WG14/N322
X3J11/94-006

ISO / IEC JTC1 / SC22
Programming languages, their environments and system software interfaces
Secretariat: CANADA (SCC)

ISO/IEC JTC1/SC22
N 1531

DECEMBER 1993

TITLE: Summary of Voting on PDAM1 to ISO/IEC 9899:1990 on Normative Addendum to Programming language C

SOURCE: Secretariat ISO/IEC JTC1/SC22

WORK ITEM: JTC1.22.20.02

STATUS: New

CROSS REFERENCE: N1443

DOCUMENT TYPE: Summary of Voting

ACTION: For information to SC22 Member Bodies.
See attached.

Address reply to: ISO/IEC JTC1/SC22 Secretariat
J.L. Côté
Treasury Board Secretariat
300 Laurier Ave. West, 10th Floor, Ottawa, Ontario, Canada K1A 0R5

4/27

SUMMARY OF VOTING ON:

Letter Ballot Reference No: SC22 N1443
Circulated by :JTC1/SC22
Circulation Date :1993-09-01
Closing Date :1993-12-10

SUBJECT: PDAM1 to ISO/IEC 9899 on Normative Addendum to Programming language C

The following responses have been received:

'P' Members supporting proposal,
without comments : 10

'P' Members supporting proposal,
with comments : 00

'P' Members not supporting proposal, : 03

'P' Members abstaining : 01

'P' Members not voting : 08

Secretariat Action:

The comments received will be forwarded to WG14 - C for recommendation on further processing of PDAM1.

ISO/IEC JTC1/SC22 LETTER BALLOT SUMMARY

PROJECT NO: JTC1.22.20.02

SUBJECT: CD9899:1990 PDAM1: Normative Addendum to ISO/IEC 9899:1990 Programming Language C

Reference Document No: N1443
Circulation Date: 1993-09-01

Ballot Document No: N1443
Closing Date:1993-12-10

Circulated To: SC22 P, L

Circulated By: Secretariat

SUMMARY OF VOTING AND COMMENTS RECEIVED

	Approve	Disapprove	Abstain	Comments	Not Voting
'P' Members					
Australia	( )	( )	(x)	( )	( )
Austria	( )	( )	( )	( )	(✓)
Belgium	(x)	( )	( )	( )	( )
Brazil	( )	( )	( )	( )	(✓)
Bulgaria	( )	( )	( )	( )	(✓)
Canada	(x)	( )	( )	( )	( )
China	( )	( )	( )	( )	(✓)
Denmark	(x)	( )	( )	( )	( )
Finland	(x)	( )	( )	( )	( )
France	( )	( )	( )	( )	(✓)
Germany	(x)	( )	( )	( )	( )
Greece	( )	( )	( )	( )	(✓)
Italy	(x)	( )	( )	( )	( )
Japan	( )	(x)	( )	(x)	( )
Netherlands	( )	(x)	( )	(x)	( )
New Zealand	(x)	( )	( )	( )	( )
Slovenia	( )	( )	( )	( )	(✓)
Sweden	(x)	( )	( )	( )	( )
Switzerland	( )	( )	( )	( )	(✓)
UK	( )	(x)	( )	(x)	( )
Ukraine	(x)	( )	( )	( )	( )
USA	(x)	( )	( )	( )	( )
'O' Members					
Argentina	( )	( )	( )	( )	( )
Cuba	( )	( )	( )	( )	( )
Czech/Slovak Re	(x)	( )	( )	( )	( )
Hungary	( )	( )	( )	( )	( )
Iceland	( )	( )	( )	( )	( )
India	( )	( )	( )	( )	( )
Poland	(x)	( )	( )	( )	( )
Portugal	( )	( )	( )	( )	( )
Romania	( )	( )	( )	( )	( )
Singapore	( )	( )	( )	( )	( )
Turkey	( )	( )	( )	( )	( )
Thailand	( )	( )	( )	( )	( )
Yugoslavia	( )	( )	( )	( )	( )

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 : 03-3431-2808
FAX : 03-3431-6493
TX:02425340 IPSJ

Japan does NOT agree to the circulation of "ISO/IEC JTC1/SC22 N1443 Proposed Draft Amendment 1 to ISO/IEC 9899:1990 Programming language C on: Normative Integrity Addendum" as a DIS.

Japan will vote YES, provided that the following comments are accepted.

---

o Japanese working paper for draft proposed "Rationale" (see attachement) should be added into the Annex as a clause "Annex B. Rationale".

o "Background" in the clause "1 Scope" (from page 1, line 28 to page 2, line 44) should be moved to the new clause "Annex B. Rationale" in the Annex.

o Page 3, line 29, "3.3 Version macro";
The top of the line started with the words "this amendment." should be correctly indented.

o Page 5, line 32, Synopsis in "4.5.2.1.1 The iswalnum function"
The error of document formatter command "iswalph<F255D or iswdigit" should be corrected.

o Page 5, line 37, Synopsis in "4.5.2.1.2 The iswalpha function"
"intiiswalpha(wint_t wc);" should be corrected to
"int iswalpha(wint_t wc);"

o Page 5, line 39, Description in "4.5.2.1.2 The iswalpha function"
"forwwhich" should be corrected to "for which".

o Page 6, line 41, Description in "4.5.2.1.8 The iswpunct function"
The reference to the footnote 9) should be added at the bottom of the sentence "... for which neither iswapace nor iswalnum is true."

o Page 15, line 49, the footnote 15)
The footnote should be moved to the bottom of the page 14.

o page 16, line 47, Description in "4.6.2.4.1 The fwprintf function"
The conversion specifier "%s" in the sentence "an array of wchar_t type using %s conversion" should be changed to "%ls".

o page 17, line 37, Description "4.6.2.4.2 The fwscanf function"
The last comma in the sentence "the conversion specifiers c, s, and..." should be deleted.

o Page 21, line 10, Description in "4.6.2.4.5 The swprintf function"
The error of document formatter command "the argument F002Bs" should be corrected.

o Page 21, line 13, Returns in "4.6.2.4.5 The swprintf function"
The error of document formatter command "@STD_PH=Returns" should be corrected.

o Page 23, line 41, Description in "4.6.2.5.4 The fputws function"
"The terminating null byte" should be changed to
"The terminating null wide character."

o Page 35, line 23, Returns in "The wctob function"
The sentence "the single-byte representation" should be changed to "the single-byte representation of that character."

---- End of Comments ----

430

Annex B
(informative)
Rationale

B.1 Background of this Amendment

Most traditional computer systems and computer languages, including C, have an assumption (sometimes undocumented) that a "character" can be handled as an atomic quantity associated with a single memory storage unit - a "byte" or something similar. This is not true in general. For example, a Japanese, Chinese, or Korean character usually requires a representation of two or three bytes; this is a *multibyte character* as defined by ISO/IEC 9899:1990 subclause 3.13. Even in the Latin word, a multibyte coded character set may appear in the near future. This conflict is called a *byte and character problem*.

A related concern in this area is how to address having at least two different meanings for string length: number of bytes and number of characters.

To cope with these problems, many technical experts, particularly in Japan, have developed their own sets of additional multibyte character functions, sometimes independently and sometimes cooperatively. Fortunately, the developed extensions are actually quite similar. It can be said that in the process they have found common features for multibyte character support. Moreover, the industry currently has many good implementations of such support.

The above in no way denigrates the important groundwork in multibyte and wide-character programming provided by ISO/IEC 9899:1990:

- Both the source and execution character sets can contain multibyte characters(with possibly different encodings), even in the "C" locale.
- Multibyte characters are permitted in comments, string literals, character constants, and header name.
- The language support wide-character constants and strings.
- The library has five basic functions that convert between multibyte and wide characters.

However, these five functions are often too restrictive and too primitive to develop portable international programs that manage characters.

Consider a simple program that wants to count the number of characters, not bytes, in its input. The prototypical program,

```
#include<stdio.h>

int main(void)
{
    int c, n = 0;

    while ((c= getchar()) ! EOF)
        n++;
    printf("Count = %d\n", n);
    return 0;
}
```

does not work as expected if the input contains multibyte characters; it always count the number of bytes. It is certainly possible to rewrite this program using just some of the five basic conversion functions, but the simplicity and elegance of the above is lost.

International Standard ISO/IEC 9899:1990 deliberately chose not to invent a more complete multibyte and wide-character library, choosing instead to await their natural development, as the C community acquired more experience with wide-characters. The task of committee ISO/IEC JTC 1/SC22/WG14 was to study the various existing implementations and, with care, develop this first amendment to ISO/IEC 9899:1990. The set of

Similarly, International Standard ISO/IEC 9899:1990 deliberately chose not to address in detail the problem of working C source code with character sets such as the national variants of ISO 646. These variants often redefine several of the punctuation characters used to write a number of C tokens. The (admittedly partial) solution adopted was to add *trigraphs* to the languages. Thus, for example, ??< can appear any where in a C program that { should appear, even within a character constant or string literals.

This amendment reponds to an international sentiment that more readable alternatives should also be provided, wherever possible. Thus, it adds to the language alternate spelling of several tokens. It also adds to library header, <iso646.h>, that defines a number of macros that expand to still other tokens which are less readable when spelled with trigraphs. Note, however, that trigraphs are still the only alternative to writing certain characters within a character constant and a string literal.

An important goal of any amendment to an international standard is to minimize *quiet changes* - changes in the definition of a programming language that transform a previously valid program, or into an invalid program that need not generate a diagnostic message, with different behavior. (By contrast, changes that invalid a previously valid program are generally considered palatable if they generate an obligatory diagnostic message at translation time.) Nevertheless, this amendment knowingly introduces two classes of quiet changes:

new tokens - The tokens %: and %:%: are just preprocessing tokens in International Standard ISO/IEC 9899:1990 but are given specific meanings in the amendment. An existing program that uses either of these tokens in a macro argument can behave differently as a result of the amendment.

new function names - Several names (with external linkage) not reserved to implementation in International Standard ISO/IEC 9899:1990, such as btowc, are now so reserved if *any* translation unit in the program includes either of the header <wctype.h> or <wchar.h>. An existing program that uses any of these names can behave differently as a result of this amendment.

B.2 Programming Model Based on Wide Characters

Using the MSE functions, a multibyte character handling program can be written as easily and as in the same style as a traditional single-byte based program.

A programming model based on MSE function is as follows: Firstly, a multibyte character or a multibyte string is read from external file into a wchar_t object or a wchar_t array object by the fgetwc function or other input functions based on the fgetwc function such as getwchar, getwc, fgetws. While this reading, a code conversion is occurred. Namely, these input functions convert the multibyte character to the corresponding wide character as if by a call to the mbtowc function. After or while necessary repeats of reading, the wchar_t objects are processed in memory by the MSE functions like iswxxx, wcstod, wcscpy, wmemcmp and so on. Finally, the resulted wchar_t objects are written to external file as a sequence of multibyte characters by the fputwc function or other output functions based on the fputwc function such as putwchar, putwc, fputws. While this writing, a code conversion is occurred. Namely, these output functions convert the wide character to the corresponding multibyte character as if by a call to the wctomb function.

In case of the formatted input/output functions, the similar programming style can be applicable except that the character code conversion may also be done through extended conversion specifiers, such as %ls, %lc.

For example, the wide character based program corresponding to the subclause "Annex B.1" can be written as follows;

```
#include<stdio.h>
#include <wchar.h>

int main(void)
{
        wint_t wc;
        int n = 0;

        while ((wc = getwchar()) ! WEOF)
                n++;
        wprintf(L"Count = %d\n", n);
        return 0;
}
```

B.3 Parallelism versus Improvement

When defining the MSE library functions, the Committee could have a possibility to chose a design policy between two candidates, such as a *parallelism* and a *improvement*.

The policy of parallelism means that an interface of function defined in this amendment is similar to the corresponding single-byte based function in ISO/IEC 9899:1990. Namely, the number of parameter of both parallel functions are exactly same, and the type of parameter and the type of return value have a simple correspondence as shown below:

```
char <-> wchar_t
int  <-> wint_t
```

An approach along this policy is relatively easy.

On other hand, the improvement means that an interface of functions in this amendment is changed from the corresponding single-byte based functions in ISO/IEC 9899:1990 in order to resolve the problems potentially contained in the existing functions. Or, the corresponding functions are not introduced in this amendment when the functionality can be substituted by other functions.

On the way of the approach of improvement, there were a lot of collision of views in order to get the most appropriate interface. Moreover, much careful considerations and discussion among various experts in this area was necessary to decide which policy should be taken for each functions. The current amendment is the result of these history.

The following is the list of the parallel functions that treat characters.

ISO/IEC 9899:1990	Amendment
isalnum	iswalnum
isalpha	iswalpha
iscntrl	iswcntrl
isdigit	iswdigit
isgraph	iswgraph
islower	iswlower
isprint	iswprint
ispunct	iswpunct
isspace	iswspace
isupper	iswupper
isxdigit	iswxdigit
tolower	towlower
toupper	towupper
fprintf	fwprintf
fscanf	fwscanf
printf	wprintf
scanf	wscanf
sprintf	swprintf
sscanf	swscanf
vfprintf	vwfprintf
vprintf	vwprintf
vsprintf	vswprintf
fgetc	fgetwc
fgets	fgetws
fputc	fputwc
fputs	fputws
getc	getwc
getchar	getwchar
putc	putwc
putchar	putwchar
ungetc	ungetwc
strtod	wcstod
strtol	wcstol
strtoul	wcstoul
memcpy	wmemcpy
memmove	wmemmove

strcat	wcscat
strncat	wcsncat
memcmp	wmemcmp
strcmp	wcscmp
strcoll	wcscoll
strncmp	wcsncmp
strxfrm	wcsxfrm
memchr	wmemchr
strchr	wcschr
strcspn	wcscspn
strpbrk	wcspbrk
strrchr	wcsrchr
strspn	wcsspn
strstr	wcsstr
memset	wmemset
strlen	wcslen
strftime	wcsftime

The following is the list of the functions that have different interface between single and wide character version. The sprintf family based on wide character have an extra size_t parameter in order to repair the security hole that the existing functions are potentially carrying. And the wide character based strtok function, that is wcstok, has an extra wchar_t ** parameter in order to eliminate the internal memory that the strtok function have to hold.

```
int sprintf(char *s, const char *format, ...);

        int swprintf(wchar_t *s, size_t n, const wchar_t *format, ...);

int vsprintf(char *s, const char *format, va_list arg);

        int vswprintf(wchar_t *s, size_t n, const wchar_t *format,
                va_list arg);

char *strtok(char *s1, const char *s2);

        wchar_t * wcstok(wchar_t * s1, const wchar_t *s2,
                wchar_t **ptr);
```

The following is the list of the functions in ISO/IEC 9899:1990 that do not have corresponding partner in the amendment because of several reasons, that is, a redundancy, a dangerous behavior, or an unnecessity in the wide character based program. Most of them can be replaced by other functions.

gets
puts
perror
atof
atoi
atol
strerror

The following is the list of the functions in this amendment that do not have corresponding partner in ISO/IEC 9899:1990. They were introduced in order to achieve the perfect conversion between the multibyte character(s) and the wide character(s), or in order to give character handling programs a flexibility and simplicity.

wctype
iswctype
wctrans
towctrans
fwide
btowc
wctob
mbsinit
mbrlen
mbrtowc
wcrtomb
mbsrtowcs
wcsrtombs

blank page

B.4 Support for ISO 646 invariant character set environment

With its rich set of operators and punctuators, the C language makes heavy demands on the ASCII character set. Even before the language was standardized, it presented problems to those who would move C to EBCDIC machines. More than one vendor provided alternate spellings for some of the tokens that used characters with no EBCDIC equivalent. With the spread of C throughout the world, such representation problems have only grown worse.

ISO 646, the international standard corresponding to ASCII, permits national variants of a number of the characters used by C. Strictly speaking, this is not a problem in representing C programs, since the necessary characters exist in all such variants. They just print funny. Displaying C programs for human edification suffers, however, since the operators and punctuators can be hard to recognize in their various altered forms.

The C Standard addresses the problem in a different way. It provides replacements at the level of individual characters using three-character sequences called 'trigraphs.' For example, '??<' is entirely equivalent to '{', even within a character constant or string literal. While this approach provides a complete solution for the known limitations of EBCDIC and ISO 646, the result is arguably not highly readable.

Thus, this Amendment provides a set of more readable 'digraphs.' These are two-character alternate spellings for several of the operators and punctuators that can be hard to read with ISO 646 national variants. Trigraphs are still required within character constants and string literals, but at least the commoner operators and punctuators can have more suggestive spellings using digraphs.

The added digraphs were intentionally kept to a minimum. Wherever possible, we instead provided alternate spellings for operators in the form of macros defined in the new header <iso646.h>. Digraphs are provided for the preprocessing operators # and ## because they cannot be replaced by macro names. Digraphs are also provided for the punctuators [,], {, and } because macro names proved to be a less readable alternative. We recognize that the solution we offer is incomplete and involves a mixture of approaches, but we believe that it can help make Standard C programs more readable.

B.5 Headers

B.5.1 <wchar.h>

B.5.1.1 Prototypes in <wchar.h>

The function prototype declarations for MSE library functions were necessary to be included in a certain header.

The Committee considered following ideas;

(1) introducing of the new headers such as <wctype.h>, <wstdio.h> or <wstring.h>, which are symmetry to headers specified in ISO 9899:1990 such as <ctype.h>, <stdio.h> or <string.h>, in order to declare MSE function prototypes in corresponding new headers

(2) declaring all of MSE function prototypes in <stdlib.h>, where wchar_t is defined

(3) introducing new header and declaring all of MSE function prototypes in the new header

(4) declaring MSE function prototypes in existing headers specified in ISO 9899:1990 related these functions

The defect of idea (1) is; the relationship between new headers with existing ones, especially their dependencies, becomes complicated. For example, two headers may be included in prior to <wstdio.h> header, as below:

```
#include <stdlib.h>
#include <stdio.h>
#include <wstdio.h>
```

The defect of idea (2) is; the program has to include many prototype declarations even if the program does not need declarations in <stdlib.h> other than existing ones. And the Committee opposed strongly to add any identifiers to existing headers.

The defect of idea (3) is; the asymmetry of existing headers and new header.

The defect of idea (4) is; the Committee opposed strongly to add any identifiers to existing headers.
The Committee decided to introduce a new header <wchar.h>, in order to declare all MSE function prototypes.

B.5.1.2 Declared types and macros in <wchar.h>

The concern that the declarations for types and macros in <wchar.h> should be specified in an efficient way has arisen.

The Committee considered that necessary header for using MSE library functions is <wchar.h> only. But there were strong oppositions to declaring some existing types such as FILE in the new header.

The declarations in <wchar.h> other than MSE prototypes are limited those that have high-independency. Existing header may be included with <wchar.h>, if necessary. And when the implementation supports MSE, two types, FILE and fpos_t declared in <stdio.h> are needed to revise suitably.

B.5.2 <wctype.h>

The committee originally intended to place all MSE functionality in a single header, <wchar.h>. We found, however, that this header was excessively large, even compared to the existing large headers <stdio.h> and <stdlib.h>. We also observed that the wide-character classification and mapping functions seemed to form a separate group. (These are functions that typically have names beginning with 'isw' or 'tow'.) A translation unit could well make use of most of the functionality of the MSE without using this separate group. Equally, a translation unit might need the wide-character classification and mapping functions

<ctype.h>. That division also reduced the size of <wchar.h> to more manageable proportions.

B.6 Wide character classification functions

The eleven isw* functions which correspond to the character testing functions defined in ISO 9899:1990 have been introduced. Each wide character testing function is specified in parallel with the matching character handling function, i.e. character v.s. wide character. However, the following changes are also introduced in addition to handling wide characters.

B.6.1 No "C" locale restriction for isw* functions

The behavior of character testing functions in ISO 9899:1990 is affected by the current locale. And some of the functions have implementation-defined aspects only when not in the "C" locale. The behavior for the "C" locale is specified as, for example, In the "C" locale, islower returns true only for the characters defined as lower-case letters (as defined in 5.2.1).
This "C" locale restriction existing for character testing functions in ISO 9899:1990 has been replaced with supersetting constraint for wide character testing functions. There is no special description about "C" locale behavior for isw* functions. In stead, the following rule is applied to any locale.
When a character c is true for a isxxx function, the corresponding wide character wc shall be true for a iswxxx function.

```
isxxx(c) != 0 ==> iswxxx(wc) !=0
```

B.6.2 Changed space character (' ') handling in iswgraph and iswpunct

Space character (' ') was treated specially in isprint, isgraph and ispunct. The space character handling in their matching wide character functions differ from the one specified in ISO 9899:1990.
The matching wide character functions use iswspace class character in stead of single space character (' '). Therefore, the behaviors of the iswgraph and iswpunct functions may differ from their matching functions in ISO 9899:1990 in this regard (See footnote for iswgraph in the Amendment).

B.7 Extensible wide character classification/mapping functions

There are eleven standard character testing functions defined in ISO 9899:1990. As the number of supported locales increases, the requirements for additional character classification grows and varies from locale to locale. To satisfy this requirement, many of existing implementations especially for non-English countries have been defining new isxxx functions such as iskanji, ishanzi, etc.. However, this approach has a name space problem and is not flexible at all to support additional classification requirements. Therefore, in this Amendment, a pair of extensible wide character classification functions, wctype and iswctype, are introduced to satisfy open-ended number of requirements for character classification. As the name of character classification is passed as an argument to the wctype functions, it does not have a problem of name space pollution. And these generic interfaces allow a program to test if the classification is available in the current locale and test locale-specific character classification, such as kanji or hiragana in Japanese.

In the same way, a pair of wide character mapping functions, wctrans and towctrans, are introduced to support locale specific character mappings. One of the example of applying this functionality is the mappings between hiragana and katakana in a Japanese character set.

B.8 The Generalized Multibyte Characters

The International Standard ISO/IEC 9899:1990 intentionally restricted the class of acceptable encodings for multibyte characters. One goal was to ensure that, at least in the initial shift state, the characters in the basic C character set have multibyte representations that are single characters with the same code as the single-byte representation. The other was to ensure that the null byte should never appear as the second or subsequent byte of any multibyte code. Hence, 'a' is always 'a' and '\0' is always '\0', to put matters most simply.

While these may be reasonable restrictions within a C program, they hamper the ability of the MSE functions to read arbitrary wide-oriented files. For example, a system may wish to represent files as sequences of ISO 10646 characters. Reading or writing such a file as a wide-oriented stream should be an easy matter. At most, the library may have to map between native and some canonical byte order in the file. In fact, it is easy to think of an ISO 10646 file as being some form of multibyte file -- except that it violates both restrictions described above. (The code for 'a' can look like the byte sequence '\0', '\0', '\0', 'a', for example.)

Thus, the MSE introduces the notion of a 'generalized multibyte encoding.' It subsumes all the ways we can currently imagine that operating systems will represent files containing characters from a large character set.

B.9 Streams and Files

B.9.1 The conversion state

It is necessary to convert between multibyte characters and wide characters during performing wide character input/output functions. The conversion state, introduced in 4.5.3.2 in the Amendment, is used in order to perform this conversion. Every wide input/output functions refer the conversion state held in the FILE object.

The conversion state in the FILE object is determined by the file position of the corresponding multibyte character stream. Knowing the current file position and the corresponding conversion state of the multibyte character stream, the wide input/output functions determine the behavior of conversion. The shift states in the state-dependent encoding is a part of the conversion state.

The wide input/output functions behave as if;

* a FILE object includes a hidden mbstate_t object and,
* the wide input/output functions apply it to the mbrtowc/wcrtomb functions in performing conversion between a multibyte character and a wide character.

B.9.2. Implementation

The Committee assumes that only wide character input/output functions can maintain the consistency between the conversion state information and the stream. The byte input/output functions may take no care of the conversion state information in the FILE object.

There is a premise that wide character input/output functions always begins at the boundary of subsequent two multibyte characters. The Committee has decided that it is intolerable for implementors to provide the wide character input/output functions whose behaviors are ensured without this premise.

Applying a byte input/output function to a wide-oriented stream may cause to locate the file position indicator of the stream at other position than the boundary of subsequent two multibyte characters. The Committee has decided to constrain the usage of the byte input/output functions because they may break the premise on a wide-oriented stream.

[Seek operation]

A fpos_t object for a stream in a state-dependent encoding includes the shift state information for the corresponding stream. In order to ensure the behavior of consequent wide character input/output functions under a state-dependent encoding environment, a seek operation should reset the conversion state corresponding to the file position as well as repositioning the file position.

The traditional seek operation functions, fseek/ftell may not available under such encoding environment that a type 'long' object is too small to hold both the conversion state information and file position indicator.

[Supporting state-dependent encodings]

Under state-dependent encodings, a FILE object should include the conversion state corresponding to the stream, because the Committee has strong intention that programmers need not handle the tremendous conversion states in wide character input/output.

There is no means for programmers to access the internal shift state in a FILE object.

[Supporting multiple encoding environment]

A "multiple encoding environment" which has two or more different

There is an environment, for example in Japan, where we have two or more encoding rules for a single character set. Most implementations for Japanese environment should take special care of these multiple encodings.

In the program execution, the wide character input/output functions get the information about the current encodings from the LC_CTYPE category of the current locale. When writing a program for a multiple encoding environment, programmer should keep the information of LC_CTYPE category of the current locale for each of opened files. At every access to the files, the corresponding LC_CTYPE category should be restored.

Because the encoding-rule information is a part of the conversion state, a hidden mbstate_t object in the FILE object holds the encoding-rule information.

The conversion state just created when a file is opened is said to have "unbound" state because it has no relations to any of the encoding rules. Just after
the first wide character input/output operation, the conversion state is bound to the encoding rule which is corresponding to the LC_CTYPE category of the current locale.

The following is the summary of the relations between some objects, the shift state, and the encoding rules.

		fpos_t	FILE
shift state		incl.	incl.
encoding rule		none(*1)	incl.
effect of changing LC_CTYPE	(unbound)	none	affected
	(bound)	none	none

(*1) Some implementations, the fpos_t object may include the encoding rule.

B.9.3 Constraint of the usage of the byte input/output functions and the wide character input/output functions.

Both the wide input/output functions and the byte input/output functions refer the same type of object(FILE object). As described in B.9.2, however, there is a constraint upon the mixed usage of the both type of the input/output functions. That is, if one of a wide input/output functions is applied to a FILE object, its stream becomes wide-oriented and no byte input/output functions shall be applied later.

The reason of this constraint is to ensure the consistency between the current file position and the current conversion state in the FILE object. Applying one of the byte input/output functions to a wide-oriented stream breaks the consistency which has been held in the FILE object, because the byte input/output functions may(or should) take no care of the conversion state information in the FILE object.

The following diagram shows state transition of the stream with input/output functions.

[[[State Transition Diagram of the Stream will be inserted here]]]

B.9.4 Streams with no difference between text and binary

In some implementations, such as UNIX, there are streams which has both types of properties, text and binary. According to such an implementation, the Committee specifies the following usage of the wide character input/output functions. A stream opened as a binary stream should be obey the constraints of the usage upon text streams when the wide character input/output functions are applied.

So implementation of the wide character input/output functions can rely on the premise that programmers use the wide character input/output functions to a binary stream under the constraints upon the text stream. Implementation may provide the wide character input/output functions which behave correctly on a

However, the behavior of the wide character input/output functions on the binary stream cannot be always ensured.

B.10 Formatted input/output functions

B.10.1 Adding l qualifier to existing formatted input/output functions

The simplest extension for the wide character input/output is to use existing formatted input/output functions with the existing (byte-oriented) stream. In this case, data consist of characters only (ie. strings) are treated as wide character and other data such as numerical data are treated as single byte characters. Though this is not a complete model for wide character processing, this is a common extension among some existing implementations in Japan. So the committee decided to take similar extension.

At first, new conversion specifiers %S and %C were introduced to handle a wide character string and a wide character respectively. After long discussions about the actual implementation and the future library directions (in 7.13.6 of ISO 9899:1990), these specifiers were withdrew. Then new qualifier l not only for c and s specifiers but also for [in the fscanf function is introduced for wide character processing. Note that even though the new qualifier is introduced as an extension, the field width and the precision still specify the number of bytes.

And in order to implement a new qualifier exactly as much as possible, new set of functions that are quite efficient for parsing wide characters and converting wide characters from/to single byte characters completely is required. Thus functions described in 4.6.5.1, 4.6.5.2, 4.6.5.3 and 4.6.5.4 are introduced.

Because this is "pure extension" to ISO 9899:1990, it has essential restrictions as follows and is expected to be implemented for encodings that are not state-dependent basically.

[Restrictions on the fscanf function]

> In a state-dependent encoding, one or more shift sequences may be included in the format and be executed as an ordinary multibyte character directive. And shift sequences may also be included in an input string. Because the fscanf function treats these shift sequences in a quite same way as for single byte characters, an unexpected match may occur or an expected match might not occur.
>
> See examples described in 4.6.2.3.2.

[Restrictions on the fprintf function]

> In a state-dependent encoding, redundant shift sequences may be written.

B.10.2 Formatted wide character input/outbut functions

In the early MSE, formatted wide character input/outbut functions were not introduced because an extension to existing formatted input/output functions seemed to be sufficient. After considering the complete model for wide character handling, the necessity of formatted wide character input/outbut functions was recognized.

Formatted wide character input/outbut functions have quite same conversion specifiers and qualifiers as existing formatted input/outbut functions including l qualifier for c, s and [conversion. Because the format string consists of wide characters and the field width and the precision specify the number of wide characters, some restrictions on existing functions are no more found in new functions. This means that wide character are read/written under the complete control of the format.

B.11 The introduction of the fwide function

While we believe that the MSE provides reasonably complete functionality for manipulating wide-oriented files, we noticed that no reliable mechanism existed for testing or setting the orientation of a stream. (You could try certain operations to see if they fail, but that is risky and still not a complete strategy.) Hence, we introduced the function fwide as a means of forcing a newly opened stream into the desired orientation without attempting any input/output on the stream. The function also serves as a passive means of testing the orientation of a stream, either before or after the orientation has been fixed.

B.12 Single-byte wide character conversion functions

Two single-byte wide character conversion functions, btowc and wctob, have been introduced in the Amendment. These functions bridge over a single-byte character handling and a wide character handling.

There is L'x' == 'x' rule for a member of basic character set specified in ISO 9899:1990. There was a discussion on this rule in either way, to relax or to tighten. In the amendment, this rule is preserved without any changes. Applying the rule to all of single byte characters brings unnecessary constraint on implementation with regards to wide character encoding. It prohibits an implementation from having a common wide character encoding for multiple multibyte encodings. On the other hand, relaxing or removing the rule was considered to be inappropriate in terms of practical implementation. The new function wctob can be used to test if a single byte character function to be used safely. For example, when a format string of the scanf function is parsed and searched for a white space character, the wctob function can be used combined with the isspace function. Please refer to the specification of iswxxx functions, as the wctob function is used to specify the relationship between isxxx function and iswxxx function.

Meanwhile, when you write practically a code of the wide character processing program, especially wide character handling library functions, it will be necessary to judge if this single byte is the first and only character of a valid multibyte character. This is the reason of the introduction of the btowc function. Moreover, with some encodings, btowc can be reduced to a simple in-line expression.

B.13 Extended multibyte and wide character conversion utilities

Although ISO/IEC 9899:1990 allows a multibyte character to have state-dependent encoding (subclause 5.2.1.2), it is not sufficient to support state-dependent encoding environments due to the following problem of the multibyte character conversion functions (subclause 7.10.7)

1. Since the Functions maintain the shift state information internally, they can not handle multiple strings at the same time.

2. The functions may generate redundant shift sequences as the output, or can not handle redundant shift sequences as the input.

3. There is an inconvenience in the multibyte string conversion functions (subclause 7.10.8) regardless of state dependency of the encoding. When the encoding error occurs, it returns -1 without any information on the location where the conversion stopped.

B.13.1 Introduction of the Conversion State

To handle multiple strings on state-dependent encoding, the committee introduced concept of conversion state. The conversion state determines the behavior of their byte-to-wide character or vise versa by the position from the beginning of the multibyte character string, and store information such as encoding, character accumulator and shift state.

The nonarray object, mbstate_t, is defined to hold the conversion state.

A zero-valued mbstate_t object assumed to be in the initial conversion state. After a zero-valued mbstate_t object is declared, but before any operations are performed on it, the object is unbound. Once a multibyte and wide character conversion utilities has been applied to the object, the object becomes bound and hold the above information.

The conversion utilities maintain the conversion state according to the encoding specified in LC_CTYPE category of the current locale. Once the conversion starts, the utility will work as if the encoding scheme were not changed on condition that :

the utility is applied to the same string,
and
the LC_CTYPE category setting is the same,
and
the conversion direction is the same.

B.13.2 Conversion Utilities

As mbstate_t object was introduced, the necessity of related functions to the object were discussed as follows:

[Function for initialization]

Though a method to initialize the object is needed, it would be better not to define too many functions in the amendment. Thus the committee decided to assume a zero-filled mbstate_t object as a initial conversion state object.

[Function for comparison]

The committee reached the conclusion that it may be impossible to define the equality between two conversion states. If two mbstate_t objects have the same value to each attributes, they could be the same. However, there may be another case that they have the different values but behave the same.

[The mbsinit function]

The function is introduced to test whether an mbstate_t object is initial shift state or not, because the initial shift state is not always a certain value. The function is necessary because many functions in the amendment treats the initial shift state as a special condition.

Regarding problems 2 and 3 described in the beginning of B.13, the committee introduced a method to distinguish an immature multibyte sequence, which requires more bytes to determine if they form a valid multibyte character, from an encoding error. When encountering an

stores the immature character information. Thus, the user can resume the pending conversion due to the immature multibyte sequence.

The new multibyte/wide string conversion utilities are made restartable by using character accumulator and shift state information. According to this enhancement, the functions have a parameter for source string as a pointer to a pointer to the string. The function updates the argument to point the position where the conversion stopped.

B.14 Column width

The number of characters to be read or written can be specified in existing formatted input/output functions. At the traditional display device that displays characters with fixed pitch, the number of characters is directly proportional to the width that is occupied by these characters. So the display format can be specified through the field width and/or the precision.

In formatted wide character input/output functions, the field width and the precision specifies the number of wide characters to be read or written. The number of wide characters is not always directly proportional to the width of them. For example, at the Japanese traditional display device, a single byte character such as an ASCII characters occupies different width from a Kanji character, while each of them is treated as one wide character. To enable to control the display format for wide characters, a set of formatted wide character input/output functions whose metric was the column width were proposed.

This proposal was supported only by Japan. Because this proposal was based on the traditional display device with a fixed width of characters and many modern display devices support a proportional pitch, it was doubtful whether input/output functions in this proposal were really needed. Though another set of functions that return the column width at any kind of display devices for a given wide character or wide character string were considered, they seemed to be out of the scope of C language. Thus all proposals regarding to the column width were withdrew.

If an implementor needs this kind of functionality, there are a few way to extend wide character output functions with keeping an implementation as the conforming implementation. For example, the field width prefixed with a '#' can specify the column width as shown below:

```
%#N  set the counting mode to "printing-positions",
     and reset the  %n  counter

%N   set the counting mode back to "widechars",
     and reset the  %n  counter.
```

--- End of Rationale ---

Nederlands Normalisatie-instituut

Kalfjeslaan 2
Postbus 5059, 2600 GB Delft
Telefoon (015) 690 390
Telefax (015) 690 190
Telex 38 144 nni nl

The NNI regrets that it has to vote against SC22/N1443. The NNI does not whish to support the solution offered for the usage of C with national variants of the ISO 646 character set. This solution is contained in clauses 2, 3 and 4.4 of N1443.

There is pressure from countries using national variants of ISO 646 to have a standard way of expressing C in the invariant subset of ISO 646 that is more aesthetically pleasing than the trigraph solution that is embedded in the current C standard. We understand their whish. But we have come to the conclusion that we do not see an acceptable way of realizing this whish. We see three criteria for proposed solutions: completeness, aesthetics and technical cleanness.

- The solution in N1341 is both incomplete and overcomplete. Incomplete because no solution is offered inside strings and literal characters. Overcomplete because, to remain consistent with macro definitions for characters in the variant subset, it also contains macro definitions for the invariant subset; e.g. *and_eq* for *&=*, because *or_eq* needs to be defined as *|=*.
- The result is barely aesthetically acceptable.
- The proposal is technically feasible, but unsatisfactory. The solution uses two technically different approaches to solve the same problem; alternate spelling of tokens and macro definitions.
  Standardization of macro definitions is, strictly speaking, not necessary. Users can create their own sets of definitions, without threatening portability.
  The use of the macro names *and_eq* and *not_eq* is confusing. *and_eq* is to be used as replacement for the assignment operator *&=*, while *not_eq* is to be used as replacement for the comparison operator *!=*.
  The proposal also seems to be in conflict with the emerging C++ standard with respect to the digraphs *<:* and *%:*.

We would further like to make the following two observations:

1. Usage of ISO 8859-1 (Latin-1), which solves this problem, is becoming widespread.
2. We expect that the proposed solution will be little used. Programs written in the ISO 646 invariant representation of C look so different from the current representation that they will be hard to maintain for people used to the current representation i.e the rest of the world. We expect that for this reason a large part of the community in countries that stated interest in this proposal will keep on using the current representation of C.
   Furthermore, only a few of the countries with national variants of ISO 646 have expressed interest in this proposal.

The proposal in clauses 2, 3 and 4.4 is not good enough to be acceptable, even as a compromise. Especially because it solves a disappearing problem. We see no reason to burden the international community with this part of N1443.

The rest of N1443 is a worthwhile document that we welcome. We will support N1443 if the objections mentioned above are taken away by removing the clauses 2, 3 and 4.4.

VOTE ON COMMITTEE DRAFT 9899:1990 PDAM 1	
Date of circulation 1993-09-01	Reference number
Closing date for voting 1993-12-10	ISO/TC 1 /SC 22 N

ISO/TEC 1 /SC 22 Title: Programming Languages, their environments and system software interfaces Secretariat: Canada, SCC	Circulated to P-members of the committee for voting on registration of the draft as a DIS, in accordance with 2.4.3 of part 1 of the IEC/ISO Directives

Please send this form, duly completed, to the secretariat indicated above.

CD 9899:1990 PDAM 1

Title

Information Technology - Programming Languages, their environments and system software interfaces - Amendment 1 to ISO/IEC 9899:1990 Programming language C
en: Integrity Addendum

☐ We agree to the circulation of the draft as a DIS in accordance with 2.5.1 of part 1 of the IEC/ISO Directives

☒ We do not agree to the circulation of the draft as a DIS
The reasons for our disagreement are the following (use a separate page as annex, if necessary)

The following have not been deleted, as requested in our vote on N 1341.

Clauses 2, 3, 3.1 and 4.4.

Deletion of these clauses will change our vote to approval.

P-member voting: UK	
Date: 10 December 1993	Signature: C M Jenkins

FORM 8 (ISO)

4/5