INTERNATIONAL STANDARD ISO/IEC 9899:1999
TECHNICAL CORRIGENDUM 3

Published 2007-11-15

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

Programming languages — C

TECHNICAL CORRIGENDUM 3

Langages de programmation — C

RECTIFICATIF TECHNIQUE 3

Technical Corrigendum 3 to ISO/IEC 9899:1989 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces.*

---

1. *Page 005, 3.9*

Change:
effective rounding mode
to:
current rounding mode

2. *Page 009, 5.1.1.2*

Paragraph 1 change footnote 5 to:
Implementations shall behave as if these separate phases occur, even though many are typically folded together in practice. Source files, translation units, and translated translation units need not necessarily be stored as files, nor need there be any one-to-one correspondence between these entities and any external representation. The description is conceptual only, and does not specify any particular implementation.

ICS 35.060 **Ref. No. ISO/IEC 9899:1999/Cor.3:2007(E)**

Published in Switzerland

3. *Page 018, 5.2.1.1*

Change paragraph 1 to:
Before any other processing takes place, each occurrence of one of the following sequences of three characters (called *trigraph sequences*[12)]) is replaced with the corresponding single character.

4. *Page 018, 5.2.1.1*

Paragraph 2, add new example paragraph before the existing example:
EXAMPLE 1:

```
??=define arraycheck(a,b) a??(b??) ??!??! b??(a??)
```

becomes

```
#define arraycheck(a,b) a[b] || b[a]
```

5. *Page 018, 5.2.1.1*

Paragraph 2 change **EXAMPLE:** to **EXAMPLE 2:**

6. *Page 024, 5.2.4.2.2*

Change paragraph 8 to:
Except for assignment and cast (which remove all extra range and precision), the values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type. The use of evaluation formats is characterized by the implementation-defined value of **`FLT_EVAL_METHOD`**:[19)]

7. *Page 036, 6.2.5*

Add a new paragraph after paragraph 22:
A type has *known constant size* if the type is not incomplete and is not a variable length array type.

8. *Page 044, 6.3.1.5*

Add to the end of paragraph 1:
(if the source value is represented in the precision and range of its type)

9. *Page 044, 6.3.1.5*

Change paragraph 2:
explicitly converted to its semantic type
to:
explicitly converted (including to its own type)

10. *Page 050, 6.4*

Change paragraph 4, last sentence to:
There is one exception to this rule: header name preprocessing tokens are recognized only within **`#include`** preprocessing directives and in implementation-defined locations within **`#pragma`** directives. In such contexts, a sequence of characters that could be either a header name or a string literal is recognized as the former.

11. *Page 054, 6.4.4*

Change the constraint section paragraph 2 to read:
Each constant shall have a type and the value of a constant shall be in the range of representable values for its type.

12. *Page 056, 6.4.4.1*

Add the following sentence as the last sentence of the paragraph after the table:
If an integer constant cannot be represented by any type in its list and has no extended integer type, then the integer constant has no type.

13. *Page 065, 6.4.7*

Change paragraph 3, last sentence to:
Header name preprocessing tokens are recognized only within **`#include`** preprocessing directives and in implementation-defined locations within **`#pragma`** directives[*)].

(* New Footnote) For an example of a header name preprocessing token used in a **`#pragma`** directive, see 6.10.9.

14. *Page 067, 6.5*

Add a footnote to paragraph 2, after the first sentence.
[*)]A floating-point status flag is not an object and can be set more than once within an expression.

15. *Page 073, 6.5.2.3*

Attach a new footnote to the words "named member" in paragraph 3:
[*)] If the member used to access the contents of a union object is not the same as the member last used to store a value in the object, the appropriate part of the object representation of the value is reinterpreted as an object representation in the new type as described in 6.2.6 (a process sometimes called "type punning"). This might be a trap representation.

16. *Page 081, 6.5.4*

Move the text of footnote 87 to normative text, after paragraph 4.

17. *Page 098, 6.7*

Change Paragraph 7:
in the case of function arguments (including in prototypes)
to:
in the case of function parameters (including in prototypes)

18. *Page 102, 6.7.2.1*

Append to Paragraph 6:
The keywords **`struct`** and **`union`** indicate that the type being specified is, respectively, a structure type or a union type.

19. *Page 106, 6.7.2.3*

Add a new paragraph following paragraph 1:
Where two declarations that use the same tag declare the same type, they shall both use the same choice of **`struct`**, **`union`**, or **`enum`**.

20. *Page 114 6.7.5*

Change 3rd sentence in paragraph 3 from:
If the nested sequence of declarators in a full declarator contains a variable length array type, the type specified by the full declarator is said to be *variably modified*.
to:
If in the nested sequence of declarators in a full declarator there is a declarator specifying a variable length array type, the type specified by the full declarator is said to be *variably modified*. Furthermore, any type derived by declarator type derivation from a variably modified type is itself variably modified.

21. *Page 116, 6.7.5.2*

Change the first sentence paragraph 2 to:
An ordinary identifier (as defined in 6.2.3) that has a variably modified type shall have either block scope and no linkage or function prototype scope.

22. *Page 122, 6.7.6*

In the syntax rules for *direct-abstract-declarator:* paragraph 1 replace:
direct-abstract-declarator$_{opt}$ **[** *assignment-expression*$_{opt}$ **]**
with:
direct-abstract-declarator$_{opt}$ **[** *type-qualifier-list*$_{opt}$ *assignment-expression*$_{opt}$ **]**
direct-abstract-declarator$_{opt}$ **[** **`static`** *type-qualifier-list*$_{opt}$ *assignment-expression* **]**
direct-abstract-declarator$_{opt}$ **[** *type-qualifier-list* **`static`** *assignment-expression* **]**

23. *Page 128, 6.7.8*

Change paragraph 24, example 1:
```
complex c = 5 + 3 * I;
```
to:
```
double complex c = 5 + 3 * I;
```

24. *Page 135, 6.8.5*

Append to paragraph 4:
The repetition occurs regardless of whether the loop body is entered from the iteration statement or by a jump.[*)]

[*)] Code jumped over is not executed. In particular, the controlling expression of a **`for`** or **`while`** statement is not evaluated before entering the loop body, nor is *clause-1* of a **`for`** statement.

25. *Page 136, 6.8.5.3*

Change paragraph 1:
If *clause-1* is a declaration, the scope of any variables it declares is the remainder of the declaration and the entire loop, including the other two expressions;
to:
If *clause-1* is a declaration, the scope of any identifiers it declares is the remainder of the declaration and the entire loop, including the other two expressions;

26. *Page 139, 6.8.6.4*

Add to footnote 136:
The representation of floating-point values may have wider range or precision and is determined by **`FLT_EVAL_METHOD`**. A cast may be used to remove this extra range and precision.

27. *Page 146, 6.10*

Change paragraph 2 to:
A *preprocessing directive* consists of a sequence of preprocessing tokens that satisfies the following constraints: The first token in the sequence is a **#** preprocessing token that (at the start of translation phase 4) is either the first character in the source file (optionally after white space containing no new-line characters) or that follows white space containing at least one new-line character. The last token in the sequence is the first new-line character that follows the first token in the sequence.[140] A new-line character ends the preprocessing directive even if it occurs within what would otherwise be an invocation of a function-like macro.

28. *Page 148, 6.10.1*

Insert new constraint paragraph after paragraph 1:
Each preprocessing token that remains after all macro replacements have occurred shall be in the lexical form of a token (6.4).

29. *Page 148, 6.10.1*

Change the third sentence in paragraph 3 to:
After all replacements due to macro expansion and the **defined** unary operator have been performed, all remaining identifiers (including those lexically identical to keywords) are replaced with the pp-number **0**, and then each preprocessing token is converted into a token.

30. *Page 150, 6.10.2*

Change paragraph 5 to:
The implementation shall provide unique mappings for sequences consisting of one or more nondigits or digits (6.4.2.1) followed by a period (**.**) and a single nondigit. The first character shall not be a digit. The implementation may ignore distinctions of alphabetical case and restrict the mapping to eight significant characters before the period.

31. *Page 151, 6.10.3*

Replace paragraph 9 with:
A preprocessing directive of the form
 # define *identifier replacement-list new-line*
defines an *object-like macro* that causes each subsequent instance of the macro name[146] to be replaced by the replacement list of preprocessing tokens that constitute the remainder of the directive. The replacement list is then rescanned for more macro names as specified below.

32. *Page 152, 6.10.3*

Change paragraph 10 to:
A preprocessing directive of the form
 # define *identifier lparen identifier-list*$_{opt}$) *replacement-list new-line*
 # define *identifier lparen ...*) *replacement-list new-line*
 # define *identifier lparen identifier-list , ...*) r*eplacement-list new-line*
defines a *function-like macro* with parameters, whose use is similar syntactically to a function call.

33. *Page 160, 6.10.8*

Add, in proper alphabetic order in the list of paragraph 2:
__STDC_MB_MIGHT_NEQ_WC__

The integer constant **1**, intended to indicate that, in the encoding for **wchar_t**, a member of the basic character set need not have a code value equal to its value when used as the lone character in an integer character constant.

34. *Page 170, 7.3.1*

Replace paragraphs 3 and 4 with:
[#3] The macros
imaginary
and
_Imaginary_I
are defined if and only if the implementation supports imaginary types;[*)] if defined, they expand to **_Imaginary** and a constant expression of type **const float _Imaginary** with the value of the imaginary unit.

[#4] The macro

I
expands to either **_Imaginary_I** or **_Complex_I**. If **_Imaginary_I** is not defined, **I** shall expand to **_Complex_I**.

[#5] Notwithstanding the provisions of subclause 7.1.3, a program may undefine and perhaps then redefine the macros **complex**, **imaginary**, and **I**.
[*)] A specification for imaginary types is in informative annex G.

35. *Page 187, 7.6*

Add a footnote to paragraph 1, after the third sentence.
[*)]A floating-point status flag is not an object and can be set more than once within an expression.

36. *Page 188, 7.6*

Change the last sentence of paragraph 5 to:
The defined macros expand to integer constant expressions with values such that bitwise ORs of all combinations of the macros result in distinct values, and furthermore, bitwise ANDs of all combinations of the macros result in zero.[*)]

[*)]The macros should be distinct powers of two.

37. *Page 214, 7.12.1*

Add to paragraph 4:
from finite arguments
in second sentence after
mathematical result is an exact infinity

38. *Page 239, 7.12.13.1*

Change paragraph 2:
the rounding mode characterized by the value of **`FLT_ROUNDS`**
to:
the current rounding mode

39. *Page 254, 7.17*

Change the last part of paragraph 2:

`wchar_t`

which is an integer type whose range of values can represent distinct codes for all members of the largest extended character set specified among the supported locales; the null character shall have the code value zero. Each member of the basic character set shall have a code value equal to its value when used as the lone character in an integer character constant if an implementation does not define **`__STDC_MB_MIGHT_NEQ_WC__`**.

40. *Page 260, 7.18.4*

Change paragraph 2:
a decimal, octal, or hexadecimal constant
to:
an unsuffixed integer constant

41. *Page 264, 7.19.1*

In paragraph 5, item 4, remove the **`perror`** after **`gets`**.

42. *Page 298, 7.19.7.7*

Add forward reference to 7.26.9

43. *Page 336, 7.22*

Paragraph 5 add:
(except **`modf`**)
after
`<complex.h>`

44. *Page 337, 7.22*

Paragraph 7 (3rd line from end) change:
`cabs(fc)`
to:
`fabs(fc)`

45. *Page 353 7.24.2.1*

Change g,G specifier in paragraph 8 to:
`g,G`

A **`double`** argument representing a floating-point number is converted in style **`f`** or **`e`** (or in style **`F`** or **`E`** in the case of a **`G`** conversion specifier), depending on the value converted and the precision. Let *P* equal the precision if nonzero, 6 if the precision is omitted, or 1 if the precision is zero. Then, if a conversion with style **`E`** would have an exponent of *X*:

—
if $P > X \geq -4$, the conversion is with style **`f`** (or **`F`**) and precision $P - (X + 1)$.

—
otherwise the conversion is with style **`e`** (or **`E`**) and precision $P - 1$.

Finally, unless the **`#`** flag is used, any trailing zeros are removed from the fractional portion of the result and the decimal-point wide character is removed if there is no fractional portion remaining.

46. *Page 402, 7.26.9*

Add new paragraph after paragraph 1:
The **`gets`** function is obsolescent, and is deprecated.

47. *Page , A.2.2*

In the syntax rules for (6.7.6) *direct-abstract-declarator:* replace line 3 with:
direct-abstract-declarator$_{opt}$ **[** *type-qualifier-list*$_{opt}$ *assignment-expression*$_{opt}$ **]**
direct-abstract-declarator$_{opt}$ **[** **`static`** *type-qualifier-list*$_{opt}$ *assignment-expression* **]**
direct-abstract-declarator$_{opt}$ **[** *type-qualifier-list* **`static`** *assignment-expression* **]**

48. *Page 446, Annex F.5*

Change paragraph 2:
correctly rounded
to:
correctly rounded (honoring the current rounding mode)

49. *Page 454, F.9*

Change paragraph 8 from:
Whether or when the trigonometric, hyperbolic, base-*e* exponential, base-*e* logarithmic, error, and log gamma functions raise the "inexact" floating-point exception is implementation-defined. For other functions, the "inexact" floating-point exception is raised whenever the rounded result is not identical to the mathematical result.
to:
Whether or when library functions raise the "inexact" floating-point exception is unspecified, unless explicitly specified otherwise.