



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.

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.