n3453 - Examples of Undefined Behavior

This document describes the issues around the development of the Examples of Undefined Behavior for Annex J.2. proposal. This document is specifically for UBSG members to use while developing the Examples document.

The Examples document is specifically for WG14. It will be a 'new work item proposal', and eventually appear as a white paper.

For each undefined behavior listed, we recommend all relevant examples will be added verbatim to Annex J.2. Other items, such as 'Reviewers' can be ignored.

Acknowledgments

This document was suggested by Aaron Ballman, and championed by David Svoboda

We would also like to thank the Undefined Behavior Study Group, including Martin Uecker for eliminating several undefined behaviors, in his "Slaying the Earthly Demons" series. Thanks also to Shivashankar Maddanimath, Joseph Myers, Clive Pygott, and Robin Rowe for reviewing this document.

History

This document was created in late 2021 in the UBSG.

It was first presented to WG14 in the Fall 2024 meeting in Minneapolis.

December 2023 commit & notes

New UB code examples (without notes): 13, 22, 70, 71, 74, 76-78, 83, 84, 88, 89, 91, 95, 97, 98, 101, 102, 114, 116-118, 122, 123, 145, 147, 150, 175-180, 190, 194, 198, 202, 204

New UB code examples with notes:

- 6: This seems to limit the scope of non-basic character sets to keywords, and no C keyword contains a non-basic character
- 15: This will compile but not link. It might not qualify; this might be the UB about x being uninitialized. Is it possible for a variably-length array to have a dimension that is not evaluated? (This might be an earthly demon).
- 64: Code sample stolen from Example 2 of C23 6.7.4.1. First example (cs = ncs) does not compile under current Clang, but other UB statements compile with just a warning.
- 26-29, 146, 149: How to compile? These look like earthly demons.
- 52-55, 82, 90, 93, 205: How to compile? Probably impl-def earthly demons?
- 77: Bad wording…refuted by example 3 in s6.7.10

- 85: Please check…code example requires two files
- 96: Platform-specific behavior, documented in comments
- 103: Duplicate UB: Isn't there another UB about macro/keyword collision?
- 148: Is there a purpose or direct way to read nullptr_t objects?
- 191: I have no idea what this means, and see no related text in section 7.24.4.8.
- 203, 206, 207: GCC/Clang don't support _Decimal* types

Development Rules

Here are some rules that the UB examples should comply with:

1. All examples on this page should compile with a popular compiler (that is, MSVC, GCC, or Clang)
2. If an example fails to compile with a popular compiler, but does compile with a more obscure compiler, then we should note the compiler used, including version number.
3. Any example that fails to build under every compiler we know of is incomplete…we should ideally find a compiler that builds it. (The compiler may produce any warnings, but should not produce any fatal errors.)
4. Finally, if we cannot produce an example that compiles, then perhaps that UB should be eliminated and any such code should be ill-formed, and require a fatal diagnostic in translation.

Tasks

If you get a few minutes to work on this page, you can do one of two things:

- Replace any TODO with a suitable coding example. If you like your example, then add your name (last name good enough) to the 'Reviewers:' section right below the TODO.
- Review any UB coding example and verify that it is good (eg it correctly illustrates the particular UB).
  - If you like the example as is, then add your name to the Reviewers section.
  - If you don't like the example, then please repair or replace the example.
    - If you modify the example, then add your name to the Reviewers section AND erase the other reviewers (since they haven't reviewed your changes yet).
- Eventually we should eliminate all the TODOs, and every code example should have a set of reviewers (preferably 3 or more) that approve of the example.

Statistics

Total Number of UB"s in document: 221

Total Number of TODO's: 26 Total Number of Code Examples from TS 17961: 37 Total Number of Code Examples from CERT C: 49 Total Number of Original Code Examples: 138

Note that the latter statistics total up to 250, and this is because some UBs have multiple code examples.

Format

The enumeration of UBs corresponding to 'C23' definitions are taken from N3220.

The term COMPILABLE appears for some of the UB's listed here. These are UBs where we cannot think of code that both demonstrates the UB and compiles without errors in GCC or Clang. What should we do about these UBs? Assuming no one can provide a compiler that builds such examples, perhaps we should state that these UBs are obsolete, or instead that these programs should be ill-formed.

The format of this document is is:

- C23 UB Number
- UB Definition
- Examples
- Notes
- Reviewers

Before final publication, the Notes and Reviewers can be removed. Any remaining TODO's can be removed, although they should have been removed by the completion process.

Legal Issues

ISO and IEC do not recognize any licenses other than their own and will not honor any other license without a long, protracted series of negotiations with their lawyers. Therefore, do not submit proprietary code, or any code that is attached to a license that must accompany the code, such as the GNU Public License. This material is intended to become an ISO/IEC document, so any contributions carry implicit permission to be published by ISO and IEC.

Citations

Many code examples are external to this document, in which case they cite their sources. The code examples come from:

- TS 17961: ISO/IEC TS 17961. *Information Technology—Programming Languages, Their Environments and System Software Interfaces—C Secure Coding Rules.* Geneva, Switzerland: ISO, 2012.
  - The following subsections of Section 5 of TS 17961 have no examples that demonstrate undefined behavior: 4, 7, 8, 9, 10 (exception), 11 (example 2), 12, 16, 17, 18, 19, 21, 24, 25, 38, 39, 42, 43, 46
- CERT C Rule (published 2016 snapshot of SEI CERT C Rule) (NCCE = Noncompliant Code Example)
- CERT C Rec (recommendations)

It is expected that before final publication of this document, all of the code examples will be inlined. The CERT code examples are reproduced with permission of the owner.

Normative Text

1. A "shall" or "shall not" requirement that appears outside of a constraint is violated (Clause 4).

See UB Example #4, which violates a 'shall' statement.

Reviewers: svoboda, UBSG

2. A nonempty source file does not end in a new-line character which is not immediately preceded by a backslash character or ends in a partial preprocessing token or comment (5.1.1.2).

CERT C Rec MSC04-C 1st NCCE

Reviewers: svoboda

3. Token concatenation produces a character sequence matching the syntax of a universal character name (5.1.1.2).

CERT C Rule PRE30-C 1st NCCE 2.1.1

Reviewers: svoboda, s.maddanimath

4. A program in a hosted environment does not define a function named main using one of the specified forms (5.1.2.3.2).

```c
#include <stdio.h>
int main(float argc) {  // Undefined Behavior
  printf("main argument count:%f\n", argc);
  return 0;
}
```

Reviewers: s.maddanimath, svoboda, UBSG, j.myers

5. The execution of a program contains a data race (5.1.2.5).

CERT C Rule CON32-C 1st NCCE 14.3.1, CON40-C 1st NCCE 14.11.1

Reviewers: svoboda

6. A character not in the basic source character set is encountered in a source file, except in an identifier, a character constant, a string literal, a header name, a comment, or a preprocessing token that is never converted to a token (5.2.1).

```
/* In UTF-8 Lowercase n-with-tilde == ñ == U+00F1 == 0xC3 0xB1 == \303 \261 */
const char mañana tomorrow[] = "tomorrow"; /* valid UTF-8 but not compilable */
// Undefined Behavior if built in UTF-8 locale
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

7. An identifier, comment, string literal, character constant, or header name contains an invalid multibyte character or does not begin and end in the initial shift state (5.2.2).

```
/* In UTF-8 Lowercase n-with-tilde == ñ == U+00F1 == 0xC3 0xB1 == \303 \261 */
const char ma\303ana[] = "tomorrow"; /* invalid UTF-8, missing \261 */
/* Undefined Behavior if built in UTF-8 locale */
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda, USBG, j.myers

8. The same identifier has both internal and external linkage in the same translation unit (6.2.2).

CERT C Rule DCL36-C 1st NCCE 3.3.1

Reviewers: svoboda, s.maddanimath

Note: the following is a different example which is more complex:

```
static int a;
int f(void) {
  int a;
  {
     extern int a; // not internal!
  }
}
```

Reviewers: uecker, j.myers

9. An object is referred to outside of its lifetime (6.2.4).

CERT C Rule DCL30-C 1st NCCE 3.1.1, 2nd NCCE 3.1.4, 3rd NCCE 3.1.6
CERT C Rule EXP35-C 1st NCCE 4.5.1, 2nd NCCE 4.5.3

Reviewers: svoboda, s.maddanimath

10. The value of a pointer to an object whose lifetime has ended is used (6.2.4).

TS17961 5.14 [nullref] EXAMPLE 5.15 [addrescape] EXAMPLE 1, 2, 3

Reviewers: svoboda

11. The value of an object with automatic storage duration is used while the object has an indeterminate representation (6.2.4, 6.7.11, 6.8).

TS17961 5.35 [uninitref] EXAMPLE 1, 2
CERT C Rule EXP33-C 4th NCCE 4.3.9,
CERT C Rec MSC22-C 3rd NCCE

Reviewers: svoboda, s.maddanimath

12. A non-value representation is read by an lvalue expression that does not have character type (6.2.6.1).

TS17961 5.35 [uninitref] EXAMPLE 3, 4

Reviewers: svoboda

13. A non-value representation is produced by a side effect that modifies any part of the object using an lvalue expression that does not have character type (6.2.6.1).

```
const int float_size = sizeof(float);
float value;
char bytes[float_size] = "yes";
memcpy(&value, bytes, float_size);
printf("value is %f\n", value); // Undefined Behavior
```

Reviewers: svoboda

14. Two declarations of the same object or function specify types that are not compatible (6.2.7).

TS17961 5.13 [funcdecl] EXAMPLE 1, 2, 4

Reviewers: svoboda

15. A program requires the formation of a composite type from a variable length array type whose size is specified by an expression that is not evaluated (6.2.7).

```
extern int x;
char bytes[x]; // Undefined Behavior
```

Reviewers: svoboda

16. Conversion to or from an integer type produces a value outside the range that can be represented (6.3.1.4).

CERT C Rule FLP34-C 1st NCCE 6.3.1, FLP36-C 1st NCCE 6.4.1

Reviewers: svoboda

17. Demotion of one real floating type to another produces a value outside the range that can be represented (6.3.2.1).

CERT C Rule FLP34-C 2nd NCCE 6.3.3

Reviewers: svoboda

18. An lvalue does not designate an object when evaluated (6.3.2.1).

Reviewers:

19. A non-array lvalue with an incomplete type is used in a context that requires the value of the designated object (6.3.2.1).

```
struct f *p;
void g(void) {
  *p;
}
```

Reviewers: uecker, j.myers

Note: Removed from J.2. by N3244

20. An lvalue designating an object of automatic storage duration that could have been declared with the register storage class is used in a context that requires the value of the designated object, but the object is uninitialized. (6.3.2.1).

```
void f(void) {
  /* register */ int x;  // address of x not taken, so x could be stored in a register
  int y = x;             // Undefined Behavior
}
```

Reviewers: svoboda, uecker, j.myers

21. An lvalue having array type is converted to a pointer to the initial element of the array, and the array object has register storage class (6.3.2.1).

```
void f(void) {
  register int a[3];
  int *p = a;  // Undefined Behavior
  a[0] = 1;
  p[0];
}
```

Reviewers: svoboda

Note: Removed from J.2. by N3244

22. An attempt is made to use the value of a void expression, or an implicit or explicit conversion (except to void) is applied to a void expression (6.3.2.2).

```
void f(int x) {
  printf("x is %d\n", x);
}

void* p = (void*) f;
int (*g)(int) = p;
int y = g(123); // Undefined Behavior
printf("y is %d\n", y);
```

Reviewers: svoboda

23. Conversion of a pointer to an integer type produces a value outside the range that can be represented (6.3.2.3).

TS17961 5.10 [intptrconv] EXAMPLE 1,2

Reviewers: svoboda

24. Conversion between two pointer types produces a result that is incorrectly aligned (6.3.2.3).

TS17961 5.11 [alignconv] EXAMPLE 1

Reviewers: svoboda

25. A pointer is used to call a function whose type is not compatible with the referenced type (6.3.2.3).

TS17961 5.6 [argcomp] EXAMPLE 1

Reviewers: svoboda, s.maddanimath

26. An unmatched ' or " character is encountered on a logical source line during tokenization (6.4).

```
char s[] = "foo
bar";
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

27. A reserved keyword token is used in translation phase 7 or 8 (5.1.1.2) for some purpose other than as a keyword (6.4.1).

```
int doN = 3;  // "÷N", Does not compile on Clang.
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

28. A universal character name in an identifier does not designate a character whose encoding falls into one of the specified ranges (6.4.2.1).

```
int \u00C6N = 1;  // "ÆN", Valid
int \u00F7N = 3;  // "÷N", Does not compile on Clang.
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

29. The initial character of an identifier is a universal character name designating a digit (6.4.2.1).

```
int \u00C6N = 1;  // "ÆN", Valid
int \u0046N = 2;  // "FN", Does not compile on Clang.
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

30. Two identifiers differ only in nonsignificant characters (6.4.2.1).

TS17961 5.13 [funcdecl] EXAMPLE 4

Reviewers: svoboda

31. The identifier ___func___ is explicitly declared (6.4.2.2).

```
void __func__(void); // Undefined Behavior
```

Reviewers: svoboda, j.myers

32. The program attempts to modify a string literal (6.4.5).

TS17961 5.28 [strmod] EXAMPLE 1, 2, 3, 4, 5

Reviewers: svoboda

33. The characters ', \, ", //, or /* occur in the sequence between the < and > delimiters, or the characters ', \, //, or /* occur in the sequence between the" delimiters, in a header name preprocessing token (6.4.7).

```
#include "dave's_hello.h"
// Undefined Behavior
```

Reviewers: svoboda, j.myers

34. A side effect on a scalar object is unsequenced relative to either a different side effect on the same scalar object or a value computation using the value of the same scalar object (6.5).

CERT C Rec PRE00-C 1st NCCE, 3rd NCCE

Reviewers: svoboda, s.maddanimath

35. An exceptional condition occurs during the evaluation of an expression (6.5.).

TS17961 5.30 [intoflow] EXAMPLE 1

Reviewers: svoboda

36. An object has its stored value accessed other than by an lvalue of an allowable type (6.5.1).

TS17961 5.1 [ptrcomp] EXAMPLE

Reviewers: svoboda

C18-37. For a call to a function without a function prototype in scope, the number of arguments does not equal the number of parameters (6.5.2.2).

TS17961 5.6 [argcomp] EXAMPLE 2

Reviewers: svoboda, s.maddanimath

C18-38. For a call to a function without a function prototype in scope where the function is defined with a function prototype, either the prototype ends with an ellipsis or the types of the arguments after default argument promotion are not compatible with the types of the parameters (6.5.2.2).

TS17961 5.6 [argcomp] EXAMPLE 3

Reviewers: svoboda, s.maddanimath

37. A function is defined with a type that is not compatible with the type (of the expression) pointed to by the expression that denotes the called function (6.5.3.3).

TS17961 5.6 [argcomp] EXAMPLE 4, 5.13 [funcdecl] EXAMPLE 3

Reviewers: svoboda, s.maddanimath

38. A member of an atomic structure or union is accessed (6.5.3.4).

```
_Atomic struct {
  int x;
} foo;
foo.x;   // Undefined Behavior
```

Reviewers: uecker, svoboda, j.myers

39. The operand of the unary * operator has an invalid value (6.5.4.2).

```
char* p = NULL;
*p;   // Undefined Behavior
```

Reviewers: svoboda, j.myers

40. A pointer is converted to other than an integer or pointer type (6.5.5).

```
struct f { struct f *x; } *p;
void g(void) {
  (struct f)p;
}
```

Note: Removed from J.2. by N3244

41. The value of the second operand of the / or % operator is zero (6.5.6).

TS17961 5.26 [diverr] EXAMPLE 1

Reviewers: svoboda, s.maddanimath

42. If the quotient a/b is not representable, the behavior of both a/b and a%b (6.5.6).

TS17961 5.26 [diverr] EXAMPLE 2

Reviewers: svoboda, s.maddanimath

43. Addition or subtraction of a pointer into, or just beyond, an array object and an integer type produces a result that does not point into, or just beyond, the same array object (6.5.7).

TS17961 5.22 [invptr] EXAMPLE 1

Reviewers: svoboda

44. Addition or subtraction of a pointer into, or just beyond, an array object and an integer type produces a result that points just beyond the array object and is used as the operand of a unary * operator that is evaluated (6.5.7).

TS17961 5.22 [invptr] EXAMPLE 4, 6, 10, 12

Reviewers: svoboda

45. Pointers that do not point into, or just beyond, the same array object are subtracted (6.5.7).

TS17961 5.36 [ptrobj] EXAMPLE, CERT C Rule ARR36-C 1st NCCE 7.3.1

Reviewers: svoboda

46. An array subscript is out of range, even if an object is apparently accessible with the given subscript (as in the lvalue expression a[1][7] given the declaration int a[4][5]) (6.5.7).

TS17961 5.22 [invptr] EXAMPLE 8

Reviewers: svoboda

47. The result of subtracting two pointers is not representable in an object of type ptrdiff_t (6.5.7).

```
#include <stdlib.h>
size_t size = 1 + (SIZE_MAX / 2); // this assumes sizeof(size_t) == sizeof(ptrdiff_t)
char* x = malloc(size);
assert(x != 0);
char* start = x;
char* too_far = x + size;
ptrdiff_t too_big = too_far - start;  // Undefined Behavior
```

Reviewers: svoboda

48. An expression is shifted by a negative number or by an amount greater than or equal to the width of the promoted expression (6.5.8).

CERT C Rule INT34-C 1st NCCE 5.5.1, 2nd NCCE 5.5.3, 3rd NCCE 5.5.5

Reviewers: svoboda

49. An expression having signed promoted type is left-shifted and either the value of the expression is negative or the result of shifting would not be representable in the promoted type (6.5.8).

CERT C Rule INT32-C 6th NCCE 5.3.8.1, CERT C Rule INT34-C 2nd NCCE 5.5.3

Reviewers: svoboda

50. Pointers that do not point to the same aggregate or union (nor just beyond the same array object) are compared using relational operators (6.5.9).

```
struct {
  int x;
  int y;
} a, b;
if (&a.y < &b.y) {  // Undefined Behavior
  // ...
}
```

Reviewers: uecker, svoboda, j.myers

51. An object is assigned to an inexactly overlapping object or to an exactly overlapping object with incompatible type (6.5.17.2).

```
const size_t limit = sizeof(int) + 1;
char bytes[limit];
int *p1 = (int*) &(bytes[0]);
int *p2 = (int*) &(bytes[1]); // overlaps with p1 (unless sizeof(int) == 1)
*p1 = 123;
*p2 = *p1; // Undefined Behavior
```

Reviewers: coates, svoboda

52. An expression that is required to be an integer constant expression does not have an integer type; has operands that are not integer constants, named constants, compound literal constants, enumeration constants, character constants, predefined constants, sizeof expressions whose results are integer constants, alignof expressions, or immediately-cast floating constants; or contains casts (outside operands to sizeof and alignof operators) other than conversions of arithmetic types to integer types (6.6).

```
enum people {
  Tom=1.5,  // Undefined Behavior
  Dick=2,
  Harry=3
};

struct s {
  int bit:1;
  int two_bits:2.5; // Undefined Behavior
  int all_the_bits;
};
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

53. A constant expression in an initializer is not, or does not evaluate to, one of the following: a named constant, a compound literal constant, an arithmetic constant expression, a null pointer constant, an address constant, or an address constant for a complete object type plus or minus an integer constant expression (6.6).

```
int square(int x) {return x*x;}

enum people {
  Tom=1,
  Dick=2,
  Harry=square(2)
};

struct s {
  int bit:1;
  int two_bits:square(2);
  int all_the_bits;
};
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

54. An arithmetic constant expression does not have arithmetic type; has operands that are not integer constants, floating constants, named and compound literal constants of arithmetic type, character constants, predefined constants, sizeof expressions whose results are integer constants, or alignof expressions; or contains casts (outside operands to sizeof or alignof operators) other than conversions of arithmetic types to arithmetic types (6.6).

```
float f = "Hello, world!";
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

55. The value of an object is accessed by an array-subscript [], member-access . or ->, address &, or indirection * operator or a pointer cast in creating an address constant (6.6).

```
int primes[] = {2, 3, 5, 7};
int first_odd_prime = primes[1]; // 3
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

56. An identifier for an object is declared with no linkage and the type of the object is incomplete after its declarator, or after its init-declarator if it has an initializer (6.7).

Note: Removed from J.2. by N3244

57. A function is declared at block scope with an explicit storage-class specifier other than extern (6.7.2).

Note: Removed from J.2. by N3244

58. A structure or union is defined without any named members (including those specified indirectly via anonymous structures and unions) (6.7.3.2).

```
struct {
  // No named members
} myStruct;
```

Reviewers: coates, svoboda

59. An attempt is made to access, or generate a pointer to just past, a flexible array member of a structure when the referenced object provides no elements for that array (6.7.3.2).

CERT C Rule ARR30-C 5th NCCE 7.1.10

Reviewers: svoboda

60. When the complete type is needed, an incomplete structure or union type is not completed in the same scope by another declaration of the tag that defines the content (6.7.3.4).

Note: Removed from J.2. by N3244

61. An attempt is made to modify an object defined with a const-qualified type through use of an lvalue with non-const-qualified type (6.7).

CERT C Rule EXP40-C 1st NCCE 4.9.1

Reviewers: svoboda

62. An attempt is made to refer to an object defined with a volatile-qualified type through use of an lvalue with non-volatile-qualified type (6.7.4).

CERT C Rule EXP32-C 1st NCCE 4.2.1

Reviewers: svoboda, s.maddanimath

63. The specification of a function type includes any type qualifiers (6.7.4).

```
typedef int fun_t(int);
const fun_t f;  // Undefined Behavior
```

Reviewers: uecker, svoboda, j.myers

Note: Removed from J.2. by N3242

64. Two qualified types that are required to be compatible do not have the identically qualified version of a compatible type (6.7.4).

```
const struct s { int mem; } cs = { 1 };
struct s ncs; // the object ncs is modifiable
typedef int A[2][3];
const A a = {{4, 5, 6}, {7, 8, 9}}; // array of array of const int
int *pi;
const int *pci;
ncs = cs;      // valid
```

```
cs = ncs;       // Undefined Behavior: violates modifiable lvalue constraint for =
pi = &ncs.mem; // valid
pi = &cs.mem;  // Undefined Behavior:  violates type constraints for =
pci = &cs.mem; // valid
pi = a[0];     // Undefined Behavior: a[0] has type "const int *"
```

Reviewers: svoboda

65. An object which has been modified is accessed through a restrict-qualified pointer to a const-qualified type, or through a restrict-qualified pointer and another pointer that are not both based on the same object (6.7.4.2).

TS17961 5.33 [restrict] EXAMPLE 1, 2, CERT C Rule EXP43-C 2st NCCE 4.11.2.1, 3rd NCCE 4.11.2.3, 5th NCCE 4.11.4.1

Reviewers: svoboda

66. A restrict-qualified pointer is assigned a value based on another restricted pointer whose associated block neither began execution before the block associated with this pointer, nor ended before the assignment (6.7.4.2).

CERT C Rule EXP43-C 1st NCCE 4.11.1.1, 4th NCCE 4.11.3.1, 6th NCCE 4.11.5.1

Reviewers: svoboda

67. A function with external linkage is declared with an inline function specifier, but is not also defined in the same translation unit (6.7.5).

```
extern inline int foo(int x);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

Note: Removed from J.2. by N3244

68. A function declared with a _Noreturn function specifier returns to its caller (6.7.5).

```
#include <stdnoreturn.h>
_Noreturn void f(void) {
  return;  // Undefined Behavior at run-time
}
```

Reviewers: uecker, svoboda, j.myers

69. The definition of an object has an alignment specifier and another declaration of that object has a different alignment specifier (6.7.6).

Note: Removed from J.2. by N3244

70. Declarations of an object in different translation units have different alignment specifiers (6.7.6).

```c
// In file1.c:
alignas(16) int a;

// In file2.c:
alignas(32) int a; // Undefined Behavior
```

Reviewers: svoboda

71. Two pointer types that are required to be compatible are not identically qualified, or are not pointers to compatible types (6.7.7.2).

```c
int i = 1, j = 2;
const int *cp = &i; // *cp is constant
int *ncp = &j;      // *ncp is modifiable
ncp = cp;           // valid
cp = ncp;           // Undefined Behavior: violates modifiable lvalue constraint for =
```

Reviewers: svoboda

72. The size expression in an array declaration is not a constant expression and evaluates at program execution time to a nonpositive value (6.7.7.3).

```c
int size = -4;

// This creates a VLA.
int arr[size];

printf("%d\n",sizeof(arr));
```

Reviewers:

73. In a context requiring two array types to be compatible, they do not have compatible element types, or their size specifiers evaluate to unequal values (6.7.7.3).

CERT C Rule EXP39-C 4th NCCE 4.8.7

Reviewers: s.maddanimath

74. A declaration of an array parameter includes the keyword static within the [and] and the corresponding argument does not provide access to the first element of an array with at least the specified number of elements (6.7.7.4).

```
const int size = 5;

int average(int numbers[static size]) {
  int sum = 0;
  for (int i = 0; i < size; i++) {
    sum += numbers[i];
  }
  return sum / size;
}

int main(void) {
  int a[3] = { 4, 1002, 27 };
  int result = average(a); // Undefined Behavior, array too small
  printf("Average is %d\n", result);
}
```

Reviewers: svoboda

75. A storage-class specifier or type qualifier modifies the keyword void as a function parameter type list (6.7.74).

```
void f(const void);
void g(register void);   // Undefined Behavior
```

Note: Removed from J.2. by N3244

Reviewers:

76. In a context requiring two function types to be compatible, they do not have compatible return types, or their parameters disagree in use of the ellipsis terminator or the number and type of parameters (after default argument promotion, when there is no parameter type list) (6.7.7.4).

```
void f(double a[6]);

void f(double a[7]); // Undefined Behavior
```

Reviewers: svoboda

77. A declaration for which a type is inferred contains a pointer, array, or function declarators (6.7.10).

```
double double_double(double x) {
  return x * 2;
}

auto (*fn)(double x) = double_double; // Undefined Behavior
```

Reviewers: svoboda

78. A declaration for which a type is inferred contains no or more than one declarators (6.7.10).

```
auto i = 3, j = 4.5; // Undefined Behavior
```

Reviewers: svoboda

79. The value of an unnamed member of a structure or union is used (6.7.11).

Note: Removed from J.2. by N3245

80. The initializer for a scalar is neither a single expression, an empty initializer, nor a single expression enclosed in braces (6.7.11).

Note: Removed from J.2. by N3246

81. The initializer for a structure or union object is neither an initializer list nor a single expression that has compatible structure or union type (6.7.11).

Note: Removed from J.2. by N3246

82. The initializer for an aggregate or union, other than an array initialized by a string literal, is not a brace-enclosed list of initializers for its elements or members (6.7.11).

```
struct st {
  int value;
};

struct st s = 0;
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

83. A function definition that does not have the asserted property is called by a function declaration or a function pointer with a type that has the unsequenced or reproducible attribute (6.7.13.8).

```
int next(int *ip) [[reproducible]] {
  return *ip++; // Undefined Behavior, not idempotent
}
```

Reviewers: svoboda

84. An identifier with external linkage is used, but in the program there does not exist exactly one external definition for the identifier, or the identifier is not used and there exist multiple external definitions for the identifier (6.9).

```
extern int x;

x = 3; // Undefined Behavior if x is never defined
printf("x is %d!\n", x);
```

Reviewers: svoboda

C18-87. An adjusted parameter type in a function definition is not a complete object type (6.9.1).

```
// no previous definition of struct foo
void f(struct foo x) {}   // Undefined Behavior
```

Note: Removed from C23

Reviewers: j.myers

85. A function that accepts a variable number of arguments is defined without a parameter type list that ends with the ellipsis notation (6.9.2).

```
int add(int first, int second) {
  return first + second;
}
```

```
#include <stdio.h>

int add(int first, int second, ...);

int main () {
  int result = add(2, 3, 5, 7, 11, -1); // Undefined Behavior
  printf("Sum is %d\n", result);
  return 0;
}
```

Reviewers: svoboda

86. The } that terminates a function is reached, and the value of the function call is used by the caller (6.9.2).

CERT C Rule MSC37-C 1st NCCE 15.4.1, 2nd NCCE 15.4.3, 3rd NCCE 15.4.3.1

Reviewers: svoboda

87. An identifier for an object with internal linkage and an incomplete type is declared with a tentative definition (6.9.3).

Note: Removed from J.2. by N3347

88. A non-directive preprocessing directive is executed (6.10).

```
# 1234
// Undefined Behavior
```

Reviewers: svoboda

89. The token defined is generated during the expansion of a #if or #elif preprocessing directive, or the use of the defined unary operator does not match one of the two specified forms prior to macro replacement (6.10.2).

```
#define FOO 1

#define concat(uc1, uc2) uc1##uc2

#if concat(def, ined) FOO
// Undefined Behavior
```

Reviewers: svoboda

90. The #include preprocessing directive that results after expansion does not match one of the two header name forms (6.10.3).

```
#define str(s) # s
#define xstr(s) str(s)
#define INCFILE(n) z ## n

#include xstr(INCFILE(3).h)
// #include "z3.h", but fails to compile if s/xstr/str/;
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

91. The character sequence in an #include preprocessing directive does not start with a letter (6.10.3).

```
#include "2file.h"
// Undefined Behavior
```

Reviewers: svoboda

92. There are sequences of preprocessing tokens within the list of macro arguments that would otherwise act as preprocessing directives (6.10.5).

CERT C Rule PRE32-C 1st NCCE 2.3.1

Reviewers: svoboda

93. The result of the preprocessing operator # is not a valid character string literal (6.10.5.2).

```
#define str(s) # s
#define xstr(s) str(s)
#define x xstr(o'clock)
// Ill-formed, lone single quote
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

94. The result of the preprocessing operator ## is not a valid preprocessing token (6.10.5.3).

```
#define my_join(a, b) a ## b
char q[] = my_join(foo, bar);
// Ill-formed, unless 'foobar' is an identifier (or similar preprocessing token)
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

95. The #line preprocessing directive that results after expansion does not match one of the two well-defined forms, or its digit sequence specifies zero or a number greater than 2147483647 (6.10.6).

```
#line 10000000000
// Undefined Behavior, > 2^32
```

Reviewers: svoboda

96. A non-STDC #pragma preprocessing directive that is documented as causing translation failure or some other form of undefined behavior is encountered (6.10.8).

```
#pragma options align=foobar
// Undefined Behavior with GCC on Darwin platforms (e.g. Mac). See
// https://gcc.gnu.org/onlinedocs/gcc/Darwin-Pragmas.html
```

Reviewers: svoboda

97. A #pragma STDC preprocessing directive does not match one of the well-defined forms (6.10.8).

```
#pragma STDC FP_CONTRACT FOOBAR
// Undefined Behavior
```

Reviewers: svoboda

98. The name of a predefined macro, or the identifier defined, is the subject of a #define or #undef preprocessing directive (6.10.10).

```
#define __FILE__ source.c
// Undefined Behavior
```

Reviewers: svoboda

99. An attempt is made to copy an object to an overlapping object by use of a library function, other than as explicitly allowed (e.g., memmove) (Clause 7).

CERT C Rule EXP43-C 4th NCCE 4.11.3.1

Reviewers: svoboda

100. A file with the same name as one of the standard headers, not provided as part of the implementation, is placed in any of the standard places that are searched for included source files (7.1.2).

CERT C Rec PRE04-C 1st NCCE

Reviewers: svoboda

101. A header is included within an external declaration or definition (7.1.2).

```
struct st {
  int s;
#include <stdio.h>
// Undefined Behavior
};
```

Reviewers: svoboda

102. A function, object, type, or macro that is specified as being declared or defined by some standard header is used before any header that declares or defines it is included (7.1.2).

```
#include <stdio.h>

double sin(double x) {
  return x - 3.0;
}

#include <math.h>
// Undefined Behavior, sin() already defined

int main(void) {
  double p = 3.14; // almost pi
  printf("sin( %f) is %f\n", p, sin(p));
  return 0;
}
```

Reviewers: svoboda

103. A standard header is included while a macro is defined with the same name as a keyword (7.1.2).

```
#define do int x;

#include <stdio.h>
// Undefined Behavior, sin() already defined
```

Reviewers: svoboda

104. The program attempts to declare a library function itself, rather than via a standard header, but the declaration does not have external linkage (7.1.2).

CERT C Rule DCL37-C 4th NCCE 3.4.7

Reviewers: svoboda, s.maddanimath

105. The program declares or defines a reserved identifier, other than as allowed by 7.1.4 (7.1.3).

TS17961 5.44 [resident] EXAMPLE 1, 2, 4, 6, 8

Reviewers: svoboda

106. The program removes the definition of a macro whose name begins with an underscore and either an uppercase letter or another underscore (7.1.3).

```
#undef __STDC_UTF_32__
// Undefined Behavior
```

Reviewers: svoboda, j.myers

107. An argument to a library function has an invalid value or a type not expected by a function with a variable number of arguments (7.1.4).

```
#include <stdio.h>

printf("Hello, %s!\n", 123);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

108. The pointer passed to a library function array parameter does not have a value such that all address computations and object accesses are valid (7.1.4).

TS17961 5.20 [libptr] EXAMPLE 1, 2, 3, 4, 5.22 [invptr] EXAMPLE 13, 5.31 [nonnullcs] EXAMPLE 2, 5.37 [taintstrcpy] EXAMPLE, 5.40 [taintformatio] EXAMPLE 2

Reviewers: svoboda

109. The macro definition of assert is suppressed to access an actual function (7.2).

CERT C Rule MSC38-C 1st NCCE 15.5.1

Reviewers: svoboda

110. The argument to the assert macro does not have a scalar type (7.2).

```
#include <assert.h>

int a[5];
assert(a);  // Undefined Behavior
```

TODO: COMPILABLE EXAMPLE? (ideally one that replaces a with a struct or union)

Reviewers: svoboda

111. The CX_LIMITED_RANGE, FENV_ACCESS, or FP_CONTRACT pragma is used in any context other than outside all external declarations or preceding all explicit declarations and statements inside a compound statement (7.3.4, 7.6.1, 7.12.2).

```c
void f(void) {
  int a;
  #pragma FP_CONTRACT OFF
  // Undefined Behavior
}
```

TODO: COMPILABLE EXAMPLE?

Reviewers: uecker, j.myers, svoboda

112. The value of an argument to a character handling function is neither equal to the value of EOF nor representable as an unsigned char (7.4).

TS17961 5.32 [chrsgnext] EXAMPLE

Reviewers: svoboda

113. A macro definition of errno is suppressed to access an actual object, or the program defines an identifier with the name errno (7.5).

TS17961 5.44 [resident] EXAMPLE 1, CERT C Rule MSC38-C 2nd NCCE 15.5.3

Reviewers: svoboda

114. Part of the program tests floating-point status flags, sets floating-point control modes, or runs under non-default mode settings, but was translated with the state for the FENV_ACCESS pragma "off" (7.6.1).

```c
#pragma STDC FENV_ACCESS OFF

int set_excepts;
feclearexcept(FE_INVALID);
set_excepts = fetestexcept(FE_INVALID); // Undefined Behavior
if (set_excepts & FE_INVALID) {
  puts("INVALID");
}
```

Reviewers: svoboda

115. The exception-mask argument for one of the functions that provide access to the floating-point status flags has a nonzero value not obtained by bitwise OR of the floating-point exception macros (7.6.4).

```
#pragma STDC FENV_ACCESS ON
int set_excepts = fetestexcept(1); // Undefined Behavior
```

Reviewers: svoboda

116. The fesetexceptflag function is used to set floating-point status flags that were not specified in the call to the fegetexceptflag function that provided the value of the corresponding fexcept_t object (7.6.4.5).

```
#pragma STDC FENV_ACCESS ON

fexcept_t excepts;
fesetexceptflag(&excepts,FE_ALL_EXCEPT);
// Undefined Behavior, excepts not set by fegetexceptflag
```

Reviewers: svoboda

117. The argument to fesetenv or feupdateenv is neither an object set by a call to fegetenv or feholdexcept, nor is it an environment macro (7.6.6.3, 7.6.6.4).

```
#pragma STDC FENV_ACCESS ON

fenv_t fenv;
fesetenv(&fenv);
// Undefined Behavior, fenv not initialized
```

Reviewers: svoboda

118. The value of the result of an integer arithmetic or conversion function cannot be represented (7.8.2.1, 7.8.2.2, 7.8.2.3, 7.8.2.4, 7.24.6.1, 7.24.6.2, 7.24.1).

```
intmax_t x = INTMAX_MIN;
intmax_t px = imaxabs(x);  // Undefined Behavior
```

Reviewers: svoboda

119. The program modifies the string pointed to by the value returned by the setlocale function (7.11.1.1).

TS17961 5.29 [libmod] EXAMPLE 1

Reviewers: svoboda

120. A pointer returned by the setlocale function is used after a subsequent call to the function, or after the calling thread has exited (7.11.1.1).

```
#include <locale.h>

char *locale1 = setlocale(LC_ALL, "");
assert(locale1);
/* ... */
char *locale2 = setlocale(LC_ALL, "");
int size = strlen(locale1);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

121. The program modifies the structure pointed to by the value returned by the localeconv function (7.11.2.1).

TS17961 5.29 [libmod] EXAMPLE 2

Reviewers: svoboda

122. A macro definition of math_errhandling is suppressed or the program defines an identifier with the name math_errhandling (7.12).

```
#include <math.h>

int math_errhandling;  // Undefined Behavior
```

Reviewers: svoboda

123. An argument to a floating-point classification or comparison macro is not of real floating type (7.12.3, 7.12.17).

```
double complex p = CMPLX( 2, 3); // 2 + 3i
if (isfinite(p) // Undefined Behavior
  // ...
```

Reviewers: svoboda

124. A macro definition of setjmp is suppressed to access an actual function, or the program defines an external identifier with the name setjmp (7.13).

```
#include <setjmp.h>

int setjmp(char* foo);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

125. An invocation of the setjmp macro occurs other than in an allowed context (7.13.2.1).

CERT C Rec MSC22-C 1st NCCE

Reviewers: svoboda

126. The longjmp function is invoked to restore a nonexistent environment (7.13.2.1).

CERT C Rec MSC22-C,2nd NCCE

Reviewers: svoboda

127. After a longjmp, there is an attempt to access the value of an object of automatic storage duration that does not have volatile-qualified type, local to the function containing the invocation of the corresponding setjmp macro, that was changed between the setjmp invocation and longjmp call (7.13.2.1).

CERT C Rec MSC22-C 3rd NCCE

Reviewers: svoboda

128. The program specifies an invalid pointer to a signal handler function (7.14.1.1).

```
#include <signal.h>

void *handler = NULL;
signal(SIG_IGN, handler);  // Undefined Behavior
```

Reviewers: svoboda

129. A signal handler returns when the signal corresponded to a computational exception (7.14.1.1).

CERT C Rule: SIG35-C,1st NCCE

Reviewers: svoboda

130. A signal handler called in response to SIGFPE, SIGILL, SIGSEGV, or any other implementation-defined value corresponding to a computational exception returns (7.14.1.1).

CERT C Rule SIG35-C 1st NCCE 12.4.1

Reviewers: svoboda

131. A signal occurs as the result of calling the abort or raise function, and the signal handler calls the raise function (7.14.1.1).

TS17961 5.5 [asyncsig] EXAMPLE 2, CERT C Rule SIG30-C 3rd NCCE 12.1.5

Reviewers: svoboda

132. A signal occurs other than as the result of calling the abort or raise function, and the signal handler refers to an 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 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 (for the same signal number) (7.14.1.1).

TS17961 5.3 [accsig] EXAMPLE, 5.5 [asyncsig] EXAMPLE 1, 3

Reviewers: svoboda

133. The value of errno is referred to after a signal occurred other than as the result of calling the abort or raise function and the corresponding signal handler obtained a SIG_ERR return from a call to the signal function (7.14.1.1).

CERT C Rule ERR32-C 1st NCCE 13.2.1

Reviewers: svoboda

134. A signal is generated by an asynchronous signal handler (7.14.1.1).

CERT C Rule SIG30-C 2nd NCCE 12.1.3

Reviewers: svoboda

135. The signal function is used in a multi-threaded program (7.14.1.1).

CERT C Rule CON37-C 1st NCCE 14.8.1

Reviewers: svoboda

136. A function with a variable number of arguments attempts to access its varying arguments other than through a properly declared and initialized va_list object, or before the va_start macro is invoked (7.16, 7.16.1.1, 7.16.1.4).

```c
#include <stdio.h>
#include <stdarg.h>

void f(int last, ...) {
  va_list args;
  int number = va_arg(args, int);  // Undefined Behavior
  va_start( args, last);           // Oops, should precede va_args!
  printf("The number is %d\n", number);
  va_end(args);
}

int main(void) {
  f(1, 2, 3);
  return 0;
}
```

Reviewers: svoboda, j.myers

137. The macro va_arg is invoked using the parameter ap that was passed to a function that invoked the macro va_arg with the same parameter (7.16).

```c
#include <stdio.h>
#include <stdarg.h>

void g(va_list args) {
  int number = va_arg(args, int);
  printf("The first variadic number is %d\n", number);
}

void f(int last, ...) {
  va_list args;
  va_start( args, last);
  g(args);
  int number = va_arg(args, int);  // Undefined Behavior
  printf("The next variadic number is %d\n", number);
  va_end(args);
}

void main(void) {
  f(1, 2, 3);
}
```

Reviewers: svoboda, j.myers

138. A macro definition of va_start, va_arg, va_copy, or va_end is suppressed to access an actual function, or the program defines an external identifier with the name va_copy or va_end (7.16.1).

```
#include <stdarg.h>

// Undefined Behavior
#undef va_arg

int va_arg(void) {return 123;}
```

Reviewers: svoboda

139. The va_start or va_copy macro is invoked without a corresponding invocation of the va_end macro in the same function, or vice versa (7.16.1, 7.16.1.2, 7.16.1.3, 7.16.1.4).

```
#include <stdarg.h>

void f(int last, ...) {
  va_list args;
  va_start( args, last);
  /* Undefined Behavior, missing va_end(args) */
}
```

Reviewers: svoboda, j.myers

140. The va_arg macro is invoked when there is no actual next argument, or with a specified type that is not compatible with the promoted type of the actual next argument, with certain exceptions (7.16.1.1).

CERT C Rec DCL10-C 1st NCCE, CERT C Rule MSC39-C 1st NCCE 15.6.1

Reviewers: svoboda

141. The type parameter to the va_arg macro does not name an object type (7.16.1.1).

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

int main(void) {
  int x[] = {1, 2};
  int y[] = {3, 4};
  my_printf("My data", x, y);
}

void my_printf(const char* prefix, ...) {
  va_list args;
  int *x;
  int *y;
  va_start( args, prefix);
  x = va_arg( args, int*);   // Valid
  y = va_arg( args, void()); // Undefined Behavior
  va_end( args);
  printf("%s: [%d, %d], [%d, %d]\n", prefix, x[0], x[1], y[0], y[1]);
}
```

TODO: COMPILABLE EXAMPLE? (ideally one that replaces a with a function pointer example)

Reviewers: svoboda, UBSG

142. Using a null pointer constant in form of an integer expression as an argument to a ... function and then interpreting it as a void* or char* (7.16.1.1).

CERT C Rec DCL10-C 1st NCCE, CERT C Rule MSC39-C 1st NCCE 15.6.1

Reviewers: svoboda

143. The va_copy or va_start macro is invoked to initialize a va_list that was previously initialized by either macro without an intervening invocation of the va_end macro for the same va_list (7.16.1.2, 7.16.1.4).

```
#include <stdarg.h>

void f(int last, ...) {
  va_list args;
  va_start( args, last);
  va_start( args, last);   // Undefined Behavior
  va_end(args);
}
```

Reviewers: svoboda, j.myers

144. The va_start macro is invoked with additional arguments that include unbalanced parentheses, or unrecognized preprocessing tokens (7.16.1.4).

CERT C Rec DCL10-C 1st NCCE, CERT C Rule MSC39-C 1st NCCE 15.6.1

Reviewers: svoboda

C18-145. The parameter parmN of a va_start macro is declared with the register storage class, with a function or array type, or with a type that is not compatible with the type that results after application of the default argument promotions (7.16.1.4).

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

void f(float last, ...) {
  register int wrong = 0;
  va_list args;
  va_start( args, wrong);  // Undefined Behavior
  int number = va_arg(args, int);  // Undefined Behavior, no arg!
  printf("The next variadic number is %d\n", number);
  va_end(args);
}
```

Reviewers: svoboda

145. The macro definition of a generic function is suppressed to access an actual function (7.17.1, 7.18).

```
#include <stdbit.h>

#undef stdc_has_single_bit
// Undefined Behavior
```

Reviewers: svoboda

146. The type parameter of an offsetof macro defines a new type (7.21).

```
int plus(int a, int b) {
  return a+b;
}

typedef int binary_f(int, int);
binary_f *add = plus;

typedef struct st {
  int num1;
  int num2;
} binary_s;

size_t z = offsetof( int (*)(int, int), num2);
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

147. When program execution reaches an unreachable() macro invocation (7.21.1).

```
if (x == 0) {
  unreachable(); // Undefined Behavior
}
```

Reviewers: svoboda

148. Arbitrarily copying or changing the bytes of or copying from a non-null pointer into a nullptr_t object and then reading that object (7.21.2).

```
int i = 1;
int* pi = &i;
nullptr_t n = nullptr;
memcpy(&n, pi, sizeof(n));
memcpy(pi, &n, sizeof(int)); // Undefined Behavior
```

Reviewers: svoboda

149. The member-designator parameter of an offsetof macro is an invalid right operand of the . operator for the type parameter, or designates a bit-field (7.21).

```
typedef struct st {
  int num1;
  int num2;
} binary_s;

size_t z = offsetof( binary_s, num3); // Oops, meant num2
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

150. The argument in an instance of one of the integer-constant macros is not a decimal, octal, or hexadecimal constant, or it has a value that exceeds the limits for the corresponding type (7.22.4).

```
  unsigned char i = UINT8_C(0x123); // Undefined Behavior, 0x123 > 2^8
```

Reviewers: svoboda

151. A byte input/output function is applied to a wide-oriented stream, or a wide character input/output function is applied to a byte-oriented stream (7.23.2).

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

```
int main(void) {
  FILE* in = fopen("foo.txt", "r");

  wchar_t wide_line[80];
  fgetws(wide_line, sizeof(wchar_t) * sizeof(wide_line), in);
  // the stream is now oriented for wide characters
  wprintf(L"The first line is: %ls", wide_line);

  char line[80];
  fgets(line, sizeof(line), in);  // Undefined Behavior
  printf("The second line is: %s", line);

  return 0;
}
```

Reviewers: svoboda, j.myers

152. Use is made of any portion of a file beyond the most recent wide character written to a wide-oriented stream (7.23.2).

Note: Removed from J.2. by N3064

153. The value of a pointer to a FILE object is used after the associated file is closed (7.23.3).

CERT C Rule FIO46-C 1st NCCE 10.12.1

Reviewers: svoboda

154. The stream for the fflush function points to an input stream or to an update stream in which the most recent operation was input (7.23.5.2).

```
#include <stdio.h>

FILE* f = fopen("foo", "r");
fflush(f);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

155. The string pointed to by the mode argument in a call to the fopen function does not exactly match one of the specified character sequences (7.23.5.3).

```
#include <stdio.h>

FILE* f = fopen("foo", "read");  // Undefined Behavior
```

Reviewers: svoboda, j.myers

156. An output operation on an update stream is followed by an input operation without an intervening call to the fflush function or a file positioning function, or an input operation on an update stream is followed by an output operation with an intervening call to a file positioning function (7.23.5.3).

TS17961 5.27 [ioileave] EXAMPLE

Reviewers: svoboda

157. An attempt is made to use the contents of the array that was supplied in a call to the setvbuf function (7.23.5.6).

```c
#include <stdio.h>

#define SIZE 1024
char buf[SIZE];
FILE _file = fopen("foo", "r");
if (file == 0 ||
    setvbuf(file, buf, buf ? IOFBF : IONBF, SIZE) != 0) {
  /* Handle error */
}
/* ... */
buf[0] = '\0';  // Undefined Behavior
```

Reviewers: svoboda, j.myers

158. There are insufficient arguments for the format in a call to one of the formatted input/output functions, or an argument does not have an appropriate type (7.23.6.1, 7.23.6.2, 7.31.2.1, 7.31.2.2).

TS17961 5.45 [invfmtstr] EXAMPLE, CERT C Rec DCL10-C 2nd NCCE

Reviewers: svoboda

159. The format in a call to one of the formatted input/output functions or to the strftime or wcsftime function is not a valid multibyte character sequence that begins and ends in its initial shift state (7.23.6.1, 7.23.6.2, 7.29.3.5, 7.31.2.1, 7.31.2.2, 7.31.5.1).

```c
#include <stdio.h>
#include <locale.h>

setlocale(LC_ALL, "UTF-8");
/* In UTF-8: the Euro symbol == '€' == U+20AC == \xE2 \x82 \xAC == \342 \202 \254 */
```

```
const char s[] = {'\xE2', '\0'};  // invalid UTF-8
printf(s);   // Undefined Behavior in UTF-8 locale
```

Reviewers: svoboda, UBSG, j.myers

160. In a call to one of the formatted output functions, a precision appears with a conversion specifier other than those described (7.23.6.1, 7.31.2.1).

```
#include <stdio.h>

void f(char c) {
  printf("%.3c", c);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

161. A conversion specification for a formatted output function uses an asterisk to denote an argument-supplied field width or precision, but the corresponding argument is not provided (7.23.6.1, 7.31.2.1).

```
#include <stdio.h>

void f(int i) {
  printf("%*iX", i);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

162. A conversion specification for a formatted output function uses a # or 0 flag with a conversion specifier other than those described (7.23.6.1, 7.31.2.1).

```
#include <stdio.h>

void f(char* s) {
  printf("%0s", s);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

163. A conversion specification for one of the formatted input/output functions uses a length modifier with a conversion specifier other than those described (7.23.6.1, 7.23.6.2, 7.31.2.1, 7.31.2.2).

```
#include <stdio.h>

void f(int* pi) {
  printf("%lp", pi);  /* 'l' not defined for %p */ }
```

Reviewers: svoboda, j.myers

164. An s conversion specifier is encountered by one of the formatted output functions, and the argument is missing the null terminator (unless a precision is specified that does not require null termination) (7.23.6.1, 7.31.2.1).

TS17961 5.31 [nonnullcs] EXAMPLE 1

Reviewers: svoboda

165. An n conversion specification for one of the formatted input/output functions includes any flags, an assignment-suppressing character, a field width, or a precision (7.23.6.1, 7.23.6.2, 7.31.2.1, 7.31.2.2).

```
#include <stdio.h>

void f(int* pi) {
  printf("%-n", pi);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

166. A % conversion specifier is encountered by one of the formatted input/output functions, but the complete conversion specification is not exactly %% (7.23.6.1, 7.23.6.2, 7.31.2.1, 7.31.2.2).

```
#include <stdio.h>

void f(void) {
  printf("%-%");  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

167. An invalid conversion specification is found in the format for one of the formatted input/output functions, or the strftime or wcsftime function (7.23.6.1, 7.23.6.2, 7.29.3.5, 7.31.2.1, 7.31.2.2, 7.31.5.1).

```
#include <stdio.h>

void f(int i) {
  printf("%q", i);  /* %q not defined */
}
```

Reviewers: svoboda, j.myers

168. The number of characters or wide characters transmitted by a formatted output function (or written to an array, or that would have been written to an array) is greater than INT_MAX (7.23.6.1, 7.31.2.1).

```
#include <stdio.h>

void f(int i) {
  printf("%*iX", INT_MAX, i);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

169. The number of input items assigned by a formatted input function is greater than INT_MAX (7.23.6.2, 7.31.2.2).

```
#include <stdio.h>
#include <limits.h>
#include <stdarg.h>

/* Assume this function is called with >INT_MAX arguments */
void f(int unused, ...) {
  va_list args;
  va_start( args, unused);

  static unsigned int size = (unsigned int) INT_MAX + 1U;

  char format_string[size*2 + 1]; // will be "%c%c%c..."
  // This assumes that SIZE_MAX > 2*UINT_MAX
  for (unsigned int i = 0; i < size; i += 2+) {
    format_string[i] = '%';
    format_string[i+1] = 'c';
  }
  format_string[size*2] = '\0';

  vscanf(format_string, args);    // Undefined Behavior
}
```

Reviewers: svoboda

170. The result of a conversion by one of the formatted input functions cannot be represented in the corresponding object, or the receiving object does not have an appropriate type (7.23.6.2, 7.31.2.2).

CERT C Rec INT05-C 1st NCCE

Reviewers: svoboda

171. A c, s, or [conversion specifier is encountered by one of the formatted input functions, and the array pointed to by the corresponding argument is not large enough to accept the input sequence (and a null terminator if the conversion specifier is s or [) (7.23.6.2, 7.31.2.2).

TS17961 5.40 [taintformatio] EXAMPLE 1

Reviewers: svoboda

172. A c, s, or [conversion specifier with an l qualifier is encountered by one of the formatted input functions, but the input is not a valid multibyte character sequence that begins in the initial shift state (7.23.6.2, 7.31.2.2).

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

setlocale(LC_ALL, "UTF-8");
/* In UTF-8: the Euro symbol == '€' == U+20AC == \xE2 \x82 \xAC == \342 \202 \254 */
const char invalid[] = {'\xE2', '\0'};  // invalid UTF-8
char c;
sscanf(invalid, "%c", &c);  // Undefined Behavior in UTF-8 locale
```

Reviewers: svoboda, j.myers

173. The input item for a %p conversion by one of the formatted input functions is not a value converted earlier during the same program execution (7.23.6.2, 7.31.2.2).

```
#include <stdio.h>

char addr[] = "0x12345678"; // not produced by this code
void* ptr;
sscanf( addr, "%p", &ptr);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

174. The vfprintf, vfscanf, vprintf, vscanf, vsnprintf, vsprintf, vsscanf, vfwprintf, vfwscanf, vswprintf, vswscanf, vwprintf, or vwscanf function is called with an improperly initialized va_list argument, or the argument is used (other than in an invocation of va_end) after the function returns (7.23.6.8, 7.23.6.9, 7.23.6.10, 7.23.6.11, 7.23.6.12, 7.23.6.13, 7.23.6.14, 7.31.2.5, 7.31.2.6, 7.31.2.7, 7.31.2.8, 7.31.2.9, 7.31.2.10).

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

```
void f(int first, ...) {
  int local = 0;
  va_list args;
  va_start( args, local);          // Oops, not first fixed arg
  vprintf("Hello, %s\n", args);    // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

175. The contents of the array supplied in a call to the fgets or fgetws function are used after a read error occurred (7.23.7.2, 7.31.3.2).

```
const int buf_size = 100;
char buf[buf_size];
FILE* f = fopen("foo", "r");
if (f == NULL) {
  printf("Can't open foo\n");
}
fgets( buf, buf_size, f);        // Oops, no check for failure
printf("Buf is now %s\n", buf); // Undefined Behavior if fgets() returned NULL
```

Reviewers: svoboda

176. The n parameter is negative or zero for a call to fgets or fgetws. (7.23.7.2, 7.31.3.2).

```
const int buf_size = -1;      // Oops, should be > 0
char buf[buf_size];
FILE* f = fopen("foo", "r");
if (f == NULL) {
  printf("Can't open foo\n");
}
fgets( buf, buf_size, f);     // Undefined Behavior
```

Reviewers: svoboda

177. The file position indicator for a binary stream is used after a call to the ungetc function where its value was zero before the call (7.23.7.10).

```
FILE* f = fopen("foo", "rb");
if (f == NULL) {
  printf("Can't open foo\n");
}
assert(ftell( f) == 0);
int c = 'A';
c = ungetc( c, f); // Undefined Behavior
```

Reviewers: svoboda

178. The file position indicator for a stream is used after an error occurred during a call to the fread or fwrite function (7.23.8.1, 7.23.8.2).

```
FILE* f = fopen("foo", "rb");
if (f == NULL) {
  printf("Can't open foo\n");
}
const int buf_size = 125;
char buffer[buf_size];
size_t bytes = fread( buffer, 1, buf_size, f);
printf("Read %ld bytes\n", bytes);
if (bytes < buf_size) { // uh-oh, error
  long pos = ftell( f); // Undefined Behavior if fread() had error
  printf("File is at %lu position \n", pos);
}
```

Reviewers: svoboda

179. A partial element read by a call to the fread function is used (7.23.8.1).

```
FILE* f = fopen("foo", "rb");
if (f == NULL) {
  printf("Can't open foo\n");
}
const int buf_size = 250;
wchar_t buffer[buf_size];
for (size_t i = 0; i < buf_size; i++) {
  buffer[i] = 0;
}
size_t bytes = fread( buffer, sizeof(wchar_t), buf_size, f);
printf("%lu bytes read\n", bytes);
if (bytes < buf_size) { // hmmm, less bytes read than expected
  printf("Possible partial byte read is %.02x, %c\n", buffer[bytes], buffer[bytes]);
  // Undefined Behavior if buffer has partially-read character
}
```

Reviewers: svoboda

180. The fseek function is called for a text stream with a nonzero offset and either the offset was not returned by a previous successful call to the ftell function on a stream associated with the same file or whence is not SEEK_SET (7.23.9.2).

```
FILE* f = fopen("foo", "r");
if (f == NULL) {
  printf("Can't open foo\n");
}
fseek(f, 5, SEEK_CUR);  // Undefined behavior
```

Reviewers: svoboda

181. The fsetpos function is called to set a position that was not returned by a previous successful call to the fgetpos function on a stream associated with the same file (7.23.9.3).

TS17961 5.41 [xfilepos] EXAMPLE

Reviewers: svoboda

182. A non-null pointer returned by a call to the calloc, malloc, realloc, or aligned_alloc function with a zero requested size is used to access an object (7.24.3).

```
#include <stdlib.h>
#include <assert.h>

size_t size = 0;
int _array = (int_) malloc(size * sizeof(int));
assert(array);
array[0] = 123;  // out-of-bounds write
```

Reviewers: svoboda, j.myers

183. The value of a pointer that refers to space deallocated by a call to the free or realloc function is used (7.24.3).

TS17961 5.2 [accfree] EXAMPLE 1,2,3

Reviewers: svoboda

184. The pointer argument to the free or realloc function is unequal to a null pointer and does not match a pointer earlier returned by a memory management function, or the space has been deallocated by a call to free or realloc (7.24.3.3, 7.24.3.7).

TS17961 5.23 [dblfree] EXAMPLE 1, 2, TS17961 5.34 [xfree] EXAMPLE 1, 2

Reviewers: svoboda

185. The value of the object allocated by the malloc function is used (7.24.3.6).

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

unsigned char* p = malloc(10);
if (p != 0) {
  printf("p[0] is %c\n", p[0]);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

186. The values of any bytes in a new object allocated by the realloc function beyond the size of the old object are used (7.24.3.7).

CERT C Rule EXP33-C 5th NCCE 4.3.11

Reviewers: svoboda, s.maddanimath

187. The program calls the exit or quick_exit function more than once, or calls both functions (7.24.4.4, 7.24.4.7).

CERT C Rule ENV32-C 1st NCCE 11.3.1

Reviewers: svoboda

188. During the call to a function registered with the atexit or at_quick_exit function, a call is made to the longjmp function that would terminate the call to the registered function (7.24.4.4, 7.24.4.7).

CERT C Rule ENV32-C 2nd NCCE 11.3.3

Reviewers: svoboda

189. The string set up by the getenv or strerror function is modified by the program (7.24.4.6, 7.26.6.3).

TS17961 5.29 [libmod] EXAMPLE 3, 4

Reviewers: svoboda

190. A signal is raised while the quick_exit function is executing (7.24.4.7).

```
void exit_handler(void) {
  raise(SIGINT); // Undefined Behavior
}

int main(void) {
```

```c
  if (atexit(exit_handler) != 0) {
    /* Handle error */
  }
  quick_exit(0);
  return 0;
}
```

Reviewers: svoboda

191. A command is executed through the system function in a way that is documented as causing termination or some other form of undefined behavior (7.24.4.8).

```c
system("ls /missing"); // should fail
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

192. A searching or sorting utility function is called with an invalid pointer argument, even if the number of elements is zero (7.24.5).

```c
#include <stdlib.h>

int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

int *arr = NULL;
qsort(arr, 0, sizeof(int), compare);
```

Reviewers: coates, svoboda

193. The comparison function called by a searching or sorting utility function alters the contents of the array being searched or sorted, or returns ordering values inconsistently (7.24.5).

```c
#include <stdlib.h>

int compare(const void *a, const void *b) {
    *(int *)a = *(int *)a + 1;  // Modify array contents!
    return (*(int *)a - *(int *)b);
}

int arr[] = {2, 1, 4, 1, 5, 9, 2, 6};
size_t n = sizeof(arr) / sizeof(arr[0]);
qsort(arr, n, sizeof(int), compare);
```

Reviewers: coates, svoboda

194. The array being searched by the bsearch function does not have its elements in proper order (7.24.5.1).

```
int compare(const void *a, const void *b) {
  return *(int*)a - *(int*)b;
}

int arr[] = {2, 3, 5, 11, 7}; // Oops, mis-ordered array
int n = sizeof(arr)/sizeof(arr[0]);
int key = 7;
int *result = (int*)bsearch(&key, arr, n, sizeof(int), compare);
// Undefined Behavior
```

Reviewers: svoboda

195. The current conversion state is used by a multibyte/wide character conversion function after changing the LC_CTYPE category (7.24.7).

fscanf() is defined in C23 s7.23.6.2. Example 6, (paragraph 23) in this section describes a hypothetical encoding with shift states, which we will use for this example:

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <memory.h>
#include <locale.h>

char str[50];
wchar_t wstr[50];
memset(wstr, 0, sizeof(wstr));
int counter = 0;
setlocale(LC_CTYPE, "HYPO");                // Use hypothetical encoding
/* Suppose standard input contains the single line: ↑X Y↓ */
fgets(str, sizeof(str), stdin);             // str == "↑X Y↓"
counter += mbtowc(wstr, str, 4);            // wstr == " X", upper state
setlocale(LC_CTYPE, "C");                   // state changed
counter += mbtowc(wstr, &str[counter], 4);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

196. A string or wide string utility function is instructed to access an array beyond the end of an object (7.26.1, 7.31.4).

CERT C Rule STR38-C 1st NCCE 8.6.1, 2nd NCCE 8.6.2, 3rd NCCE 8.6.4

Reviewers: svoboda

197. A string or wide string utility function is called with an invalid pointer argument, even if the length is zero (7.26.1, 7.31.4).

```c
#include <string.h>

char* c = 0;
int length = strlen(c); // Undefined Behavior
```

Reviewers: svoboda, j.myers

198. The contents of the destination array are used after a call to the strxfrm, strftime, wcsxfrm, or wcsftime function in which the specified length was too small to hold the entire null-terminated result (7.26.4.5, 7.29.3.5, 7.31.4.4.4, 7.31.5.1).

```c
char src[] = "This is a test";
const int string_size = strlen(src) + 1;
char dest[string_size - 1];   // Oops, too small
int length = strxfrm(dest, src, string_size);
// Undefined Behavior
```

Reviewers: svoboda

199. A sequence of calls of the strtok function is made from different threads (7.26.5.9).

```c
#include <string.h>
#include <threads.h>

int bar(void*) {
  char *t = strtok(NULL, "#,"); // Undefined Behavior
  return t[0];
}


char str[] = "?a???b,,,#c";
char *t = strtok(str, "?");
thrd_t thr;
if (thrd_success != thrd_create(&thr, bar, 0)) {
  /* Handle error */
}

t = strtok(NULL, ","); // Undefined Behavior

int retval;
if (thrd_success != thrd_join(thr, &retval)) {
  /* Handle error */
}
```

Reviewers: svoboda, CliveP, robin-rowe, j.myers

200. The first argument in the very first call to the strtok or wcstok is a null pointer (7.26.5.9, 7.31.4.5.8).

```
#include <string.h>

static char str[] = "?a???b,,,#c";
char *t;
t = strtok(NULL, "?");  // Undefined Behavior
```

Reviewers: svoboda, j.myers

201. A pointer returned by the strerror function is used after a subsequent call to the function, or after the calling thread has exited (7.26.6.3).

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

char* inval = strerror(EINVAL);
char* perm = strerror(EPERM);
printf("Invalid: %s\n", inval); /* Undefined Behavior, invalidated by perm */
printf("Permission: %s\n", perm);
```

Reviewers: svoboda, j.myers

202. The type of an argument to a type-generic macro is not compatible with the type of the corresponding parameter of the selected function (7.27).

```
double complex dc = CMPLX( 2, 3); // 2 + 3i
double result = dsqrt(dc);   // Undefined Behavior
```

Reviewers: svoboda

203. Arguments for generic parameters of a type-generic macro are such that some argument has a corresponding real type that is of standard floating type and another argument is of decimal floating type (7.27).

```
double d = 3.0;
_Decimal64 d64 = 2.0;
double result = remainder(d64, d);   // Undefined Behavior
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

204. Arguments for generic parameters of a type-generic macro are such that neither <math.h> and <complex.h> define a function whose generic parameters have the determined corresponding real type (7.27).

```
  double d = 3.0;
  float f = 2.0;
  double result = remainder(f, d);    // Undefined Behavior
```

Reviewers: svoboda

205. A complex argument is supplied for a generic parameter of a type-generic macro that has no corresponding complex function (7.27).

```
float complex fc = CMPLX(2, 3); // 2 + 3i
double result = ceil(fc);       // Undefined Behavior
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

206. A decimal floating argument is supplied for a generic parameter of a type-generic macro that expects a complex argument (7.27).

```
double d = 3.0;
_Decimal64 d64 = 2.0;
double result = remainder(d64, d);    // Undefined Behavior
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

207. A standard floating or complex argument is supplied for a generic parameter of a type-generic macro that expects a decimal floating type argument (7.27).

```
double d = 3.0;
_Decimal64 d64 = 2.0;
double result = d64div(d64, d);    // Undefined Behavior
```

TODO: COMPILABLE EXAMPLE?

Reviewers: svoboda

208. A non-recursive mutex passed to mtx_lock is locked by the calling thread (7.28.4.3).

```
#include <threads.h>


mtx_t m;
```

```
if (thrd_success != mtx_init(&m, mtx_plain)) {
  /* Handle error */
}

if (thrd_success != mtx_lock(&m)) {
  /* Handle error */
}

if (thrd_success != mtx_lock(&m)) {  // Undefined Behavior
  /* Handle error */
}

mtx_destroy(&m);
```

Reviewers: svoboda, j.myers

209. The mutex passed to mtx_timedlock does not support timeout (7.28.4.4).

```
#include <threads.h>
#include <time.h>

mtx_t m;
if (thrd_success != mtx_init(&m, mtx_plain)) { /* Oops, should be mtx_timed */
  /* Handle error */
}

struct timespec ts;
if (0 == timespec_get(&ts, TIME_UTC)) {
  /* Handle error */
}
ts.tv_sec += 1; /* 1 second from now */

if (thrd_success != mtx_timedlock(&m, &ts)) { // Undefined Behavior
  /* Handle error */
}

mtx_destroy(&m);
```

Reviewers: svoboda, j.myers

210. The mutex passed to mtx_unlock is not locked by the calling thread (7.28.4.6).

```
#include <threads.h>

mtx_t m;
```

```
if (thrd_success != mtx_init(&m, mtx_plain)) {
  /* Handle error */
}

if (thrd_success != mtx_unlock(&m)) {   // Undefined Behavior
  /* Handle error */
}

mtx_destroy(&m);
```

Reviewers: svoboda, j.myers

211. The thread passed to thrd_detach or thrd_join was previously detached or joined with another thread (7.28.5.3, 7.28.5.6).

CERT C Rule CON39-C 1st NCCE 14.10.1

Reviewers: svoboda

212. The tss_create function is called from within a destructor (7.28.6.1).

```
#include <threads.h>

void destructor(void* arg) {
  tss_t key;
  if (thrd_success != tss_create(&key, 0)) { // Undefined Behavior
    /* Handle error */
  }
}

int func(void*) {
  tss_t key;
  if (thrd_success != tss_create(&key, destructor)) {
    /* Handle error */
  }
  static char str[] = "Hello";
  tss_set(key, str);
  return 0;
}

int foo(void*) {
  thrd_t thr;
  if (thrd_success != thrd_create(&thr, func, 0)) {
    /* Handle error */
  }
  int retval;
  if (thrd_success != thrd_join(thr, &retval)) {
    /* Handle error */
  }
  return 0;
}
```

Reviewers: svoboda, j.myers

213. The key passed to tss_delete, tss_get, or tss_set was not returned by a call to tss_create before the thread commenced executing destructors (7.28.6.2, 7.28.6.3, 7.28.6.4).

```
#include <threads.h>

void destructor(void* arg) {
  tss_t key;
  if (thrd_success != tss_create(&key, 0)) { // Undefined Behavior
    /* Handle error */
  }
}

int func(void*) {
  tss_t key;
  static char str[] = "Hello";
  tss_set(key, str); /* Undefined Behavior, key not initialized by tss_create() */
  return 0;
}

int foo(void*) {
  thrd_t thr;
  if (thrd_success != thrd_create(&thr, func, 0)) {
    /* Handle error */
  }

  int retval;
  if (thrd_success != thrd_join(thr, &retval)) {
    /* Handle error */
  }
  return 0;
}
```

Reviewers: svoboda, j.myers

214. An attempt is made to access the pointer returned by the time conversion functions after the thread that originally called the function to obtain it has exited (7.29.3).

```
#include <stdio.h>
#include <threads.h>
#include <time.h>

char* now = 0;

int bar(void*) {
  time_t n1;
  if ((time_t) -1 == time(&n1)) {
    /* Handle error */
  }
  struct tm* n2 = localtime(&n1);
  now = asctime(n2);
  return 0;
}

void foo(void) {
  thrd_t thr;
  if (thrd_success != thrd_create(&thr, bar, 0)) {
    /* Handle error */
  }

  int retval;
  if (thrd_success != thrd_join(thr, &retval)) {
    /* Handle error */
  }

  printf("The time is %s\n", now); // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

215. At least one member of the broken-down time passed to asctime contains a value outside its normal range, or the calculated year exceeds four digits or is less than the year 1000 (7.29.3.1).

CERT C Rule MSC33-C 1st NCCE 15.3.1

Reviewers: svoboda

216. The argument corresponding to an s specifier without an l qualifier in a call to the fwprintf function does not point to a valid multibyte character sequence that begins in the initial shift state (7.31.2.11).

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

setlocale(LC_ALL, "UTF-8");
/* In UTF-8: the Euro symbol == '€' == U+20AC == \xE2 \x82 \xAC == \342 \202 \254 */
const char invalid[] = {'\xE2', '\0'};    // invalid UTF-8
fwprintf(stdout, L"The string is %ls\n", invalid);
/* Undefined Behavior in UTF-8 locale */
```

Reviewers: svoboda, j.myers

217. In a call to the wcstok function, the object pointed to by ptr does not have the value stored by the previous call for the same wide string (7.31.4.5.8).

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

wchar_t input[20] = L"foo bar baz";
wchar_t* buffer;
wchar_t* token = wcstok(input, L" ", &buffer);
while (token) {
  wprintf(L"%ls\n", token);
  wcscpy(buffer, input);              // buffer changed
  token = wcstok(0, L" ", &buffer);   // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

218. An mbstate_t object is used inappropriately (7.31.6).

CERT C Rule EXP33-C 3rd NCCE 4.3.7

Reviewers: svoboda, s.maddanimath

219. The value of an argument of type wint_t to a wide character classification or case mapping function is neither equal to the value of WEOF nor representable as a wchar_t (7.32.1).

```
#include <wctype.h>

int main(void) {
  int flag = iswalpha(WINT_MIN);
  // Undefined Behavior if sizeof(wchar_t) < sizeof(wint_t)
```

Reviewers: svoboda, j.myers

220. The iswctype function is called using a different LC_CTYPE category from the one in effect for the call to the wctype function that returned the description (7.32.2.2.1).

See UB 191 for background on LC_CTYPE categories. (editor; UB 191 of which version?)

```c
#include <wchar.h>
#include <locale.h>

int f(wint_t wc) {
  wctype_t alpha = wctype("alpha");
  setlocale(LC_CTYPE, "C");     // state changed
  return iswctype(wc, alpha);   // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

221. The towctrans function is called using a different LC_CTYPE category from the one in effect for the call to the wctrans function that returned the description (7.32.3.2.1).

```c
#include <wchar.h>
#include <wctype.h>
#include <locale.h>

wint_t f(wint_t wc) {
  wctype_t lower = wctrans("tolower");
  setlocale(LC_CTYPE, "C");     // state changed
  return towctrans(wc, lower);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers