



Programming languages — C

TECHNICAL CORRIGENDUM 2

Langages de programmation — C

RECTIFICATIF TECHNIQUE 2

Technical corrigendum 2 to International Standard ISO/IEC 9899:1990 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Page 6

In subclause 5.1.2.1, page 6, delete:

There are otherwise no reserved external identifiers.

Page 7

In subclause 5.1.2.2.3, page 7, add at the end of the first sentence the footnote:

In accordance with subclause 6.1.2.4, objects with automatic storage duration declared in **main** will no longer have storage guaranteed to be reserved in the former case even where they would in the latter.

Page 11

In subclause 5.2.1.2, page 11, change the third bullet item:

wherein each sequence of multibyte characters begins in an *initial shift state* and enters other implementation-defined *shift states*

to:

wherein each sequence of multibyte characters begins in an *initial shift state* and enters other locale-specific *shift states*

Page 22

In subclause 6.1.2.4, page 22, first paragraph, change:

There are two storage durations: static and automatic.

to:

There are three storage durations: static, automatic, and allocated. Allocated storage is described in 7.10.3.

Page 25

In subclause 6.1.2.6, page 25, first paragraph, change:

Moreover, two structure, union, or enumerated types declared in separate translation units are compatible if they have the same number of members, the same member names, and compatible member types; for two structures, the members shall be in the same order; for two structures or unions, the bit-fields shall have the same widths; for two enumerated types, the members shall have the same values.

to:

Moreover, two structure, union, or enumerated types declared in separate translation units are compatible if at least one is an incomplete type or if they have the same number of members, the same member names, and compatible member types; for two complete structure types, the members shall be in the same order; for two complete structure or union types, the bit-fields shall have the same widths; for two enumerated types, the members shall have the same values.

Page 31

In subclause 6.1.4, page 31, change the last paragraph of Semantics (before the Example) from:

Identical string literals of either form need not be distinct. If the program attempts to modify a string literal of either form, the behavior is undefined.

to:

These arrays need not be distinct provided their elements have the appropriate values. If the program attempts to modify such an array, the behavior is undefined.

Page 36

In subclause 6.2.2.1, page 36, change the parenthetical remark in the final sentence of the first paragraph:

(including, recursively, any member of all contained structures or unions)

to:

(including, recursively, any member or element of all contained aggregates or unions)

Page 41

In subclause 6.3.2.2, page 41, second paragraph, change:

If the expression that denotes the called function has a type that includes a prototype, the arguments are implicitly converted, as if by assignment, to the types of the corresponding parameters.

to:

If the expression that denotes the called function has a type that includes a prototype, the arguments are implicitly converted, as if by assignment, to the types of the corresponding parameters, taking the type of each parameter to be the unqualified version of its declared type.

Page 45

In subclause 6.3.4, page 45, change the paragraph under Constraints:

Unless the type name specifies void type, the type name shall specify qualified or unqualified scalar type and the operand shall have scalar type.

to:

Unless the type name specifies a void type, the type name shall specify qualified or unqualified scalar type and the operand shall have scalar type.

Page 61

In subclause 6.5.2.2, page 61, second paragraph of Semantics, change:

Each enumerated type shall be compatible with an integer type; the choice of type is implementation-defined.

to:

Each enumerated type shall be compatible with an integer type. The choice of type is implementation-defined, but shall be capable of representing the values of all the members of the enumeration.

In subclause 6.5.2.2, page 61, append to Semantics:

The enumerated type is complete at the } that terminates the list of enumerator declarations.