

WG14 N2668

Title: Indeterminate Values and Trap Representations

Authors: Martin Uecker and Jens Gustedt

Date: 2021-02-28

1. Introduction

The basics of C's memory object model as described in 6.2.6 can be summarized as follows: An object representation (except for bit fields) consists of a series of bytes. When accessed using an lvalue expression an object representation can be interpreted as having a certain type. A type defines the set of values an expression can have. For each type, there is a mapping of representation bytes to the values of the type; the bytes representing a value are the object representation of the value. A value can have different object representations, but an object representation represents at most one value. There are byte patterns (called trap representations) that do not represent a value of the given type and reading or storing them is undefined. On the other hand, the bytes of an object representation can be accessed safely using lvalue expressions of character type because character types do not have trap representations. C follows this model fairly consistently, although sometimes the wording is not entirely precise. In particular, we have the following definitions:

3.15

1 object

region of data storage in the execution environment, the contents of which can represent values

2 Note 1 to entry: When referenced, an object can be interpreted as having a particular type; see 6.3.2.1.

3.19.2

1 indeterminate value

either an unspecified value or a trap representation

3.19.4

1 trap representation

an object representation that need not represent a value of the object type

The definition of "indeterminate value" is "either an unspecified value or a trap representation" and then we define a "trap representation" as something that does "not represent a value". This is contradictory and causes confusion.

The wording proposed here (relative to N2596) tries to make the standard text more precise by resolving this inconsistency. The intention is to stay as close as possible to the original wording while making the minimal changes necessary to make the use of "indeterminate" consistent with how the terms value and representation are used in the C standard. Specifically, it is proposed to replace the term "indeterminate value" (of an object) by the term "indeterminate state" (of an object). This change can be made consistently throughout the standard as the term "indeterminate value" is almost always associated with an object (the one exception are pointer values, see below), i.e. there are no "indeterminate values" of

expressions as reading “indeterminate values” from an object does not yield an “indeterminate value” but either unspecified values or – for trap representations - has undefined behavior. An implementation could of course introduce the concept of an indeterminate value for expressions which read objects in indeterminate state, but this is then in the realm of undefined behavior from the point of the standard. It should be noted that C++ has the concept of an “indeterminate value” for expressions, which comes with some special rules which define behavior in some special cases (e.g. initialization). C does not have these rules.

The proposed changes are summarized in the following:

1. The definition of “indeterminate value” is changed to “indeterminate state”
2. In the core language, phrases such as „the value of the object is indeterminate“ are changed to „the state of the object is indeterminate” or the “object has an indeterminate state“. Please note that his terminology is not new but already used in some parts of the standard, in particular for atomics.

One aspect defers special attention: Pointers become indeterminate when lifetime of the pointed-to object ends. This rule appears in the context of section 6.2.4 “Storage duration of objects”. As the concept of “indeterminate value” should now only be applied to objects there is the question is what then happens to “pointer values” which are currently being used in evaluations when the lifetime of the pointed-to object ends. This may occur in several contexts:

- When a pointer to a automatic object is returned from a function.
- When free or realloc are implemented with macros such that no sequence point appears before freeing the object.
- When in a concurrent program where the object is freed by a different thread.

Similar issues can appear when integers are converted to pointers. We will use the term “invalid value” for such pointer values. This term is also already used for such values in the section for the indirection operator (6.5.3.2) The planned TS 6010 for pointer provenance will address these questions in detail and provide precise definitions for “valid” and “invalid” pointer values.

3. In the library, similar changes are made, but in some cases the word “indeterminate” was not correct and changed to “unspecified” or “not valid” based on the kind of entity in question. The specific choices are explained in the following: `errno` is an lvalue which designates an object and its state can become indeterminate (7.14.1.1). File content consisting of multi-byte characters may become invalid but not indeterminate (7.21.2). Padding bytes become unspecified (7.24.4.1). *content of arrays* may become indeterminate (7.24.4.5,7.27.3.5) or – for character types - may become unspecified (7.21.5.6,7.21.7.2). The term *the content of the array* is used quite often in the standard but not very precise in this context. It is suggested to change it to *the members of the array* which makes it clear that objects are meant. A *file position indicator* is an object and its state can become indeterminate (7.21.7.10,7.21.8.1,7.21.8.2). The state of an newly allocated object is indeterminate (7.22.3.1,7.22.3.4). Additional bytes obtained with `realloc` are unspecified (7.22.3.5). The *conversion state* is described by an object

whose state can become indeterminate (7.22.7). The state of an nthread-specific storage pointer created by `tss_create` function can become indeterminate (7.26.6.1). A `va_list` is an object and its state can become indeterminate.

2.1 Definitions

3.19.2

1 indeterminate ~~value~~-state

state of an object corresponding to an object representation that either **represents** an **unspecified value** or **is** a **trap representation**

3.19.3

1 unspecified value

valid value of the relevant type where this document imposes no requirements on which value is chosen in any instance

2 Note 1 to entry: An unspecified value cannot be a trap representation.

3.19.4

1 trap representation

an **object representation** that ~~need~~-**does not represent** a **value** of the object type

2.2 Core Language

5.1.2.3 Program execution

5 When the processing of the abstract machine is interrupted by receipt of a signal, the values of objects that are neither lock-free atomic objects nor of type `volatile sig_atomic_t` are unspecified, as is the state of the dynamic floating-point environment. The **value state** of any **object** modified by the handler that is neither a lock-free atomic object nor of type `volatile sig_atomic_t` becomes **indeterminate** when the handler exits, as does the state of the dynamic floating-point environment if it is modified by the handler and not restored to its original state.

6.2.4 Storage durations of objects

2 The lifetime of an object is the portion of program execution during which storage is guaranteed to be reserved for it. An object exists, has a constant address,³⁶⁾ and retains its last-stored value throughout its lifetime.³⁷⁾ If an object is referred to outside of its lifetime, the behavior is undefined. The value of a **pointer** becomes **invalid indeterminate** when the object it the pointers points to (or just past) reaches the end of its lifetime. **The state of a pointer object that stores such a value becomes indeterminate.**

6 For such an object that does not have a variable length array type, its lifetime extends from entry into the block with which it is associated until execution of that block ends in any way. (Entering an enclosed block or calling a function suspends, but does not end, execution of the current block.) If the block is entered recursively, a new instance of the object is created each time. The initial **value state** of the **object** is- **indeterminate**. If an initialization is specified for the object, it is performed each time the declaration or compound literal is reached in the execution of the block; otherwise, the **value state of the object** becomes **indeterminate** each time the declaration is reached.

7 For such an object that does have a variable length array type, its lifetime extends from the declaration of the object until execution of the program leaves the scope of the declaration.³⁸⁾ If the scope is entered recursively, a new instance of the object is created each time. **The initial value state of** the object is **indeterminate**.

6.2.6 Representations of types

6.2.6.1 General

4 Values stored in non-bit-field objects of any other object type **consist of** are represented using $n \times \text{CHAR_BIT}$ bits, where n is the size of an object of that type, in bytes. **An object that stores the value** may be copied into an object of type unsigned char [n] (e.g., by memcpy); the resulting set of bytes is called the object representation of the value. Values stored in bit-fields consist of m bits, where m is the size specified for the bit-field. The object representation is the set of m bits the bit-field comprises in the addressable storage unit holding it. Two values (other than NaNs) with the same object representation compare equal, but values that compare equal may have different object representations.

5 Certain object representations need not represent a value of the object type. If **the stored value of an object has** such a **representation and** is read by an lvalue expression that does not have character type, the behavior is undefined. If such a representation is produced by a side effect that modifies all or any part of the object by an lvalue expression that does not have character type, the behavior is undefined.56) Such a representation is called a trap representation.

6 When a value is stored in an object of structure or union type, including in a member object, the bytes of the object representation that correspond to any padding bytes take unspecified values. 52) The **value object representation corresponding to the state** of a structure or union **object** is never a **trap representation**, even though the **value object representation corresponding to the state** of a member of the structure or union **object** may be a **trap representation**.

6.3.2.3 Pointers

5 An integer may be converted to any pointer type. Except as previously specified, the result is implementation-defined, might not be correctly aligned, might not point to an entity of the referenced type, and might **be a trap representation produce an indeterminate state when stored into an object**.71)

6.5.2.5 Compound literals

16 Note that if an iteration statement were used instead of an explicit goto and a label, the lifetime of the unnamed object would be the body of the loop only, and on entry next time around **p** would **have an have indeterminate value state**, which would result in undefined behavior.

6.7.2.1 Structure and union specifiers

28 The assignment:

```
*s1 = *s2;
```

only copies the member n ; if any of the **array elements** are within the first sizeof (struct s) bytes of the structure, they **might be copied or simply overwritten with indeterminate values** are set to **an indeterminate state, that may or may not coincide with a copy of the representation of the elements of the source array**.

6.7.9 Initialization

9 Except where explicitly stated otherwise, for the purposes of this subclause unnamed members of objects of structure and union type do not participate in initialization. Unnamed **members** of structure objects have **indeterminate value state** even after initialization.

10 If an **object** that has automatic storage duration is not initialized explicitly, its **state-value** is indeterminate.

6.8 Statements and blocks

3 A block allows a set of declarations and statements to be grouped into one syntactic unit. The initializers of objects that have automatic storage duration, and the variable length array declarators of ordinary identifiers with block scope, are evaluated and the values are stored in the objects (**including storing an indeterminate value in the state of objects** without an initializer **becomes indeterminate**) each time the declaration is reached in the order of execution, as if it were a statement, and within each declaration in the order that declarators appear.

6.8.4.2 The switch statement

7 EXAMPLE In the artificial program fragment

```
switch (expr)
```

```
{
```

```
int i = 4;
```

```
f(i);
```

```
case 0:
```

```
i = 17;
```

```
/* falls through into default code */
```

```
default:
```

```
printf("%d\n", i);
```

```
}
```

the object whose identifier is `i` exists with automatic storage duration (within the block) but is never initialized, and thus if the controlling expression has a nonzero value, the call to the `printf` function will access an **object with an indeterminate value state**. Similarly, the call to the function `f` cannot be reached.

2.3. Library

7.13.2.1 The longjmp function

3 All accessible objects have values, and all other components of the abstract machine²⁷¹) have state, as of the time the `longjmp` function was called, except that **the states-values-of-objects** of automatic storage duration that are local to the function containing the invocation of the corresponding `setjmp` macro that do not have volatile-qualified type and have been changed between the `setjmp` invocation and `longjmp` call are **indeterminate**.

7.14.1.1 The signal function

5 If the signal occurs other than as the result of calling the `abort` or `raise` function, the behavior is undefined if the signal handler refers to any object with static or thread storage duration that is not a lock-free atomic object other than by assigning a value to an object declared as volatile `sig_atomic_t`, or the signal handler calls any function in the standard library other than

— the `abort` function,

— the `_Exit` function,

— the `quick_exit` function,

— the functions in `<stdatomic.h>` (except where explicitly stated otherwise) when the atomic arguments are lock-free,

— the `atomic_is_lock_free` function with any atomic argument, or

— the `signal` function with the first argument equal to the signal number corresponding to the signal that caused the invocation of the handler. Furthermore, if such a call to the `signal` function results in a `SIG_ERR` return, the **value-of-object designated by** `errno` is **in an indeterminate state**.²⁷⁴)

7.16 Variable arguments <stdarg.h>

3 The type declared is

va_list

which is a complete object type suitable for holding information needed by the macros va_start, va_arg, va_end, and va_copy. If access to the varying arguments is desired, the called function shall declare an object (generally referred to as ap in this subclause) having type va_list. The **object ap** may be passed as an argument to another function; if that function invokes the va_arg macro with parameter ap, the **value state** of **ap** in the calling function is **indeterminate** and shall be passed to the va_end macro prior to any further reference to ap.275)

7.17.2.1 The ATOMIC_VAR_INIT macro

2 The ATOMIC_VAR_INIT macro expands to a token sequence suitable for initializing an atomic object of a type that is initialization-compatible with value. An atomic **object** with automatic storage duration that is not explicitly initialized is initially **in an indeterminate state**; however, the default (zero) initialization for objects with static or thread-local storage duration is guaranteed to produce a valid state.277)

7.17.8 Atomic flag type and operations

4 The macro ATOMIC_FLAG_INIT may be used to initialize an atomic_flag to the clear state. An **atomic_flag** that is not explicitly initialized with ATOMIC_FLAG_INIT is initially **in an indeterminate state**.

7.21.2 Streams

5 Byte input/output functions shall not be applied to a wide-oriented stream and wide character input/output functions shall not be applied to a byte-oriented stream. The remaining stream operations do not affect, and are not affected by, a stream's orientation, except for the following additional restrictions:

— Binary wide-oriented streams have the file-positioning restrictions ascribed to both text and binary streams.

— For wide-oriented streams, after a successful call to a file-positioning function that leaves the file position indicator prior to the end-of-file, a wide character output function can overwrite a partial multibyte character; any **file contents beyond the byte(s) written are may henceforth not consist of valid multibyte characters indeterminate**.

7.21.3 Files

4 A file may be disassociated from a controlling stream by closing the file. Output streams are flushed (any unwritten buffer contents are transmitted to the host environment) before the stream is disassociated from the file. The **value-of-a-pointer-to-lifetime of** a FILE object **is indeterminate after ends when** the associated file is closed (including the standard text streams). Whether a file of zero length (on which no characters have been written by an output stream) actually exists is implementation-defined.

7.21.5.6 The setvbuf function

2 The setvbuf function may be used only after the stream pointed to by stream has been associated with an open file and before any other operation (other than an unsuccessful call to setvbuf) is performed on the stream. The argument mode determines how stream will be buffered, as follows:

_IOFBF causes input/output to be fully buffered;

_IOLBF causes input/output to be line buffered;

_IONBF causes input/output to be unbuffered.

If buf is not a null pointer, the array it points to may be used instead of a buffer allocated by the setvbuf function297) and the argument size specifies the size of the array; otherwise, size may determine the size of a buffer allocated by the setvbuf function. **The contents the members of the array** at any time **are have unspecified values indeterminate**.

7.21.6.8 The vfprintf function

313)As the functions vfprintf, vfscanf, vprintf, vscanf, vsnprintf, vsprintf, and vsscanf invoke the va_arg macro, **the value of arg** after the return is **in an indeterminate state**.

7.21.7.2 The fgets function

3 The fgets function returns s if successful. If end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read error occurs during the operation, **the members of the array contents are have unspecified values indeterminate** and a null pointer is returned.

7.21.7.10 The ungetc function

5 A successful call to the ungetc function clears the end-of-file indicator for the stream. The value of the file position indicator for the stream after reading or discarding all pushed-back characters shall be the same as it was before the characters were pushed back.315) For a text stream, the value of its file position indicator after a successful call to the ungetc function is unspecified until all pushed-back characters are read or discarded. For a binary stream, its **file position indicator** is decremented by each successful call to the ungetc function; if its value was zero before a call, it is **in an indeterminate state** after the call.316)

7.21.8.1 The fread function

2 The fread function reads, into the array pointed to by ptr, up to nmemb elements whose size is specified by size, from the stream pointed to by stream. For each object, size calls are made to the fgetc function and the results stored, in the order read, in an array of unsigned char exactly overlaying the object. The file position indicator for the stream (if defined) is advanced by the number of characters successfully read. If an error occurs, the resulting **value-state** of the **file position indicator** for the stream is **indeterminate**. If a partial element is read, its **value state** is **indeterminate**.

7.21.8.2 The fwrite function

2 The fwrite function writes, from the array pointed to by ptr, up to nmemb elements whose size is specified by size, to the stream pointed to by stream. For each object, size calls are made to the fputc function, taking the values (in order) from an array of unsigned char exactly overlaying the object. The file position indicator for the stream (if defined) is advanced by the number of characters successfully written. If an error occurs, the resulting **value-state** of the **file position indicator** for the stream is **indeterminate**

7.22.3.1 The aligned_alloc function

2 The aligned_alloc function allocates space for an **object** whose alignment is specified by alignment, whose size is specified by size, and whose **value state** is **indeterminate**. If the value of alignment is not a valid alignment supported by the implementation the function shall fail by returning a null pointer.

7.22.3.4 The malloc function

2 The malloc function allocates space for an **object** whose size is specified by size and whose **value state** is **indeterminate**.

7.22.3.5 The realloc function

2 The realloc function deallocates the old object pointed to by ptr and returns a pointer to a new object that has the size specified by size. The contents of the new object shall be the same as that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any **bytes** in the new object beyond the size of the old object have **indeterminateunspecified values**.

7.22.7 Multibyte/wide character conversion functions

1 The behavior of the multibyte character functions is affected by the LC_CTYPE category of the current locale. For a state-dependent encoding, each of the mbtowc and wctomb functions is placed into its initial conversion state prior to the first call to the function and can be returned to that 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. It is implementation-defined whether internal conversion state has thread storage duration, and whether a newly created thread has the same state as the current thread at the time of creation, or the initial conversion state. A call with s as a null pointer causes these functions to return a nonzero value if encodings have state dependency, and zero otherwise. Changing the LC_CTYPE category causes the **internal object describing the conversion state** of the mbtowc and wctomb functions to be **in an indeterminate state**.

7.24.4.1 The memcmp function

335)The **contents of "holes" values of unused bytes-used as padding** for purposes of alignment within structure objects are **indeterminateunspecified**. Strings shorter than their allocated space and unions can also cause problems in comparison.

7.24.4.5 The strxfrm function

3 The strxfrm function returns the length of the transformed string (not including the terminating null character). If the value returned is n or more, the **contentsmembers of the array** pointed to by s1 are **in an indeterminate state**.

7.26.6.1 The tss_create function

6 If the tss_create function is successful, it sets the thread-specific storage pointed to by key to a value that uniquely identifies the newly created pointer and returns thrd_success; otherwise, thrd_error is returned and the **thread-specific storage** pointed to by key is set to an **indeterminate state value**.

7.27.3.5 The strftime function

8 If the total number of resulting characters including the terminating null character is not more than maxsize, the strftime function returns the number of characters placed into the array pointed to by s not including the terminating null character. Otherwise, zero is returned and the **contentsmembers of the array** are **in an indeterminate state**.

7.29.2.6 The vfwscanf function

366)As the functions vfwprintf, vswprintf, vfwscanf, vwprintf, vwsconf, and vswscanf invoke the va_arg macro, the **value state of-arg** after the return is **indeterminate**.

7.29.3.2 The fgetws function

3 The fgetws function returns s if successful. If end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read or encoding error occurs during the operation, the **array contentsmembers** are **in an indeterminate state** and a null pointer is returned.

7.29.4.4.4 The wcsxfrm function

3 The wcsxfrm function returns the length of the transformed wide string (not including the terminating null wide character). If the value returned is n or greater, the **contentsmembers of the array** pointed to by s1 are **in an indeterminate state**.

7.29.5.1 The wcsftime function

3 If the total number of resulting wide characters including the terminating null wide character is not more than maxsize, the wcsftime function returns the number of wide characters placed into the array pointed to by s not including the terminating null wide character. Otherwise, zero is returned and the **contents** members of the array are **in an indeterminate state**.

J.2 Undefined behavior

— The value of an **object** with automatic storage duration is used while **it the object is in an indeterminate state** (6.2.4, 6.7.9, 6.8).

K.3.5.3.10 The vprintf_s function

432)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 state** of-arg after the return is indeterminate.

K.3.5.3.11 The vscanf_s function

434)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 state** of-arg after the return is indeterminate.

K.3.5.3.14 The vsscanf_s function

437)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 state** of-arg after the return is indeterminate.

K.3.6.4 Multibyte/wide character conversion functions

1 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. 447) Changing the LC_CTYPE category causes the **internal object describing the conversion state** of these functions to be **in an indeterminate state**.

K.3.9.1.7 The vfwscanf_s function

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

K.3.9.1.10 The vswscanf_s function

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

K.3.9.1.12 The vwscanf_s function

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

L.2.2

1 bounded undefined behavior

undefined behavior (3.4.3) that does not perform an out-of-bounds store.

2 Note 1 to entry: The behavior might perform a trap.

3 Note 2 to entry: Any **values** produced or stored might be **indeterminateunspecified values**.

Acknowledgements: We want to thank Martin Sebor for helpful discussions.