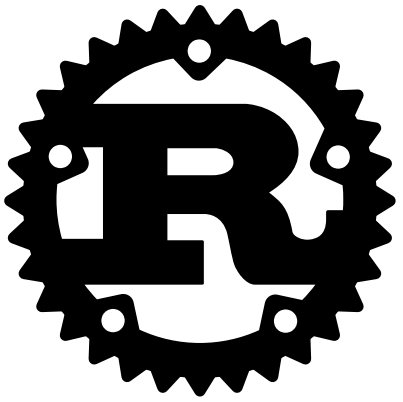
RUST



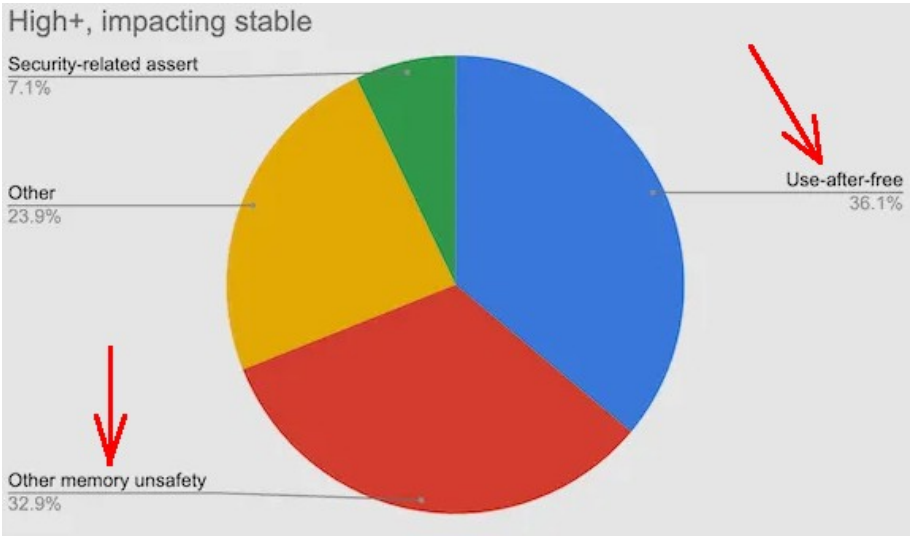
Krüpl Zsolt 2022. nov. 10.

Napirend

- Áttekintés
- Technikai részletek
- Bemutató

Biztonságos programozás

Microsoft: **70 percent** of all security bugs are **memory safety issues** *(2006 … 2018)*
https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/



Biztonság: próbálkozások

Cyclone: AT&T labs 2002 – 2011
http://cyclone.thelanguage.org/

Safe-C:
https://www.safe-c.org

Checked-C: Microsoft 2015 – 2020
https://www.microsoft.com/en-us/research/project/checked-c/

Újabban: Projekt Verona – Microsoft
https://www.microsoft.com/en-us/research/project/project-verona/

Rust előnyei

- **Elsődleges szempont a biztonságos kód.**
  - Amit lehetséges, azt ellenőrzése alatt tart a fordító.
    1. lehetőleg fordítási időben
    2. futásidejű ellenőrzéssel
  - hibakezelés kikényszerítése
  - szigorú nyelv
- **Gyors futású, memóriahatékony**
  - nincs semmiféle futtatókörnyezete
  - se szemétgyűjtő (GC)
- **Rendszerprogramozásra** alkalmas nyelv. Felhasználható:
  - Mikrovezérlőtől szerverekig
  - Webassembly, CUDA
- Modern megoldások, jól dokumentált

Rendszerprogramozás

- Operációs rendszer
- Eszközmeghajtó
- Fájlrendszer
- Adatbázis
- Olcsó eszközön futó megbízható kódok
- Titkosítás
- Média codec
- Média feldolgozás
  (hangfelismerés, képszerkesztés, …)
- Szöveg olvasás, konverziók

- Magasabb szintű nyelvek implementálása
- Hálózati szoftverek
- Virtualizáció és konténer
- Tudományos szimuláció
- Játékprogram

Rust árnyoldalai

- **Nehéz a tanulási görbéje**
  - szokatlan megoldások, főként az **„ownership” és „borrowing”**.
  - **szigorú nyelv**

- **Lassú a fordítás** a C és C++-hoz képest
  - ellenőrzések
  - a sok publikált forráskód-rekesz *(crate)* elkényelmesíti a fejlesztőt

- **Egyelőre egyetlen fordító implementáció van** *(https://www.rust-lang.org).*
  - mrustc kezdetleges
  - GCC 13 – alapszintű Rust fordítás már lesz benne

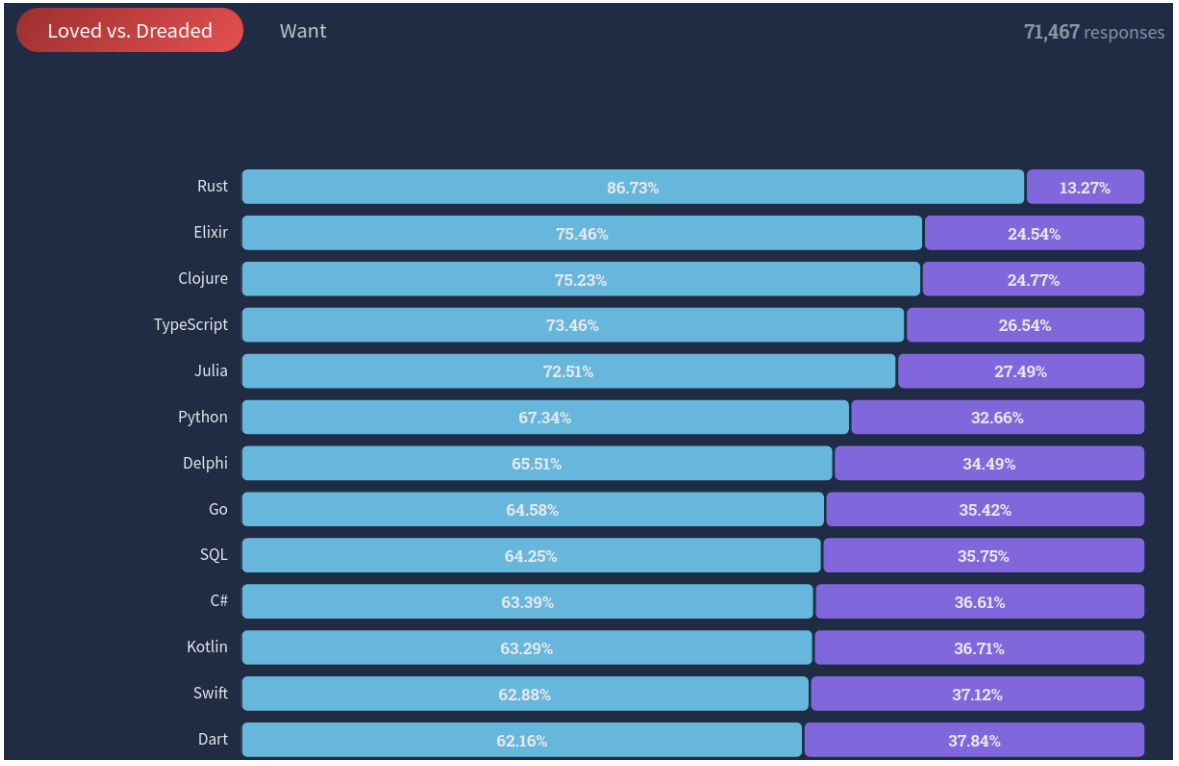
Hol használják?

- Ma már sok szoftverben; teljes mértékben vagy részben
- **Linux kernel 6.1** *(https://kernel.org/)* verziójától a kernel is **C és Rust**.
- **Járműipar** is kezdi magának felfedezni

- **Elsődleges célterület:**
  - konzolalkalmazások
  - webassembly
  - hálózati alkalmazások, kiszolgálók
  - beágyazott rendszerek

- Többen kérdezik: ki fogják dobni a C és C++ nyelveken írt szoftvereket?

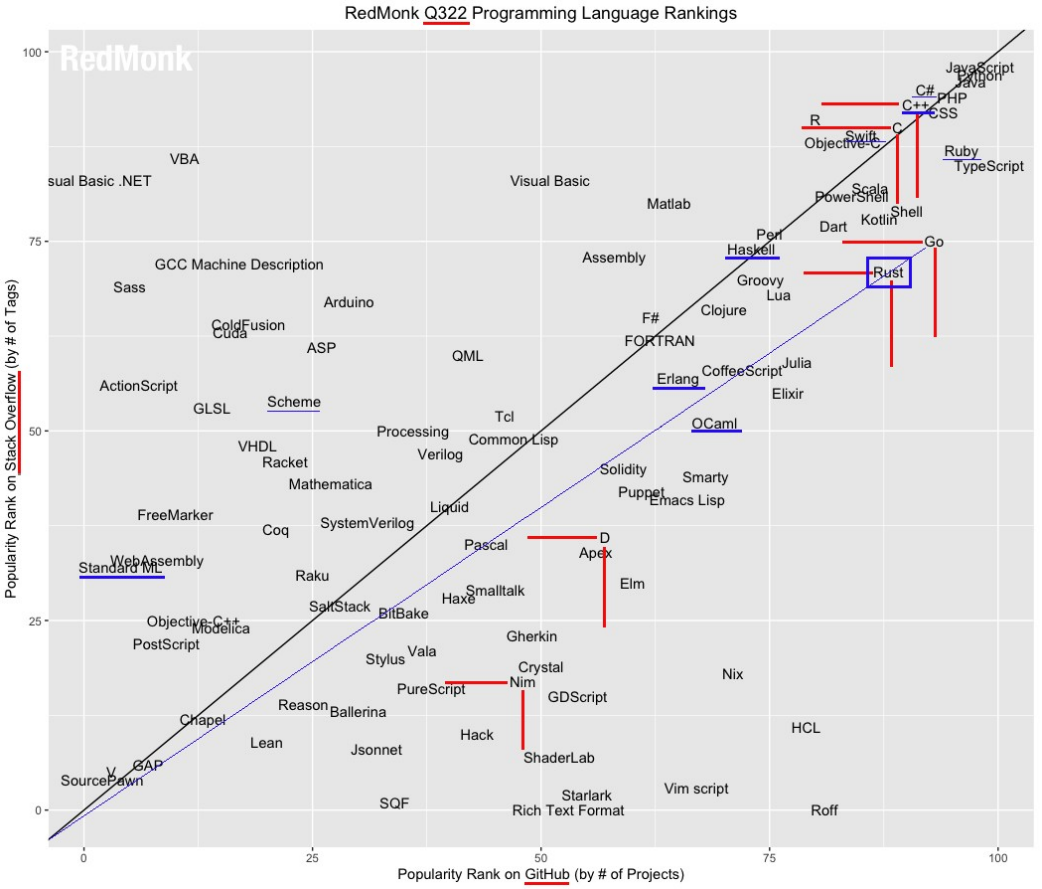
A legjobban szeretett nyelv



https://survey.stackoverflow.co/2022/#most-loved-dreaded-and-wanted-language-love-dread

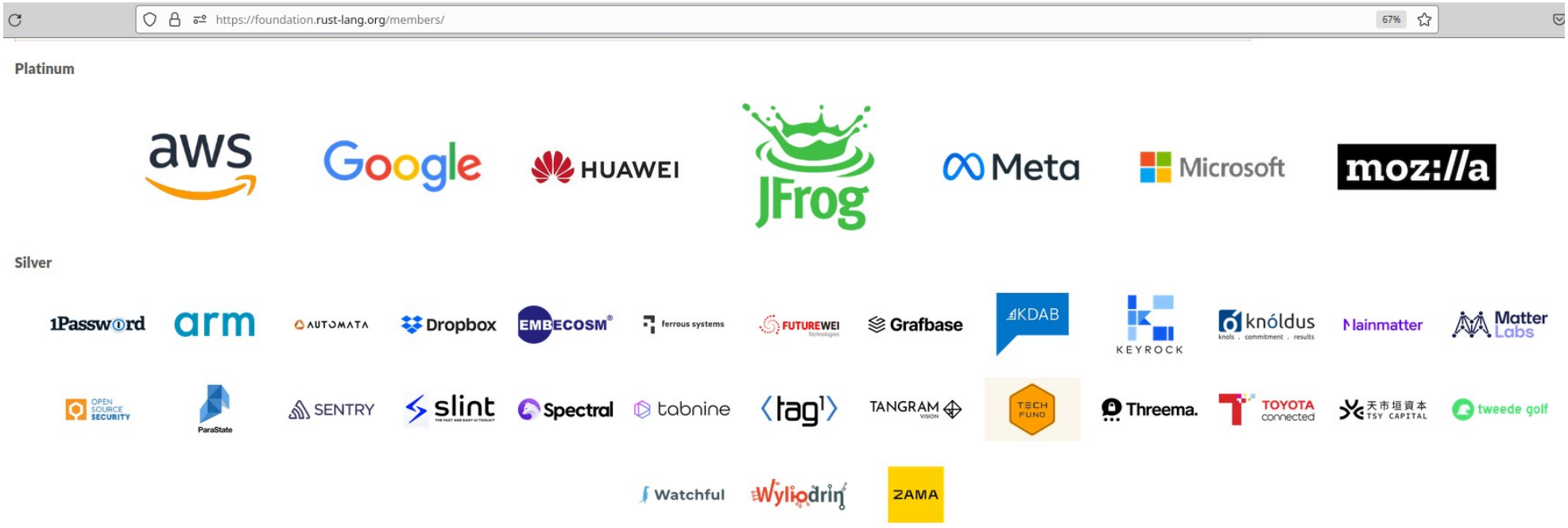
A megjelenését követően minden évben első helyen (2016 ...).

GitHub és Stack Overflow weboldalak



https://redmonk.com/sogrady/2022/10/20/language-rankings-6-22/

Rust alapítvány



Rust története

Channel	Version	Will be stable on
Stable	1.65	November 03 2022
Beta	1.66	December 15 2022
Nightly	1.67	January 26 2023

- 2006 Graydon Hoare – Mozilla
- 2010 bejelentés
- 2012 béta *(Rust 0.1 – 2012. jan. 20.)*
  https://github.com/rust-lang/rust/blob/master/RELEASES.md
- **2015 Rust 1.0**
  https://blog.rust-lang.org/
- **3 évente mérföldkő** (edition = ”2021” 1.56)
  - visszafelé kompatibilitás
  - 6 hetente új kiadás, apróbb újításokkal

Memóriakezelés evolúciója

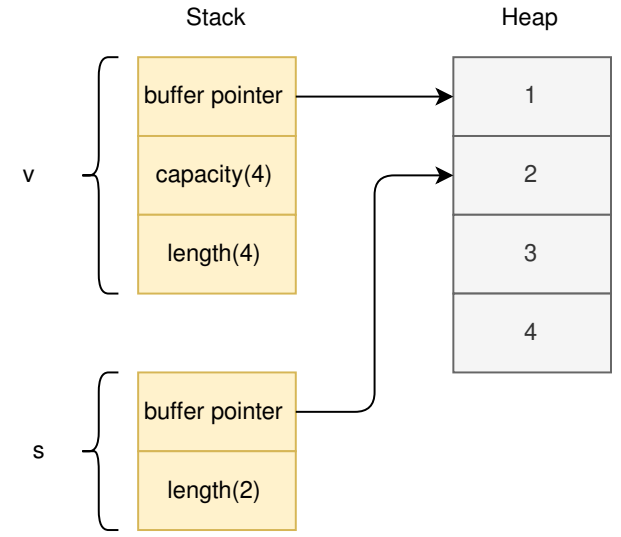
1. Manuális memóriakezelés
   - **C**: külső függvények *(malloc, free)*
   - **C++**: nyelvi elem lett *(new, delete)*
   - **Carbon:** memóriakezelése leginkább szintaktikában változott – kísérleti fázisban
     https://github.com/carbon-language/carbon-lang

2. Automatikus, szemétgyűjtő *(GC)* alapon: **Java** *(1995)*
   - **D** – https://dlang.org
   - **Go** – https://go.dev
   - **Nim** – https://nim-lang.org – RC vagy GC alapon

3. GC-t **nélkülöző** automatikus memóriakezelés
   - **Rust**
   - **Projekt Verona** – kísérleti fázisban

Vektor és szelet *(slice)*

```rust
fn teszt1(vt: &[i32]) {        // slice
    println!("t1: {:?}", vt);
}

fn teszt2(vt: Vec<i32>) {
    println!("t2: {:?}", vt);
    // itt fog felszabadulni
}

fn main() {
    let mut v = Vec::new();
    v.push(3);

    teszt1(&v);
    v.push(4);
    teszt1(&v[1..]);

    teszt2(v);
    v.push(5);  // Fordítási hiba, nincs már 'v'
}
```




```
17 |     teszt2(v);
   |            - value moved here
18 |     v.push(5);  // Fordítási hiba, nincs már 'v'
   |     ^^^^^^^^^ value borrowed here after move
```

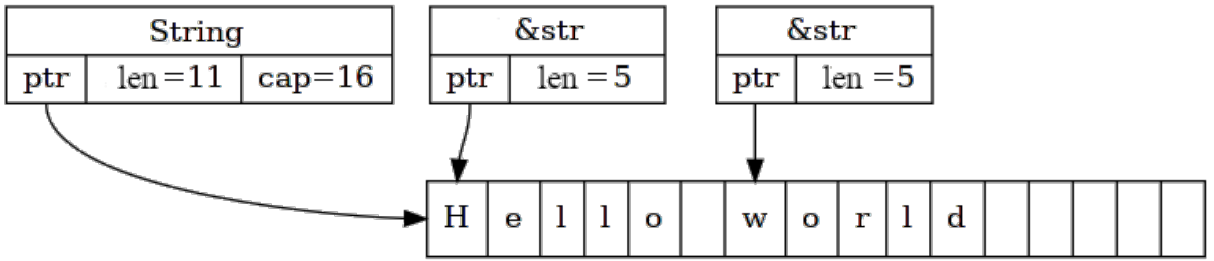
„Use-after-free”: C++ és Rust

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v { 10, 11 };
    int *vptr = &v[1];      // v[1]-re mutat
    cout << *vptr << endl; // oké, 11
    // ...
    // ...
    v.push_back(12);        // reallokálhat új helyre
    // ...
    // ...
    cout << *vptr << endl; // "use-after-free"
}
```

```rust
fn main() {
    let mut v = vec![10, 11];
    let vptr = &v[1];    // v[1]-re mutat
    println!("{}", vptr);    // oké, 11
    // ...
    // ...
    v.push(12);              // reallokálhat új helyre
    // ...
    // ...
    println!("{}", vptr); // a fordítás hibával leáll
}
```

```
laptop:/tmp$ g++ -Wall -Wextra -O2 pelda.cpp -o pelda
laptop:/tmp$ ./pelda
11
5
laptop:/tmp$
```

String és &str




```
let szavak = txt.split(' ');
```

- C-vel ellentétben nem ’\0’ végződés határol
- **String:** kizárólag **UTF8**
- **capacity, len:** UTF8 byte-ok számát mutatja, nem a karakterszámot

Enum: összekevert cimkék

```cpp
#include <iostream>

using namespace std;

enum Color { red, green, blue };
enum Colortable { col_white, col_red, col_black };

int main() {
    Colortable r = col_red;  // hoppá!
    switch(r) {
        case Color::red: cout << "red" << endl; break;
        case green: cout << "green" << endl; break;
        case blue : cout << "blue"  << endl; break;
    }
}

// --- biztonságos: C++11 óta lehetséges ez is ---
enum class Color { red, green, blue };
enum class Colortable { col_white, col_red, col_black };
```

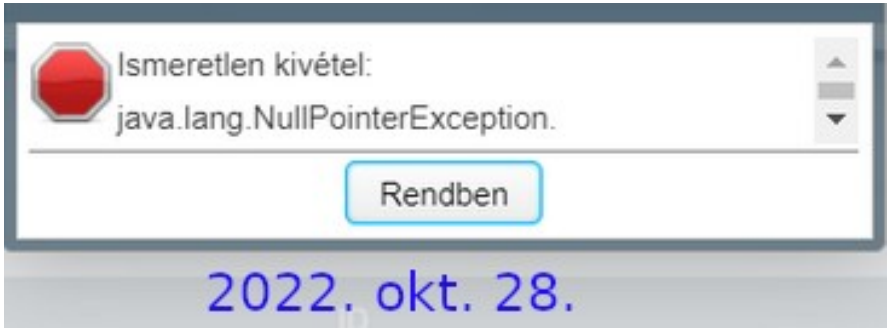
```rust
enum Color { Red, Green, Blue }
enum Colortable { ColWhite, ColRed, ColBlack }

fn main() {
    let r = Color::Red;    // kötelező!
    // let r = Colortable::ColRed;
    match r {
        Color::Red => println!("red"),
        Color::Green => println!("green"),
        Color::Blue => println!("blue"),
    }
}
```

Enum további érdekességei

```
enum Color {   // C-szerű
    Red,
    Green,
    Blue,
}

enum Akármi {  // Enum cimkék változókkal
    Egyik(u8, u8, u8),
    Másik(String, Vec<i32>),
    Harmadik,
}

fn main() {
    let valami = Akármi::Egyik(11, 22, 33);
    match valami {
        Akármi::Egyik(r, g, b) => println!("RGB: {} {} {}", r, g, b),
        Akármi::Másik(neve, értékei) => println!("{}: {:?}", neve, értékei),
        Akármi::Harmadik => println!("Harmadik eset teljesült."),
    }
}
```

Biztonság: NULL pointer nélkül

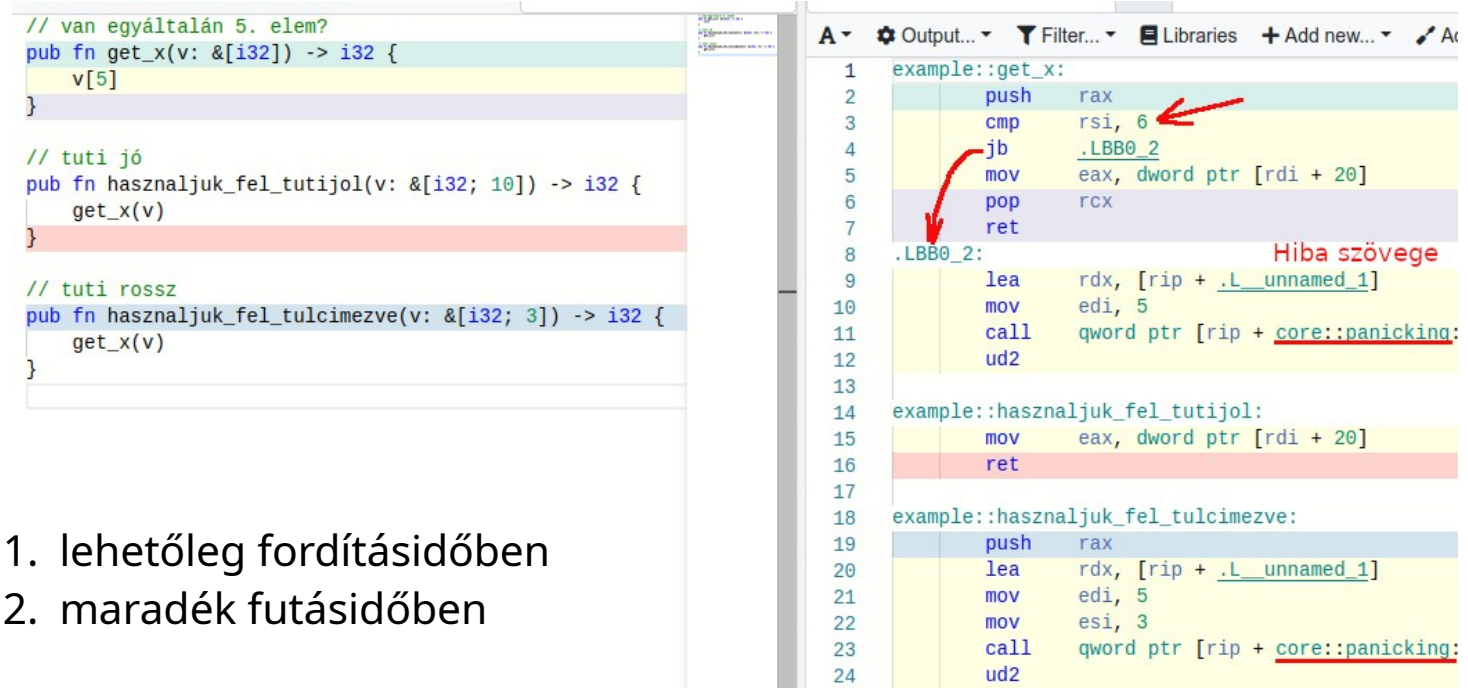
- NULL pointer az **adatcsatornában**
- Nem ritka, hogy lekelezetlenül marad



Google: „Null Pointer References: The Billion Dollar Mistake”

- Rust: nincs NULL pointer, helyette **enum**
  - Option<**T**> → Some(**T**) vagy None
  - Result<**T, E**> → Ok(**T**) vagy Err(**E**)
- Lekezelése:
  - if let Some(x) = … { … } else { … }
  - let Some(x) = … else { panic!(…); } – Rust 1.65
  - match … { Some(x) => …, None => … }
  - unwrap() és társai → körültekintően velük
  - ? operátorral való hibafelterjesztés

Fordító: hogyan fog ellenőrizni?



1. lehetőleg fordításidőben
2. maradék futásidőben

https://rust.godbolt.org/ *(-O opcióval)*

Tetszetős megoldások: iterátor

Iterátor for ciklusban:

```
let v = vec![10, 20, 30, 40, 50];

for &s in v {
    acc += s;
}

for s in 0..10 {
    acc += s;
}

for s in (0..100).step_by(5).skip(3).take(4) {
    acc += s;
}
```

Iterátor közvetlenül:

```
let acc = (0..100).step_by(5).take(4).sum();
```

Egy szálon és több szálon:

```
fn teszt(a: &[i32], b: &[i32]) -> Vec<i32> {
    a.iter().zip(b).map(|(&x, &y)| x * y).collect()
}
```

```
use rayon::prelude::*;

fn teszt_parallel(a: &[i32], b: &[i32]) -> Vec<i32> {
    a.par_iter().zip(b).map(|(&x, &y)| x * y).collect()
}
```

Iterátor szövegre

```
let txt = "Árvíztűrő журнал";

let byteok: Vec<u8> = txt.bytes().collect();
let karakterek: Vec<char> = txt.chars().collect(); // unicode ch vektor

let revchars: Vec<char> = txt.chars().rev().collect();
let revstring: String = txt.chars().rev().collect();
```

Osztály: egyszerű példa

```rust
struct Counter {
    state: u64,
}

impl Counter {
    fn new(x: u64) -> Self { // tetszőleges
        Counter { state: x }
    }

    fn add(&mut self){
        self.state += 1;
    }

    fn get(&self) -> u64 {
        self.state
    }
}
```

```rust
fn main() {
    let mut ct = Counter { state: 2 }; // ha látható
    let mut ct2 = Counter::new(5); // konstruktorral

    ct.add();
    ct.add();

    ct2.add();

    println!("Számláló:  {}", ct.get());
    println!("Számláló2: {}", ct2.get());
}
```

ASM: https://rust.godbolt.org/z/r3qGffEa1

Párhuzamos programozás

- **rayon::par_iter**
- **std::thread::spawn()**
  - mpsc::channel()
- **Async / await**:
  - vagy std::future
  - vagy futures-rs https://crates.io/crates/futures

Makrók

```
println!("Felkiáltójel");
let v: Vec<u128> = vec![1, 2, 3];
```

```
#[macro_export]
macro_rules! Vec {
    ( $( $x:expr ),* ) => {
        {
            let mut temp_vec = Vec::new();
            $(
                temp_vec.push($x);
            )*
            temp_vec
        }
    };
}
```

Rustc, Cargo, Rustup

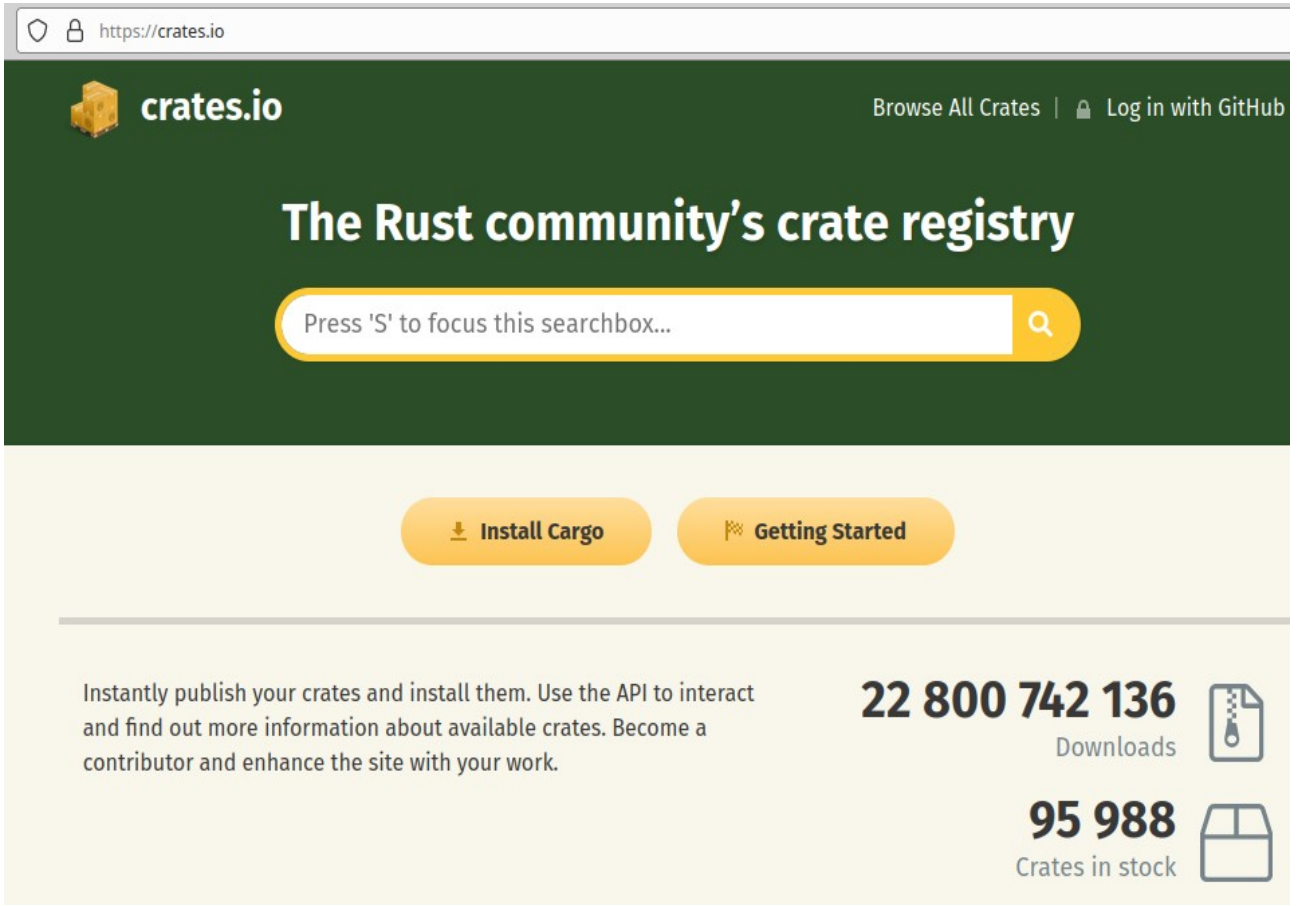
- „gcc” helyett **rustc**
  - Ritkán használjuk magában
- „make” helyett: **cargo**
  - Fordít + letölti a szükséges csomagokat *(Cargo.toml)*
  - Futtat
  - Forráskódot formáz
  - Unit testet futtat
  - Dokumentáció generátora van
  - ...
- Telepítéshez, frissítéshez: **rustup**

Unit teszt

```rust
pub fn add(a: i32, b: i32) -> i32 {
    a + b + 1
}

pub fn mul(a: i32, b: i32) -> i32 {
    a * b
}

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn lassuk_add() {
        assert_eq!(add(2, 3), 5);
    }

    #[test]
    fn lassuk_mul() {
        assert_eq!(mul(2, 3), 6);
    }
}
```

```
laptop:/tmp/unit_teszt$ cargo test
   Compiling unit_teszt v0.1.0 (/tmp/unit_teszt)
    Finished test [unoptimized + debuginfo] target(s) in 0.65s
     Running unittests src/lib.rs (target/debug/deps/unit_teszt-7b1792d8ea556

running 2 tests
test tests::lassuk_mul ... ok
test tests::lassuk_add ... FAILED

failures:

---- tests::lassuk_add stdout ----
thread 'tests::lassuk_add' panicked at 'assertion failed: `(left == right)`
  left: `6`,
 right: `5`', src/lib.rs:15:9
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace


failures:
    tests::lassuk_add

test result: FAILED. 1 passed; 1 failed; 0 ignored; 0 measured; 0 filtered out

error: test failed, to rerun pass '--lib'
laptop:/tmp/unit_teszt$
```

Rekeszek – https://crates.io



Cargo.toml

```
[dependencies]
num-complex = "0.4"
```

Verziószámozás: x.y.z

Sokakat elriaszt a 0-val kezdés.

„Bare metal programming”

```
#![no_std]
#![no_main]

// ...
use stm32f1xx_hal::{pac, prelude::*};

#[entry]
fn main() -> ! {
    let p = pac::Peripherals::take().unwrap();
    // ...
    let mut rcc = p.RCC.constrain();
    let clocks = rcc
        .cfgr
        .use_hse(8.mhz())
        .sysclk(72.mhz())
        .pclk1(24.mhz())
        .freeze(&mut flash.acr);
    // ...
    let mut gpioa = p.GPIOA.split(&mut rcc.apb2);
    let txpin = gpioa.pa2.into_alternate_push_pull(&mut gpioa.crl);
    // ...
    // ...
}
```

Mikrovezérlővel óvatosabban

- Stabilitással nem lesz probléma
- „Hogyan lehet ezt megoldani?”
- Fórumon is kevesebb a segíteni tudó
- **Viszont lehetséges.**

https://hackaday.com/wp-content/uploads/2015/12/rust.jpg

Unsafe – egyszerű példakód

```
extern "C" {
    fn abs(input: i32) -> i32;      // libc.so abs() függvénye (man 3 abs)
}

fn cfv_abs(val: i32) -> i32 {
    unsafe { abs(val) }
}




// ---- felhasználása -----

fn main() {
    let x = -10;
    println!("C-ből: {}", cfv_abs(x)); // C-ből a példa kedvéért
    println!("Rust:  {}", x.abs());    // Rust normál abs()
}
```

Köszönöm az eddigi figyelmet!

A folytatásban online bemutató következik