TECHNICAL REPORT

ISO/IEC TR 24731-1

First edition
2007-09-01

Information technology — Programming languages, their environments and system software interfaces — Extensions to the C library

Part 1: Bounds-checking interfaces

Technologies de l'information — Langages de programmation, leurs environnements et leurs systèmes d'interface de logiciel — Extensions à la bibliothèque C —

Partie 1: Interfaces des contrôles des bornes

Reference number
ISO/IEC TR 24731-1:2007(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 24731-1, which is a Technical Report of type 2, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

ISO/IEC 24731 consists of the following part, under the general title *Information technology — Programming languages, their environments and system software interfaces — Extensions to the C library*:

— *Part 1: Bounds-checking interfaces* [Technical Report]

Introduction

Traditionally, the C library has contained many functions that trust the programmer to provide output character arrays big enough to hold the result being produced. Not only do these functions not check that the arrays are big enough, they frequently lack the information needed to perform such checks. While it is possible to write safe, robust, and error-free code using the existing library, the library tends to promote programming styles that lead to mysterious failures if a result is too big for the provided array.

A common programming style is to declare character arrays large enough to handle most practical cases. However, if these arrays are not large enough to handle the resulting strings, data can be written past the end of the array overwriting other data and program structures. The program never gets any indication that a problem exists, and so never has a chance to recover or to fail gracefully.

Worse, this style of programming has compromised the security of computers and networks. Buffer overflows can often be exploited to run arbitrary code with the permissions of the vulnerable (defective) program.

If the programmer writes runtime checks to verify lengths before calling library functions, then those runtime checks frequently duplicate work done inside the library functions, which discover string lengths as a side effect of doing their job.

This Technical Report provides alternative functions for the C library that promote safer, more secure programming. The functions verify that output buffers are large enough for the intended result and return a failure indicator if they are not. Data is never written past the end of an array. All string results are null terminated.

This Technical Report also addresses another problem that complicates writing robust code: functions that are not re-entrant because they return pointers to static objects owned by the function. Such functions can be troublesome since a previously returned result can change if the function is called again, perhaps by another thread.

Information technology — Programming languages, their environments and system software interfaces — Extensions to the C library —

Part 1:
Bounds-checking interfaces

1. Scope

This Technical Report specifies a series of extensions of the programming language C, specified by International Standard ISO/IEC 9899:1999. These extensions can be useful in the mitigation of security vulnerabilities in programs, and consist of a new predefined macro, and new functions, macros, and types declared or defined in existing standard headers.

International Standard ISO/IEC 9899:1999 provides important context and specification for this Technical Report. Clauses 3 and 4 of this Technical Report are to be read as if they were merged into Clauses 3 and 4 of ISO/IEC 9899:1999. Clause 5 of this Technical Report is to be read as if it were merged into Subclause 6.10.8 of ISO/IEC 9899:1999. Clause 6 of this Technical Report is to be read as if it were merged into the parallel structure of named Subclauses of Clause 7 of ISO/IEC 9899:1999. Statements made in ISO/IEC 9899:1999, whether about the language or library, apply to this Technical Report unless a corresponding section of this Technical Report states otherwise. In particular, Subclause 7.1.4 ("Use of library functions") of ISO/IEC 9899:1999 applies to this Technical Report.

2. Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9899:1999, *Programming languages — C*

ISO/IEC 9899:1999/Cor.1:2001, *Programming languages — C — Technical Corrigendum 1*

ISO/IEC 9899:1999/Cor.2:2004, *Programming languages — C — Technical Corrigendum 2*

ISO 31−11:1992, *Quantities and units — Part 11: Mathematical signs and symbols for use in the physical sciences and technology*

ISO/IEC 2382−1:1993, *Information technology — Vocabulary — Part 1: Fundamental terms*

3. Terms, definitions, and symbols

For the purposes of this document, the terms and definitions given in ISO/IEC 9899:1999, ISO/IEC 2382−1, and the following apply. Other terms are defined where they appear in *italic* type. Mathematical symbols not defined in this Technical Report are to be interpreted according to ISO 31−11.

3.1

runtime-constraint

requirement on a program when calling a library function

NOTE 1 Despite the similar terms, a runtime-constraint is not a kind of constraint as defined by ISO/IEC 9899:1999, Subclause 3.8, and need not be diagnosed at translation time.

NOTE 2 Implementations verify that the runtime-constraints for a library function are not violated by the program. See Subclause 6.1.4.

4. Conformance

If a "shall" or "shall not" requirement that appears outside of a constraint or runtime-constraint is violated, the behavior is undefined.

5. Predefined macro names

The following macro name is conditionally defined by the implementation:

__STDC_LIB_EXT1__ The integer constant **200509L**, intended to indicate conformance to this Technical Report.[1)]

1) The intention is that this will remain an integer constant of type **long int** that is increased with each revision of this Technical Report.

6. Library

6.1 Introduction

6.1.1 Standard headers

The functions, macros, and types declared or defined in Clause 6 and its subclauses are not declared or defined by their respective headers if **__STDC_WANT_LIB_EXT1__** is defined as a macro which expands to the integer constant **0** at the point in the source file where the appropriate header is included.

The functions, macros, and types declared or defined in Clause 6 and its subclauses are declared and defined by their respective headers if **__STDC_WANT_LIB_EXT1__** is defined as a macro which expands to the integer constant **1** at the point in the source file where the appropriate header is included.[2)]

It is implementation-defined whether the functions, macros, and types declared or defined in Clause 6 and its subclauses are declared or defined by their respective headers if **__STDC_WANT_LIB_EXT1__** is not defined as a macro at the point in the source file where the appropriate header is included.[3)]

Within a preprocessing translation unit, **__STDC_WANT_LIB_EXT1__** shall be defined identically for all inclusions of any headers from Clause 6. If **__STDC_WANT_LIB_EXT1__** is defined differently for any such inclusion, the implementation shall issue a diagnostic as if a preprocessor error directive was used.

2) Future revisions of this Technical Report may define meanings for other values of **__STDC_WANT_LIB_EXT1__**.

3) Subclause 7.1.3 of ISO/IEC 9899:1999 reserves certain names and patterns of names that an implementation may use in headers. All other names are not reserved, and a conforming implementation may not use them. While some of the names defined in Clause 6 and its subclauses are reserved, others are not. If an unreserved name is defined in a header when **__STDC_WANT_LIB_EXT1__** is not defined, then the implementation is not conforming.

6.1.2 Reserved identifiers

Each macro name in any of the following subclauses is reserved for use as specified if it is defined by any of its associated headers when included; unless explicitly stated otherwise (see ISO/IEC 9899:1999 Subclause 7.1.4).

All identifiers with external linkage in any of the following subclauses are reserved for use as identifiers with external linkage if any of them are used by the program. None of them are reserved if none of them are used.

Each identifier with file scope listed in any of the following subclauses is reserved for use as a macro name and as an identifier with file scope in the same name space if it is defined by any of its associated headers when included.

6.1.3 Use of errno

An implementation may set **`errno`** for the functions defined in this Technical Report, but is not required to.

6.1.4 Runtime-constraint violations

Most functions in this Technical Report include as part of their specification a list of runtime-constraints. These runtime-constraints are requirements on the program using the library.[4)]

Implementations shall verify that the runtime-constraints for a function are not violated by the program. If a runtime-constraint is violated, the implementation shall call the currently registered runtime-constraint handler (see **`set_constraint_handler_s`** in **`<stdlib.h>`**). Multiple runtime-constraint violations in the same call to a library function result in only one call to the runtime-constraint handler. It is unspecified which one of the multiple runtime-constraint violations cause the handler to be called.

If the runtime-constraints section for a function states an action to be performed when a runtime-constraint violation occurs, the function shall perform the action before calling the runtime-constraint handler. If the runtime-constraints section lists actions that are prohibited when a runtime-constraint violation occurs, then such actions are prohibited to the function both before calling the handler and after the handler returns.

The runtime-constraint handler might not return. If the handler does return, the library function whose runtime-constraint was violated shall return some indication of failure as given by the returns section in the function's specification.

4) Although runtime-constraints replace many cases of undefined behavior from International Standard ISO/IEC 9899:1999, undefined behavior still exists in this Technical Report. Implementations are free to detect any case of undefined behavior and treat it as a runtime-constraint violation by calling the runtime-constraint handler. This license comes directly from the definition of undefined behavior.

6.2 Errors <errno.h>

The header **<errno.h>** defines a type.

The type is

errno_t

which is type **int**.[5)]

5) As a matter of programming style, **errno_t** may be used as the type of something that deals only with the values that might be found in **errno**. For example, a function which returns the value of **errno** might be declared as having the return type **errno_t**.

6.3 Common definitions <stddef.h>

The header **<stddef.h>** defines a type.

The type is

rsize_t

which is the type **size_t**.[6)]

6) See the description of the **RSIZE_MAX** macro in **<stdint.h>**.

6.4 Integer types <stdint.h>

The header **<stdint.h>** defines a macro.

The macro is

RSIZE_MAX

which expands to a value[7)] of type **size_t**. Functions that have parameters of type **rsize_t** consider it a runtime-constraint violation if the values of those parameters are greater than **RSIZE_MAX**.

Recommended practice

Extremely large object sizes are frequently a sign that an object's size was calculated incorrectly. For example, negative numbers appear as very large positive numbers when converted to an unsigned type like **size_t**. Also, some implementations do not support objects as large as the maximum value that can be represented by type **size_t**.

For those reasons, it is sometimes beneficial to restrict the range of object sizes to detect programming errors. For implementations targeting machines with large address spaces, it is recommended that **RSIZE_MAX** be defined as the smaller of the size of the largest object supported or **(SIZE_MAX >> 1)**, even if this limit is smaller than the size of some legitimate, but very large, objects. Implementations targeting machines with small address spaces may wish to define **RSIZE_MAX** as **SIZE_MAX**, which means that there is no object size that is considered a runtime-constraint violation.

7) The macro **RSIZE_MAX** need not expand to a constant expression.

6.5 Input/output <stdio.h>

The header **<stdio.h>** defines several macros and two types.

The macros are

L_tmpnam_s

which expands to an integer constant expression that is the size needed for an array of **char** large enough to hold a temporary file name string generated by the **tmpnam_s** function;

TMP_MAX_S

which expands to an integer constant expression that is the maximum number of unique file names that can be generated by the **tmpnam_s** function.

The types are

errno_t

which is type **int**; and

rsize_t

which is the type **size_t**.

6.5.1 Operations on files

6.5.1.1 The tmpfile_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
errno_t tmpfile_s(FILE * restrict * restrict streamptr);
```

Runtime-constraints

streamptr shall not be a null pointer.

If there is a runtime-constraint violation, **tmpfile_s** does not attempt to create a file.

Description

The **tmpfile_s** function creates a temporary binary file that is different from any other existing file and that will automatically be removed when it is closed or at program termination. If the program terminates abnormally, whether an open temporary file is removed is implementation-defined. The file is opened for update with **"wb+"** mode with the meaning that mode has in the **fopen_s** function (including the mode's effect on exclusive access and file permissions).

If the file was created successfully, then the pointer to **FILE** pointed to by **streamptr** will be set to the pointer to the object controlling the opened file. Otherwise, the pointer to **FILE** pointed to by **streamptr** will be set to a null pointer.

Recommended practice

It should be possible to open at least **TMP_MAX_S** temporary files during the lifetime of the program (this limit may be shared with **tmpnam_s**) and there should be no limit on the number simultaneously open other than this limit and any limit on the number of open files (**FOPEN_MAX**).

Returns

The **tmpfile_s** function returns zero if it created the file. If it did not create the file or there was a runtime-constraint violation, **tmpfile_s** returns a non-zero value.

6.5.1.2 The tmpnam_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
errno_t tmpnam_s(char *s, rsize_t maxsize);
```

Runtime-constraints

s shall not be a null pointer. **maxsize** shall be less than or equal to **RSIZE_MAX**. **maxsize** shall be greater than the length of the generated file name string.

Description

The **tmpnam_s** function generates a string that is a valid file name and that is not the same as the name of an existing file.[8)] The function is potentially capable of generating **TMP_MAX_S** different strings, but any or all of them may already be in use by existing files and thus not be suitable return values. The lengths of these strings shall be less than the value of the **L_tmpnam_s** macro.

The **tmpnam_s** function generates a different string each time it is called.

8) Files created using strings generated by the **tmpnam_s** function are temporary only in the sense that their names should not collide with those generated by conventional naming rules for the implementation. It is still necessary to use the **remove** function to remove such files when their use is ended, and before program termination. Implementations should take care in choosing the patterns used for names returned by **tmpnam_s**. For example, making a thread id part of the names avoids the race condition and possible conflict when multiple programs run simultaneously by the same user generate the same temporary file names.

It is assumed that **s** points to an array of at least **maxsize** characters. This array will be set to generated string, as specified below.

The implementation shall behave as if no library function except **tmpnam** calls the **tmpnam_s** function.[9)]

Recommended practice

After a program obtains a file name using the **tmpnam_s** function and before the program creates a file with that name, the possibility exists that someone else may create a file with that same name. To avoid this race condition, the **tmpfile_s** function should be used instead of **tmpnam_s** when possible. One situation that requires the use of the **tmpnam_s** function is when the program needs to create a temporary directory rather than a temporary file.

Returns

If no suitable string can be generated, or if there is a runtime-constraint violation, the **tmpnam_s** function writes a null character to **s[0]** (only if **s** is not null and **maxsize** is greater than zero) and returns a non-zero value.

Otherwise, the **tmpnam_s** function writes the string in the array pointed to by **s** and returns zero.

Environmental limits

The value of the macro **TMP_MAX_S** shall be at least 25.

6.5.2 File access functions

6.5.2.1 The fopen_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
errno_t fopen_s(FILE * restrict * restrict streamptr,
    const char * restrict filename,
    const char * restrict mode);
```

Runtime-constraints

None of **streamptr**, **filename**, or **mode** shall be a null pointer.

9) An implementation may have **tmpnam** call **tmpnam_s** (perhaps so there is only one naming convention for temporary files), but this is not required.

If there is a runtime-constraint violation, **fopen_s** does not attempt to open a file. Furthermore, if **streamptr** is not a null pointer, **fopen_s** sets ***streamptr** to the null pointer.

Description

The **fopen_s** function opens the file whose name is the string pointed to by **filename**, and associates a stream with it.

The **mode** string shall be as described for **fopen**, with the addition that modes starting with the character **'w'** or **'a'** may be preceded by the character **'u'**, see below:

uw	truncate to zero length or create text file for writing, default permissions
ua	append; open or create text file for writing at end-of-file, default permissions
uwb	truncate to zero length or create binary file for writing, default permissions
uab	append; open or create binary file for writing at end-of-file, default permissions
uw+	truncate to zero length or create text file for update, default permissions
ua+	append; open or create text file for update, writing at end-of-file, default permissions
uw+b *or* **uwb+**	truncate to zero length or create binary file for update, default permissions
ua+b *or* **uab+**	append; open or create binary file for update, writing at end-of-file, default permissions

To the extent that the underlying system supports the concepts, files opened for writing shall be opened with exclusive (also known as non-shared) access. If the file is being created, and the first character of the mode string is not **'u'**, to the extent that the underlying system supports it, the file shall have a file permission that prevents other users on the system from accessing the file. If the file is being created and first character of the mode string is **'u'**, then by the time the file has been closed, it shall have the system default file access permissions.[10)]

If the file was opened successfully, then the pointer to **FILE** pointed to by **streamptr** will be set to the pointer to the object controlling the opened file. Otherwise, the pointer to **FILE** pointed to by **streamptr** will be set to a null pointer.

Returns

The **fopen_s** function returns zero if it opened the file. If it did not open the file or if there was a runtime-constraint violation, **fopen_s** returns a non-zero value.

---

10) These are the same permissions that the file would have been created with by **fopen**.

6.5.2.2 The `freopen_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
errno_t freopen_s(FILE * restrict * restrict newstreamptr,
     const char * restrict filename,
     const char * restrict mode,
     FILE * restrict stream);
```

Runtime-constraints

None of **`newstreamptr`**, **`mode`**, and **`stream`** shall be a null pointer.

If there is a runtime-constraint violation, **`freopen_s`** neither attempts to close any file associated with **`stream`** nor attempts to open a file. Furthermore, if **`newstreamptr`** is not a null pointer, **`fopen_s`** sets **`*newstreamptr`** to the null pointer.

Description

The **`freopen_s`** function opens the file whose name is the string pointed to by **`filename`** and associates the stream pointed to by **`stream`** with it. The **`mode`** argument has the same meaning as in the **`fopen_s`** function (including the mode's effect on exclusive access and file permissions).

If **`filename`** is a null pointer, the **`freopen_s`** function attempts to change the mode of the stream to that specified by **`mode`**, as if the name of the file currently associated with the stream had been used. It is implementation-defined which changes of mode are permitted (if any), and under what circumstances.

The **`freopen_s`** function first attempts to close any file that is associated with **`stream`**. Failure to close the file is ignored. The error and end-of-file indicators for the stream are cleared.

If the file was opened successfully, then the pointer to **`FILE`** pointed to by **`newstreamptr`** will be set to the value of **`stream`**. Otherwise, the pointer to **`FILE`** pointed to by **`newstreamptr`** will be set to a null pointer.

Returns

The **`freopen_s`** function returns zero if it opened the file. If it did not open the file or there was a runtime-constraint violation, **`freopen_s`** returns a non-zero value.

6.5.3 Formatted input/output functions

Unless explicitly stated otherwise, if the execution of a function described in this subclause causes copying to take place between objects that overlap, the objects take on unspecified values.

6.5.3.1 The `fprintf_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int fprintf_s(FILE * restrict stream,
     const char * restrict format, ...);
```

Runtime-constraints

Neither **`stream`** nor **`format`** shall be a null pointer. The **`%n`** specifier[11] (modified or not by flags, field width, or precision) shall not appear in the string pointed to by **`format`**. Any argument to **`fprintf_s`** corresponding to a **`%s`** specifier shall not be a null pointer.

If there is a runtime-constraint violation,[12] the **`fprintf_s`** function does not attempt to produce further output, and it is unspecified to what extent **`fprintf_s`** produced output before discovering the runtime-constraint violation.

Description

The **`fprintf_s`** function is equivalent to the **`fprintf`** function except for the explicit runtime-constraints listed above.

Returns

The **`fprintf_s`** function returns the number of characters transmitted, or a negative value if an output error, encoding error, or runtime-constraint violation occurred.

---

11) It is not a runtime-constraint violation for the characters **`%n`** to appear in sequence in the string pointed at by **`format`** when those characters are not a interpreted as a **`%n`** specifier. For example, if the entire format string was **`%%n`**.

12) Because an implementation may treat any undefined behavior as a runtime-constraint violation, an implementation may treat any unsupported specifiers in the string pointed to by **`format`** as a runtime-constraint violation.

6.5.3.2 The `fscanf_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int fscanf_s(FILE * restrict stream,
     const char * restrict format, ...);
```

Runtime-constraints

Neither **`stream`** nor **`format`** shall be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation,[13] the **`fscanf_s`** function does not attempt to perform further input, and it is unspecified to what extent **`fscanf_s`** performed input before discovering the runtime-constraint violation.

Description

The **`fscanf_s`** function is equivalent to **`fscanf`** except that the **`c`**, **`s`**, and **`[`** conversion specifiers apply to a pair of arguments (unless assignment suppression is indicated by a **`*`**). The first of these arguments is the same as for **`fscanf`**. That argument is immediately followed in the argument list by the second argument, which has type **`rsize_t`** and gives the number of elements in the array pointed to by the first argument of the pair. If the first argument points to a scalar object, it is considered to be an array of one element.[14]

A matching failure occurs if the number of elements in a receiving object is insufficient to hold the converted input (including any trailing null character).

---

13) Because an implementation may treat any undefined behavior as a runtime-constraint violation, an implementation may treat any unsupported specifiers in the string pointed to by **`format`** as a runtime-constraint violation.

14) If the format is known at translation time, an implementation may issue a diagnostic for any argument used to store the result from a **`c`**, **`s`**, or **`[`** conversion specifier if that argument is not followed by an argument of a type compatible with **`rsize_t`**. A limited amount of checking may be done if even if the format is not known at translation time. For example, an implementation may issue a diagnostic for each argument after **`format`** that has of type pointer to one of **`char`**, **`signed char`**, **`unsigned char`**, or **`void`** that is not followed by an argument of a type compatible with **`rsize_t`**. The diagnostic could warn that unless the pointer is being used with a conversion specifier using the **`hh`** length modifier, a length argument must follow the pointer argument. Another useful diagnostic could flag any non-pointer argument following **`format`** that did not have a type compatible with **`rsize_t`**.

Returns

The **fscanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **fscanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

EXAMPLE 1 The call:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
/* ... */
int n, i; float x; char name[50];
n = fscanf_s(stdin, "%d%f%s", &i, &x, name, (rsize_t) 50);
```

with the input line:

```
25 54.32E-1 thompson
```

will assign to **n** the value 3, to **i** the value 25, to **x** the value 5.432, and to **name** the sequence **thompson\0**.

EXAMPLE 2 The call:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
/* ... */
int n; char s[5];
n = fscanf_s(stdin, "%s", s, sizeof s);
```

with the input line:

```
hello
```

will assign to **n** the value 0 since a matching failure occurred because the sequence **hello\0** requires an array of six characters to store it.

6.5.3.3 The printf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int printf_s(const char * restrict format, ...);
```

Runtime-constraints

format shall not be a null pointer. The **%n** specifier[15] (modified or not by flags, field width, or precision) shall not appear in the string pointed to by **format**. Any argument to **printf_s** corresponding to a **%s** specifier shall not be a null pointer.

15) It is not a runtime-constraint violation for the characters **%n** to appear in sequence in the string pointed at by **format** when those characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **%%n**.

If there is a runtime-constraint violation, the **printf_s** function does not attempt to produce further output, and it is unspecified to what extent **printf_s** produced output before discovering the runtime-constraint violation.

Description

The **printf_s** function is equivalent to the **printf** function except for the explicit runtime-constraints listed above.

Returns

The **printf_s** function returns the number of characters transmitted, or a negative value if an output error, encoding error, or runtime-constraint violation occurred.

6.5.3.4 The scanf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int scanf_s(const char * restrict format, ...);
```

Runtime-constraints

format shall not be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the **scanf_s** function does not attempt to perform further input, and it is unspecified to what extent **scanf_s** performed input before discovering the runtime-constraint violation.

Description

The **scanf_s** function is equivalent to **fscanf_s** with the argument **stdin** interposed before the arguments to **scanf_s**.

Returns

The **scanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **scanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

6.5.3.5 The snprintf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int snprintf_s(char * restrict s, rsize_t n,
     const char * restrict format, ...);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. **n** shall neither equal zero nor be greater than **RSIZE_MAX**. The **%n** specifier[16] (modified or not by flags, field width, or precision) shall not appear in the string pointed to by **format**. Any argument to **snprintf_s** corresponding to a **%s** specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if **s** is not a null pointer and **n** is greater than zero and less than **RSIZE_MAX**, then the **snprintf_s** function sets **s[0]** to the null character.

Description

The **snprintf_s** function is equivalent to the **snprintf** function except for the explicit runtime-constraints listed above.

The **snprintf_s** function, unlike **sprintf_s**, will truncate the result to fit within the array pointed to by **s**.

Returns

The **snprintf_s** function returns the number of characters that would have been written had **n** been sufficiently large, not counting the terminating null character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than **n**.

16) It is not a runtime-constraint violation for the characters **%n** to appear in sequence in the string pointed at by **format** when those characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **%%n**.

6.5.3.6 The `sprintf_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int sprintf_s(char * restrict s, rsize_t n,
     const char * restrict format, ...);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. **n** shall neither equal zero nor be greater than **RSIZE_MAX**. The number of characters (including the trailing null) required for the result to be written to the array pointed to by **s** shall not be greater than **n**. The **%n** specifier[17] (modified or not by flags, field width, or precision) shall not appear in the string pointed to by **format**. Any argument to **sprintf_s** corresponding to a **%s** specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if **s** is not a null pointer and **n** is greater than zero and less than **RSIZE_MAX**, then the **sprintf_s** function sets **s[0]** to the null character.

Description

The **sprintf_s** function is equivalent to the **sprintf** function except for the parameter **n** and the explicit runtime-constraints listed above.

The **sprintf_s** function, unlike **snprintf_s**, treats a result too big for the array pointed to by **s** as a runtime-constraint violation.

Returns

If no runtime-constraint violation occurred, the **sprintf_s** function returns the number of characters written in the array, not counting the terminating null character. If an encoding error occurred, **sprintf_s** returns a negative value. If any other runtime-constraint violation occurred, **sprintf_s** returns zero.

17) It is not a runtime-constraint violation for the characters **%n** to appear in sequence in the string pointed at by **format** when those characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **%%n**.

6.5.3.7 The `sscanf_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int sscanf_s(const char * restrict s,
     const char * restrict format, ...);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the **sscanf_s** function does not attempt to perform further input, and it is unspecified to what extent **sscanf_s** performed input before discovering the runtime-constraint violation.

Description

The **sscanf_s** function is equivalent to **fscanf_s**, except that input is obtained from a string (specified by the argument **s**) rather than from a stream. Reaching the end of the string is equivalent to encountering end-of-file for the **fscanf_s** function. If copying takes place between objects that overlap, the objects take on unspecified values.

Returns

The **sscanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **sscanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

6.5.3.8 The `vfprintf_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vfprintf_s(FILE * restrict stream,
     const char * restrict format,
     va_list arg);
```

Runtime-constraints

Neither **stream** nor **format** shall be a null pointer. The **%n** specifier[18] (modified or not by flags, field width, or precision) shall not appear in the string pointed to by **format**. Any argument to **vfprintf_s** corresponding to a **%s** specifier shall not be a null pointer.

If there is a runtime-constraint violation, the **vfprintf_s** function does not attempt to produce further output, and it is unspecified to what extent **vfprintf_s** produced output before discovering the runtime-constraint violation.

Description

The **vfprintf_s** function is equivalent to the **vfprintf** function except for the explicit runtime-constraints listed above.

Returns

The **vfprintf_s** function returns the number of characters transmitted, or a negative value if an output error, encoding error, or runtime-constraint violation occurred.

6.5.3.9 The vfscanf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vfscanf_s(FILE * restrict stream,
     const char * restrict format,
     va_list arg);
```

Runtime-constraints

Neither **stream** nor **format** shall be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the **vfscanf_s** function does not attempt to perform further input, and it is unspecified to what extent **vfscanf_s** performed input before discovering the runtime-constraint violation.

18) It is not a runtime-constraint violation for the characters **%n** to appear in sequence in the string pointed at by **format** when those characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **%%n**.

Description

The **vfscanf_s** function is equivalent to **fscanf_s**, with the variable argument list replaced by **arg**, which shall have been initialized by the **va_start** macro (and possibly subsequent **va_arg** calls). The **vfscanf_s** function does not invoke the **va_end** macro.[19)]

Returns

The **vfscanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **vfscanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

6.5.3.10 The vprintf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vprintf_s(const char * restrict format,
     va_list arg);
```

Runtime-constraints

format shall not be a null pointer. The **%n** specifier[20)] (modified or not by flags, field width, or precision) shall not appear in the string pointed to by **format**. Any argument to **vprintf_s** corresponding to a **%s** specifier shall not be a null pointer.

If there is a runtime-constraint violation, the **vprintf_s** function does not attempt to produce further output, and it is unspecified to what extent **vprintf_s** produced output before discovering the runtime-constraint violation.

Description

The **vprintf_s** function is equivalent to the **vprintf** function except for the explicit runtime-constraints listed above.

---

19) As the functions **vfprintf_s**, **vfscanf_s**, **vprintf_s**, **vscanf_s**, **vsnprintf_s**, **vsprintf_s**, and **vsscanf_s** invoke the **va_arg** macro, the value of **arg** after the return is indeterminate.

20) It is not a runtime-constraint violation for the characters **%n** to appear in sequence in the string pointed at by **format** when those characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **%%n**.

Returns

The **vprintf_s** function returns the number of characters transmitted, or a negative value if an output error, encoding error, or runtime-constraint violation occurred.

6.5.3.11 The vscanf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vscanf_s(const char * restrict format,
     va_list arg);
```

Runtime-constraints

format shall not be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the **vscanf_s** function does not attempt to perform further input, and it is unspecified to what extent **vscanf_s** performed input before discovering the runtime-constraint violation.

Description

The **vscanf_s** function is equivalent to **scanf_s**, with the variable argument list replaced by **arg**, which shall have been initialized by the **va_start** macro (and possibly subsequent **va_arg** calls). The **vscanf_s** function does not invoke the **va_end** macro.[21)]

Returns

The **vscanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **vscanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

21) As the functions **vfprintf_s**, **vfscanf_s**, **vprintf_s**, **vscanf_s**, **vsnprintf_s**, **vsprintf_s**, and **vsscanf_s** invoke the **va_arg** macro, the value of **arg** after the return is indeterminate.

6.5.3.12 The `vsnprintf_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vsnprintf_s(char * restrict s, rsize_t n,
     const char * restrict format,
     va_list arg);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. **n** shall neither equal zero nor be greater than **RSIZE_MAX**. The **%n** specifier[22] (modified or not by flags, field width, or precision) shall not appear in the string pointed to by **format**. Any argument to **vsnprintf_s** corresponding to a **%s** specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if **s** is not a null pointer and **n** is greater than zero and less than **RSIZE_MAX**, then the **vsnprintf_s** function sets **s[0]** to the null character.

Description

The **vsnprintf_s** function is equivalent to the **vsnprintf** function except for the explicit runtime-constraints listed above.

The **vsnprintf_s** function, unlike **vsprintf_s**, will truncate the result to fit within the array pointed to by **s**.

Returns

The **vsnprintf_s** function returns the number of characters that would have been written had **n** been sufficiently large, not counting the terminating null character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than **n**.

22) It is not a runtime-constraint violation for the characters **%n** to appear in sequence in the string pointed at by **format** when those characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **%%n**.

6.5.3.13 The `vsprintf_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vsprintf_s(char * restrict s, rsize_t n,
     const char * restrict format,
     va_list arg);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. **n** shall neither equal zero nor be greater than **RSIZE_MAX**. The number of characters (including the trailing null) required for the result to be written to the array pointed to by **s** shall not be greater than **n**. The **%n** specifier[23] (modified or not by flags, field width, or precision) shall not appear in the string pointed to by **format**. Any argument to **vsprintf_s** corresponding to a **%s** specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if **s** is not a null pointer and **n** is greater than zero and less than **RSIZE_MAX**, then the **vsprintf_s** function sets **s[0]** to the null character.

Description

The **vsprintf_s** function is equivalent to the **vsprintf** function except for the parameter **n** and the explicit runtime-constraints listed above.

The **vsprintf_s** function, unlike **vsnprintf_s**, treats a result too big for the array pointed to by **s** as a runtime-constraint violation.

Returns

If no runtime-constraint violation occurred, the **vsprintf_s** function returns the number of characters written in the array, not counting the terminating null character. If an encoding error occurred, **vsprintf_s** returns a negative value. If any other runtime-constraint violation occurred, **vsprintf_s** returns zero.

23) It is not a runtime-constraint violation for the characters **%n** to appear in sequence in the string pointed at by **format** when those characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **%%n**.

6.5.3.14 The vsscanf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vsscanf_s(const char * restrict s,
      const char * restrict format,
      va_list arg);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the **vsscanf_s** function does not attempt to perform further input, and it is unspecified to what extent **vsscanf_s** performed input before discovering the runtime-constraint violation.

Description

The **vsscanf_s** function is equivalent to **sscanf_s**, with the variable argument list replaced by **arg**, which shall have been initialized by the **va_start** macro (and possibly subsequent **va_arg** calls). The **vsscanf_s** function does not invoke the **va_end** macro.[24)]

Returns

The **vsscanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **vscanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

24) As the functions **vfprintf_s**, **vfscanf_s**, **vprintf_s**, **vscanf_s**, **vsnprintf_s**, **vsprintf_s**, and **vsscanf_s** invoke the **va_arg** macro, the value of **arg** after the return is indeterminate.

6.5.4 Character input/output functions

6.5.4.1 The `gets_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
char *gets_s(char *s, rsize_t n);
```

Runtime-constraints

s shall not be a null pointer. **n** shall neither be equal to zero nor be greater than **RSIZE_MAX**. A new-line character, end-of-file, or read error shall occur within reading **n-1** characters from **stdin**.[25)]

If there is a runtime-constraint violation, **s[0]** is set to the null character, and characters are read and discarded from **stdin** until a new-line character is read, or end-of-file or a read error occurs.

Description

The **gets_s** function reads at most one less than the number of characters specified by **n** from the stream pointed to by **stdin**, into the array pointed to by **s**. No additional characters are read after a new-line character (which is discarded) or after end-of-file. The discarded new-line character does not count towards number of characters read. A null character is written immediately after the last character read into the array.

If end-of-file is encountered and no characters have been read into the array, or if a read error occurs during the operation, then **s[0]** is set to the null character, and the other elements of **s** take unspecified values.

Recommended practice

The **fgets** function allows properly-written programs to safely process input lines too long to store in the result array. In general this requires that callers of **fgets** pay attention to the presence or absence of a new-line character in the result array. Consider using **fgets** (along with any needed processing based on new-line characters) instead of **gets_s**.

25) The **gets_s** function, unlike **gets**, makes it a runtime-constraint violation for a line of input to overflow the buffer to store it. Unlike **fgets**, **gets_s** maintains a one-to-one relationship between input lines and successful calls to **gets_s**. Programs that use **gets** expect such a relationship.

Returns

The **gets_s** function returns **s** if successful. If there was a runtime-constraint violation, or if end-of-file is encountered and no characters have been read into the array, or if a read error occurs during the operation, then a null pointer is returned.

6.6 General utilities <stdlib.h>

The header **<stdlib.h>** defines three types.

The types are

errno_t

which is type **int**; and

rsize_t

which is the type **size_t**; and

constraint_handler_t

which has the following definition

```
typedef void (*constraint_handler_t)(
     const char * restrict msg,
     void * restrict ptr,
     errno_t error);
```

6.6.1 Runtime-constraint handling

6.6.1.1 The set_constraint_handler_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
constraint_handler_t set_constraint_handler_s(
     constraint_handler_t handler);
```

Description

The **set_constraint_handler_s** function sets the runtime-constraint handler to be **handler**. The runtime-constraint handler is the function to be called when a library function detects a runtime-constraint violation. Only the most recent handler registered with **set_constraint_handler_s** is called when a runtime-constraint violation occurs.

When the handler is called, it is passed the following arguments in the following order:

1. A pointer to a character string describing the runtime-constraint violation.
2. A null pointer or a pointer to an implementation defined object.
3. If the function calling the handler has a return type declared as **errno_t**, the return value of the function is passed. Otherwise, a positive value of type **errno_t** is passed.

The implementation has a default constraint handler that is used if no calls to the **set_constraint_handler_s** function have been made. The behavior of the default handler is implementation-defined, and it may cause the program to exit or abort.

If the **handler** argument to **set_constraint_handler_s** is a null pointer, the implementation default handler becomes the current constraint handler.

Returns

The **set_constraint_handler_s** function returns a pointer to the previously registered handler.[26)]

6.6.1.2 The abort_handler_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
void abort_handler_s(
     const char * restrict msg,
     void * restrict ptr,
     errno_t error);
```

Description

A pointer to the **abort_handler_s** function shall be a suitable argument to the **set_constraint_handler_s** function.

The **abort_handler_s** function writes a message on the standard error stream in an implementation-defined format. The message shall include the string pointed to by **msg**. The **abort_handler_s** function then calls the **abort** function.[27)]

Returns

The **abort_handler_s** function does not return to its caller.

26) If the previous handler was registered by calling **set_constraint_handler_s** with a null pointer argument, a pointer to the implementation default handler is returned (not NULL).

27) Many implementations invoke a debugger when the **abort** function is called.

6.6.1.3 The `ignore_handler_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
void ignore_handler_s(
     const char * restrict msg,
     void * restrict ptr,
     errno_t error);
```

Description

A pointer to the **`ignore_handler_s`** function shall be a suitable argument to the **`set_constraint_handler_s`** function.

The **`ignore_handler_s`** function simply returns to its caller.[28)]

Returns

The **`ignore_handler_s`** function returns no value.

6.6.2 Communication with the environment

6.6.2.1 The `getenv_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t getenv_s(size_t * restrict len,
          char * restrict value, rsize_t maxsize,
          const char * restrict name);
```

Runtime-constraints

`name` shall not be a null pointer. **`maxsize`** shall neither equal zero nor be greater than **`RSIZE_MAX`**. If **`maxsize`** is not equal to zero, then **`value`** shall not be a null pointer.

If there is a runtime-constraint violation, the integer pointed to by **`len`** is set to 0 (if **`len`** is not null), and the environment list is not searched.

---

28) If the runtime-constraint handler is set to the **`ignore_handler_s`** function, any library function in which a runtime-constraint violation occurs will return to its caller. The caller can determine whether a runtime-constraint violation occurred based on the library function's specification (usually, the library function returns a non-zero **`errno_t`**).

Description

The **getenv_s** function searches an *environment list*, provided by the host environment, for a string that matches the string pointed to by **name**.

If that name is found then **getenv_s** performs the following actions. If **len** is not a null pointer, the length of the string associated with the matched list member is stored in the integer pointed to by **len**. If the length of the associated string is less than **maxsize**, then the associated string is copied to the array pointed to by **value**.

If that name is not found then **getenv_s** performs the following actions. If **len** is not a null pointer, zero is stored in the integer pointed to by **len**. If **maxsize** is greater than zero, then **value[0]** is set to the null character.

The set of environment names and the method for altering the environment list are implementation-defined.

Returns

The **getenv_s** function returns zero if the specified **name** is found and the associated string was successfully stored in **value**. Otherwise, a non-zero value is returned.

6.6.3 Searching and sorting utilities

These utilities make use of a comparison function to search or sort arrays of unspecified type. Where an argument declared as **size_t nmemb** specifies the length of the array for a function, if **nmemb** has the value zero on a call to that function, then the comparison function is not called, a search finds no matching element, sorting performs no rearrangement, and the pointer to the array may be null.

The implementation shall ensure that the second argument of the comparison function (when called from **bsearch_s**), or both arguments (when called from **qsort_s**), are pointers to elements of the array.[29] The first argument when called from **bsearch_s** shall equal **key**.

The comparison function shall not alter the contents of either the array or search key. The implementation may reorder elements of the array between calls to the comparison function, but shall not otherwise alter the contents of any individual element.

When the same objects (consisting of **size** bytes, irrespective of their current positions in the array) are passed more than once to the comparison function, the results shall be

---

29) That is, if the value passed is **p**, then the following expressions are always valid and nonzero:

```
((char *)p - (char *)base) % size == 0
(char *)p >= (char *)base
(char *)p < (char *)base + nmemb * size
```


consistent with one another. That is, for **qsort_s** they shall define a total ordering on the array, and for **bsearch_s** the same object shall always compare the same way with the key.

A sequence point occurs immediately before and immediately after each call to the comparison function, and also between any call to the comparison function and any movement of the objects passed as arguments to that call.

6.6.3.1 The bsearch_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
void *bsearch_s(const void *key, const void *base,
     rsize_t nmemb, rsize_t size,
     int (*compar)(const void *k, const void *y,
                   void *context),
     void *context);
```

Runtime-constraints

Neither **nmemb** nor **size** shall be greater than **RSIZE_MAX**. If **nmemb** is not equal to zero, then none of **key**, **base**, or **compar** shall be a null pointer.

If there is a runtime-constraint violation, the **bsearch_s** function does not search the array.

Description

The **bsearch_s** function searches an array of **nmemb** objects, the initial element of which is pointed to by **base**, for an element that matches the object pointed to by **key**. The size of each element of the array is specified by **size**.

The comparison function pointed to by **compar** is called with three arguments. The first two point to the **key** object and to an array element, in that order. The function shall return an integer less than, equal to, or greater than zero if the **key** object is considered, respectively, to be less than, to match, or to be greater than the array element. The array shall consist of: all the elements that compare less than, all the elements that compare equal to, and all the elements that compare greater than the **key** object, in that order.[30)]

30) In practice, this means that the entire array has been sorted according to the comparison function.

The third argument to the comparison function is the **context** argument passed to **bsearch_s**. The sole use of **context** by **bsearch_s** is to pass it to the comparison function.[31)]

Returns

The **bsearch_s** function returns a pointer to a matching element of the array, or a null pointer if no match is found or there is a runtime-constraint violation. If two elements compare as equal, which element is matched is unspecified.

6.6.3.2 The qsort_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t qsort_s(void *base, rsize_t nmemb, rsize_t size,
     int (*compar)(const void *x, const void *y,
                   void *context),
     void *context);
```

Runtime-constraints

Neither **nmemb** nor **size** shall be greater than **RSIZE_MAX**. If **nmemb** is not equal to zero, then neither **base** nor **compar** shall be a null pointer.

If there is a runtime-constraint violation, the **qsort_s** function does not sort the array.

Description

The **qsort_s** function sorts an array of **nmemb** objects, the initial element of which is pointed to by **base**. The size of each object is specified by **size**.

The contents of the array are sorted into ascending order according to a comparison function pointed to by **compar**, which is called with three arguments. The first two point to the objects being compared. The function shall return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. The third argument to the comparison function is the **context** argument passed to **qsort_s**. The sole use of **context** by **qsort_s** is to pass it to the comparison function.[32)]

If two elements compare as equal, their relative order in the resulting sorted array is unspecified.

---

31) The **context** argument is for the use of the comparison function in performing its duties. For example, it might specify a collating sequence used by the comparison function.

32) The **context** argument is for the use of the comparison function in performing its duties. For example, it might specify a collating sequence used by the comparison function.

Returns

The **qsort_s** function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

6.6.4 Multibyte/wide character conversion functions

The behavior of the multibyte character functions is affected by the **LC_CTYPE** category of the current locale. For a state-dependent encoding, each function is placed into its initial conversion state by a call for which its character pointer argument, **s**, is a null pointer. Subsequent calls with **s** as other than a null pointer cause the internal conversion state of the function to be altered as necessary. A call with **s** as a null pointer causes these functions to set the int pointed to by their **status** argument to a nonzero value if encodings have state dependency, and zero otherwise.[33] Changing the **LC_CTYPE** category causes the conversion state of these functions to be indeterminate.

6.6.4.1 The wctomb_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t wctomb_s(int * restrict status,
     char * restrict s,
     rsize_t smax,
     wchar_t wc);
```

Runtime-constraints

Let *n* denote the number of bytes needed to represent the multibyte character corresponding to the wide character given by **wc** (including any shift sequences).

If **s** is not a null pointer, then **smax** shall not be less than *n*, and **smax** shall not be greater than **RSIZE_MAX**. If **s** is a null pointer, then **smax** shall equal zero.

If there is a runtime-constraint violation, **wctomb_s** does not modify the int pointed to by **status**, and if **s** is not a null pointer, no more than **smax** elements in the array pointed to by **s** will be accessed.

Description

The **wctomb_s** function determines *n* and stores the multibyte character representation of **wc** in the array whose first element is pointed to by **s** (if **s** is not a null pointer). The number of characters stored never exceeds **MB_CUR_MAX** or **smax**. If **wc** is a null wide

33) If the locale employs special bytes to change the shift state, these bytes do not produce separate wide character codes, but are grouped with an adjacent multibyte character.

character, a null byte is stored, preceded by any shift sequence needed to restore the initial shift state, and the function is left in the initial conversion state.

The implementation shall behave as if no library function calls the **wctomb_s** function.

If **s** is a null pointer, the **wctomb_s** function stores into the int pointed to by **status** a nonzero or zero value, if multibyte character encodings, respectively, do or do not have state-dependent encodings.

If **s** is not a null pointer, the **wctomb_s** function stores into the int pointed to by **status** either *n* or −1 if **wc**, respectively, does or does not correspond to a valid multibyte character.

In no case will the int pointed to by **status** be set to a value greater than the **MB_CUR_MAX** macro.

Returns

The **wctomb_s** function returns zero if successful, and a non-zero value if there was a runtime-constraint violation or **wc** did not correspond to a valid multibyte character.

6.6.5 Multibyte/wide string conversion functions

The behavior of the multibyte string functions is affected by the **LC_CTYPE** category of the current locale.

6.6.5.1 The mbstowcs_s function

Synopsis

```
#include <stdlib.h>
errno_t mbstowcs_s(size_t * restrict retval,
     wchar_t * restrict dst, rsize_t dstmax,
     const char * restrict src, rsize_t len);
```

Runtime-constraints

Neither **retval** nor **src** shall be a null pointer. If **dst** is not a null pointer, then neither **len** nor **dstmax** shall be greater than **RSIZE_MAX**. If **dst** is a null pointer, then **dstmax** shall equal zero. If **dst** is not a null pointer, then **dstmax** shall not equal zero. If **dst** is not a null pointer and **len** is not less than **dstmax**, then a null character shall occur within the first **dstmax** multibyte characters of the array pointed to by **src**.

If there is a runtime-constraint violation, then **mbstowcs_s** does the following. If **retval** is not a null pointer, then **mbstowcs_s** sets ***retval** to **(size_t)(-1)**. If **dst** is not a null pointer and **dstmax** is greater than zero and less than **RSIZE_MAX**, then **mbstowcs_s** sets **dst[0]** to the null wide character.

Description

The **mbstowcs_s** function converts a sequence of multibyte characters that begins in the initial shift state from the array pointed to by **src** into a sequence of corresponding wide characters. If **dst** is not a null pointer, the converted characters are stored into the array pointed to by **dst**. Conversion continues up to and including a terminating null character, which is also stored. Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multibyte character, or (if **dst** is not a null pointer) when **len** wide characters have been stored into the array pointed to by **dst**.[34] If **dst** is not a null pointer and no null wide character was stored into the array pointed to by **dst**, then **dst[len]** is set to the null wide character. Each conversion takes place as if by a call to the **mbrtowc** function.

Regardless of whether **dst** is or is not a null pointer, if the input conversion encounters a sequence of bytes that do not form a valid multibyte character, an encoding error occurs: the **mbstowcs_s** function stores the value **(size_t)(-1)** into ***retval**. Otherwise, the **mbstowcs_s** function stores into ***retval** the number of multibyte characters successfully converted, not including the terminating null character (if any).

All elements following the terminating null wide character (if any) written by **mbstowcs_s** in the array of **dstmax** wide characters pointed to by **dst** take unspecified values when **mbstowcs_s** returns.[35]

If copying takes place between objects that overlap, the objects take on unspecified values.

Returns

The **mbstowcs_s** function returns zero if no runtime-constraint violation and no encoding error occurred. Otherwise, a non-zero value is returned.

6.6.5.2 The wcstombs_s function

Synopsis

```
#include <stdlib.h>
errno_t wcstombs_s(size_t * restrict retval,
    char * restrict dst, rsize_t dstmax,
    const wchar_t * restrict src, rsize_t len);
```


34) Thus, the value of **len** is ignored if **dst** is a null pointer.

35) This allows an implementation to attempt converting the multibyte string before discovering a terminating null character did not occur where required.

Runtime-constraints

Neither **retval** nor **src** shall be a null pointer. If **dst** is not a null pointer, then neither **len** nor **dstmax** shall be greater than **RSIZE_MAX**. If **dst** is a null pointer, then **dstmax** shall equal zero. If **dst** is not a null pointer, then **dstmax** shall not equal zero. If **dst** is not a null pointer and **len** is not less than **dstmax**, then the conversion shall have been stopped (see below) because a terminating null wide character was reached or because an encoding error occurred.

If there is a runtime-constraint violation, then **wcstombs_s** does the following. If **retval** is not a null pointer, then **wcstombs_s** sets ***retval** to **(size_t)(-1)**. If **dst** is not a null pointer and **dstmax** is greater than zero and less than **RSIZE_MAX**, then **wcstombs_s** sets **dst[0]** to the null character.

Description

The **wcstombs_s** function converts a sequence of wide characters from the array pointed to by **src** into a sequence of corresponding multibyte characters that begins in the initial shift state. If **dst** is not a null pointer, the converted characters are then stored into the array pointed to by **dst**. Conversion continues up to and including a terminating null wide character, which is also stored. Conversion stops earlier in two cases:

— when a wide character is reached that does not correspond to a valid multibyte character;

— (if **dst** is not a null pointer) when the next multibyte character would exceed the limit of *n* total bytes to be stored into the array pointed to by **dst**. If the wide character being converted is the null wide character, then *n* is the lesser of **len** or **dstmax**. Otherwise, *n* is the lesser of **len** or **dstmax-1**.

If the conversion stops without converting a null wide character and **dst** is not a null pointer, then a null character is stored into the array pointed to by **dst** immediately following any multibyte characters already stored. Each conversion takes place as if by a call to the **wcrtomb** function.[36)]

Regardless of whether **dst** is or is not a null pointer, if the input conversion encounters a wide character that does not correspond to a valid multibyte character, an encoding error occurs: the **wcstombs_s** function stores the value **(size_t)(-1)** into ***retval**. Otherwise, the **wcstombs_s** function stores into ***retval** the number of bytes in the resulting multibyte character sequence, not including the terminating null character (if any).

36) If conversion stops because a terminating null wide character has been reached, the bytes stored include those necessary to reach the initial shift state immediately before the null byte. However, if the conversion stops before a terminating null wide character has been reached, the result will be null terminated, but might not end in the initial shift state.

All elements following the terminating null character (if any) written by **wcstombs_s** in the array of **dstmax** elements pointed to by **dst** take unspecified values when **wcstombs_s** returns.[37)]

If copying takes place between objects that overlap, the objects take on unspecified values.

Returns

The **wcstombs_s** function returns zero if no runtime-constraint violation and no encoding error occurred. Otherwise, a non-zero value is returned.

37) When **len** is not less than **dstmax**, the implementation might fill the array before discovering a runtime-constraint violation.

6.7 String handling <string.h>

The header **<string.h>** defines two types.

The types are

errno_t

which is type **int**; and

rsize_t

which is the type **size_t**.

6.7.1 Copying functions

6.7.1.1 The memcpy_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t memcpy_s(void * restrict s1, rsize_t s1max,
     const void * restrict s2, rsize_t n);
```

Runtime-constraints

Neither **s1** nor **s2** shall be a null pointer. Neither **s1max** nor **n** shall be greater than **RSIZE_MAX**. **n** shall not be greater than **s1max**. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, the **memcpy_s** function stores zeros in the first **s1max** characters of the object pointed to by **s1** if **s1** is not a null pointer and **s1max** is not greater than **RSIZE_MAX**.

Description

The **memcpy_s** function copies **n** characters from the object pointed to by **s2** into the object pointed to by **s1**.

Returns

The **memcpy_s** function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

6.7.1.2 The memmove_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t memmove_s(void *s1, rsize_t s1max,
     const void *s2, rsize_t n);
```

Runtime-constraints

Neither **s1** nor **s2** shall be a null pointer. Neither **s1max** nor **n** shall be greater than **RSIZE_MAX**. **n** shall not be greater than **s1max**.

If there is a runtime-constraint violation, the **memmove_s** function stores zeros in the first **s1max** characters of the object pointed to by **s1** if **s1** is not a null pointer and **s1max** is not greater than **RSIZE_MAX**.

Description

The **memmove_s** function copies **n** characters from the object pointed to by **s2** into the object pointed to by **s1**. This copying takes place as if the **n** characters from the object pointed to by **s2** are first copied into a temporary array of **n** characters that does not overlap the objects pointed to by **s1** or **s2**, and then the **n** characters from the temporary array are copied into the object pointed to by **s1**.

Returns

The **memmove_s** function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

6.7.1.3 The strcpy_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strcpy_s(char * restrict s1,
     rsize_t s1max,
     const char * restrict s2);
```

Runtime-constraints

Neither **s1** nor **s2** shall be a null pointer. **s1max** shall not be greater than **RSIZE_MAX**. **s1max** shall not equal zero. **s1max** shall be greater than **strnlen_s(s2, s1max)**. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if **s1** is not a null pointer and **s1max** is greater than zero and not greater than **RSIZE_MAX**, then **strcpy_s** sets **s1[0]** to the null character.

Description

The **strcpy_s** function copies the string pointed to by **s2** (including the terminating null character) into the array pointed to by **s1**.

All elements following the terminating null character (if any) written by **strcpy_s** in the array of **s1max** characters pointed to by **s1** take unspecified values when **strcpy_s** returns.[38)]

Returns

The **strcpy_s** function returns zero[39)] if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

6.7.1.4 The strncpy_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strncpy_s(char * restrict s1,
      rsize_t s1max,
      const char * restrict s2,
      rsize_t n);
```

Runtime-constraints

Neither **s1** nor **s2** shall be a null pointer. Neither **s1max** nor **n** shall be greater than **RSIZE_MAX**. **s1max** shall not equal zero. If **n** is not less than **s1max**, then **s1max** shall be greater than **strnlen_s(s2, s1max)**. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if **s1** is not a null pointer and **s1max** is greater than zero and not greater than **RSIZE_MAX**, then **strncpy_s** sets **s1[0]** to the null character.

---

38) This allows an implementation to copy characters from **s2** to **s1** while simultaneously checking if any of those characters are null. Such an approach might write a character to every element of **s1** before discovering that the first element should be set to the null character.

39) A zero return value implies that all of the requested characters from the string pointed to by **s2** fit within the array pointed to by **s1** and that the result in **s1** is null terminated.

Description

The **strncpy_s** function copies not more than **n** successive characters (characters that follow a null character are not copied) from the array pointed to by **s2** to the array pointed to by **s1**. If no null character was copied from **s2**, then **s1[n]** is set to a null character.

All elements following the terminating null character (if any) written by **strncpy_s** in the array of **s1max** characters pointed to by **s1** take unspecified values when **strncpy_s** returns.[40)]

Returns

The **strncpy_s** function returns zero[41)] if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

EXAMPLE 1 The **strncpy_s** function can be used to copy a string without the danger that the result will not be null terminated or that characters will be written past the end of the destination array.

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
/* ... */
char src1[100] = "hello";
char src2[7] = {'g', 'o', 'o', 'd', 'b', 'y', 'e'};
char dst1[6], dst2[5], dst3[5];
int r1, r2, r3;
r1 = strncpy_s(dst1, 6, src1, 100);
r2 = strncpy_s(dst2, 5, src2, 7);
r3 = strncpy_s(dst3, 5, src2, 4);
```

The first call will assign to **r1** the value zero and to **dst1** the sequence **hello\0**.
The second call will assign to **r2** a non-zero value and to **dst2** the sequence **\0**.
The third call will assign to **r3** the value zero and to **dst3** the sequence **good\0**.

---

40) This allows an implementation to copy characters from **s2** to **s1** while simultaneously checking if any of those characters are null. Such an approach might write a character to every element of **s1** before discovering that the first element should be set to the null character.

41) A zero return value implies that all of the requested characters from the string pointed to by **s2** fit within the array pointed to by **s1** and that the result in **s1** is null terminated.

6.7.2 Concatenation functions

6.7.2.1 The `strcat_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strcat_s(char * restrict s1,
     rsize_t s1max,
     const char * restrict s2);
```

Runtime-constraints

Let *m* denote the value **`s1max - strnlen_s(s1, s1max)`** upon entry to **`strcat_s`**.

Neither **`s1`** nor **`s2`** shall be a null pointer. **`s1max`** shall not be greater than **`RSIZE_MAX`**. **`s1max`** shall not equal zero. *m* shall not equal zero.[42] *m* shall be greater than **`strnlen_s(s2,`** *m***`)`**. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if **`s1`** is not a null pointer and **`s1max`** is greater than zero and not greater than **`RSIZE_MAX`**, then **`strcat_s`** sets **`s1[0]`** to the null character.

Description

The **`strcat_s`** function appends a copy of the string pointed to by **`s2`** (including the terminating null character) to the end of the string pointed to by **`s1`**. The initial character from **`s2`** overwrites the null character at the end of **`s1`**.

All elements following the terminating null character (if any) written by **`strcat_s`** in the array of **`s1max`** characters pointed to by **`s1`** take unspecified values when **`strcat_s`** returns.[43]

Returns

The **`strcat_s`** function returns zero[44] if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

---

42) Zero means that **`s1`** was not null terminated upon entry to **`strcat_s`**.

43) This allows an implementation to append characters from **`s2`** to **`s1`** while simultaneously checking if any of those characters are null. Such an approach might write a character to every element of **`s1`** before discovering that the first element should be set to the null character.

44) A zero return value implies that all of the requested characters from the string pointed to by **`s2`** were appended to the string pointed to by **`s1`** and that the result in **`s1`** is null terminated.

6.7.2.2 The `strncat_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strncat_s(char * restrict s1,
      rsize_t s1max,
      const char * restrict s2,
      rsize_t n);
```

Runtime-constraints

Let *m* denote the value **`s1max - strnlen_s(s1, s1max)`** upon entry to **`strncat_s`**.

Neither **`s1`** nor **`s2`** shall be a null pointer. Neither **`s1max`** nor **`n`** shall be greater than **`RSIZE_MAX`**. **`s1max`** shall not equal zero. *m* shall not equal zero.[45] If **`n`** is not less than *m*, then *m* shall be greater than **`strnlen_s(s2, `***m***`)`**. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if **`s1`** is not a null pointer and **`s1max`** is greater than zero and not greater than **`RSIZE_MAX`**, then **`strncat_s`** sets **`s1[0]`** to the null character.

Description

The **`strncat_s`** function appends not more than **`n`** successive characters (characters that follow a null character are not copied) from the array pointed to by **`s2`** to the end of the string pointed to by **`s1`**. The initial character from **`s2`** overwrites the null character at the end of **`s1`**. If no null character was copied from **`s2`**, then **`s1[s1max-`***m***`+n]`** is set to a null character.

All elements following the terminating null character (if any) written by **`strncat_s`** in the array of **`s1max`** characters pointed to by **`s1`** take unspecified values when **`strncat_s`** returns.[46]

45) Zero means that **`s1`** was not null terminated upon entry to **`strncat_s`**.

46) This allows an implementation to append characters from **`s2`** to **`s1`** while simultaneously checking if any of those characters are null. Such an approach might write a character to every element of **`s1`** before discovering that the first element should be set to the null character.

Returns

The **strncat_s** function returns zero[47] if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

EXAMPLE 1 The **strncat_s** function can be used to copy a string without the danger that the result will not be null terminated or that characters will be written past the end of the destination array.

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
/* ... */
char s1[100] = "good";
char s2[6] = "hello";
char s3[6] = "hello";
char s4[7] = "abc";
char s5[1000] = "bye";
int r1, r2, r3, r4;
r1 = strncat_s(s1, 100, s5, 1000);
r2 = strncat_s(s2, 6, "", 1);
r3 = strncat_s(s3, 6, "X", 2);
r4 = strncat_s(s4, 7, "defghijklmn", 3);
```

After the first call **r1** will have the value zero and **s1** will contain the sequence **goodbye\0**.
After the second call **r2** will have the value zero and **s2** will contain the sequence **hello\0**.
After the third call **r3** will have a non-zero value and **s3** will contain the sequence **\0**.
After the fourth call **r4** will have the value zero and **s4** will contain the sequence **abcdef\0**.

6.7.3 Search functions

6.7.3.1 The strtok_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
char *strtok_s(char * restrict s1,
     rsize_t * restrict s1max,
     const char * restrict s2,
     char ** restrict ptr);
```

Runtime-constraints

None of **s1max**, **s2**, or **ptr** shall be a null pointer. If **s1** is a null pointer, then ***ptr** shall not be a null pointer. The value of ***s1max** shall not be greater than **RSIZE_MAX**. The end of the token found shall occur within the first ***s1max** characters of **s1** for the first call, and shall occur within the first ***s1max** characters of where searching resumes on subsequent calls.

If there is a runtime-constraint violation, the **strtok_s** function does not indirect through the **s1** or **s2** pointers, and does not store a value in the object pointed to by **ptr**.

47) A zero return value implies that all of the requested characters from the string pointed to by **s2** were appended to the string pointed to by **s1** and that the result in **s1** is null terminated.

Description

A sequence of calls to the **strtok_s** function breaks the string pointed to by **s1** into a sequence of tokens, each of which is delimited by a character from the string pointed to by **s2**. The fourth argument points to a caller-provided **char** pointer into which the **strtok_s** function stores information necessary for it to continue scanning the same string.

The first call in a sequence has a non-null first argument and **s1max** points to an object whose value is the number of elements in the character array pointed to by the first argument. The first call stores an initial value in the object pointed to by **ptr** and updates the value pointed to by **s1max** to reflect the number of elements that remain in relation to **ptr**. Subsequent calls in the sequence have a null first argument and the objects pointed to by **s1max** and **ptr** are required to have the values stored by the previous call in the sequence, which are then updated. The separator string pointed to by **s2** may be different from call to call.

The first call in the sequence searches the string pointed to by **s1** for the first character that is *not* contained in the current separator string pointed to by **s2**. If no such character is found, then there are no tokens in the string pointed to by **s1** and the **strtok_s** function returns a null pointer. If such a character is found, it is the start of the first token.

The **strtok_s** function then searches from there for the first character in **s1** that *is* contained in the current separator string. If no such character is found, the current token extends to the end of the string pointed to by **s1**, and subsequent searches in the same string for a token return a null pointer. If such a character is found, it is overwritten by a null character, which terminates the current token.

In all cases, the **strtok_s** function stores sufficient information in the pointer pointed to by **ptr** so that subsequent calls, with a null pointer for **s1** and the unmodified pointer value for **ptr**, shall start searching just past the element overwritten by a null character (if any).

Returns

The **strtok_s** function returns a pointer to the first character of a token, or a null pointer if there is no token or there is a runtime-constraint violation.

EXAMPLE

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
static char str1[] = "?a???b,,,#c";
static char str2[] = "\t \t";
char *t, *ptr1, *ptr2;
rsize_t max1 = sizeof(str1);
rsize_t max2 = sizeof(str2);
```



```
t = strtok_s(str1, &max1, "?", &ptr1);     // t points to the token "a"
t = strtok_s(NULL, &max1, ",", &ptr1);     // t points to the token "??b"
t = strtok_s(str2, &max2, " \t", &ptr2);   // t is a null pointer
t = strtok_s(NULL, &max1, "#,", &ptr1);    // t points to the token "c"
t = strtok_s(NULL, &max1, "?", &ptr1);     // t is a null pointer
```

6.7.4 Miscellaneous functions

6.7.4.1 The `strerror_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strerror_s(char *s, rsize_t maxsize,
     errno_t errnum);
```

Runtime-constraints

s shall not be a null pointer. **maxsize** shall not be greater than **RSIZE_MAX**. **maxsize** shall not equal zero.

If there is a runtime-constraint violation, then the array (if any) pointed to by **s** is not modified.

Description

The **strerror_s** function maps the number in **errnum** to a locale-specific message string. Typically, the values for **errnum** come from **errno**, but **strerror_s** shall map any value of type **int** to a message.

If the length of the desired string is less than **maxsize**, then the string is copied to the array pointed to by **s**.

Otherwise, if **maxsize** is greater than zero, then **maxsize-1** characters are copied from the string to the array pointed to by **s** and then **s[maxsize-1]** is set to the null character. Then, if **maxsize** is greater than 3, then **s[maxsize-2]**, **s[maxsize-3]**, and **s[maxsize-4]** are set to the character period (**.**).

Returns

The **strerror_s** function returns zero if the length of the desired string was less than **maxsize** and there was no runtime-constraint violation. Otherwise, the **strerror_s** function returns a non-zero value.

6.7.4.2 The `strerrorlen_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
size_t strerrorlen_s(errno_t errnum);
```

Description

The **`strerrorlen_s`** function calculates the length of the (untruncated) locale-specific message string that the **`strerror_s`** function maps to **`errnum`**.

Returns

The **`strerrorlen_s`** function returns the number of characters (not including the null character) in the full message string.

6.7.4.3 The `strnlen_s` function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
size_t strnlen_s(const char *s, size_t maxsize);
```

Description

The **`strnlen_s`** function computes the length of the string pointed to by **`s`**.

Returns

If **`s`** is a null pointer,[48)] then the **`strnlen_s`** function returns zero.

Otherwise, the **`strnlen_s`** function returns the number of characters that precede the terminating null character. If there is no null character in the first **`maxsize`** characters of **`s`** then **`strnlen_s`** returns **`maxsize`**. At most the first **`maxsize`** characters of **`s`** shall be accessed by **`strnlen_s`**.

48) Note that the **`strnlen_s`** function has no runtime-constraints. This lack of runtime-constraints along with the values returned for a null pointer or an unterminated string argument make **`strnlen_s`** useful in algorithms that gracefully handle such exceptional data.

6.8 Date and time <time.h>

The header **<time.h>** defines two types.

The types are

errno_t

which is type **int**; and

rsize_t

which is the type **size_t**.

6.8.1 Components of time

A broken-down time is *normalized* if the values of the members of the **tm** structure are in their normal rages.[49)]

6.8.2 Time conversion functions

Like the **strftime** function, the **asctime_s** and **ctime_s** functions do not return a pointer to a static object, and other library functions are permitted to call them.

6.8.2.1 The asctime_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
errno_t asctime_s(char *s, rsize_t maxsize,
     const struct tm *timeptr);
```

Runtime-constraints

Neither **s** nor **timeptr** shall be a null pointer. **maxsize** shall not be less than 26 and shall not be greater than **RSIZE_MAX**. The broken-down time pointed to by **timeptr** shall be normalized. The calendar year represented by the broken-down time pointed to by **timeptr** shall not be less than calendar year 0 and shall not be greater than calendar year 9999.

If there is a runtime-constraint violation, there is no attempt to convert the time, and **s[0]** is set to a null character if **s** is not a null pointer and **maxsize** is not zero and is not greater than **RSIZE_MAX**.

49) The normal ranges are defined in Subclause 7.23.1 of ISO/IEC 9899:1999.

Description

The **asctime_s** function converts the normalized broken-down time in the structure pointed to by **timeptr** into a 26 character (including the null character) string in the form

```
Sun Sep 16 01:03:52 1973\n\0
```

The fields making up this string are (in order):

1. The name of the day of the week represented by **timeptr->tm_wday** using the following three character weekday names: Sun, Mon, Tue, Wed, Thu, Fri, and Sat.
2. The character space.
3. The name of the month represented by **timeptr->tm_mon** using the following three character month names: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, and Dec.
4. The character space.
5. The value of **timeptr->tm_mday** as if printed using the **fprintf** format **"%2d"**.
6. The character space.
7. The value of **timeptr->tm_hour** as if printed using the **fprintf** format **"%.2d"**.
8. The character colon.
9. The value of **timeptr->tm_min** as if printed using the **fprintf** format **"%.2d"**.
10. The character colon.
11. The value of **timeptr->tm_sec** as if printed using the **fprintf** format **"%.2d"**.
12. The character space.
13. The value of **timeptr->tm_year + 1900** as if printed using the **fprintf** format **"%4d"**.
14. The character new line.
15. The null character.

Recommended practice

The **strftime** function allows more flexible formatting and supports locale-specific behavior. If you do not require the exact form of the result string produced by the **asctime_s** function, consider using the **strftime** function instead.

Returns

The **asctime_s** function returns zero if the time was successfully converted and stored into the array pointed to by **s**. Otherwise, it returns a non-zero value.

6.8.2.2 The ctime_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
errno_t ctime_s(char *s, rsize_t maxsize,
     const time_t *timer);
```

Runtime-constraints

Neither **s** nor **timer** shall be a null pointer. **maxsize** shall not be less than 26 and shall not be greater than **RSIZE_MAX**.

If there is a runtime-constraint violation, **s[0]** is set to a null character if **s** is not a null pointer and **maxsize** is not equal zero and is not greater than **RSIZE_MAX**.

Description

The **ctime_s** function converts the calendar time pointed to by **timer** to local time in the form of a string. It is equivalent to

```
asctime_s(s, maxsize, localtime_s(timer))
```

Recommended practice

The **strftime** function allows more flexible formatting and supports locale-specific behavior. If you do not require the exact form of the result string produced by the **ctime_s** function, consider using the **strftime** function instead.

Returns

The **ctime_s** function returns zero if the time was successfully converted and stored into the array pointed to by **s**. Otherwise, it returns a non-zero value.

6.8.2.3 The gmtime_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
struct tm *gmtime_s(const time_t * restrict timer,
     struct tm * restrict result);
```


Runtime-constraints

Neither **timer** nor **result** shall be a null pointer.

If there is a runtime-constraint violation, there is no attempt to convert the time.

Description

The **gmtime_s** function converts the calendar time pointed to by **timer** into a broken-down time, expressed as UTC. The broken-down time is stored in the structure pointed to by **result**.

Returns

The **gmtime_s** function returns **result**, or a null pointer if the specified time cannot be converted to UTC or there is a runtime-constraint violation.

6.8.2.4 The localtime_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
struct tm *localtime_s(const time_t * restrict timer,
     struct tm * restrict result);
```

Runtime-constraints

Neither **timer** nor **result** shall be a null pointer.

If there is a runtime-constraint violation, there is no attempt to convert the time.

Description

The **localtime_s** function converts the calendar time pointed to by **timer** into a broken-down time, expressed as local time. The broken-down time is stored in the structure pointed to by **result**.

Returns

The **localtime_s** function returns **result**, or a null pointer if the specified time cannot be converted to local time or there is a runtime-constraint violation.

6.9 Extended multibyte and wide character utilities <wchar.h>

The header **<wchar.h>** defines two types.

The types are

errno_t

which is type **int**; and

rsize_t

which is the type **size_t**.

Unless explicitly stated otherwise, if the execution of a function described in this subclause causes copying to take place between objects that overlap, the objects take on unspecified values.

6.9.1 Formatted wide character input/output functions

6.9.1.1 The fwprintf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <wchar.h>
int fwprintf_s(FILE * restrict stream,
     const wchar_t * restrict format, ...);
```

Runtime-constraints

Neither **stream** nor **format** shall be a null pointer. The **%n** specifier[50] (modified or not by flags, field width, or precision) shall not appear in the wide string pointed to by **format**. Any argument to **fwprintf_s** corresponding to a **%s** specifier shall not be a null pointer.

If there is a runtime-constraint violation, the **fwprintf_s** function does not attempt to produce further output, and it is unspecified to what extent **fwprintf_s** produced output before discovering the runtime-constraint violation.

Description

The **fwprintf_s** function is equivalent to the **fwprintf** function except for the explicit runtime-constraints listed above.

---

50) It is not a runtime-constraint violation for the wide characters **%n** to appear in sequence in the wide string pointed at by **format** when those wide characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **L"%%n"**.

Returns

The **fwprintf_s** function returns the number of wide characters transmitted, or a negative value if an output error, encoding error, or runtime-constraint violation occurred.

6.9.1.2 The fwscanf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <wchar.h>
int fwscanf_s(FILE * restrict stream,
     const wchar_t * restrict format, ...);
```

Runtime-constraints

Neither **stream** nor **format** shall be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the **fwscanf_s** function does not attempt to perform further input, and it is unspecified to what extent **fwscanf_s** performed input before discovering the runtime-constraint violation.

Description

The **fwscanf_s** function is equivalent to **fwscanf** except that the **c**, **s**, and **[** conversion specifiers apply to a pair of arguments (unless assignment suppression is indicated by a *****). The first of these arguments is the same as for **fwscanf**. That argument is immediately followed in the argument list by the second argument, which has type **size_t** and gives the number of elements in the array pointed to by the first argument of the pair. If the first argument points to a scalar object, it is considered to be an array of one element.[51)]

A matching failure occurs if the number of elements in a receiving object is insufficient to hold the converted input (including any trailing null character).

---

51) If the format is known at translation time, an implementation may issue a diagnostic for any argument used to store the result from a **c**, **s**, or **[** conversion specifier if that argument is not followed by an argument of a type compatible with **rsize_t**. A limited amount of checking may be done if even if the format is not known at translation time. For example, an implementation may issue a diagnostic for each argument after **format** that has of type pointer to one of **char**, **signed char**, **unsigned char**, or **void** that is not followed by an argument of a type compatible with **rsize_t**. The diagnostic could warn that unless the pointer is being used with a conversion specifier using the **hh** length modifier, a length argument must follow the pointer argument. Another useful diagnostic could flag any non-pointer argument following **format** that did not have a type compatible with **rsize_t**.

Returns

The **fwscanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **fwscanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

6.9.1.3 The snwprintf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <wchar.h>
int snwprintf_s(wchar_t * restrict s,
     rsize_t n,
     const wchar_t * restrict format, ...);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. **n** shall neither equal zero nor be greater than **RSIZE_MAX**. The **%n** specifier[52] (modified or not by flags, field width, or precision) shall not appear in the wide string pointed to by **format**. Any argument to **snwprintf_s** corresponding to a **%s** specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if **s** is not a null pointer and **n** is greater than zero and less than **RSIZE_MAX**, then the **snwprintf_s** function sets **s[0]** to the null wide character.

Description

The **snwprintf_s** function is equivalent to the **swprintf** function except for the explicit runtime-constraints listed above.

The **snwprintf_s** function, unlike **swprintf_s**, will truncate the result to fit within the array pointed to by **s**.

Returns

The **snwprintf_s** function returns the number of wide characters that would have been written had **n** been sufficiently large, not counting the terminating wide null

52) It is not a runtime-constraint violation for the wide characters **%n** to appear in sequence in the wide string pointed at by **format** when those wide characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **L"%%n"**.

character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than **n**.

6.9.1.4 The swprintf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <wchar.h>
int swprintf_s(wchar_t * restrict s, rsize_t n,
     const wchar_t * restrict format, ...);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. **n** shall neither equal zero nor be greater than **RSIZE_MAX**. The number of wide characters (including the trailing null) required for the result to be written to the array pointed to by **s** shall not be greater than **n**. The **%n** specifier[53] (modified or not by flags, field width, or precision) shall not appear in the wide string pointed to by **format**. Any argument to **swprintf_s** corresponding to a **%s** specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if **s** is not a null pointer and **n** is greater than zero and less than **RSIZE_MAX**, then the **swprintf_s** function sets **s[0]** to the null wide character.

Description

The **swprintf_s** function is equivalent to the **swprintf** function except for the explicit runtime-constraints listed above.

The **swprintf_s** function, unlike **snwprintf_s**, treats a result too big for the array pointed to by **s** as a runtime-constraint violation.

Returns

If no runtime-constraint violation occurred, the **swprintf_s** function returns the number of wide characters written in the array, not counting the terminating null wide character. If an encoding error occurred or if **n** or more wide characters are requested to be written, **swprintf_s** returns a negative value. If any other runtime-constraint violation occurred, **swprintf_s** returns zero.

---

53) It is not a runtime-constraint violation for the wide characters **%n** to appear in sequence in the wide string pointed at by **format** when those wide characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **L"%%n"**.

6.9.1.5 The swscanf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <wchar.h>
int swscanf_s(const wchar_t * restrict s,
     const wchar_t * restrict format, ...);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the **swscanf_s** function does not attempt to perform further input, and it is unspecified to what extent **swscanf_s** performed input before discovering the runtime-constraint violation.

Description

The **swscanf_s** function is equivalent to **fwscanf_s**, except that the argument **s** specifies a wide string from which the input is to be obtained, rather than from a stream. Reaching the end of the wide string is equivalent to encountering end-of-file for the **fwscanf_s** function.

Returns

The **swscanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **swscanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

6.9.1.6 The vfwprintf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
int vfwprintf_s(FILE * restrict stream,
     const wchar_t * restrict format,
     va_list arg);
```


Runtime-constraints

Neither **stream** nor **format** shall be a null pointer. The **%n** specifier[54] (modified or not by flags, field width, or precision) shall not appear in the wide string pointed to by **format**. Any argument to **vfwprintf_s** corresponding to a **%s** specifier shall not be a null pointer.

If there is a runtime-constraint violation, the **vfwprintf_s** function does not attempt to produce further output, and it is unspecified to what extent **vfwprintf_s** produced output before discovering the runtime-constraint violation.

Description

The **vfwprintf_s** function is equivalent to the **vfwprintf** function except for the explicit runtime-constraints listed above.

Returns

The **vfwprintf_s** function returns the number of wide characters transmitted, or a negative value if an output error, encoding error, or runtime-constraint violation occurred.

6.9.1.7 The vfwscanf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
int vfwscanf_s(FILE * restrict stream,
     const wchar_t * restrict format, va_list arg);
```

Runtime-constraints

Neither **stream** nor **format** shall be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the **vfwscanf_s** function does not attempt to perform further input, and it is unspecified to what extent **vfwscanf_s** performed input before discovering the runtime-constraint violation.

54) It is not a runtime-constraint violation for the wide characters **%n** to appear in sequence in the wide string pointed at by **format** when those wide characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **L"%%n"**.

Description

The **vfwscanf_s** function is equivalent to **fwscanf_s**, with the variable argument list replaced by **arg**, which shall have been initialized by the **va_start** macro (and possibly subsequent **va_arg** calls). The **vfwscanf_s** function does not invoke the **va_end** macro.[55)]

Returns

The **vfwscanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **vfwscanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

6.9.1.8 The vsnwprintf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <wchar.h>
int vsnwprintf_s(wchar_t * restrict s,
     rsize_t n,
     const wchar_t * restrict format,
     va_list arg);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. **n** shall neither equal zero nor be greater than **RSIZE_MAX**. The **%n** specifier[56)] (modified or not by flags, field width, or precision) shall not appear in the wide string pointed to by **format**. Any argument to **vsnwprintf_s** corresponding to a **%s** specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if **s** is not a null pointer and **n** is greater than zero and less than **RSIZE_MAX**, then the **vsnwprintf_s** function sets **s[0]** to the null wide character.

Description

The **vsnwprintf_s** function is equivalent to the **vswprintf** function except for the explicit runtime-constraints listed above.

---

55) As the functions **vfwscanf_s**, **vwscanf_s**, and **vswscanf_s** invoke the **va_arg** macro, the value of **arg** after the return is indeterminate.

56) It is not a runtime-constraint violation for the wide characters **%n** to appear in sequence in the wide string pointed at by **format** when those wide characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **L"%%n"**.

The **vsnwprintf_s** function, unlike **vswprintf_s**, will truncate the result to fit within the array pointed to by **s**.

Returns

The **vsnwprintf_s** function returns the number of wide characters that would have been written had **n** been sufficiently large, not counting the terminating null character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than **n**.

6.9.1.9 The vswprintf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <wchar.h>
int vswprintf_s(wchar_t * restrict s,
     rsize_t n,
     const wchar_t * restrict format,
     va_list arg);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. **n** shall neither equal zero nor be greater than **RSIZE_MAX**. The number of wide characters (including the trailing null) required for the result to be written to the array pointed to by **s** shall not be greater than **n**. The **%n** specifier[57] (modified or not by flags, field width, or precision) shall not appear in the wide string pointed to by **format**. Any argument to **vswprintf_s** corresponding to a **%s** specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if **s** is not a null pointer and **n** is greater than zero and less than **RSIZE_MAX**, then the **vswprintf_s** function sets **s[0]** to the null wide character.

Description

The **vswprintf_s** function is equivalent to the **vswprintf** function except for the explicit runtime-constraints listed above.

The **vswprintf_s** function, unlike **vsnwprintf_s**, treats a result too big for the array pointed to by **s** as a runtime-constraint violation.

57) It is not a runtime-constraint violation for the wide characters **%n** to appear in sequence in the wide string pointed at by **format** when those wide characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **L"%%n"**.

Returns

If no runtime-constraint violation occurred, the **vswprintf_s** function returns the number of wide characters written in the array, not counting the terminating null wide character. If an encoding error occurred or if **n** or more wide characters are requested to be written, **vswprintf_s** returns a negative value. If any other runtime-constraint violation occurred, **vswprintf_s** returns zero.

6.9.1.10 The vswscanf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <wchar.h>
int vswscanf_s(const wchar_t * restrict s,
     const wchar_t * restrict format,
     va_list arg);
```

Runtime-constraints

Neither **s** nor **format** shall be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the **vswscanf_s** function does not attempt to perform further input, and it is unspecified to what extent **vswscanf_s** performed input before discovering the runtime-constraint violation.

Description

The **vswscanf_s** function is equivalent to **swscanf_s**, with the variable argument list replaced by **arg**, which shall have been initialized by the **va_start** macro (and possibly subsequent **va_arg** calls). The **vswscanf_s** function does not invoke the **va_end** macro.[58)]

Returns

The **vswscanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **vswscanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

---

58) As the functions **vfwscanf_s**, **vwscanf_s**, and **vswscanf_s** invoke the **va_arg** macro, the value of **arg** after the return is indeterminate.

6.9.1.11 The vwprintf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <wchar.h>
int vwprintf_s(const wchar_t * restrict format,
     va_list arg);
```

Runtime-constraints

format shall not be a null pointer. The **%n** specifier[59] (modified or not by flags, field width, or precision) shall not appear in the wide string pointed to by **format**. Any argument to **vwprintf_s** corresponding to a **%s** specifier shall not be a null pointer.

If there is a runtime-constraint violation, the **vwprintf_s** function does not attempt to produce further output, and it is unspecified to what extent **vwprintf_s** produced output before discovering the runtime-constraint violation.

Description

The **vwprintf_s** function is equivalent to the **vwprintf** function except for the explicit runtime-constraints listed above.

Returns

The **vwprintf_s** function returns the number of wide characters transmitted, or a negative value if an output error, encoding error, or runtime-constraint violation occurred.

6.9.1.12 The vwscanf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <wchar.h>
int vwscanf_s(const wchar_t * restrict format,
     va_list arg);
```

Runtime-constraints

format shall not be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

---

59) It is not a runtime-constraint violation for the wide characters **%n** to appear in sequence in the wide string pointed at by **format** when those wide characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **L"%%n"**.

If there is a runtime-constraint violation, the **vwscanf_s** function does not attempt to perform further input, and it is unspecified to what extent **vwscanf_s** performed input before discovering the runtime-constraint violation.

Description

The **vwscanf_s** function is equivalent to **wscanf_s**, with the variable argument list replaced by **arg**, which shall have been initialized by the **va_start** macro (and possibly subsequent **va_arg** calls). The **vwscanf_s** function does not invoke the **va_end** macro.[60)]

Returns

The **vwscanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **vwscanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

6.9.1.13 The wprintf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <wchar.h>
int wprintf_s(const wchar_t * restrict format, ...);
```

Runtime-constraints

format shall not be a null pointer. The **%n** specifier[61)] (modified or not by flags, field width, or precision) shall not appear in the wide string pointed to by **format**. Any argument to **wprintf_s** corresponding to a **%s** specifier shall not be a null pointer.

If there is a runtime-constraint violation, the **wprintf_s** function does not attempt to produce further output, and it is unspecified to what extent **wprintf_s** produced output before discovering the runtime-constraint violation.

Description

The **wprintf_s** function is equivalent to the **wprintf** function except for the explicit runtime-constraints listed above.

---

60) As the functions **vfwscanf_s**, **vwscanf_s**, and **vswscanf_s** invoke the **va_arg** macro, the value of **arg** after the return is indeterminate.

61) It is not a runtime-constraint violation for the wide characters **%n** to appear in sequence in the wide string pointed at by **format** when those wide characters are not a interpreted as a **%n** specifier. For example, if the entire format string was **L"%%n"**.

Returns

The **wprintf_s** function returns the number of wide characters transmitted, or a negative value if an output error, encoding error, or runtime-constraint violation occurred.

6.9.1.14 The wscanf_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <wchar.h>
int wscanf_s(const wchar_t * restrict format, ...);
```

Runtime-constraints

format shall not be a null pointer. Any argument indirected though in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the **wscanf_s** function does not attempt to perform further input, and it is unspecified to what extent **wscanf_s** performed input before discovering the runtime-constraint violation.

Description

The **wscanf_s** function is equivalent to **fwscanf_s** with the argument **stdin** interposed before the arguments to **wscanf_s**.

Returns

The **wscanf_s** function returns the value of the macro **EOF** if an input failure occurs before any conversion or if there is a runtime-constraint violation. Otherwise, the **wscanf_s** function returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

6.9.2 General wide string utilities

6.9.2.1 Wide string copying functions

6.9.2.1.1 The wcscpy_s function

Synopsis

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <wchar.h>
errno_t wcscpy_s(wchar_t * restrict s1,
     rsize_t s1max,
     const wchar_t * restrict s2);
```


