SPIRIT Platform Blueprint

SPIRIT C Language Profile

(SPIRIT Issue 3.0)

Network Management Forum

SPIRIT Platform Blueprint

SPIRIT C Language Profile (SPIRIT Issue 3.0)

ISBN: N/A
Document Number: J406

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to:

Network Management Forum
1201 Mount Kemble Avenue
Morristown, NJ 07960
U.S.A.

Tel: +1 201 425 1900

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this C Language Profile is to define a standard C-language interface between application programs and systems software for the SPIRIT environment. Any application program written to this C-language interface is portable across SPIRIT-conforming platforms, regardless of its vendor. This C-language interface is intended to be used in conjunction with other SPIRIT APIs that have a C-language binding. An example is the Base System API.

This C-language interface complies with the ISO C Language standard, ISO/IEC 9899 (ISO C), with its (proposed) amendment, ISO/IEC 9899/AM1.[1] Since all the mandatory requirements of ISO C are also applicable to this interface, any implementation that conforms to this interface is also ISO C-compliant. However, as this interface defines some additional requirements to ISO C, an application conforming to this interface may not always strictly conform to ISO C.

1.2 Reading Instructions

This document describes the differences between ISO C and the SPIRIT C-language interface specification.

Specification elements that are identical to ISO C are not restated in this document. In ISO C, readers should replace ''This International Standard'' with ''This specification'' when applying the technical contents to the SPIRIT specification.

The indications ''The same as ISO/IEC 9899.'', ''The same as ISO/IEC 9899/AM1.'' or ''The same as ISO/IEC 9899 with ISO/IEC 9899/AM1.'' mean that the technical contents of this SPIRIT specification are identical to ISO C.

As far as possible, the organisation of ISO/IEC 9899 is maintained in this profile. New clauses, such as ''Wide-character classification and mapping utilities'' and ''Extended multibyte and wide-character utilities'' in the corresponding clauses of this profile were introduced by ISO/IEC 9899/AM1. Also, a clause of ''Code conversion utilities'' is added to the end of the ''Library'' section.

Italic font is used to indicate additional requirements to ISO C specified by SPIRIT C.

1. ISO/IEC 9899:1990, Programming Languages — C (technically identical to ANSI standard X3.159-1989).
   ISO/IEC 9899:1990/Amendment 1:1994, Multibyte Support Extensions (MSE) for ISO C.

Chapter 2

C Language Profile

This profile is organised according to ISO/IEC 9899.

1. Scope

   The same as ISO/IEC 9899.

2. Normative References

   The same as ISO/IEC 9899.

3. Definitions and Conventions

   In this C language profile, ‘‘shall’’ is interpreted as a requirement on an implementation or a program; conversely, ‘‘shall not’’ is interpreted as a prohibition.

   For the purposes of this profile, the following definitions apply. Other terms are defined at their first appearance, indicated by italic type. Terms explicitly defined in this profile do not refer implicitly to similar terms defined elsewhere. Terms not defined in this profile are to be interpreted according to ISO 2382[2] vocabulary standards.

3.1 Alignment

   The same as ISO/IEC 9899.

3.2 Argument

   The same as ISO/IEC 9899.

3.3 Bit

   The same as ISO/IEC 9899.

3.4 Byte

   The same as ISO/IEC 9899.

3.5 Character

   The same as ISO/IEC 9899.

3.6 Constraints

   The same as ISO/IEC 9899.

3.7 Diagnostic Message

2. ISO 2382, Information Technology — Vocabulary.

The same as ISO/IEC 9899.

3.8 Forward References

The same as ISO/IEC 9899.

3.9 Implementation

The same as ISO/IEC 9899.

3.10 Implementation-defined Behaviour

The same as ISO/IEC 9899.

3.11 Implementation Limits

The same as ISO/IEC 9899.

3.12 Multibyte Character

The same as ISO/IEC 9899.

3.13 Object

The same as ISO/IEC 9899.

3.14 Parameter

The same as ISO/IEC 9899.

3.15 Undefined Behaviour

The same as ISO/IEC 9899.

3.16 Unspecified Behaviour

The same as ISO/IEC 9899.

3.17 Conformance Document

A document provided by an implementor that contains implementation details as described in the ‘‘Documentation’’ clause of the ‘‘Compliance’’ section.

4. Compliance

A conformance document with the following information shall be available for an implementation claiming conformance to this profile.

The conformance document shall not contain information about extended facilities or capabilities outside the scope of this profile.

The conformance document shall contain a statement that indicates the full name and date of the SPIRIT specification that applies.

The conformance document shall describe the limits values found in the **<limits.h>** and **<float.h>** headers, translation limits, stating values, the conditions under which those values may change, and the limits of such variations, if any.

The conformance document shall describe the behaviour of the implementation for all implementation-defined features defined in this profile. The conformance document may specify the behaviour of the implementation for those features where this profile states that implementations may vary or where features are identified as undefined or unspecified.

No features, other than those described in this profile, shall be present in the conformance document.

The phrases ‘‘shall document’’ or ‘‘shall be documented’’ in this profile mean that documentation of the feature shall appear in the conformance document, as described above.

5. Environment

The same as ISO/IEC 9899.

5.1 Conceptual Models

The same as ISO/IEC 9899.

5.1.1 Translation Environment

The same as ISO/IEC 9899.

5.1.1.1 Program Structure

The same as ISO/IEC 9899.

5.1.1.2 Translation Phases

The same as ISO/IEC 9899.

5.1.1.3 Diagnostics

The same as ISO/IEC 9899.

5.1.2 Execution Environments

The same as ISO/IEC 9899.

5.1.2.1 Freestanding Environment

The same as ISO/IEC 9899.

5.1.2.2 Hosted Environment

The same as ISO/IEC 9899.

5.1.2.2.1 Program Startup

The same as ISO/IEC 9899.

5.1.2.2.2 Program Execution

The same as ISO/IEC 9899.

5.1.2.2.3 Program Termination

The same as ISO/IEC 9899.

5.1.2.3 Program Execution

The same as ISO/IEC 9899.

5.2 Environmental Considerations

5.2.1 Character Sets

The same as ISO/IEC 9899.

5.2.1.1 Trigraph Sequences

The same as ISO/IEC 9899.

5.2.1.2 Multibyte Characters

The same as ISO/IEC 9899.

5.2.2 Character Display Semantics

The same as ISO/IEC 9899.

5.2.3 Signals and Interrupts

The same as ISO/IEC 9899.

5.2.4 Environmental Limits

The same as ISO/IEC 9899.

5.2.4.1 Translation Limits

The same as ISO/IEC 9899.

5.2.4.2 Numerical Limits

The same as ISO/IEC 9899.

5.2.4.2.1 Size of Integral Type **<limits.h>**

The same as ISO/IEC 9899.

5.2.4.2.2 Characteristics of Floating Types **<float.h>**

The technical requirements of this clause are the same as ISO/IEC 9899 except for the implementation limit values shown in the paragraph below.

The values given in the following lists are replaced by implementation-defined expressions, equal or greater in magnitude (absolute value) to those shown and with the same sign:

FLT R___ADIX2
FLT M___ANT_DIG
DBL M___ANT_DIG
DBL M___AINT_DIG
FLT D___IG6
DBL D___IG14
LDBL __DIG14
FLT M___IN_EXP
DBL M___IN_EXP
LDBL __MIN_EXP
FLT M___IN_10_EXP-37
DBL M___IN_10_EXP-78
LDBL __MIN_10_EXP-78
FLT M___AX_EXP
DBL M___AX_EXP
LDBL __MAX_EXP
FLT M___AX_10_EXP+38
DBL M___AX_10_EXP+75
LDBL __MAX_10_EXP+75

The values given in the following list are replaced by implementation-defined expressions with values that are equal to or greater than those shown:

FLT M___AX 1E+38
DBL M___AX 1E+75
LDBL __MAX 1E+75

The values given in the following list shall be replaced by implementation-defined expressions with values that are equal to or less than those shown:

FLT E___PSILON1E-5
DBL E___PSILON1E-13
LDBL __EPSILON1E-13
FLT M___IN1E-37
DBL M___IN1E-78
LDBL __MIN1E-78

6. Language

The same as ISO/IEC 9899.

6.1 Lexical Elements

The same as ISO/IEC 9899.

6.1.1 Keywords

The same as ISO/IEC 9899.

6.1.2 Identifiers

Syntax The same as ISO/IEC 9899.

Description The same as ISO/IEC 9899.

Constraints The same as ISO/IEC 9899.

Semantics The same as ISO/IEC 9899.

Implementation Limits

Implementation restricts the significance of an external name to eight characters (*rather than six characters*).

The implementation treats at least the first 31 characters of an internal name (a macro name or an identifier that does not have an external linkage) as significant. Corresponding lower-case and upper-case letters are different. The implementation may further restrict the significance of an external name (an identifier that has an external linkage) to eight characters and may ignore distinctions of alphabetical case for such names. These limitations on identifiers are all implementation-defined.

Any identifiers that differ in a significant character are different identifiers. If two identifiers differ in a non-significant character, the behaviour is undefined.

6.1.2.1 Scopes of Identifiers

The same as ISO/IEC 9899.

6.1.2.2 Linkages of Identifiers

The same as ISO/IEC 9899.

6.1.2.3 Name Spaces of Identifiers

The same as ISO/IEC 9899.

6.1.2.4 Storage Durations of Objects

The same as ISO/IEC 9899.

6.1.2.5 Types

The same as ISO/IEC 9899.

6.1.2.6 Compatible Type and Composite Type

The same as ISO/IEC 9899.

6.1.3 Constants

The same as ISO/IEC 9899.

6.1.3.1 Floating Constants

The same as ISO/IEC 9899.

6.1.3.2 Integer Constants

The same as ISO/IEC 9899.

6.1.3.3 Enumeration Constants

The same as ISO/IEC 9899.

6.1.3.4 Character Constants

The same as ISO/IEC 9899.

6.1.4 String Literals

The same as ISO/IEC 9899.

6.1.5 Operators

The same as ISO/IEC 9899 with ISO/IEC 9899/AM1.

6.1.6 Punctuators

The same as ISO/IEC 9899 with ISO/IEC 9899/AM1.

6.1.7 Header Names

The same as ISO/IEC 9899.

6.1.8 Preprocessing Numbers

The same as ISO/IEC 9899.

6.1.9 Comments

The same as ISO/IEC 9899.

6.2 Conversions

The same as ISO/IEC 9899.

6.2.1 Arithmetic Operands

6.2.1.1 Characters and Integers

The same as ISO/IEC 9899.

6.2.1.2 Signed and Unsigned Integers

When an **unsigned** integer is converted to its corresponding **signed** integer, all bits are preserved.

When a value with **integral** type is converted to another **integral** type, if the value can be represented by the new type, its value is unchanged.

When a **signed** integer is converted to an **unsigned** integer with equal or greater size, if the value of the **signed** integer is non-negative, its value is unchanged. Otherwise, if the **unsigned** integer has greater size, the **signed** integer is first promoted to the **signed** integer corresponding to the **unsigned** integer; the value is converted to **unsigned** by adding to it one greater than the largest number that can be represented in the **unsigned** integer type.

When a value with **integral** type is demoted to an **unsigned** integer with smaller size, the result is the non-negative remainder on division by the number one greater than the largest **unsigned** number that can be represented in the type with smaller size. When a value with **integral** type is demoted to a **signed** integer with smaller size, the least significant bits are preserved, and when an **unsigned** integer is converted to its corresponding **signed** integer, all values are preserved.

6.2.1.3 Floating and Integral

The same as ISO/IEC 9899.

6.2.1.4 Floating Types

The same as ISO/IEC 9899.

6.2.1.5 Usual Arithmetic Conversions

The same as ISO/IEC 9899.

6.2.2 Other Operands

6.2.2.1 Lvalues and Function Designators

The same as ISO/IEC 9899.

6.2.2.2 void

The same as ISO/IEC 9899.

6.2.2.3 Pointers

The same as ISO/IEC 9899.

6.3 Expressions

The same as ISO/IEC 9899.

6.3.1 Primary Expressions

The same as ISO/IEC 9899.

6.3.2 Postfix Operators

The same as ISO/IEC 9899.

6.3.2.1 Array Subscripting

The same as ISO/IEC 9899.

6.3.2.2 Function Calls

The same as ISO/IEC 9899.

6.3.2.3 Structure and Union Members

The same as ISO/IEC 9899.

6.3.2.4 Postfix Increment and Decrement Operators

The same as ISO/IEC 9899.

6.3.3 Unary Operators

The same as ISO/IEC 9899.

6.3.3.1 Prefix Increment and Decrement Operators

The same as ISO/IEC 9899.

6.3.3.2 Address and Indirection Operators

The same as ISO/IEC 9899.

6.3.3.3 Unary Arithmetic Operators

The same as ISO/IEC 9899.

6.3.3.4 The *sizeof* Operator

The same as ISO/IEC 9899.

6.3.4 Cast Operators

The same as ISO/IEC 9899.

6.3.5 Multiplicative Operators

Syntax — The same as ISO/IEC 9899.

Constraints — The same as ISO/IEC 9899.

Semantics — The usual arithmetic conversions are performed on the operands.

The result of the * operand is the product of the operands. The result of the / operand is the quotient from the division of the first operand by the second; the result of the % operator is the remainder. In both operations, if the value of the second operand is zero, the behaviour is undefined.

When integers are divided and the division is inexact, if both operands are positive the result of the / operator is the largest integer less the algebraic quotient. The result of the % operator is positive. If either operand is negative, the result of the / operator is the integer of lesser magnitude, the nearest to the algebraic quotient. If the quotient a/b is representable, the expression (a/b) * b + a%b shall equal a.

6.3.6 Additive Operators

The same as ISO/IEC 9899.

6.3.7 Bitwise Shift Operators

Syntax — The same as ISO/IEC 9899.

Constraints — The same as ISO/IEC 9899.

Semantics — The integral promotion is performed on each of the operands. The type of result is the promoted left operand. If the value of the right operand is negative or is greater than or equal to the width in bits of the promoted left operand, the behaviour is undefined.

The result of E1 << E2 is E1 left-shifted E2 bit operations; vacated bits are filled with zeros. If E1 has an **unsigned** type, the value of the results is E1 multiplied by the quantity, 2 raised to the power E2, reduced modulo ULONG_MAX+1 if E1 has type **unsigned long**. Otherwise this is UINT_MAX+1. (The constants ULONG_MAX and UINT_MAX are defined in the header **<limits.h>**.)

The result of E1 >> E2 is E1 right-shifted E2 bit positions. If E1 has an **unsigned** type or if E1 has a **signed** type and a non-negative value, the value of the result is an integral part of the quotient of E1 divided by the quantity, 2 raised to the power E2. If E1 has a **signed** type and a negative value, the sign is preserved; vacated bits are filled with ones in a two's-complement representation.

6.3.8 Relational Operators

The same as ISO/IEC 9899.

6.3.9 Equality Operators

The same as ISO/IEC 9899.

6.3.10 Bitwise AND Operator

The same as ISO/IEC 9899.

6.3.11 Bitwise Exclusive OR Operator

The same as ISO/IEC 9899.

6.3.12 Bitwise Inclusive OR Operator

The same as ISO/IEC 9899.

6.3.13 Logical AND Operator

The same as ISO/IEC 9899.

6.3.14 Logical OR Operator

The same as ISO/IEC 9899.

6.3.15 Conditional Operator

The same as ISO/IEC 9899.

6.3.16 Assignment Operators

The same as ISO/IEC 9899.

6.3.16.1 Simple Assignment

The same as ISO/IEC 9899.

6.3.16.2 Compound Assignment

The same as ISO/IEC 9899.

6.3.17 Comma Operator

The same as ISO/IEC 9899.

6.4 Constant Expressions

The same as ISO/IEC 9899.

6.5 Declarations

The same as ISO/IEC 9899.

6.5.1 Storage-class Specifiers

The same as ISO/IEC 9899.

6.5.2 Type Specifiers

The same as ISO/IEC 9899.

6.5.2.1 Structure and Union Specifiers

The same as ISO/IEC 9899.

6.5.2.2 Enumeration Specifiers

The same as ISO/IEC 9899.

6.5.2.3 Tags

The same as ISO/IEC 9899.

6.5.3 Type Qualifiers

The same as ISO/IEC 9899.

6.5.4 Declarators

The same as ISO/IEC 9899.

6.5.4.1 Pointer Declarators

The same as ISO/IEC 9899.

6.5.4.2 Array Declarators

The same as ISO/IEC 9899.

6.5.4.3 Function Declarators (Including Prototypes)

The same as ISO/IEC 9899.

6.5.5 Type Names

The same as ISO/IEC 9899.

6.5.6 Type Definitions

The same as ISO/IEC 9899.

6.5.7 Initialization

	The same as ISO/IEC 9899.
6.6	Statements
	The same as ISO/IEC 9899.
6.6.1	Labeled Statements
	The same as ISO/IEC 9899.
6.6.2	Compound Statement or Block
	The same as ISO/IEC 9899.
6.6.3	Expression and Null Statements
	The same as ISO/IEC 9899.
6.6.4	Selection Statements
	The same as ISO/IEC 9899.
6.6.4.1	The *if* Statement
	The same as ISO/IEC 9899.
6.6.4.2	The *switch* Statement
	The same as ISO/IEC 9899.
6.6.5	Iteration Statements
	The same as ISO/IEC 9899.
6.6.5.1	The *while* Statement
	The same as ISO/IEC 9899.
6.6.5.2	The *do* Statement
	The same as ISO/IEC 9899.
6.6.5.3	The *for* Statement
	The same as ISO/IEC 9899.
6.6.6	Jump Statements
	The same as ISO/IEC 9899.
6.6.6.1	The *goto* Statement
	The same as ISO/IEC 9899.
6.6.6.2	The *continue* Statement
	The same as ISO/IEC 9899.
6.6.6.3	The *break* Statement
	The same as ISO/IEC 9899.
6.6.6.4	The *return* Statement
	The same as ISO/IEC 9899.
6.7	External Definitions

The same as ISO/IEC 9899.

6.7.1 Function Definitions

The same as ISO/IEC 9899.

6.7.2 External Object Definitions

The same as ISO/IEC 9899.

6.8 Preprocessing Directives

The same as ISO/IEC 9899.

6.8.1 Conditional Inclusion

The same as ISO/IEC 9899.

6.8.2 Source File Inclusion

The same as ISO/IEC 9899.

6.8.3 Macro Replacement

The same as ISO/IEC 9899.

6.8.3.1 Argument Substitution

The same as ISO/IEC 9899.

6.8.3.2 The # Operator

The same as ISO/IEC 9899.

6.8.3.3 The ## Operator

The same as ISO/IEC 9899.

6.8.3.4 Rescanning and Further Replacement

The same as ISO/IEC 9899.

6.8.3.5 Scope of Macro Definitions

The same as ISO/IEC 9899.

6.8.4 Line Control

The same as ISO/IEC 9899.

6.8.5 Error Directive

The same as ISO/IEC 9899.

6.8.6 Pragma Directive

The same as ISO/IEC 9899.

6.8.7 Null Directive

The same as ISO/IEC 9899.

6.8.8 Predefined Macro Names

The following macro names shall be defined in the implementation:

__LINE__ The line number of the current source line (a decimal constant).

__FILE__ The presumed name of the source line (a character string literal).

__DATE__ The date of translation of the source file (a character string literal of the form ‘‘Mmm dd yyyy’’, where the names of the months are the same as those generated by the *asctime*( ) function, and the first character of dd is a space character if the value is less than 10). If the date of translation is not available, an implementation-defined valid date shall be supplied.

__TIME__ The time of translation of the source file (a character string literal of the form ‘‘hh:mm:ss’’ as in the time generated by the *asctime*( ) function). If the time of translation is not available, an implementation-defined valid time shall be supplied.

__STDC__ The decimal constant 1, intended to indicate a conforming implementation of ISO/IEC 9899.

__STDC_VERSION__
The decimal constant 199407, intended to indicate an implementation conforming to ISO/IEC 9899/AM1.

The value of the predefined macros (except for __LINE__and __FILE__) remain constant throughout the translation unit.

None of these macro names, nor the identifier defined, shall be the subject of a **#define** or a **#undef** preprocessing directive. All predefined macro names are described with a leading underscore followed by an upper-case letter or a second underscore.

Note: The decimal constant 199407 for __STDC_VERSION__ is subject to change at the official publication of ISO/IEC 9899/AM1.

6.9 Future Language Directions

6.9.1 External Names

The same as ISO/IEC 9899.

6.9.2 Character Escape Sequences

The same as ISO/IEC 9899.

6.9.3 Storage-class Specifiers

The same as ISO/IEC 9899.

6.9.4 Function Declarators

The same as ISO/IEC 9899.

6.9.5 Function Definitions

The same as ISO/IEC 9899.

6.9.6 Array Parameters

The same as ISO/IEC 9899.

7. Library

7.1 Introduction

7.1.1 Definitions of Terms

The same as ISO/IEC 9899 with ISO/IEC 9899/AM1.

7.1.2 Standard Headers

The same as ISO/IEC 9899 with ISO/IEC 9899/AM1.

7.1.3 Reserved Identifiers

The same as ISO/IEC 9899.

7.1.4 Errors **<errno.h>**

The same as ISO/IEC 9899 with ISO/IEC 9899/AM1.

7.1.5 Limits **<float.h>** and **<limits.h>**

The same as ISO/IEC 9899.

7.1.6 Common Definitions **<stddef.h>**

The same as ISO/IEC 9899.

7.1.7 Use of Library Functions

The same as ISO/IEC 9899.

7.2 Diagnostics **<assert.h>**

The same as ISO/IEC 9899.

7.2.1 Program Diagnostics

7.2.1.1 The *assert* Macro

The same as ISO/IEC 9899.

7.3 Character Handling **<ctype.h>**

The same as ISO/IEC 9899.

7.3.1 Character Testing Functions

The same as ISO/IEC 9899.

7.3.1.1 The *isalnum*( ) Function

The same as ISO/IEC 9899.

7.3.1.2 The *isalpha*( ) Function

The same as ISO/IEC 9899.

7.3.1.3 The *iscntrl*( ) Function

The same as ISO/IEC 9899.

7.3.1.4 The *isdigit*( ) Function

The same as ISO/IEC 9899.

7.3.1.5 The *isgraph*( ) Function

The same as ISO/IEC 9899.

7.3.1.6 The *islower*( ) Function

The same as ISO/IEC 9899.

7.3.1.7 The *isprint*( ) Function

The same as ISO/IEC 9899.

7.3.1.8 The *ispunct*( ) Function

The same as ISO/IEC 9899.

7.3.1.9 The *isspace*( ) Function

The same as ISO/IEC 9899.

7.3.1.10 The *isupper*( ) Function

The same as ISO/IEC 9899.

7.3.1.11 The *isxdigit*( ) Function

The same as ISO/IEC 9899.

7.3.2 Character Case Mapping Functions

7.3.2.1 The *tolower*( ) Function

The same as ISO/IEC 9899.

7.3.2.2 The *toupper*( ) Function

The same as ISO/IEC 9899.

7.4 Localization **<locale.h>**

The same as ISO/IEC 9899.

7.4.1 Locale Control

7.4.1.1 The *setlocale*( ) Function

The same as ISO/IEC 9899.

7.4.2 Numeric Formatting Convention Inquiry

7.4.2.1 The *localeconv*( ) Function

Synopsis The same as ISO/IEC 9899.

Description The *localeconv*( ) function sets the components of an object with type **struct lconv** with values appropriate for the formatting of numeric quantities (monetary and otherwise) according to the rules of the current locale.

The members of the structure with type **char *** are pointers to strings, any of which (except decimal_point) can point to "" to indicate that the value is not available in the current locale or is of zero length. The members with type **char** are non-negative numbers, any of which can be {CHAR_MAX} to indicate that the value is not available in the current locale. The members include the following:

char *decimal_point
The decimal-point character used to format non-monetary quantities.

char *thousands_sep
The character used to separate groups of digits before the

decimal-point character in formatted non-monetary quantities.

char *grouping
A string whose elements indicate the size of each group of digits in formatted non-monetary quantities.

char *int_curr_symbol
The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in ISO 4217[3]. The fourth character (immediately preceding the null character) is the character used to separate the international currency symbol from the monetary quantity.

char *currency_symbol
The local currency symbol applicable to the current locale.

char *mon_decimal_point
The decimal-point used to format monetary quantities.

char *mon_thousands_sep
The separator for groups of digits before the decimal-point in formatted monetary quantities.

char *mon_grouping
A string whose elements indicate the size of each group of digits in formatted monetary quantities.

char *positive_sign
The string used to indicate a non-negative-valued formatted monetary quantity.

char *negative_sign
The string used to indicate a negative-value formatted monetary quantity.

char int_frac_digits
The number of fractional digits (those after the decimal-point) to be displayed in a internationally formatted monetary quantity.

char frac_digits
The number of fractional digits (those after the decimal-point) to be displayed in a formatted monetary quantity.

char p_cs_precedes
Set to 1 or 0 if the **currency_symbol** or ***int_curr_symbol*** respectively precedes or succeeds the value for a non-negative formatted monetary quantity.

3. ISO 4217:1987, Codes for the Representation of Currencies and Funds.

char p_sep_by_space
Set to 1 or 0 if the **currency_symbol** or ***int_curr_symbol*** from the value for a non-negative formatted monetary quantity. *Set to 1 if a space separates the symbol from the value; and set to 2 of a space separates the symbol and the sign string, if adjacent.*

char n_cs_precedes
Set to 1 or 0 if the **currency_symbol** or ***int_curr_symbol*** respectively precedes or succeeds the value for a negative formatted monetary quantity.

char n_sep_by_space
Set to 1 or 0 if the **currency_symbol** or ***int_curr_symbol*** from the value for a negative formatted monetary quantity. *Set to 1 of a space separates the symbol from the value; and set to 2 of a space separates the symbol and the sign string, if adjacent.*

char p_sign_posn
Set to a value indicating the positioning of the **positive_sign** for a non-negative formatted monetary quantity.

char n_sign_posn
Set to a value indicating the positioning of the **negative_sign** for a negative formatted monetary quantity.

The elements of grouping and **mon_grouping** are interpreted according to the following:

char__ MAX
No further grouping is to be performed.

0 The previous element is to be repeatedly used for the remainder of the digits.

other
The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

The value of **p_sign_posn** and **n_sign_posn** is interpreted according to the following:

0 Parentheses surround the quantity and **currency_symbol** or ***int_curr_symbol***.

1 The sign string precedes the quantity and **currency_symbol** or ***int_curr_symbol***.

2 The sign string succeeds the quantity and **currency_symbol** or ***int_curr_symbol***.

3 The sign string immediately precedes the **currency_symbol** or ***int_curr_symbol***.

4 The sign string immediately succeeds the **currency_symbol** or ***int_curr_symbol***.

	Returns	The same as ISO/IEC 9899.
	Example	The same as ISO/IEC 9899.
7.5	Mathematics **<math.h>**	
	The same as ISO/IEC 9899.	
7.5.1	Treatment of Error Conditions	
	The same as ISO/IEC 9899.	
7.5.2	Trigonometric Functions	
7.5.2.1	The *acos*( ) Function	
	The same as ISO/IEC 9899.	
7.5.2.2	The *asin*( ) Function	
	The same as ISO/IEC 9899.	
7.5.2.3	The *atan*( ) Function	
	The same as ISO/IEC 9899.	
7.5.2.4	The *atan2*( ) Function	
	The same as ISO/IEC 9899.	
7.5.2.5	The *cos*( ) Function	
	The same as ISO/IEC 9899.	
7.5.2.6	The *sin*( ) Function	
	The same as ISO/IEC 9899.	
7.5.2.7	The *tan*( ) Function	
	The same as ISO/IEC 9899.	
7.5.3	Hyperbolic Functions	
7.5.3.1	The *cosh*( ) Function	
	The same as ISO/IEC 9899.	
7.5.3.2	The *sinh*( ) Function	
	The same as ISO/IEC 9899.	
7.5.3.3	The *tanh*( ) Function	
	The same as ISO/IEC 9899.	
7.5.4	Exponential and Logarithmic Functions	
7.5.4.1	The *exp*( ) Function	
	The same as ISO/IEC 9899.	
7.5.4.2	The *frexp*( ) Function	
	The same as ISO/IEC 9899.	
7.5.4.3	The *ldexp*( ) Function	

The same as ISO/IEC 9899.

7.5.4.4 The *log*( ) Function

The same as ISO/IEC 9899.

7.5.4.5 The *log10*( ) Function

The same as ISO/IEC 9899.

7.5.4.6 The *modf*( ) Function

The same as ISO/IEC 9899.

7.5.5 Power Functions

7.5.5.1 The *pow*( ) Function

The same as ISO/IEC 9899.

7.5.5.2 The *sqrt*( ) Function

The same as ISO/IEC 9899.

7.5.6. Nearest Integer, Absolute Value and Remainder Functions

7.5.6.1 The *ceil*( ) Function

The same as ISO/IEC 9899.

7.5.6.2 The *fabs*( ) Function

The same as ISO/IEC 9899.

7.5.6.3 The *floor*( ) Function

The same as ISO/IEC 9899.

7.5.6.4 The *fmod*( ) Function

Synopsis — The same as ISO/IEC 9899.

Description — The same as ISO/IEC 9899.

Returns — The *fmod*( ) function returns the value x–i * y, for some integer i such that, if y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y. If y is zero, *fmod*( ) function returns zero and errno may be set to EDOM.

7.6 Non-local Jumps **<setjump.h>**

The same as ISO/IEC 9899.

7.6.1 Save Calling Environment

7.6.1.1 The *setjump* Macro

The same as ISO/IEC 9899.

7.6.2 Restore Calling Environment

7.6.2.1 The *longjmp*( ) Function

The same as ISO/IEC 9899.

7.7 Signal Handling **<signal.h>**

The same as ISO/IEC 9899.

7.7.1 Specify Signal Handling

7.7.1.1 The *signal*( ) Function

The same as ISO/IEC 9899.

7.7.2 Send Signal

7.7.2.1 The *raise*( ) Function

The same as ISO/IEC 9899.

7.8 Variable Arguments **<stdarg.h>**

The same as ISO/IEC 9899.

7.8.1 Variable Argument List Access Macros

The same as ISO/IEC 9899.

7.8.1.1 The *va_start* Macro

The same as ISO/IEC 9899.

7.8.1.2 The *va_arg* Macro

The same as ISO/IEC 9899.

7.8.1.3 The *va_end* Macro

The same as ISO/IEC 9899.

7.9 Input/Output **<stdio.h>**

7.9.1 Introduction

The same as ISO/IEC 9899 with ISO/IEC 9899/AM1.

7.9.2 Streams

The same as ISO/IEC 9899 with ISO/IEC 9899/AM1.

7.9.3 Files

The same as ISO/IEC 9899 with ISO/IEC 9899/AM1.

7.9.4 Operations on Files

7.9.4.1 The *remove*( ) Function

The same as ISO/IEC 9899.

7.9.4.2 The *rename*( ) Function

The same as ISO/IEC 9899.

7.9.4.3 The *tmpfile*( ) Function

The same as ISO/IEC 9899.

7.9.4.4 The *tmpnam*( ) Function

The same as ISO/IEC 9899.

7.9.5 File Access Functions

7.9.5.1 The *fclose*( ) Function

The same as ISO/IEC 9899.

7.9.5.2 The *fflush*( ) Function

The same as ISO/IEC 9899.

7.9.5.3 The *fopen*( ) Function

The same as ISO/IEC 9899.

7.9.5.4 The *freopen*( ) Function

The same as ISO/IEC 9899.

7.9.5.5 The *setbuf*( ) Function

The same as ISO/IEC 9899.

7.9.5.6 The *setvbuf*( ) Function

The same as ISO/IEC 9899.

7.9.6 Formatted Input/Output Functions

7.9.6.1 The *fprintf*( ) Function

Synopsis The same as ISO/IEC 9899.

Description The *fprintf*( ) function writes output to the **stream** pointed to by **stream**, under control of the string pointed to by **format** that specifies how subsequent arguments are covered for output. If there are insufficient arguments for the format, the behaviour is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated (as always) but are otherwise ignored. The *fprintf*( ) function returns when the end of the format string is encountered.

The format shall be a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives:

- ordinary multibyte characters (not %), which are copied unchanged to the output stream
- conversion specifications, each of which results in fetching zero or more subsequent arguments.

Conversions can be applied to the nth argument after the format in the argument list, rather than to the next unused argument. In this case, the conversion character % (see below) is replaced by the sequence **%n$***, where n is a decimal integer in the range [1, {NL_ARGMAX}], giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages.*

In format strings containing the **%n$** *form of conversion specifications, numbered arguments in the argument list can be referenced from the format string as many times as required.*

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

- Zero or more *flags* (in any order) that modify the meaning of the conversion specification.
- An optional minimum *field width*. If the converted value has fewer characters than the field width, it will be padded with spaces (by default) on the left (or right, if the left adjustment flag, described later, has been given) to the field width. The field width takes the form of an asterisk ***** (described later) or a decimal integer.
- An optional *precision* that gives the minimum number of digits to appear for the **d, i, o, u, x** and **X** conversions, the number of digits to appear after the decimal-point character for **e, E** and **f** conversions, the maximum number of significant digits for the **g** and **G** conversions, or the maximum number of characters to be written from a string in **s** conversion. The precision takes the form of a period (.) followed either by an asterisk ***** (described later) or by an optional decimal integer; if only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behaviour is undefined.
- An optional **h** specifying that a following **d, i, o, u, x** or **X** conversion specifier applies to a **short int** or **unsigned short int** argument (the argument will have been promoted according to the integral promotions, and its value shall be converted to **short int** or **unsigned short int** before printing); an optional **h** specifying that a following **n** conversion specifier applies to a pointer to a **short int** argument; an optional **l** (ell) specifying that a following **d, i, o, x** or **X** conversion specifier applies to a **long int** or **unsigned long int** argument; an optional **l** specifying that a following **n** conversion specifier applies to a pointer to a **long int** argument; *an optional* **l** *specifying that a following* **c** *conversion specifier applies to a* **wint_t** *argument; an optional* **l** *specifying that a following* **s** *conversion specifier applies to a pointer to a* **wchar_t** *argument;* or an optional **L** specifying that a following **e, E, f, g** or **G** conversion specifier applies to a **long double** argument; if an **h, l** or **L** appears with any other conversion specifier, the behaviour is undefined.
- A character that specifies the type of conversion to be applied.

As noted above, a field width or precision or both may be indicated by an asterisk. In this case, an **int** argument supplies the field width or precision. The arguments specifying field width or precision or both shall appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a - flag followed by a positive field width. A negative precision argument is taken as if the precision were

omitted. *In format strings containing the* **%n$** *form of a conversion specification, a field width or precision may be indicated by the sequence* ***m$****, where m is a decimal integer in the range [1, {NL_ARGMAX}] giving the position in the argument list (after the format argument) of an integer argument containing the field width or precision, for example:*

```
printf ("%1$d:%2$.3$d:%4$.3$d0,
hour, min, precision, sec);
```

*The format can contain either numbered argument specifications (***%n$** *and* ***m$****), or numbered argument specifications (% and* *******), but normally not both. The only exception to this is that %% can be mixed with the %m$ form. The results of mixing numbered and unnumbered argument specifications in a format string are undefined. When numbered argument specifications are used, specifying the Nth argument requires that all the leasing arguments, from the first to the (N-1)th, are specified in the format string.*

The flag characters and their meanings are:

’ *The integer portion of the result of a decimal conversion (%1, %d, %u, %f, %g or %G) will be formatted with ‘‘thousands’’ grouping characters. For other conversions the behaviour is undefined. The non-monetary grouping character is used.*

\- The result of the conversion will be left-justified within the field. (It will be right-justified if this flag is not specified.)

\+ The result of a **signed** conversion will always begin with a plus or minus sign. (It will begin with a sign only when a negative value is converted if this flag is not specified.)

space If the first character of a signed conversion is not a sign, or of a signed conversion results in no characters, a space will be prefixed to the result. If the *space* and **+** flags both appear, the space flag will be ignored.

\# The result is to be converted to an ‘‘alternate form’’. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For x (or X) conversion, a non-zero result will have 0x (or 0X) prefixed to it. For **e, E, f, g** and **G** conversions, the result will always contain a decimal-point character, even if no digits follow it. (Normally, a decimal-point character appears in the result of these conversions only if a digit follows it.) For **g** and **G** conversions, trailing zeros will not be removed from the results. For other conversions, the behaviour is undefined.

0 For **d, i, o, u, x, X, e, E, f, g** and **G** conversions, leading zeros (following any indication of sign or base) are used to pad the field width; no space padding is

performed. If the 0 and - flags both appear, the 0 flag will be ignored. For **d, i, o, u, x** and **X** conversions, if a precision is specified, the 0 flag will be ignored. For other conversions, the behaviour is undefined.

The conversion specifiers and their meanings are:

d,i — The **int** argument is converted to signed decimal in the style [-]dddd. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no character.

o,u,x,X — The **unsigned int** argument is converted to **unsigned octal** (o), **unsigned decimal** (u), or **unsigned hexadecimal** notation (x or X) in the style dddd; the letters abcdef are used for x conversion and the letters ABCDEF for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

f — The double argument is converted to decimal notation in the style [-]ddd.ddd, where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is zero and the # flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.

e,E — The double argument is converted to the style [-]d.ddde1dd, where there is one digit before the decimal-point character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero and the # flag is not specified, no decimal-point character appears. The value is rounded to the appropriate number of digits. The **E** conversion specifier will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.

g,G — The double argument is converted in style for **e** (or in style **E** in the case of a **G** conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as 1. The style used depends on the value converted; style **e** (or **E**) will be used only if the exponent resulting from such a

conversion is less than –4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result; a decimal-point character appears only if it is followed by a digit.

c — If no **l** qualifier is present, the **int** argument is converted to an **unsigned char**, and the resulting character is written. Otherwise, the **wint_t** argument is converted as if by an ls conversion specification with no precision and an argument that points to a two-element array of **wchar_t**, the first element containing **wint_t** argument to the **lc** conversion specification and the second a null wide character.

s — If no **l** qualifier is present, the argument shall be a pointer to an array of character type. Characters from the array are written up to (but not including) a terminating null character. If the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array shall contain a null character.

If an **l** *qualifier is present, the argument shall be a pointer to an array of* **wchar_t** *type. Wide characters from the array are converted to multibyte characters (each as if by a call to the wcrtomb( ) function, with the conversion state described by an* **mbstate_t** *object initialized to zero before the first wide character is converted) up to and including a terminating null wide character. The resulting multibyte characters are written up to (but not including) the terminating null character (byte).*

If no precision is specified, the array shall contain a null wide character.

If a precision is specified, no more than that many characters (bytes) are written (including shift sequences, if any), and the array shall contain a null wide character if, to equal the multibyte character sequence length given by the precision, the function would need to access a wide character one past the end of the array. In no case is partial multibyte character written.

P — The argument shall be a pointer to **void**. The value of the pointer is converted to a sequence of printable characters, in an implementation-defined manner.

n — The argument shall be a pointer to an integer into which is written the number of characters written to the output stream so far by this call to *fprintf*( ). No argument is converted.

% — A % is written. No argument is converted. The complete conversion specification shall be %%.

If a conversion specification is invalid, the behaviour is undefined.

If any argument is, or points to, a union or an aggregate (except for an array of character type using **%**s conversion, or a pointer using **%p** conversion), the behaviour is undefined.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

Returns — The same as ISO/IEC 9899.

Example — The same as ISO/IEC 9899.

7.9.6.2 The *fscanf*( ) Function

Synopsis — The same as ISO/IEC 9899.

Description — The *fscanf*( ) function reads input from the stream pointed to by **stream**, under control of the string pointed to by **format** that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointed to the objects to receive the converted input. If there are insufficient arguments for the format, the behaviour is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated (as always) but are otherwise ignored.

Conversions can be applied to the nth argument after the format in the argument list, rather than to the next unused argument. In this case, the conversion character % (see below) is replaced by the sequence **%n$***, where* **n** *is a decimal integer in the range [1,{NL_ARGMAX}]. This feature provides for the definition of a format string that selects arguments in an order appropriate to specific languages. In format strings containing the* **%n$** *form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the format string more than once.*

The format can contain either form of a conversion specification; that is, % or **%n$***, but the two forms cannot normally be mixed within a single format string. The only exception to this is that %% or %* can be mixed with the* **%n$** *form.*

The format shall be a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: one or more white-space characters, an ordinary multibyte character (neither % nor a white-space character), or a conversion specification. Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

- An optional assignment-suppressing character *****.
- An optional non-zero decimal integer that specifies the minimum field width.

- An optional **h**, **l** (ell) or **L** indicating the size of the receiving object. The conversion specifiers **d**, **i** and **n** shall be preceded by **h** if the corresponding argument is a pointer to **short int** rather than a pointer to **int**, or by **i** if it is a pointer to **long int**. Similarly, the conversion specifiers **o**, **u** and **x** shall be preceded by **h** if the corresponding argument is a pointer to **unsigned short int** rather than a pointer to **unsigned int**, or by **i** if it is a pointer to **unsigned long int**. The conversion specifiers **c**, **s** and [shall be preceded by **i** if the corresponding argument is a pointer to **wchar_t** rather than a pointer to a character type. Finally, the conversion specifiers **e**, **f** and **g** shall be preceded by **i** if the corresponding argument is a pointer to double rather than a pointer to a float, or by **L** if it is a pointer to **long double**. If an **h**, **l** or **L** appears with any other conversion specifier, the behaviour is undefined.
- A character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

The *fscanf*( ) function executes each directive of the format in turn. If a directive fails, as detailed below, the *fscanf*( ) function returns. Failures are described as input failures (due to the unavailability of input characters) or matching failures (due to inappropriate input).

A directive composed of white-space character(s) is executed by reading input up to the first non-white-space character (which remains unread), or until no more characters can be read.

A directive that is an ordinary multibyte character is executed by reading the next characters of the stream. If one of the characters differs from one composing the directive, the directive fails, and the differing and subsequent characters remain unread.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each specifier. A conversion specification is executed in the following steps.

Input white-space characters (as specified by the *isspace*( ) function) are skipped, unless the specification includes a **[**, **c** or **n** specifier.

An input item is read from the stream, unless the specification includes an n specifier. An input item is defined as the longest matching sequence of input characters, unless that exceeds a specified field width, in which case it is the initial sequence of that length in the sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails; this condition is matching failure unless end-of-file, an encoding error or a read error prevented input from the stream, in which case it is an input failure.

Except in the case of a % specifier, the input item (or, in the case of a **%n** directive, the count of input characters) is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is matching failure. Unless assignment suppression was indicated by a *****, the result of the conversion is placed in the object pointed to by the first argument following the format argument that has not already received a conversion result if the conversion specification is introduced by %, or in the nth argument if introduced by the character sequence **%n$**. If the object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behaviour is undefined.

The following conversion specifiers are valid:

d — Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the *strtol*( ) function with the value for the **base** argument. The corresponding argument shall be a pointer to an integer.

i — Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the *strtol*( ) function with the value 0 for the **base** argument. The corresponding argument shall be a pointer to an integer.

o — Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the *strtol*( ) function with the value 8 for the **base** argument. The corresponding argument shall be a pointer to an integer.

u — Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the *strtol*( ) function with the value 10 for the **base** argument. The corresponding argument shall be a pointer to unsigned integer.

x — Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the *strtol*( ) function with the value 16 for the **base** argument. The corresponding argument shall be a pointer to unsigned integer.

e,f,g — Matches an optionally signed floating-point number, whose format is the same as expected for the subject string of the *strtod*( ) function. The corresponding argument shall be a pointer to floating.

s — Matches a sequence of non-white-space characters. *If no* **i** *qualifier is present, the corresponding argument shall be a pointer to a character array large enough to accept the sequence and terminating with a null character, which will be added automatically. If an* **i**

qualifier is present, the input shall be a sequence of multibyte characters that begins in the initial shift state. Each multibyte character is converted to a wide character as if by a call to the mbrtowc( ) function, with the conversion state described by an **mbstate_t** *object initialized to zero before the first multibyte character is converted.* The corresponding argument shall be a pointer to an array of **wchar_t** large enough to accept the sequence and the terminating null wide character, which will be added automatically.

[Matches a non-empty sequence of characters from a set of expected characters (the *scanset*). If no l qualifier is present, the corresponding argument shall be a pointer to a character array large enough to accept the sequence and a terminating null character, which will be added automatically. If an **l** qualifier is present, the input shall be a sequence of multibyte characters that begins in the initial shift state. Each multibyte character is converted to a wide character by a call to the *mbrtowc*( ) function, with the conversion state described by an **mbstate_t** object initialized to zero before the first multibyte character is converted. The corresponding argument shall be a pointer to an array of **wchar_t** large enough to accept the sequence and terminating null wide character, which will be added automatically. The conversion specifier includes all subsequent characters in the format string up to and including the matching right bracket]. The characters between the brackets is a circumflex (^), in which case the *scanset* contains all characters that do not appear in the *scanlist* between the circumflex and the right bracket. If the conversion specifier begins with [ ] or [^], the right bracket character is in the *scanlist* and the next bracket character is the matching right bracket that ends the specification; otherwise the first bracket character is the one that ends the specification. If a - character is in the *scanlist* and is not the first, nor the second where the first character is a ^, nor the last character, the behaviour is implementation-defined.

c Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the directive). If no **i** qualifier is present the corresponding argument shall be a pointer to a character array large enough to accept the sequence. No null character is added. If an **i** qualifier is present, the input shall be a sequence of multibyte characters that begins in the initial shift state. Each multibyte character in the sequence is converted to a wide character by a call to the *mbrtowc*( ) function, with the conversion state described by an **mbstate_t** object initialized to zero before the first multibyte character is

converted. The corresponding argument shall be a pointer to the initial element of an array of **wchar_t** large enough to accept the resulting sequence of wide characters. No null wide character is added.

p Matches an implementation-defined set of sequences, which should be the same as the set of sequences that may be produced by the **%p** conversion of the *fprintf*( ) function. The corresponding argument shall be a pointer to a pointer to **void**. The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise the behaviour of the **%p** conversion is undefined.

n No input is consumed. The corresponding argument shall be a pointer to an integer into which is to be written the number of characters read from the input stream so far by this call to the *fscanf*( ) function. Execution of a **%n** directive does not increment the assignment count returned at the completion of execution of the **fscanf**( ) function.

% Matches a single %; no conversion or assignment occurs. The complete conversion specification shall be %%.

If a conversion specification is invalid, the behaviour is undefined.

The conversion specifiers **E**, **G** and **X** are also valid and behave the same as, respectively, **e**, **g** and **x**.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any characters matching the current directive have been read (other than leading white space, where permitted), execution of the current directive terminates with an input failure; otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive (if any) is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space (including new-line characters) is left unread unless matched by a directive. The success of literal matches and suppressed assignments is not directly determinable other than via the **%n** directive.

Returns The same as ISO/IEC 9899.

Examples The same as ISO/IEC 9899.

7.9.6.3 The *printf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fprintf*( ) function is applied.

7.9.6.4 The *scanf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fscanf*( ) function is applied.

7.9.6.5 The *sprintf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fprintf*( ) function is applied.

7.9.6.6 The *sscanf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fscanf*( ) function is applied.

7.9.6.7 The *vfprintf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fprintf*( ) function is applied.

7.9.6.8 The *vprintf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fprintf*( ) function is applied.

7.9.7 Character Input/Output Functions

7.9.7.1 The *fgetc*( ) Function

The same as ISO/IEC 9899.

7.9.7.2 The *fgets*( ) Function

The same as ISO/IEC 9899.

7.9.7.3 The *fputc*( ) Function

The same as ISO/IEC 9899.

7.9.7.4 The *fputs*( ) Function

The same as ISO/IEC 9899.

7.9.7.5 The *getc*( ) Function

The same as ISO/IEC 9899.

7.9.7.6 The *getchar*( ) Function

The same as ISO/IEC 9899.

7.9.7.7 The *gets*( ) Function

The same as ISO/IEC 9899.

7.9.7.8 The *putc*( ) Function

The same as ISO/IEC 9899.

7.9.7.9 The *putchar*( ) Function

The same as ISO/IEC 9899.

7.9.7.10 The *puts*( ) Function

The same as ISO/IEC 9899.

7.9.7.11 The *ungetc*( ) Function

The same as ISO/IEC 9899.

7.9.8 Direct Input/Output Functions

7.9.8.1 The *fread*( ) Function

The same as ISO/IEC 9899.

7.9.8.2 The *fwrite*( ) Function

The same as ISO/IEC 9899.

7.9.8.3 The *fgetpos*( ) Function

The same as ISO/IEC 9899.

7.9.8.4 The *fseek*( ) Function

The same as ISO/IEC 9899.

7.9.8.5 The *fsetpos*( ) Function

The same as ISO/IEC 9899.

7.9.8.6 The *ftell*( ) Function

The same as ISO/IEC 9899.

7.9.8.7 The *rewind*( ) Function

The same as ISO/IEC 9899.

7.9.9 Error Handling Functions

7.9.9.1 The *clearerr*( ) Function

The same as ISO/IEC 9899.

7.9.9.2 The *feof*( ) Function

The same as ISO/IEC 9899.

7.9.9.3 The *ferror*( ) Function

The same as ISO/IEC 9899.

7.9.9.4 The *perror*( ) Function

The same as ISO/IEC 9899.

7.10 General Utilities **<stdlib.h>**

The same as ISO/IEC 9899.

7.10.1 String Conversion Functions

The same as ISO/IEC 9899.

7.10.1.1 The *atof*( ) Function

The same as ISO/IEC 9899.

7.10.1.2 The *atoi*( ) Function

The same as ISO/IEC 9899.

7.10.1.3 The *atol*( ) Function

The same as ISO/IEC 9899.

7.10.1.4 The *strtod*( ) Function

The same as ISO/IEC 9899.

7.10.1.5 The *strtol*( ) Function

The same as ISO/IEC 9899.

7.10.1.6 The *strtoul*( ) Function

The same as ISO/IEC 9899.

7.10.2 Pseudo-random Sequence Generation Functions

7.10.2.1 The *rand*( ) Function

Synopsis	The same as ISO/IEC 9899.
Description	The *rand*( ) function computes a sequence of pseudo-random integers in the range 0 to RAND_MAX, with a period at least 4294967296.
Returns	The same as ISO/IEC 9899.
Environmental	The same as ISO/IEC 9899.

7.10.2.2 The *srand*( ) Function

The same as ISO/IEC 9899.

7.10.3 Memory Management Functions

The same as ISO/IEC 9899.

7.10.3.1 The *calloc*( ) Function

The same as ISO/IEC 9899.

7.10.3.2 The *free*( ) Function

The same as ISO/IEC 9899.

7.10.3.3 The *malloc*( ) Function

The same as ISO/IEC 9899.

7.10.3.4 The *realloc*( ) Function

The same as ISO/IEC 9899.

7.10.4 Communication with the Environment

7.10.4.1 The *abort*( ) Function

The same as ISO/IEC 9899.

7.10.4.2 The *atexit*( ) Function

The same as ISO/IEC 9899.

7.10.4.3 The *exit*( ) Function

The same as ISO/IEC 9899.

7.10.4.4 The *getenv*( ) Function

The same as ISO/IEC 9899.

7.10.4.5 The *system*( ) Function

The same as ISO/IEC 9899.

7.10.5 Searching and Sorting Utilities

7.10.5.1 The *bsearch*( ) Function

The same as ISO/IEC 9899.

7.10.5.2 The *qsort*( ) Function

The same as ISO/IEC 9899.

7.10.6 Integer Arithmetic Functions

7.10.6.1 The *abs*( ) Function

The same as ISO/IEC 9899.

7.10.6.2 The *div*( ) Function

The same as ISO/IEC 9899.

7.10.6.3 The *labs*( ) Function

The same as ISO/IEC 9899.

7.10.6.4 The *ldiv*( ) Function

The same as ISO/IEC 9899.

7.10.7 Multibyte Character Functions

The same as ISO/IEC 9899.

7.10.7.1 The *mblen*( ) Function

The same as ISO/IEC 9899.

7.10.7.2 The *mbtowc*( ) Function

The same as ISO/IEC 9899.

7.10.7.3 The *wctomb*( ) Function

The same as ISO/IEC 9899.

7.10.8 Multibyte String Functions

7.10.8.1 The *mbstowcs*( ) Function

The same as ISO/IEC 9899.

7.10.8.2 The *wcstombs*( ) Function

The same as ISO/IEC 9899.

7.11 String Handling **<string.h>**

7.11.1 String Function Conventions

The same as ISO/IEC 9899.

7.11.2 Copying Functions

7.11.2.1 The *memcpy*( ) Function

The same as ISO/IEC 9899.

7.11.2.2 The *memmove*( ) Function

The same as ISO/IEC 9899.

7.11.2.3 The *strcpy*( ) Function

The same as ISO/IEC 9899.

7.11.2.4 The *strncpy*( ) Function

The same as ISO/IEC 9899.

7.11.3 Concatenation functions

7.11.3.1 The *strcat*( ) Function

The same as ISO/IEC 9899.

7.11.3.2 The *strncat*( ) Function

The same as ISO/IEC 9899.

7.11.4 Comparison Functions

The same as ISO/IEC 9899.

7.11.4.1 The *memcmp*( ) Function

The same as ISO/IEC 9899.

7.11.4.2 The *strcmp*( ) Function

The same as ISO/IEC 9899.

7.11.4.3 The *strcoll*( ) Function

The same as ISO/IEC 9899.

7.11.4.4 The *strncmp*( ) Function

The same as ISO/IEC 9899.

7.11.4.5 The *strxfrm*( ) Function

The same as ISO/IEC 9899.

7.11.5 Search Functions

7.11.5.1 The *memchr*( ) Function

The same as ISO/IEC 9899.

7.11.5.2 The *strchr*( ) Function

The same as ISO/IEC 9899.

7.11.5.3 The *strcspn*( ) Function

The same as ISO/IEC 9899.

7.11.5.4 The *strpbrk*( ) Function

The same as ISO/IEC 9899.

7.11.5.5 The *strrchr*( ) Function

The same as ISO/IEC 9899.

7.11.5.6 The *strspn*( ) Function

The same as ISO/IEC 9899.

7.11.5.7 The *strstr*( ) Function

The same as ISO/IEC 9899.

7.11.5.8 The *strtok*( ) Function

The same as ISO/IEC 9899.

7.11.6 Miscellaneous Functions

7.11.6.1 The *memset*( ) Function

The same as ISO/IEC 9899.

7.11.6.2 The *strerror*( ) Function

The same as ISO/IEC 9899.

7.11.6.3 The *strlen*( ) Function

The same as ISO/IEC 9899.

7.12 Date and Time **<time.h>**

7.12.1 Components of Time

The same as ISO/IEC 9899.

7.12.2 Time Manipulation Functions

7.12.2.1 The *clock*( ) Function

The same as ISO/IEC 9899.

7.12.2.2 The *difftime*( ) Function

The same as ISO/IEC 9899.

7.12.2.3 The *mktime*( ) Function

The same as ISO/IEC 9899.

7.12.2.4 The *time*( ) Function

The same as ISO/IEC 9899.

7.12.3 Time Conversion Functions

The same as ISO/IEC 9899.

7.12.3.1 The *asctime*( ) Function

The same as ISO/IEC 9899.

7.12.3.2 The *ctime*( ) Function

The same as ISO/IEC 9899.

7.12.3.3 The *gmtime*( ) Function

The same as ISO/IEC 9899.

7.12.3.4 The *localtime*( ) Function

The same as ISO/IEC 9899.

7.12.3.5 The *strftime*( ) Function

Synopsis The same as ISO/IEC 9899.

Description The *strftime*( ) function places characters into the array pointed to be **s** as controlled by the string pointed to by **format**. The **format** shall be a multibyte character sequence, beginning and ending in its initial shift state. The **format** string consists of zero or more conversion specifiers and ordinary multibyte characters. A conversion specifier consists of a % character followed by a character that determines the behaviour of the conversion specifier. All ordinary multibyte characters (including the terminating null character) are copies unchanged into the array. If copying takes place between objects that overlap, the behaviour is undefined. No more than **maxsize** characters are placed into the array. Each conversion specifier is replaced by appropriate characters as described in the following list. The appropriate characters are determined by the **LC_TIME** category of the current locale and by the value contained in the structure pointed to by **timeptr**.

%a is replaced by the locale's abbreviated weekday name.

%A is replaced by the locale's full weekday name.

%b is replaced by the locale's abbreviated month name.

%B is replaced by the locale's full month name.

%c is replaced by the locale's appropriate date and time representation.

%C *is replaced by the century number (the year divided by 100 and truncated to an integer) as a decimal number [00-99].*

%d is replaced by the day of the month as a decimal number [01,31].

%D *same as%m/%d/%y.*

%e *is replaced by the day of the month as a decimal number [1,31]; a single digit is replaced by a space*.

%h *same as%b*.

%H is replaced by the hour (24-hour clock) as a decimal number [00,23].

%I is replaced by the hour (12-hour clock) as a decimal number [01,12].

%j is replaced by the day of the year as a decimal number [001,366].

%m is replaced by the month as a decimal number [01,12].

%M is replaced by the minutes as a decimal number.

%n *is replaced by a newline character.*

%p *is replaced by the locale's equivalent of either a.m. or p.m.*

%r *is replaced by the time in a.m. and p.m. notation; on the C locale this is equivalent to %I: %M: %S˜%p.*

%R *is replaced by the time in 24-hour notation (%H: %M).*

%S is replaced by the second as a decimal number [00,61].

%t *is replaced by a tab character.*

%T *is replaced by the time (%H: %M: %S).*

%u *is replaced by the weekday as a decimal number [1,7], with 1 representing Monday.*

%U is replaced by the week number of the year (Sunday as the first day of the week) as a decimal number [00,53].

%V *is replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1; otherwise, it is week 53 of the previous year, and the next week is week 1.*

%w is replaced by the weekday as a decimal number [0,6], with 0 representing Sunday.

%W is replaced by the week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be week 0.

%x is replaced by the locale's appropriate date representation.

%X is replaced by the locale's appropriate time representation.

%y is replaced by the year without century as a decimal number [00,99].

%Y is replaced by the year with century as a decimal number.

%Z is replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists.

%% is replaced by%.

If a conversion specification does not correspond to any of the above, the behaviour is undefined.

Modified Conversion Specifiers

Some conversion specifiers can be modified by the **E** *or* **O** *modifier characters to indicate that an alternative format or specification should be used rather than one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale, the behaviour will be as if the unmodified conversion specification were used.*

%Ec *is replaced by the locale's alternative appropriate date and time representation.*

%EC *is replaced by the name of the base year (period) in the locale's alternative representation.*

%Ex *is replaced by the locale's alternative date representation.*

%EX *is replaced by the locale's alternative time representation.*

%Ey *is replaced by the offset from %EC (year only) in the locale's alternative representation.*

%EY *is replaced by the full alternative year representation.*

%Od *is replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero, otherwise with leading spaces.*

%Oe *is replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading spaces.*

%OH *is replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.*

%OI *is replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.*

%Om *is replaced by the month using the locale's alternative numeric symbols.*

%OM *is replaced by the minutes using the locale's alternative numeric symbols.*

%OS *is replaced by the seconds using the locale's alternative numeric symbols.*

%Ou *is replaced by the weekday as a number in the locale's alternative representation (Monday = 1).*

%OU *is replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %U) using the locale's alternative numeric symbols.*

%OV *is replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %V) using the locale's alternative numeric symbols.*

%Ow *is replaced by the number of the weekday (Sunday = 0) using the locale's alternative numeric symbols.*

%OW *is replaced by the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.*

%Oy *is replaced by the year (offset from %C) in the locale's alternative representation and using the locale's alternative symbols.*

Returns The same as ISO/IEC 9899.

7.13 Alternative Spelling **<iso646.h>**

The same as ISO/IEC 9899/AM1.

7.14 Wide-character Classification and Mapping Utilities **<wctype.h>**

The same as ISO/IEC 9899/AM1.

7.14.1 Wide-character Classification Utilities

The same as ISO/IEC 9899/AM1.

7.14.1.1 Wide-character Classification Functions

The same as ISO/IEC 9899/AM1.

7.14.1.1.1 The *iswalnum*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.1.1.2 The *iswalpha*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.1.1.3 The *iswcntrl*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.1.1.4 The *iswdigit*( ) Function

7.14.1.1.5 The *iswgraph*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.1.1.6 The *iswlower*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.1.1.7 The *iswprint*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.1.1.8 The *iswpunct*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.1.1.9 The *iswspace*( ) Functions

The same as ISO/IEC 9899/AM1.

7.14.1.1.10 The *iswupper*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.1.1.11 The *iswxdigit*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.1.2 Extensible Wide-character Classification Functions

The same as ISO/IEC 9899/AM1.

7.14.1.2.1 The *wctype*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.1.2.2 The *iswctype*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.2 Wide-character Mapping Utilities

The same as ISO/IEC 9899/AM1.

7.14.2.1 Wide-character Case-mapping Functions

7.14.2.1.1 The *towlower*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.2.1.2 The *towupper*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.2.2 Extensible Wide-character Mapping Functions

The same as ISO/IEC 9899/AM1.

7.14.2.2.1 The *wctrans*( ) Function

The same as ISO/IEC 9899/AM1.

7.14.2.2.2 The *towctrans*( ) Function

The same as ISO/IEC 9899/AM1.

7.15 Extended Multibyte and Wide-character Utilities **<wchar.h>**

The same as ISO/IEC 9899/AM1.

7.15.1 Formatted Wide-character Functions

The same as ISO/IEC 9899/AM1.

7.15.1.1 Formatted Wide-character Input/Output Functions

7.15.1.1.1 The *fwprintf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fprintf*( ) function is applied.

7.15.1.1.2 The *fwscanf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fscanf*( ) function is applied.

7.15.1.1.3 The *wprintf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fprintf*( ) function is applied.

7.15.1.1.4 The *wscanf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fscanf*( ) function is applied.

7.15.1.1.5 The *swprintf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fprintf*( ) function is applied.

7.15.1.1.6 The *swscanf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fscanf*( ) function is applied.

7.15.1.1.7 The *vfwprintf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fprintf*( ) function is applied.

7.15.1.1.8 The *vwprintf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fprintf*( ) function is applied.

7.15.1.1.9 The *vswprintf*( ) Function

In addition to the requirements specified by ISO/IEC 9899 with ISO/IEC 9899/AM1, the same modification to the *fprintf*( ) function is applied.

7.15.1.2 Wide-character Input/Output Functions

7.15.1.2.1 The *fgetwc*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.1.2.2 The *fgetws*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.1.2.3 The *fputwc*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.1.2.4 The *fputws*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.1.2.5 The *getwc*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.1.2.6 The *getwchar*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.1.2.7 The *putwc*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.1.2.8 The *putwchar*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.1.2.9 The *ungetwc*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.1.2.10 The *fwide*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2 General Wide-string Utilities

The same as ISO/IEC 9899/AM1.

7.15.2.1 Wide-string Numeric Conversion Functions

7.15.2.1.1 The *wcstod*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.1.2. The *wcstol*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.1.3 The *wcstoul*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.2 Wide-string Copying Functions

7.15.2.2.1 The *wcscpy*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.2.2 The *wcsncpy*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.3 Wide-string Concatenation Functions

7.15.2.3.1 The *wcscat*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.3.2 The *wcsncat*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.4 Wide-string Comparison Functions

The same as ISO/IEC 9899/AM1.

7.15.2.4.1 The *wcscmp*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.4.2 The *wcscoll*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.4.3 The *wcsncmp*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.4.4 The *wcsxfrm*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.5 Wide-string Search Functions

7.15.2.5.1 The *wcschr*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.5.2 The *wcscspn*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.5.3 The *wcspbrk*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.5.4 The *wcsrchr*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.5.5 The *wcsspn*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.5.6 The *wcsstr*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.5.7 The *wcstok*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.5.8 The *wcslen*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.6 Wide-character Array Functions

The same as ISO/IEC 9899/AM1.

7.15.2.6.1 The *wmemchr*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.6.2 The *wmemcmp*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.6.3 The *wmemcpy*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.6.4 The *wmemmove*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.2.6.5 The *wmemset*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.3 Date and Time

7.15.3.1 Wide-string Time Conversion Functions

7.15.3.1.1 The *wcsftime*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.4 Extended Multibyte and Wide-character Conversion Utilities

The same as ISO/IEC 9899/AM1.

7.15.4.1 Single-byte Wide-character Conversion Functions

7.15.4.1.1 The *btowc*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.4.1.2 The *wctob*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.4.2 Conversion State Functions

7.15.4.2.1 The *mbsinit*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.4.3 Restartable Multibyte/Wide-character Conversion Functions

The same as ISO/IEC 9899/AM1.

7.15.4.3.1 The *mbrlen*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.4.3.2 The *mbrtowc*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.4.3.3 The *wcrtomb*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.4.4 Restartable Multibyte/Wide-string Conversion Functions

The same as ISO/IEC 9899/AM1.

7.15.4.4.1 The *mbsrtowcs*( ) Function

The same as ISO/IEC 9899/AM1.

7.15.4.4.2 The *wcsrtombs*( ) Function

The same as ISO/IEC 9899/AM1.

7.16 Future Library Directions

The same as ISO/IEC 9899.

7.16.1 Errors **<errno.h>**

The same as ISO/IEC 9899.

7.16.2 Character Handling **<ctype.h>**

The same as ISO/IEC 9899.

7.16.3 Localization **<loclae.h>**

The same as ISO/IEC 9899.

7.16.4 Mathematics **<math.h>**

The same as ISO/IEC 9899.

7.16.5 Signal Handling **<signal.h>**

The same as ISO/IEC 9899.

7.16.6 Input/Output **<stdio.h>**

The same as ISO/IEC 9899.

7.16.7 General Utilities **<stdlib.h>**

The same as ISO/IEC 9899.

7.16.8 String Handling **<string.h>**

The same as ISO/IEC 9899.

7.16.9 Wide-character Classification and Mapping Utilities **<wctype.h>**

The same as ISO/IEC 9899/AM1.

7.16.10 Extended Multibyte and Wide-character Utilities **<wchar.h>**

The same as ISO/IEC 9899 with ISO/IEC 9899/AM1.

Appendix A

X/Open C and MIA C

A.1 C-language Interface Specifications

The SPIRIT C-language interface specification (SPIRIT C) is based on ISO C, X/Open C[4] and MIA C.[5] Figure A-1 on page 50 shows the relationship between them.

4. The definition of X/Open C is both ISO C (a direct reference to ISO/IEC 9899) and Common Usage C, which is defined in Chapters 1 to 4 of X/Open Specification, 1988, 1989, February 1992, Programming Languages, Issue 3 (ISBN: 1-872630-39-1, C214).
5. Multivendor Integration Architecture, Volume 4: Division 2, Application Program Interface Specifications, Part 3-3: Programming Language C, 31 March 1992.

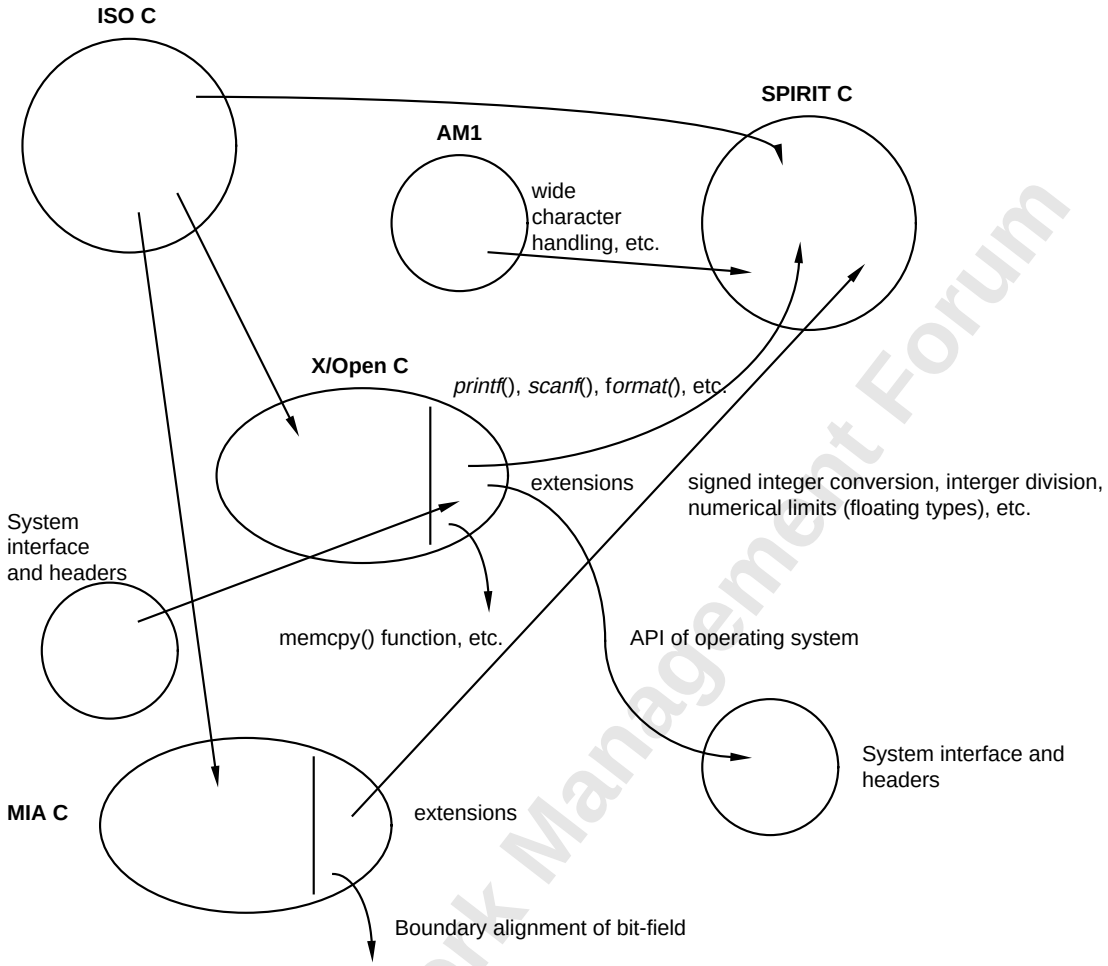

Figure A-1 C-language Interface Specifications

A.2 Differences

Table A-1 shows the differences between SPIRIT C, X/Open C (XPG4) and MIA C.

Table A-1 SPIRIT C, X/Open C and MIA C

No.	Items	XPG4	MIA	SPIRIT	SPIRIT Rationale
1	The name of header for wide character string handling functions.	X		AM1	The header's name is specified by ISO/IEC 9899/AM1.
2	**wint_t** and **wctype_t** data types and WEOF macro.	X		AM1	The data types and the macro are specified by ISO/IEC 9899/AM1.
3	*isascii*( ) and *toascii*( ) function.	X		—	The functions should be specified as system interfaces.
4	*iswalnum*( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *fgetwc*( ), *fgetws*( ), *fputwc*( ), *fputws*( ), *getwc*( ), *getwchar*( ), *putwc*( ), *putwchar*( ), *towlower*( ), *towupper*( ), *ungetwc*( ), *wctype*( ), *wcscoll*( ), *wcsftime*( ), *wcstod*( ), *wcstok*( ), *wcstol*( ), *wcstoul*( ) and *wcsxfrm*( ) functions.	X		AM1	The functions are specified by ISO/IEC 9899/AM1.
5	*wcswidth*( ) and *wcwidth*( ) functions.	X		—	The functions should be specified as system interfaces.
6	Behaviour of *localconv*( ) function.	X		XPG4	The XPG4 specification is a reasonable extension to ISO C.
7	Behaviour of functions included in **<math.h>**.	X		—	+Inf, –Ing and NaN are ANSI/IEEE floating point representation-dependent descriptions. SPIRIT C should be independent of any bit representation.
8	Constants and external variables in **<math.h>**.	X		—	The macros are additional requirements to ISO C. They are not general requirements.
9	*erf*( ), *erfc*( ), *gamma*( ), *hypot*( ), *lgamma*( ) and external variables, *signgam*( ), *y0*( ), *y1*( ) and *yn*( ) functions.	X		—	The functions should be specified as system interfaces.

No.	Items	XPG4	MIA	SPIRIT	SPIRIT Rationale
10	%n$ format specification for *printf*( ) and *scanf*( ) family functions.	X		XPG4	This notation is a simple extension to ISO C and could contribute to improved portability of internationalised applications.
11	′ flag character for *printf*( ) family functions.	X		XPG4	This notation is a simple extension to ISO C and could contribute to improved portability of internationalised applications.
12	Infinity and NaN string representation for *printf*( ) family functions.	X		—	+Inf, –Inf and NaN are ANSI/IEEE floating point representation-dependent descriptions. SPIRIT C should be independent of any bit representation.
13	%C and %S conversion character for *printf*( ) and *scanf*( ) family functions.	X		AM1	Equivalent notations (that is, %wc and %ws) are specified by ISO/IEC 9899/AM1.
14	Parameter of *fseek*( ) function.	X		—	The description is a kind of guide for users of this function. It does not specify any behaviour of the function.
15	*memcopy*( ), *strfmon*( ), *nl_langinfo*( ), *iconv*( ) family, *catopen*( ), *catclose*( ) and *catgets*( ) functions.	X		—	The functions should be specified as system interfaces.
16	Additional macros in **<limits.h>**.	X		—	The macros should be specified as system interfaces.
17	Period of *rand*( ) function.	X		XPG4	The implementation limit could make sense and could contribute to application portability.
18	Requirement of full specification of ISO C *versus* Common Usage C.	X		—	XPG4 says that Common Usage C is obsolescent.
19	Conversion into signed integer type of smaller or equal size.		X	MIA	To specify ISO C implementation-defined behaviour could contribute to application portability.

No.	Items	XPG4	MIA	SPIRIT	SPIRIT Rationale
20	Result of the division (/, %) of negative integer.		X	MIA	To specify ISO C implementation-defined behaviour could contribute to application portability.
21	Result of the right shift when the first operand is a signed integer type and has a negative value.		X	MIA	To specify ISO C implementation-defined behaviour could contribute to application portability.
22	Behaviour of *freopen*( ) function.	X	X	—	MIA specification conflicts with both XPG4 and POSIX. On the other hand, specification of specific *errno* that is not specified by ISO C is out of scope.
23	Return value of the *fmod*( ) function when either argument is zero.	X	X	*1	Return value zero required by MIA is also allowed by XPG4. There is no description on *errno* in MIA. *1 Compromise between XPG4 and MIA; that is, returns zero and [EDOM] may be set to *errno*.
24	Addition of signals.	X	X	—	These macros should be specified as system interfaces. They are not general requirements.
25	Behaviour of the *rename*( ) function when there exists a file with the same name specified as a second argument.	X	X	—	The XPG4 and MIA C specifications are completely different. These specifications would not contribute to application portability.
26	Boundary alignment of bit-field.		X	—	MIA specification conflicts with ISO C when multiple 0 width bit-fields are declared continuously.
27	Data type of *wcschr*( ) and *wcsrchr*( ) functions.	X	X	AM1	The functions are specified by ISO/IEC 9899/AM1.
28	Number of significant initial characters in an external identifier.	X	X	MIA	The limit is an implementation limit of ISO C. Conditions for conforming implementations would contribute to improved application portability.

No.	Items	XPG4	MIA	SPIRIT	SPIRIT Rationale
29	Numerical value in **<limits.h>**(MB_LEN_MAX).	X	X	—	This implementation limit depends on the locale supported by a conforming implementation. It is not a general requirement.
30	Numerical value in **<float.h>**.	X	X	MIA	The limit is an implementation limit of ISO C. Conditions for conforming implementations would contribute to improved application portability.
31	*wcscpy*( ), *wcsncpy*( ), *wcscat*( ), *wcscmp*( ), *wcsncmp*( ), *wcschr*( ), *wcscspn*( ), *wcspbrk*( ), *wcsrchr*( ), *wcsspn*( ) and *wcslen*( ) functions.	X	X	AM1	The functions are specified by ISO/IEC 9899/AM1.
32	*wcswcs*( ) function.	X	X	AM1	Equivalent function (that is, the *wcsstr*( ) function) is specified by ISO/IEC 9899/AM1. Only the function name is different.