++std-all reflector suggesting changes to the responses to the ANSI public review comments (document X3J16/96-0185 = WG21/N1003). These suggestions will be considered as amendments to that document during its consideration at the November meeting.

The following is taken from Neal's posting (c++std-all-1547), except that I have added wording to implement the suggestion where that was needed. In such cases, my suggested wording is enclosed in [[double brackets]].

I note that a few of these responses should be changed, for example:

. . . . . .
SUGGESTION 1:

Template argument deduction (14.10.2).

In the example after paragraph 11, an argument of "aa" is deduced to be of type char*. I think this is wrong and potentially unsafe, since the effect of modifying a character literal is undefined and often harmful. Rather, the type deduced from a string literal should be const char*, so that if the generated function attempts to modify the string, the compiler can detect an error.
-> Rejected.
-> This would break C compatibility and C++ code too much.
-> (For example, function overload resolution would resolve
-> differently).

==========> Should be changed to accepted

[[Accepted: see 2.13.4 [lex.string] and 4.2 [conv.array].]]

. . . . . .
SUGGESTION 2:

An example at the end of [14.10.3] shows, once again, a type of char* being deduced from a string literal. I believe it should be const char*.
-> Rejected, see 14.10.2 above.

==========> Should be changed to accepted

. . . . . .
SUGGESTION 3:

9.6 Conversion to void

12.3.2 par. 1 says:
"If conversion-type-id is 'void' ..., the program is ill-formed"

It seems to me an unnecessary restriction to exclude user-defined

conversions to void, because it is well-defined, when voiding happens.

-> The language has been relaxed to allow declarations of user-defined
-> operator void. See 12.3.2 [class.conv.fct]

==========> We should note that such a conversion operator will never be used.

[[...See 12.3.1 [class.conv.fct]; however, there is no context in which such an operator will be implicitly invoked.]]

. . . . . .
SUGGESTION 4:

11.2 (Revision 1)
ISSUE 2) Function pointers and C linkage

Original code:

```
class foo
{
  // details omitted
  static int compare(void* key1, void* key2);
};
...
tree = tavl_init(foo::compare);    // pass function pointer
```

The problem is that class foo's implementation uses a C library (for handling threaded AVL trees), and this C library needs to be passed function pointers. The seventh compiler has different calling conventions for C and C++. Seeking a *portable* solution, the following change was suggested by the compiler vendor:

```
class foo
{
  // details omitted
  static int _cdecl compare(void* key1, void* key2);
};
...
tree = tavl_init(foo::compare);    // pass function pointer
```

The problem here is that _cdecl is not part of the C++ standard.

-> Accepted.
-> The extern "C" language linkage is not part of a pointer to function
-> type.

==========> The response is contradictory.
Reword based on recent decision.

[[Accepted. The language linkage is now part of the type of a function; see 7.5 [dcl.link]. However, the example as written must be modified to use a non-member comparison function, because member functions cannot be given a non-C++ language linkage.]]

. . . . . .
SUGGESTION 5:

15- Comment from Mok-Kong Shen
Received by email
email address: Mok-Kong.Shen@lrz-muenchen.de
Was comment T17 in the post-Monterey mailing document.

Subject: Multidimensional Arrays (8.3.4)

Abstract: The C++ multidimensional arrays are inferior to those of e.g. Fortran and thus need to be improved for the language to gain wider acceptance in the fields of engineering and scientific numerical computations hithertofore absolutely dominated by Fortran. It is suggested that a new data type be added to the C++ standard for that purpose.

-> Rejected.
-> Request for an extension.

==========> We should mention valarray

[[...Request for an extension. However, the standard library does provide the valarray class, along with related classes and functions (see 26.4 [lib.valarray.synopsis]), that are intended for use in applications such as those mentioned.]]