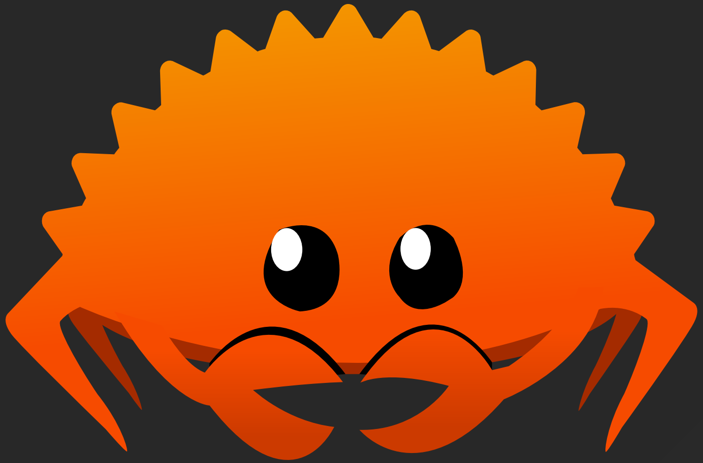


Rust

Intro & model pamięci

Łukasz Neumann

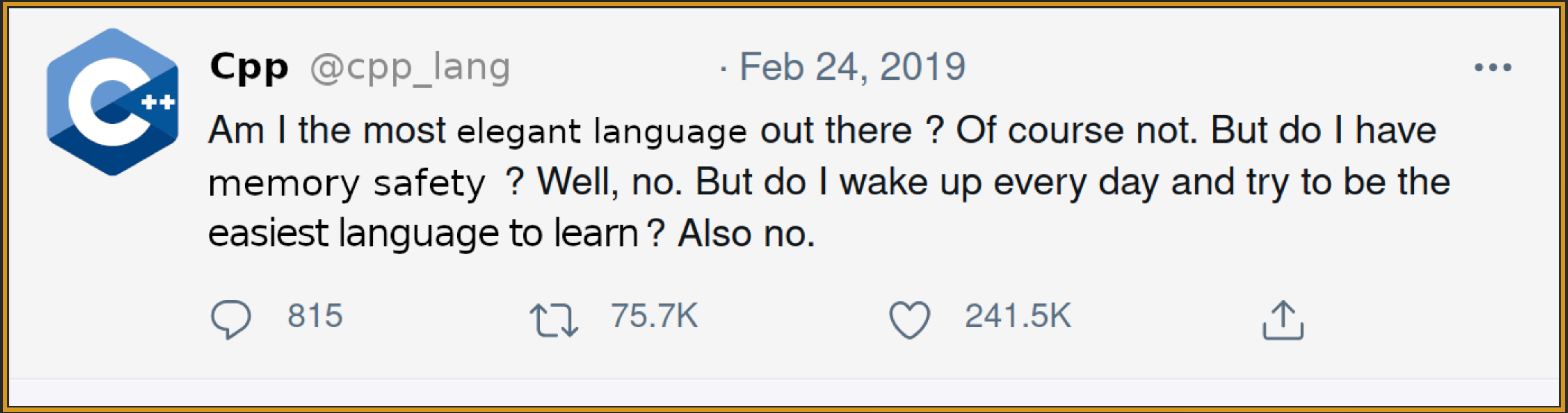
lukasz.neumann@pw.edu.pl

https://staff.elka.pw.edu.pl/~lneumann/rust_1.pdf

Anty-agenda

- unsafe
- makra ❤️
- społeczność
- FFI
- cross-platform
- async (?)

This is not a language war talk



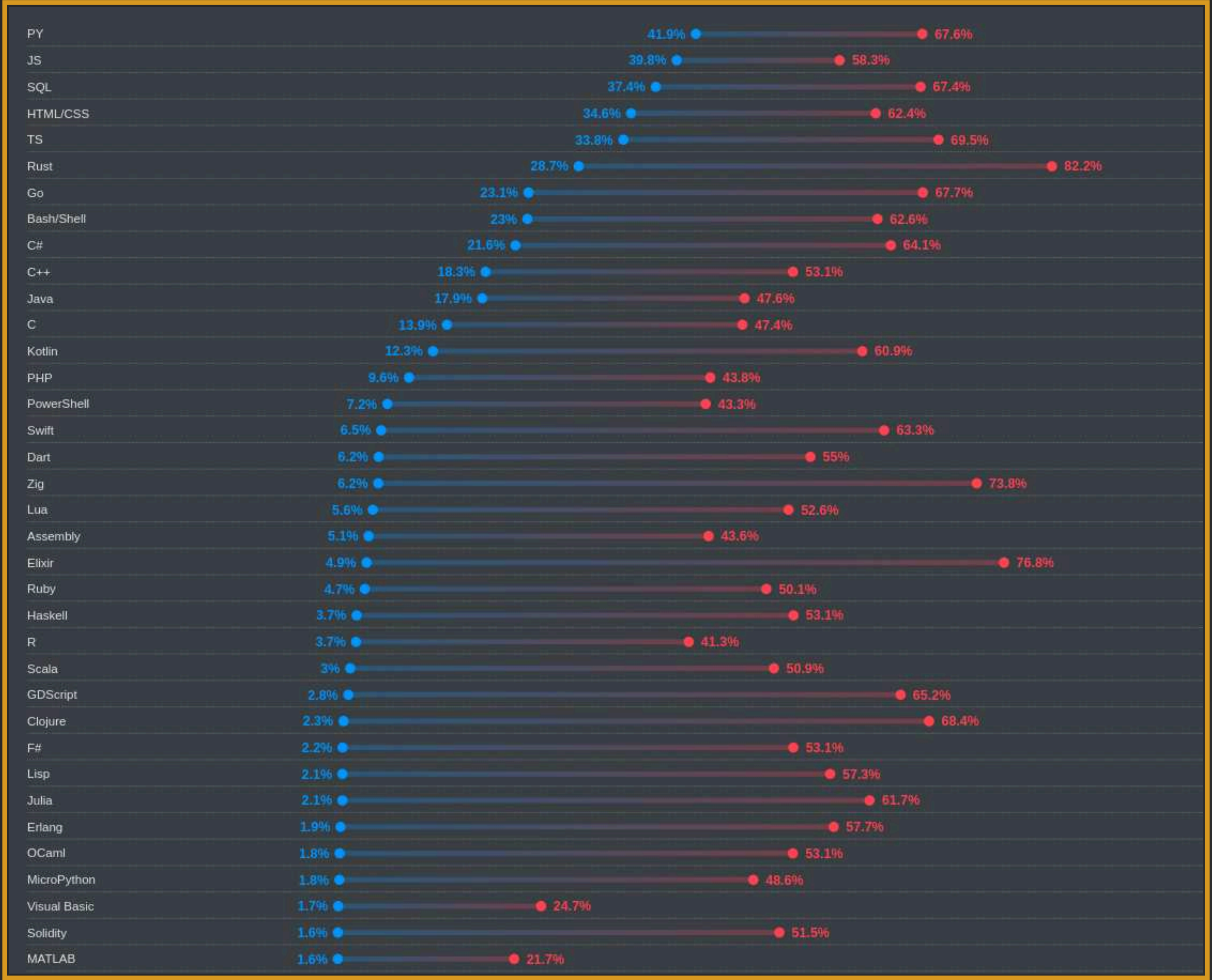
Cpp @cpp_lang · Feb 24, 2019

Am I the most elegant language out there ? Of course not. But do I have memory safety ? Well, no. But do I wake up every day and try to be the easiest language to learn ? Also no.

815 75.7K 241.5K

Hype driven development

https://survey.stackoverflow.co/2024/technology/#admired-and-desired



Źródło bólu w dużych projektach

- Chrome: 70% of high/critical vulnerabilities are memory unsafety
- Firefox: 72% of vulnerabilities in 2019 are memory unsafety
- 0days: 81% of in the wild 0days (P0 dataset) are memory unsafety
- Microsoft: 70% of all MSRC tracked vulnerabilities are memory unsafety
- Ubuntu: 65% of kernel CVEs in USNs in a 6-month sample are memory unsafety
- Android: More than 65% of high/critical vulnerabilities are memory unsafety
- macOS: 71.5% of Mojave CVEs are due to memory unsafety

https://www.usenix.org/conference/enigma2021/presentation/gaynor

Memory safety problems

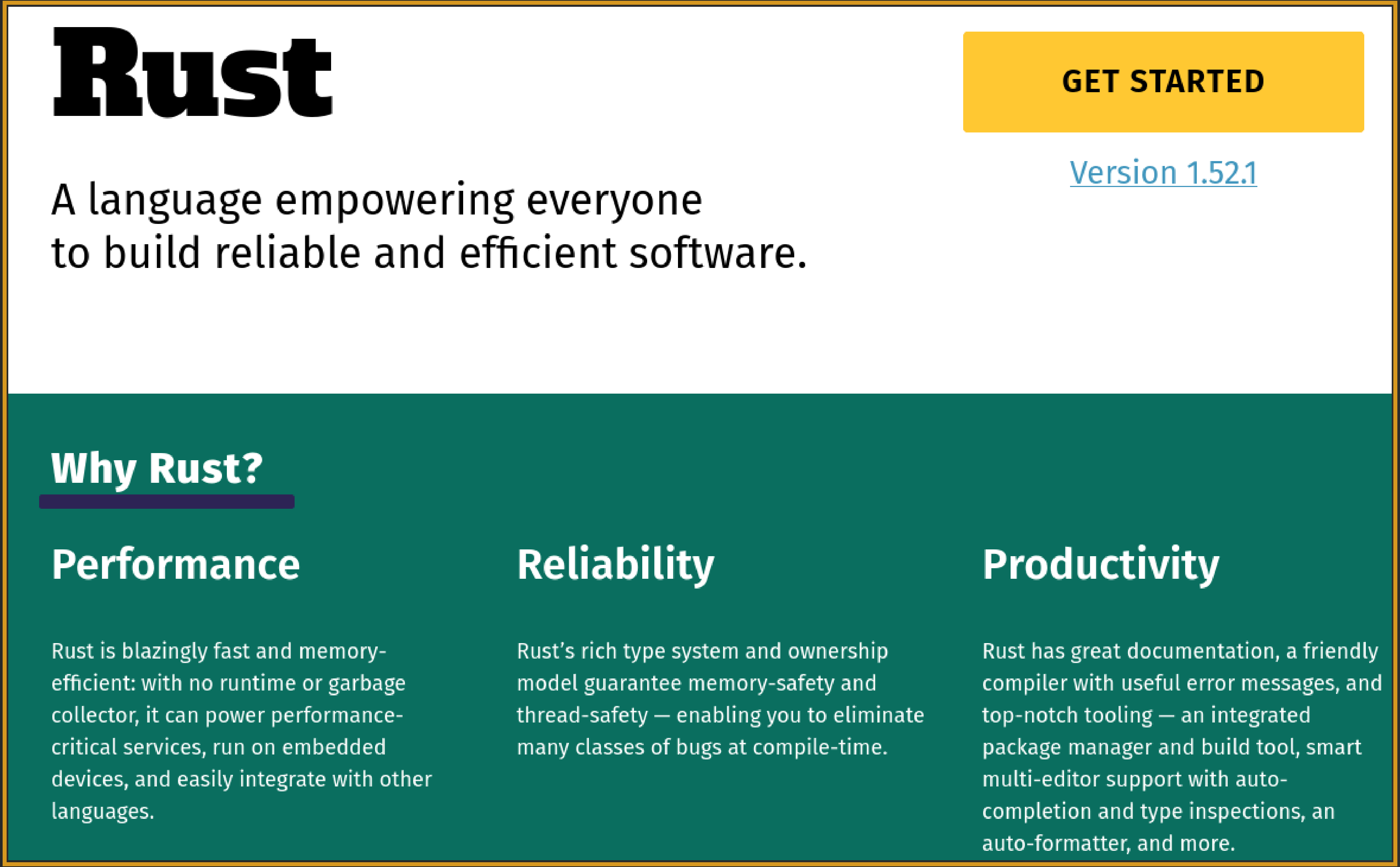
- Access errors
  - Buffer overflow
  - Buffer over-read
  - Data race condition
  - Use after free
- Uninitialized variables
  - Null pointer dereference
  - Wild pointers
- Memory leak
  - Stack exhaustion
  - Heap exhaustion
  - Double free
  - Invalid free

Todd Howard: "If you're running low on memory, you can reboot the original Xbox and the user can't tell. You can throw a screen up. When Morrowind loads sometimes you get a very long load. That's us rebooting the Xbox."

Case study - Cloudflare

> When crashes do occur an engineer needs to spend time to diagnose how it happened and what caused it. Since Pingora's inception we've served a few hundred trillion requests and have yet to crash due to our service code.
>
> In fact, Pingora crashes are so rare we usually find unrelated issues when we do encounter one. Recently we discovered a kernel bug soon after our service started crashing. We've also discovered hardware issues on a few machines, **in the past ruling out rare memory bugs caused by our software even after significant debugging was nearly impossible.**

https://blog.cloudflare.com/how-we-built-pingora-the-proxy-that-connects-cloudflare-to-the-internet/



Rust

GET STARTED

Version 1.52.1

A language empowering everyone
to build reliable and efficient software.

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Const-by-default

```rust
1 fn main() {
2     let a = 7;
3     a += 1;
4 }
```

```
error[E0384]: cannot assign twice to immutable variable `a`
 --> const_by_default.rs:3:5
  |
2 |     let a = 7;
  |         - first assignment to `a`
3 |     a += 1;
  |     ^^^^^^ cannot assign twice to immutable variable
  |
help: consider making this binding mutable
  |
2 |     let mut a = 7;
  |         +++
```

Obowiązkowa inicjalizacja

```rust
let to_find: i32;
let values: Vec<i32>;
let found = values.iter().find(|&&x| x == to_find);
println!("{found:?}");
```

```cpp
int to_find;
std::vector<int> values;
auto found = std::find(values.begin(), values.end(), to_find);
std::cout << *found << std::endl;
```

Obowiązkowa inicjalizacja

```
❯ rustc lack_of_init.rs
error[E0381]: used binding `values` isn't initialized
 --> lack_of_init.rs:4:17
  |
3 |     let values: Vec<isize>;
  |         ------ binding declared here but left uninitialized
4 |     let found = values.iter().find(|&&x| x == to_find);
  |                 ^^^^^^^^^^^^^ `values` used here but it isn't initialized

error[E0381]: used binding `to_find` isn't initialized
 --> lack_of_init.rs:4:36
  |
2 |     let to_find: isize;
  |         ------- binding declared here but left uninitialized
3 |     let values: Vec<isize>;
4 |     let found = values.iter().find(|&&x| x == to_find);
  |                                    ^^^^^      ------- borrow occurs due to use in closure
  |                                    |
  |                                    `to_find` used here but it isn't initialized
```

```
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 lack_of_init.cpp && ./a.out
[1]    40941 segmentation fault (core dumped)  ./a.out
```

Real-life case: https://github.com/bitcoin/bitcoin/pull/17449

Undefined Behavior

- Undefined behavior is the result of executing a program whose behavior is **prescribed** to be unpredictable.
- This is different from:
  - **unspecified** behavior, for which the language specification *does not prescribe* a result
  - **implementation-defined** behavior that defers to the documentation of another component of the platform (such as the ABI or the translator documentation).
- The only possible UB in safe Rust is a *result of a bug* in the compiler or in the standard library

Zarządzanie pamięcią

Resource Acquisition Is Initialization

```rust
fn dummy() {
    let mut text = String::from("Hi");
    text += " world!";
    println!("{text}");
    println!("{}", text.len());
}
```

```cpp
void dummy() {
    auto text = std::string("Hi");
    text += " world!";
    std::cout << text << std::endl;
    std::cout << text.size() << std::endl;
}
```

Borrow checker - windykator

1. Each value in Rust has a variable that’s called its **owner**.
2. There can only be **one** owner at a time.
3. When the owner goes **out of scope**, the value will be dropped.

Właściciel - przykłady

```rust
let v1 = vec![1, 2, 3];
let v2 = v1;
println!("{} {}", v1.len(), v2.len());
```

```cpp
std::vector<int> v1{1, 2, 3};
auto v2 = std::move(v1);
std::cout << v1.size() << " " << v2.size() << std::endl;
```

Właściciel - przykłady

```
❯ rustc assignment.rs
error[E0382]: borrow of moved value: `v1`
 --> assignment.rs:4:23
  |
2 |     let v1 = vec![1, 2, 3];
  |         -- move occurs because `v1` has type `Vec<i32>`, which does not implement the `Copy` trait
3 |     let v2 = v1;
  |              -- value moved here
4 |     println!("{} {}", v1.len(), v2.len());
  |                       ^^ value borrowed here after move
  |
help: consider cloning the value if the performance cost is acceptable
  |
3 |     let v2 = v1.clone();
  |                ++++++++
```

```
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 assignment.cpp && ./a.out
0 3
```

Referencje

1. At any given time, you can have **either**:
   - *one mutable* reference
   - *any* number of *immutable* references
2. References must **always** be valid.

Mieszanie mutowalności

```rust
let mut greetings = vec!["czesc", "hi", "hello"];

let hello = &greetings[2];
println!("{hello}");

greetings.pop();
println!("{hello}");
```

```cpp
std::vector<std::string> greetings = {"czesc", "hi", "hello"};

const auto& hello = greetings[2];
std::cout << hello << std::endl;

greetings.pop_back();
std::cout << hello << std::endl;
```

Mieszanie mutowalności

```
❯ rustc borrow.rs
error[E0502]: cannot borrow `greetings` as mutable because it is also borrowed as
immutable
  ⟶ borrow.rs:7:5
    |
4   |     let hello = &greetings[2];
    |                 ---------- immutable borrow occurs here
...
7   |     greetings.pop();
    |     ^^^^^^^^^^^^^^^ mutable borrow occurs here
8   |     println!("{hello}");
    |               ------- immutable borrow later used here
```

```
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 -fsanitize=undefined borrow.cpp
hello
hello
```

Undefined Behavior i jego skutki

```cpp
std::vector<std::string> greetings = {"czesc", "hi", "hello from the other side"};

const auto& hello = greetings[2];
std::cout << hello << std::endl;

greetings.pop_back();
std::cout << hello << std::endl;
```

```
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 -fsanitize=undefined borrow.cpp && ./a.out
hello from the other side
vYwUn
7other side
```

Dwa ‘typy’ referencji - przykład wredny

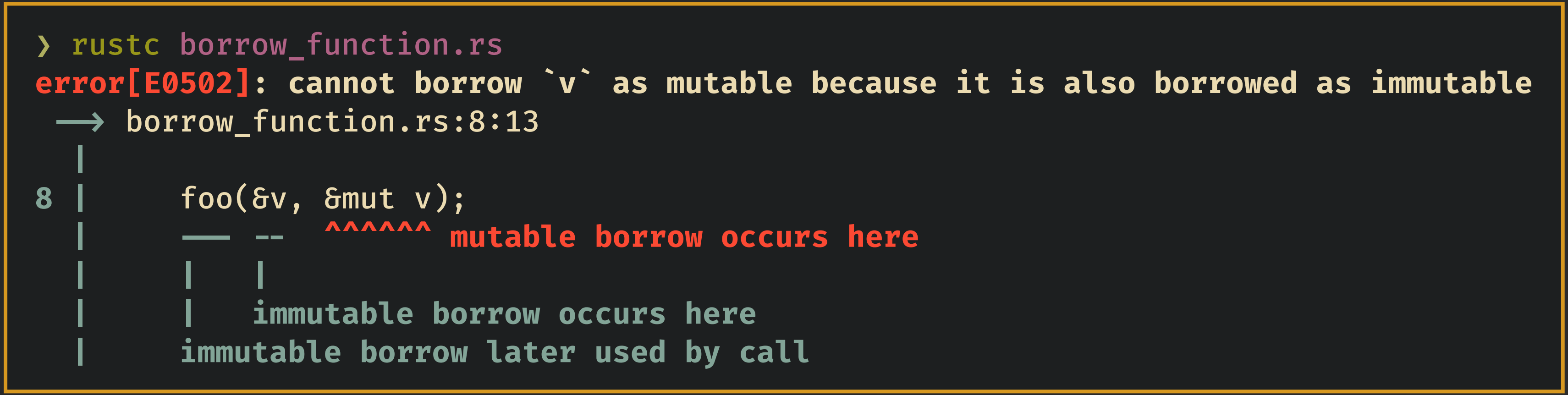
```rust
fn foo(first_vec: &Vec<i32>, second_vec: &mut Vec<i32>) {
    second_vec.clear();
    println!("{}", first_vec[0]);
}

fn main() {
    let mut v = vec![1, 2, 3];
    foo(&v, &mut v);
}
```

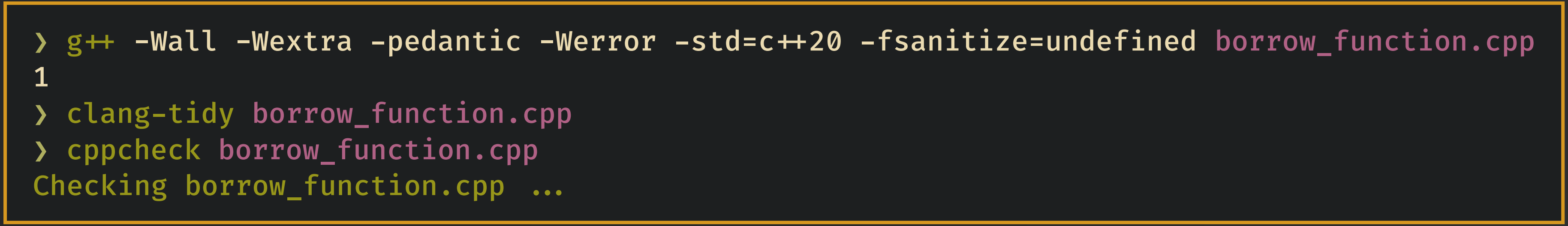
```cpp
void foo(const std::vector<int> &first_vec, std::vector<int> &second_vec) {
    second_vec.clear();
    std::cout << first_vec[0] << std::endl;
}

int main() {
    std::vector<int> v = {1, 2, 3};
    foo(v, v);
}
```

Dwa ‘typy’ referencji - przykład wredny



```
❯ rustc borrow_function.rs
error[E0502]: cannot borrow `v` as mutable because it is also borrowed as immutable
 ──> borrow_function.rs:8:13
  |
8 |     foo(&v, &mut v);
  |     ─── --  ^^^^^^ mutable borrow occurs here
  |     |   |
  |     |   immutable borrow occurs here
  |     immutable borrow later used by call
```



```
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 -fsanitize=undefined borrow_function.cpp
1
❯ clang-tidy borrow_function.cpp
❯ cppcheck borrow_function.cpp
Checking borrow_function.cpp ...
```

Życie po życiu (obiektu)

```rust
fn get_ref() -> &'static i32 {
    let my_number = 7;
    let ref_ = &my_number;
    ref_
}

fn main() {
    let ref_ = get_ref();
    println!("{ref_}");
}
```

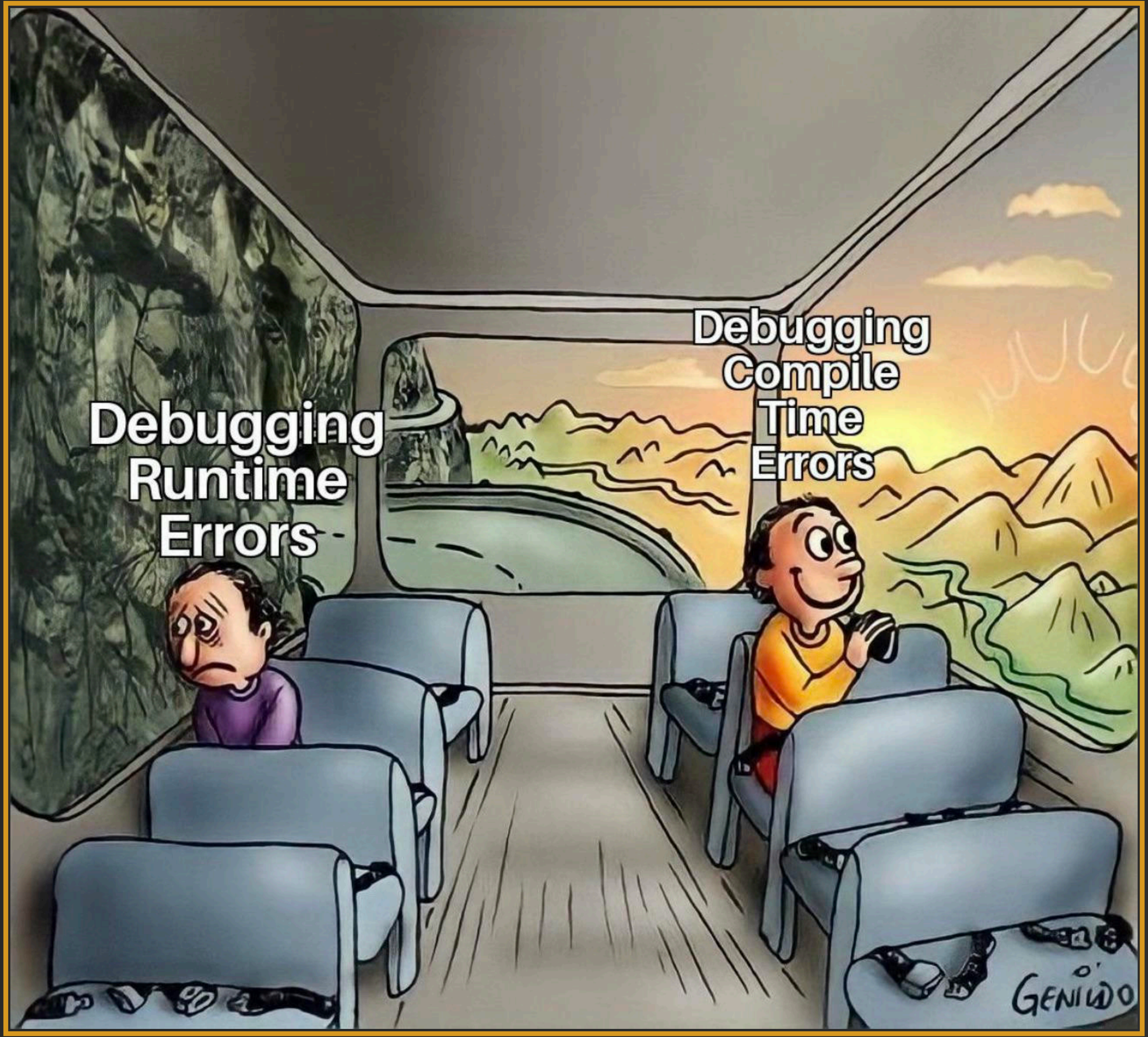
```cpp
const int& get_ref() {
    int my_number = 7;
    const int& ref = my_number;
    return ref;
}

int main() {
    const auto& ref = get_ref();
    std::cout << ref << std::endl;
}
```

Życie po życiu (obiektu)

```
❯ rustc out_of_scope.rs
error[E0515]: cannot return value referencing local variable `my_number`
 --> out_of_scope.rs:4:5
  |
3 |     let ref_ = &my_number;
  |                ---------- `my_number` is borrowed here
4 |     ref_
  |     ^^^^ returns a value referencing data owned by the current function
```

```
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 -fsanitize=undefined out_of_scope.cpp
22031
```


Debugging Runtime Errors
Debugging Compile Time Errors
GENILDO

Struktury

```rust
struct Rectangle {
    pub width: f32,
    height: f32,
}

impl Rectangle {
    fn diagonal_length(&self) -> f32 {
        (self.width.powf(2.) + self.height.powf(2.)).sqrt()
    }

    fn area(&self) -> f32 {
        self.width * self.height
    }

    fn perimeter(&self) -> f32 {
        2.0 * (self.width + self.height)
    }
}
```

Cechy

```rust
trait Shape {

    fn area(&self) -> f32;

    fn perimeter(&self) -> f32;

    fn area_to_perimeter(&self) -> f32 {
        self.area() / self.perimeter()
    }
}
```

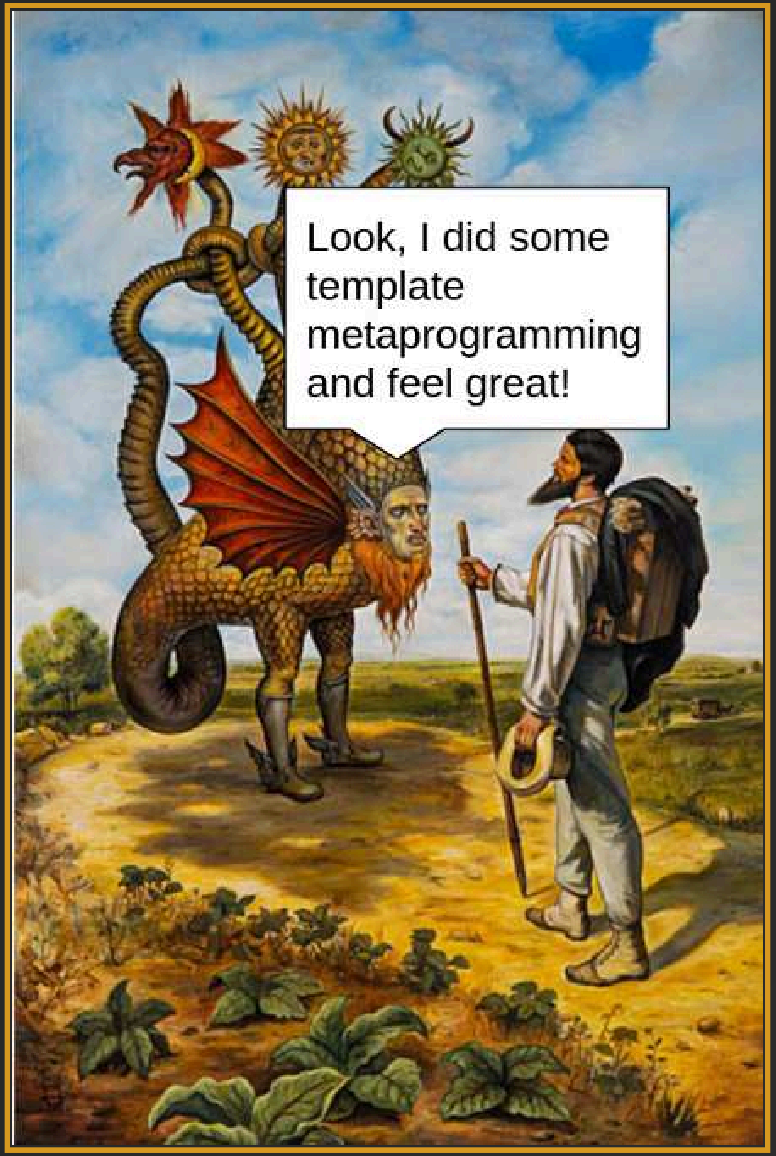
Cechy - implementacja

```rust
struct Rectangle {
    pub width: f32,
    height: f32,
}

impl Rectangle {
    fn diagonal_length(&self) -> f32 {
        (self.width.powf(2.) + self.height.powf(2.)).sqrt()
    }
}

impl Shape for Rectangle {
    fn area(&self) -> f32 {
        self.width * self.height
    }

    fn perimeter(&self) -> f32 {
        2.0 * (self.width + self.height)
    }
}
```


Look, I did some template metaprogramming and feel great!

Generics

```rust
fn print_shape_info<T>(shape: &T) {
    println!("Area: {}", shape.area());
    println!("Perimeter: {}", shape.perimeter());
    println!("Ratio: {}", shape.area_to_perimeter());
}
```

```
❯ rustc generics_wrong.rs
error[E0599]: no method named `area` found for reference `&T` in the current scope
  --> generics_wrong.rs:32:32
   |
32 |     println!("Area: {}", shape.area());
   |                                ^^^^ method not found in `&T`
   |
   = help: items from traits can only be used if the type parameter is bounded by the trait
help: the following trait defines an item `area`, perhaps you need to restrict type parameter `T` with it:
   |
31 | fn print_shape_info<T: Shape>(shape: &T) {
   |                      +++++++
```

And now for something completely different

```cpp
struct Cat {
    unsigned int weight;
    std::string name;
};

struct Bag {
    std::optional<std::reference_wrapper<Cat>> cat;
};

Cat& chonkiest(Cat& x, Cat& y) {
    return x.weight > y.weight ? x : y;
}

int main() {
    Cat x = {7, "Kotlet"};
    Bag gift;
    Cat y = {18, "Kotara Konstantynopolitańczykowianeczka"};
    gift.cat.emplace(chonkiest(x, y));
    std::cout << gift.cat->get().name << std::endl;
}
```

And now for something completely different

```cpp
struct Cat {
    unsigned int weight;
    std::string name;
};

struct Bag {
    std::optional<std::reference_wrapper<Cat>> cat;
};

Cat& chonkiest(Cat& x, Cat& y) {
    return x.weight > y.weight ? x : y;
}

int main() {
    Cat x = {7, "Kotlet"};
    Bag gift;
    {
        Cat y = {18, "Kotara Konstantynopolitańczykowianeczka"};
        gift.cat.emplace(chonkiest(x, y));
        std::cout << gift.cat->get().name << std::endl;
    }
}
```

Antywzorzec - próba ominięcia `std::unique_ptr`

```cpp
struct Cat {
    unsigned int weight;
    std::string name;
};

struct Bag {
    std::optional<std::reference_wrapper<Cat>> cat;
};

Cat& chonkiest(Cat& x, Cat& y) {
    return x.weight > y.weight ? x : y;
}

int main() {
    Cat x = {7, "Kotlet"};
    Bag gift;
    {
        Cat y = {18, "Kotara Konstantynopolitańczykowianeczka"};
        gift.cat.emplace(chonkiest(x, y));
    }
    std::cout << gift.cat->get().name << std::endl;
}
```

Antywzorzec - próba ominięcia `std::unique_ptr`

```
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 -fsanitize=undefined cat.cpp && ./a.out
Rbd:Q
     +Qnopolitańczykowianeczka
```

Prosty ko[dt] w Ruście

```rust
struct Cat {
    pub weight: u32,
    pub name: String,
}

fn chonkiest(x: &Cat, y: &Cat) → &Cat {
    if x.weight > y.weight {
        x
    } else {
        y
    }
}

fn main() {
    let x = Cat {weight: 7, name: String::from("Kotlet")};
    let chonky;  // ← uninitialized
    let y = Cat {weight: 1, name: String::from("Kotara")};
    chonky = chonkiest(&x, &y);
    println!("{}", chonky.name);
}
```

Jaki będzie zakres życia referencji?

```rust
struct Cat {
    pub weight: u32,
    pub name: String,
}

fn chonkiest(x: &Cat, y: &Cat) -> &Cat {
    if x.weight > y.weight {
        x
    } else {
        y
    }
}

fn main() {
    let x = Cat {weight: 7, name: String::from("Kotlet")};
    let chonky;  // <- uninitialized
    {
        let y = Cat {weight: 1, name: String::from("Kotara")};
        chonky = chonkiest(&x, &y);
    }
    println!("{}", chonky.name);
}
```

Błąd kompilacji - brak lifetimes

```
❯ rustc cat.rs
error[E0106]: missing lifetime specifier
 --> cat.rs:6:35
  |
6 | fn chonkiest(x: &Cat, y: &Cat) -> &Cat {
  |                 ----     ----     ^ expected named lifetime parameter
  |
  = help: this function's return type contains a borrowed value, but the signature does not
say whether it is borrowed from `x` or `y`
help: consider introducing a named lifetime parameter
  |
6 | fn chonkiest<'a>(x: &'a Cat, y: &'a Cat) -> &'a Cat {
  |             ++++     ++          ++          ++
```

Lifetimes

```rust
{
    let r;               // ---------+-- 'a
                         //          |
    {                    //          |
        let x = 5;       // -+-- 'b  |
        r = &x;          //  |       |
    }                    // -+       |
                         //          |
    println!("{r}");     //          |
}                        // ---------+
```

```rust
{
    let x = 5;           // ----------+-- 'b
                         //           |
    let r = &x;          // --+-- 'a  |
                         //   |       |
    println!("{r}");     //   |       |
                         // --+       |
}                        // ----------+
```

Naprawiony ko[dt]

```rust
struct Cat {
    pub weight: u32,
    pub name: String,
}

fn chonkiest_cat<'a>(x: &'a Cat, y: &'a Cat) -> &'a Cat {
    if x.weight > y.weight {
        x
    } else {
        y
    }
}

fn main() {
    let x = Cat {weight: 7, name: String::from("Kotlet")};
    let chonky;  // <- uninitialized
    {
        let y = Cat {weight: 1, name: String::from("Kotara")};
        chonky = chonkiest_cat(&x, &y);
    }
    println!("{}", chonky.name);
}
```

Wymuszenie zgodności życia zmiennych

```
❯ rustc cat_with_lifetimes.rs
error[E0597]: `y` does not live long enough
  --> cat_with_lifetimes.rs:19:36
   |
18 |         let y = Cat {weight: 1, name: String::from("Kotara")};
   |             - binding `y` declared here
19 |         chonky = chonkiest_cat(&x, &y);
   |                                    ^^ borrowed value does not live long enough
20 |     }
   |     - `y` dropped here while still borrowed
21 |     println!("{}", chonky.name);
   |                    ----------- borrow later used here
```

Arytmetyczna dogrzewka

```rust
fn main() {
    let x1: u16 = 1;
    let x2 = 2_u16;
    println!("{}", x1 - x2);
}
```

```cpp
int main() {
    uint16_t x1 = 1;
    uint16_t x2 = 2;
    std::cout << x1 - x2 << std::endl;
}
```

Arytmetyczna dogrzewka - compiler screams

```
❯ rustc u16_subtraction_naive.rs
error: this arithmetic operation will overflow
 --> u16_subtraction_naive.rs:4:20
  |
4 |     println!("{}", x1 - x2);
  |                    ^^^^^^^ attempt to compute `1_u16 - 2_u16`, which would overflow
  |
  = note: `#[deny(arithmetic_overflow)]` on by default
```

Arytmetyczna dogrzewka - zaciemniony kod

```rust
fn subtract(val1: u16, val2: u16) {
    println!("{}", val1 - val2);
}

fn main() {
    let x1 = 1;
    let x2 = 2;
    subtract(x1, x2);
}
```

```cpp
void subtract(uint16_t val1, uint16_t val2) {
    std::cout << val1 - val2 << std::endl;
}

int main() {
    uint16_t x1 = 1;
    uint16_t x2 = 2;
    subtract(x1, x2);
}
```

Plusy i minusy rzutowania

```
❯ rustc u16_subtraction.rs && ./u16_subtraction
thread 'main' panicked at u16_subtraction.rs:2:20:
attempt to subtract with overflow
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
❯ rustc -O u16_subtraction.rs && ./u16_subtraction
65535
```

```
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 u16_subtraction.cpp && ./a.out
-1
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 -O2 u16_subtraction.cpp && ./a.out
-1
```

Rozmiar typów i jego wpływ na zachowanie

```rust
fn subtract(val1: u32, val2: u32) {
    println!("{}", val1 - val2);
}

fn main() {
    let x1 = 1;
    let x2 = 2;
    subtract(x1, x2);
}
```

```cpp
void subtract(uint32_t val1, uint32_t val2) {
    std::cout << val1 - val2 << std::endl;
}

int main() {
    uint32_t x1 = 1;
    uint32_t x2 = 2;
    subtract(x1, x2);
}
```

Rozmiar typów i jego wpływ na zachowanie

```
❯ rustc u32_subtraction.rs && ./u32_subtraction
thread 'main' panicked at u32_subtraction.rs:2:20:
attempt to subtract with overflow
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
❯ rustc -O u32_subtraction.rs && ./u32_subtraction
4294967295
```

```
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 u32_subtraction.cpp && ./a.out
4294967295
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 -O2 u32_subtraction.cpp && ./a.out
4294967295
```

???

```rust
fn multiply(x1: u16, x2: u16){
    println!("{}", x1 * x2);
}

fn main() {
    let x1: u16 = 0xffff;
    let x2: u16 = 0xffff;
    multiply(x1, x2);
}
```

```cpp
void multiply(uint16_t val1, uint16_t val2) {
    std::cout << val1 * val2 << std::endl;
}

int main() {
    uint16_t x1 = 0xffff;
    uint16_t x2 = 0xffff;
    multiply(x1, x2);
}
```

???

```
❯ rustc u16_multiplication.rs && ./u16_multiplication
thread 'main' panicked at u16_multiplication.rs:2:20:
attempt to multiply with overflow
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
❯ rustc -O u16_multiplication.rs && ./u16_multiplication
1
```

```
❯ g++ -Wall -Wextra -pedantic -Werror -std=c++20 -O2 -fsanitize=undefined u16_multiplication.cpp && ./a.out
u16_multiplication.cpp:6:25: runtime error: signed integer overflow: 65535 * 65535 cannot be represented in
type 'int'
-131071
```

Usual arithmetic conversions - C++

> Many binary operators that expect operands of arithmetic or enumeration type cause conversions and yield result types in a similar way. The purpose is to yield a common type, which is also the type of the result. This pattern is called the usual arithmetic conversions, which are defined as follows:
>
> - If either operand is of scoped enumeration type, no conversions are performed; if the other operand does not have the same type, the expression is ill-formed.
> - If either operand is of type `long double`, the other shall be converted to `long double`.
> - Otherwise, if either operand is `double`, the other shall be converted to `double`.
> - Otherwise, if either operand is `float`, the other shall be converted to `float`.
> - Otherwise, the **integral promotions** shall be performed on both operands.

https://eel.is/c++draft/expr.arith.conv

Integral promotion rules - excerpt - C++

> A prvalue of an integer type other than `bool`, `char8_t`, `char16_t`, `char32_t`, or `wchar_t` whose integer conversion rank is less than the rank of `int` can be converted to a prvalue of type `int` if `int` can represent all the values of the source type; otherwise, the source prvalue can be converted to a prvalue of type `unsigned int`.

GONNA TELL MY KIDS THIS IS BJARNE STROUSTRUP WITH THE ORIGINAL VERSION OF THE C++ STANDARD
Capitol Hill
11:29 AM

https://eel.is/c++draft/conv.prom

Jak to wygląda w “rzeczywistości”?

```rust
fn main() {
    let buffer = ["y"; 50];
    for j in 0..9 {
        println!("{}", j * 0x20000001);
        if buffer[0] == "x" {break;}
    }
}
```

```cpp
#include <iostream>

int main() {
    char buf[50] = "y";
    for (int j = 0; j < 9; ++j) {
        std::cout << (j * 0x20000001) << std::endl;
        if (buf[0] == 'x') break;
    }
}
```

Do czego kompilatorowi UB?

1. Kompilator zakłada, że przekręcenie licznika nigdy nie nastąpi:

```
for (int j = 0; j < 9 * 0x20000001; p += 0x20000001) {
    std::cout << p << std::endl;
    if (buf[0] == 'x') break;
}
```

2. Kompilator wie, że:
   - `INT_MAX < 9 * 0x20000001`
   - `j` nie może być większe niż `INT_MAX`
3. Kompilator zamienia `j < 9 * 0x20000001` na `true`

Referencje/materiały

- https://doc.rust-lang.org/book/
- https://doc.rust-lang.org/std/
- https://medium.com/nearprotocol/understanding-rust-lifetimes-e813bcd405fa
- https://shafik.github.io/c++/2021/12/30/usual_arithmetic_confusions.html