ISO/IEC JTC1/SC22
Languages
Secretariat: CANADA (SCC)

ISO/IEC JTC1/SC22
N619

FEBRUARY 1989

TITLE:	Second DP9899:Information Processing-Programming Language C and Letter Ballot
SOURCE:	Secretariat ISO/IEC JTC1/SC22
WORK ITEM:	JTC1.22.20
STATUS:	Supersedes N413
CROSS REFERENCE:	N463
DOCUMENT TYPE:	Second DP9899-Programming Language C
ACTION:	For review and comments by SC22 Member Bodies. SC22 Member Bodies are requested to complete the enclosed Letter Ballot and return it to the SC22 Secretariat **BY 1989-06-30.**

Address reply to: ISO/IEC JTC1/SC22 Secretariat
J.L. Côté, 140 O'Connor St., 10th Floor
Ottawa, Ont., Canada K1A 0R5
Telephone: (613)957-2496 Telex: 053-3336

DRAFT PROPOSAL ISO/IEC DP 9899	
date 1989-03-03	reference number ISO/IEC JTC1/22N619
supersedes document N413	

THIS DOCUMENT IS STILL UNDER STUDY AND SUBJECT TO CHANGE. IT SHOULD NOT BE USED FOR REFERENCE PURPOSES.

work item number JTC1.22.20

ISO/IEC JTC 1/SC22

Title LANGUAGES

Secretariat CANADA, SCC

Circulated to P- and O-members of the JTC, technical committees and organizations in liaison for:

- discussion at
- comments by 1989-06-30
- voting by (P-members only) 1989-06-39

Title

INFORMATION PROCESSING - PROGRAMMING LANGUAGES C

Reference language version: ☐ English ☐ French

Introductory note

- First DP9899, document N413 circulated to SC22 Member Bodies for comments and and voting by 1988-01-22.

- The Summary of Voting and comments received on DP9899 were contained in document N463.

- Document N463 was submitted to SC22/WG14 - C for review and consideration in preparing a revised version of DP9899.

- The revised version of DP9899 is hereby circulated to SC22 Member Bodies for comment and vote.

89/61872

VOTE ON ISO/IEC DRAFT PROPOSAL DP9889	
date 1989-03-03	reference number ISO/IEC JTC 1 /SC 22 **N** 619
This ~~ballot~~ duly completed shall reach the JTC/SC secretariat by 1989-06-03	

P-members of the ISO/IEC joint technical committee concerned have an obligation to vote.

ISO/IEC JTC 1 **/SC** 22 Title LANGUAGES Secretariat CANADA, SCC	Circulated to SC22 'P' Member Bodies for voting SC22 'O' Member Bodies for information SC22 'L' Organization for comments

Subject of the ballot

SECOND **DP** 9899

Title: INFORMATION PROCESSING - PROGRAMMING LANGUAGE C

Please put a cross in the appropriate box(es)

Approval of the draft:

☐ as presented

☐ with comments as given below (use separate page as annex, if necessary)

- ☐ general
- ☐ technical
- ☐ editorial

☐ **Disapproval of the draft for reasons below (use separate page as annex, if necessary)**

- ☐ Acceptance of these reasons and appropriate changes in the text will change our vote to approval

☐ **Abstention** (for reasons below)

P-member voting:

date

signature

11 000

89/61872

Information Technology Standards Commission of Japan

ITSCJ

Information Processing Society of Japan

Kikai Shinko Building No. 3-5-8 Shiba-Koen Minato-ku, Tokyo 105, JAPAN

TEL : 03-431-2808
FAX : 03-431-6493
TX:02425340 IPSJ J

attachment 3

GENERAL COMMENTS

We will approve the DP9899 provided that the ISO standard should be identical with the ANSI C standard.

TECHNICAL COMMENTS

1. Editorial Errors

The following editorial errors should be corrected.

1.1 §3.1.2.2 Linkages of identifiers (P.22 L.14 - L.15)

"If the declaration of an identifier for an object or a function has file scope and contains the storage-class specifier static, the identifier has internal linkage."
should be change to
"If an identifier for an object or a function has file scope and it's declaration contains the storage-class specifier static, the identifier has internal linkage."

1.2 §3.3.2.3 Structure and union members (P.43 L.5 - L.7)

The use of the term "common initial sequence" apears before the definition is given in italics. The term should be defined at the first appearance.

1.3 §4.5 MATHEMATICS <math.h> (P.112 L.2 - L.3)

There is no specification about the relation between the type "double" and the "double-precision in this Standrd.

Therefore
"The functions take double-precision arguments and return double-precision values".
should be changed to
"The functions take type double arguments and return type double values."

89/67 809

Information Technology Standards Commission of Japan

ITSCJ

Information Processing Society of Japan

Kikai Shinko Building No. 3-5-8 Shiba-Koen Minato-ku, Tokyo 105, JAPAN

TEL : 03-431-2808
FAX : 03-431-6493
TX:02425340 IPSJ J

1.4 §A.6.1 Unspecified behavior (P.198 L.15 - L.16)

In this Standard there is no specification about the following statement "The order in which expressions are evaluated — in any order conforming to the precedence rules, even in the presence of parentheses(§3.3)."

Therefore the above item in appendix should be deleted.

1.5 §A.6.2 Undefined behavior (P.199 L.6 - L.8)

"Header name" is missed in the following sentence.
"A character not in the required character set is encountered in a source file, except in a prepocessing token that is never converted to a token, a character constant, a string literal, or a comment (§2.2.1)

Therefore the above sentence should be changed to
"A character not in the required character set is encountered in a source file, except in a prepocessing token that is never converted to a token, a character constant, a string literal, a comment, or a header name(§2.2.1)."

1.6 §A.6.2 Undefined behavior (P.199 L.17 - L.18)

In §3.1.2.2 there is no specification about the following statement "An identifier with external linkage is used but there does not exist exactly one external definition in the program for the identifier(§3.1.2.2)."
The concerning description has been moved to §3.7.

Therefore the above statement should be changed to
"An identifier with external linkage is used but there does not exist exactly one external definition in the program for the identifier(§3.7)."

1.7 §A.6.3.9 Structures, unions, enumerations, and bit-fields (P.204 L.33)

"The order of allocation of bit-fiels within an int(§3.5.2.1)."
should be changed to
"The order of allocation of bit-fiels within an unit (§3.5.2.1)."

89/04809

2. PROPOSALS

2.1 Const qualified type and cashing optimization (P.54 L.1 - L.2)

Acording to the Constraints in §3.3.16.1 the following program fragment seems to be correct.

```
int         *  ip;
const int  * cip;
/*      */
cip = ip;            /* The right type has no qulifier, so */
                     /* left the left type has has all the */
                     /* qualifier of the type pointed to   */
                     /* by the right.                      */
```

Furthermore, we think that an implementation may easily do "cashing optimization" for const qualified type.

```
const int       i = 1;

void f( int * b )
{
        const int    * cip = &i;

        g( *cip );
        *b = 2;
        h( *cip );
}
```

It is tempting to generate the call to g as g((r = *cip)) and the call to h as h(r) (where r is a register variable). We think optimizationa safe because the *cip has type const int, and cip is not modified by the assignment to *b.

Is our interpretation correct ?

If it is correct, the folloeing statement will be wrong.

```
int      i = 1;
int   * ip = &i;

void f( void )
{
        const int    * cip ;

        cip = ip;          /*  A1 */
        g( *cip );
        *ip = 2;
        h( *cip );
}
```

89/64809

Actually, the call to g should behave as if the source expression is g(1)
and call to h should behave as if the source expression is h(2).
But an implementation which does "cashing optimization" may generate the call
to g as g((r = *cip)) and the call to h as h(r). Therefore the call to h
may behave as if the source expression is h(1).
We think it is caused by the assignment expression A1.

Proposal:

Add the following COMMON WARNING.

In simple assignment, both operands are pointer or unqualified version of
compatible types, and the type pointed to by the left has const qualifier,
but the type pointed to by the right does not have it.

89/64809

BSi

attachment 4

UK COMMENTS ON ISO/IEC SECOND DP 9899
INFORMATION PROCESSING - PROGRAMMING LANGUAGE C

The UK approves ISO/IEC Second DP 9899 - N619, with comments attached.

SRB/srb
29 June 1989.

89/64809

Subject: ISO Document DP9899 - N619
(Second DP9899 - Programming Language C)

Based on a review of the subject document, the UK votes YES though with considerable misgivings, overcome only by accepting that there is a general wish among the 'C' community for a standard to be published as soon as possible, even if it has deficiencies. An excessive number of undefined aspects and ambiguities remain in the document and those identified in the UK are listed in an attached document. Our yes vote is on the understanding that WG14 will proceed immediately to produce an ancillary document addressing such issues not already resolved, by the time the primary document reaches DIS stage, and that this document, once agreed also be adopted by the US member body for its domestic standard and by any other member body with equivalent domestic standards.

The UK believes that the first revision of the 'C' standard, incorporating the interpretations in the additional document just referred to, should be developed by WG14 directly and that the terms of reference of WG14 (Paris resolution) should be revised accordingly.

Comments on the December 7, 1988 Draft Proposed C standard

1) page 2 line 36, page 3 line 6. Definition of Byte and Character are self referencing and circular.

2) page 2-3. Definitions should be in alphabetic order.

3) page 3 line 30. Paragraph should be given its own heading.

4) page 3 line 32. "... or by the omission ...", but this is "Unspecified behaviour" as defined on line 21 above.

5) page 6 line 37. "Preporcessing directives are executes and macro invocations are expanded"

page 87 line 1-5. "A preprocessing directive ... and is ended by the next new-line character"

page 90 line 40. "within the sequence of preprocessing tokens ... new-line is considered a normal white-space character"

These three statements are not sufficient to define the following:

```
#define f(a,b) a+b
#if f(1,
      2)
...
```

It should be defined whether the preprocessing directive rule or macro expansion wins.

6) page 8. It should be a constraint error or explictly undefined if no function called main exists in the executable.

7) page 21 Implementation limits. It is currently undefined behaviour if the implementation does not treat 31 characters as being significant. It should be a constraint error if the implementation does not support this feature.

8) page 21 line 26. It is undefined behaviour if label names are not unique within a function. This should be an error.

9) page 32 line 37 " ... [illegible]

10) page 35 line 13. A nit pick really.

in:

```
        int i;
        enum foo{a=1} x;
        i=(0,x) +1;    /* illegal */
        i=x+1     ;    /* legal */
```

The type of (0,x) is enum foo. However, it is not an object and the draft standard does not explicitly allow it to occur in an int context.

Does the committee intend this behaviour?

12) page 38. ".. with the value 0", should refer to the NULL macro.

13) page 39 footnote 33. Hang over from noalias days?

14) sizeof has type size_t. It should be a constraint error or explicitly undefined if the definition of size_t obtained from an include file does not match what the compiler considers the type of sizeof to be (the compiler has to maintain an internal type for sizeof in case the user does not include a header that defines this type).

The above comments also apply to ptrdiff_t, page 48 line 34.

15) An inferior implementation may flag a statement as being in error, while a quality implementation treats it as being legal. Consider:

```
char *p;
p=1-1;
```

A quality implementation would fold 1-1 to give zero. If the NULL macro is defined as 0 then the assignment is legal, it is the null-pointer-constant.

An inferior implementation would not fold the 1-1 and would flag the assignment as being in error.

The standard should explictly state that in an expression context only unfolded constants may become null-pointer-constant.

16) page 41 line 30. Suggest change of "... in the innermost block containing ..." be changed to "... in the innermost block, after any explicit declarations, containing ...". This extra wording keeps:

```
int (*g)()=f;
...
f();
```

illegal.

Without it the implicit declaration that occur on encountering the function call f() might make the initialiser given on the declaration of g legal. A one pass compiler would already have flagged the initialiser on g as being in error.

An even better solution would be to implicitly declare f with file scope.

89/64809

17) page 53 line 13. "... or differently qualified versions of a compatible type, the result has the composite type; ...", but no rules are given on page 26 for creating a composite type for the expression. In:

```
x ? (const int *)y : (int *) z
```

what is the composite type of (const int *) and (int *) ?

also in:

```
x ? (const int *)y : (const void *) z
```

the result type is (void *), contradicting line 11 above.

18) page 58 line 16. This constraint currently allows zero sized structs, as in:

```
struct { struct a{int b;};}
```

The reference to "... no named members, ..." in page 61 line 25 presumably refers to bit fields.

19) page 68. What is the type of constant-expression in array declarations. In the following what are the array sizes?

```
int a1[32767+1];
int a2[65000*65000];
```

20) page 68 clause 3.5.4.3. An ambiguity in needs resolving in the parsing of the following:

a) `int x(T (U));`

b) `int x(T (U (int a, char b)));`

In (a) U is the type of the parameter to a function returning type T. From page 69 line 2: "In a paramter declaration, a single typedef name in parentheses is taken to be an abstract declarator that specifies a function with a single parameter, not as redundant parentheses around the identifier for a declarator."

Thus in the case of (b):

1) U could be a redundantly parenthesized name of a function which takes a parameter-type-list and returns type T, or

2) U could be the type returned by a function which takes a parameter-type-list; which in turn is the single parameter of a function returning type T.

21) page 71 line 5. It is undefined whether the following is legal or illegal:

```
typedef t[];
t a={1,2}, b{3,4,5};
```

The behaviour for this case should be defined.

89/64809

22) page 72 line 43. Change "... its value is indeterminate." to "... its value is undefined.".

23) page 81 line 24. Change "... whose return type is void" to "... whose unqualified return type is void".

24) page 83 line 34-35. The fact that the storage-class specifier and type-specifier may be omitted, defaulting to extern and int respectively should be in the semantics clause, not an example.

25) page 84 line 1. The fact that parameter declarations may be omitted and default to int ought to be in the semantics section.

26) pointer to multi-dimensional arrays. Neils query.

28) page 82 line 22. Implies that there may be more than one external definition provided the object does not occur in an expression.

It should be a constraint or explictly undefined if this situation occurs.

29) There is no semantics given for the replacement oj object macros. This should be defined.

30) page 90. In:

```
#define f(a) a*g
#define g(a) f(a)
f(2)(9)
```

it should be defined whether this results in 2*f(9) or 2*9*g

31) page 168 line 36. "... first call ..." should read "... all calls ..."

SRB/DJ/srb
29 June 1989

89/64809