

Everywhere the Light Touches (And Doesn't): Undefined Behavior in C

Sam Stuewe

June 27, 2019



OVERVIEW

Behavior

A Mental Model

Real-World Example

Origin

Coping Mechanisms

Works Cited

BEHAVIOR

Four Categories of Non-portable Behavior:

- ▶ unspecified
- ▶ implementation-defined
- ▶ undefined
- ▶ locale-specific*

“EVERWHERE THE LIGHT TOUCHES”

- ▶ Unspecified (§3.4.4)
 - ▶ use of an unspecified value, or other behavior where this International Standard provides two or more possibilities and imposes no further requirements on which is chosen in any instance
 - ▶ Ex. the order in which arguments to a function are evaluated
- ▶ Implementation-defined (§3.4.1)
 - ▶ unspecified behavior where each implementation documents how the choice is made
 - ▶ Ex. the propagation of the high-order bit when right-shifting a signed integer

“THAT SHADOWY PLACE”

- ▶ Undefined (§3.4.3)
 - ▶ behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements
 - ▶ Ex. integer overflow*

POLL: #1

How many Undefined Behaviors are there?

POLL: #2

Where do you go to find out if something is UB?

SCAR



A “shall” or “shall not” requirement that appears outside of a constraint is violated (Annex J.2, Item 1)

“THAT SHADOWY PLACE”

- ▶ Undefined (§3.4.3)
 - ▶ behavior, **upon use** of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes **no requirements**
 - ▶ Ex. integer overflow*

“NO REQUIREMENTS”?

Things the compiler *might does not need to do*:

- ▶ generate comparable instructions
- ▶ print a diagnostic (warning/error)
- ▶ cease translation
- ▶ ignore the code entirely

“NO REQUIREMENTS”?

Things the compiler *can* (technically) do:

- ▶ generate instructions that always return the integer 42
- ▶ attempt to format your hard drive
- ▶ literally anything else, including nothing

“UPON USE”?

If any step in a program's execution has undefined behavior, then the entire execution is without meaning. This is important: it's not that evaluating $(1 << 32)$ has an unpredictable result, but rather that the entire execution of a program that evaluates this expression is meaningless. Also, it's not that the execution is meaningful up to the point where undefined behavior happens: the bad effects can actually precede the undefined operation. (*John Regehr, "A Guide to Undefined Behavior in C and C++"*)

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

int main() {
    int *p = (int*)malloc(sizeof(int));
    int *q = (int*)realloc(p, sizeof(int));
    *p = 1;
    *q = 2;
    if (p == q)
        printf("%d %d\n", *p, *q);
}
```

```
$ clang -O1 realloc.c; ./a.out  
1 2
```

POLL: #3 (OPEN-ENDED)

What would you require for the following cases?

- ▶ integer overflow
- ▶ integer division by 0
- ▶ a `_Noreturn`-marked function returns

Why would the C Committee have done this?

- ▶ enabling simpler implementation
- ▶ enabling performance optimizations
- ▶ no clear standard / competing standards
- ▶ no reasonable alternative
- ▶ laziness*

THE BAD NEWS

There is no good way to know that a codebase does not contain UB.

“RECLAIMING THE PRIDELANDS”

- ▶ “Hakuna Matata”?
- ▶ “use a different language”
- ▶ reading the standards
 - ▶ ISO-IEC 9899:2018 (>500 pages; 200\$US)
 - ▶ POSIX.1-2017 (>3000 pages; free)
 - ▶ Intel 64 / IA-32 Software Developer Manual (>4900 pages; free)
- ▶ compiler tooling
 - ▶ warning flags: `-Wall, -Wextra, etc.`
 - ▶ feature flags: `-fno-strict-aliasing, -fwrapv, etc.`
 - ▶ sanitizers: `-fsanitize=undefined`
- ▶ static analyzers: `splint, scan-build`
- ▶ formal verification: `frama-c`

- ▶ Mufasa image, uncredited (could not recover original source)
- ▶ Scar image, *The Lion King*
- ▶ Chris Lattner, “What Every C Programmer Should Know About Undefined Behavior”
- ▶ John Regehr, “A Guide to Undefined Behavior in C and C++”
- ▶ John Regehr, “Undefined Behavior Consequences Contest Winners”
- ▶ ISO-IEC 9899:2018, §3 and Annex J
- ▶ POSIX.1-2017