Doc. No.: P3403R0
Project: Programming Language - C++
Author: Andrew Tomazos
Audience: EWG
Date: 2024-08-18

The Undefined Behavior Question

The primary language design decision of the related proposals WG21:P1494, WG14:N3128 and WG21:P3352 can be summarized by posing the **Undefined Behavior Question (hereafter "The Question")** which is as follows:

"Should undefined operations be allowed to affect observable operations that happen before them?"

The Question can be answered either (a) YES, (b) NO; or (c) IT DEPENDS ON XYZ.

(If The Question is answered NO, then the first **part** of a **program,** that happens before the first undefined operation, continues to be required to **correctly** produce its observable behavior (despite the undefined operation). That is why this topic is also called "**Partial Program Correctness**".)

Q1. What's an operation?

A1. An operation is a piece of the execution of a program. The execution of a program consists of the execution of a set of operations. For each operation:

1. Control passes to the operation. (start)
2. The operation produces effects and behavior. (middle)
3. The operation completes. (end)

It is the central term of **operation**al semantics, which is the semantic style in which the C++ standard is specified.

Q2. What's an example of an operation in C++?

A2. An example of an operation is:

```
x = 42;
```

This is an assignment operation (aka “write” operation). It produces an effect: The value 42 is assigned to the variable x.

Q3. What’s an undefined operation?

A3. A subset of operations are identified by the standard as being undefined operations. They are more commonly known as operations that are “undefined behavior” or just “UB”.

Q4. What’s an example of an undefined operation?

A4. Examples of undefined operations include:

- Modifying a const variable through a const_cast
- [[assume(false)]]
- Dereferencing a null pointer
- Dividing an integer by zero
- Calling a standard library function with an unsatisfied precondition
- Overflowing a signed integer addition operation
- Reading an array out of bounds
- Data races / infinite loops
- Many others

Q5. What’s an observable operation?

A5. Observable operations are operations that produce observable behavior. These include operations that:

- Control interactive devices (eg display output or await/accept input).
- Writing data to files (and other filesystem modifications)
- Reads or writes of volatile variables
- Other similar things (eg sending packets to/from a network)

They are the behaviors that are at the intersection of (a) the standard requires them; (b) interact with the external world (i.e. external to the program); and (c) you can detect them (aka “see them”, “observe them”).

Observable operations cannot be inlined or optimized. The program must produce its observable behavior. (In fact, the observable behavior is all a program is required to produce.)

Q6. Why are volatile reads/writes observable operations?

A6. Volatile was designed so that you could connect memory addresses to external mechanisms. For example, you might have a hardware device register at a special address in

memory. By allocating a volatile variable at that memory address, it allows you to read/write to that hardware register with a normal C++ variable. This is similar to controlling an interactive device, so implementations are required to produce volatile reads/writes (they cannot be optimized away, reordered, folded etc).

Q7. What does it mean for an operation X to happen before an operation Y?

A7. If an operation X is required to happen before an operation Y, it just means that operation X has to complete before control passes to operation Y. So the effects of operation X are applied to the world prior to the effects of operation Y being applied to the world. For example:

```
int main() {
   print("abc");
   print("def");
}
```

This program prints “abcdef” and not “defabc” because `print(“abc”)` happens before `print(“def”)`

Q8. How do previous C and C++ standards answer The Question?

A8. The status of previous standards is as follows:

- The C11 standard (and previous) gave an ambiguous answer. There is no consensus on how it answered The Question.
- The C++23 standard (and previous) gave an unambiguous YES answer to The Question.
- The C23 standard gives an unambiguous NO answer to The Question (due to the adoption of WG14:N3128 by WG14)

Q9. Do undefined operations affect observable operations that happen before them, in practice?

A9. If an implementation is compliant with C23, then they are not allowed to. Prior to C23 incidents were extremely rare. The few known cases prior to C23 exclusively involved volatile reads/writes, and it is unclear whether they were intentional or accidental (compiler bugs). WG14 is satisfied that they are to be considered compiler bugs.

Q10. How should C++26 answer The Question?

A10. That is up to the committee to decide. EWG will be discussing this question in Wroclaw. The paper authors in this area are in contact and hope to reach consensus on a recommendation prior to Wroclaw.

Q11. What is the “abstract interpretation” / “abstract rule” of undefined behavior?

A11. The “abstract interpretation” and “abstract rule” both mean a YES answer to The Question. I.e. undefined operations are ALLOWED to affect observable operations that happen before them.

Q12. What is the “concrete interpretation” / “concrete rule” of undefined behavior?

A12. The “concrete interpretation” and “concrete rule” both mean a NO answer to The Question. I.e. undefined operations are BANNED from affecting observable operations that happen before them.

Q13. What’s an “observable checkpoint”?

A13. WG21:P1494R3 proposes creating a new language primitive called an “observable checkpoint”. It proposes answering The Question IT DEPENDS ON XYZ, where the XYZ is “if there is an observable checkpoint that happens between the observable operation and the undefined operation.”

If there is an observable checkpoint that happens after an observable operation and happens before an undefined operation - then the undefined operation is BANNED from affecting the observable operation. If there is no such observable checkpoint that happens between, then the undefined operation is ALLOWED to affect the observable operation.

Q14. What’s the alternative to “observable checkpoints”?

A14. WG14:N3128/WG21:P3352 both propose answering NO to The Question. This is a matter of changing the specification of undefined operations in the standard to say they cannot affect observable operations that happen before them. There are multiple different but equivalent ways that could be formulated, and there is some on-going discussion about which is the best way.

The way WG14:N3128 made the change to C23 was to say in part “All observable behavior shall appear as specified in this document even when there is a subsequent operation with undefined behavior in the execution of the program.”.

The proposed wording for WG21:P3352 (C++ version of WG14:N3128) has the same effect. It says in part “It follows that undefined operations cannot affect observable behaviors that happen before them. Assumptions implied by potential undefined operations cannot “reverse time travel” and alter an observable behavior that happens before“

References

[WG14:N3128]
N3128
Date: 2023-05-05
Title: Taming the Demons -- Undefined Behavior and Partial Program Correctness
Author: Martin Uecker

[WG21:P1494]
P1494R3: Partial program correctness
Audience: EWG; CWG; LEWG; SG22
S. Davis Herring <herring@lanl.gov>
Los Alamos National Laboratory
May 20, 2024

[WG21:P3352]
P3352R0 Taming the Demons (C++ version) – Undefined Behavior and Partial Program Correctness
Project: Programming Language - C++ (WG21)
Authors: Andrew Tomazos, Martin Uecker
Date: 2024-07-08
Audience: EWG, SG22
Target: C++26