

RUST

INTRO FOR C++ PROGRAMMERS



Created by [Kai / @_cibo_](#) and [Ben / @predatorhat](#)

```
fn main() {  
    println!("Hello, there!");  
}
```

rust-lang.org

play.rust-lang.org

- No garbage collector
- No big runtime
- Automatic memory management
- Zero-cost abstractions

SYNTAX

- Imperative
- C/C++-like: curly braces and semicolons as function and statement delimiter.
- Freeform: indentation isn't significant

C++

Rust

auto x = 42;

let x = 42;

std::unique_ptr<T>

Box<T>

std::shared_ptr<T>

Rc<T> and Arc<T>

Destructors

trait Drop

__asm

asm! macro

SELLING POINT?

Quite similar to C++. So why do we need another language?

SELLING POINT?

Rust's type system!

```
void logError(const char* msg, int* ints) {
    fprintf(stderr, "%s: %i\n", msg, ints[0]);
}

int main() {
    int* ptr = (int*)malloc (SIZE);
    // ...
    if (err) {
        abrt = 1;
        free(ptr);
    }
    // ...
    if (abrt) {
        logError("operation aborted before commit", ptr);
    }
}
```

CWE-416: Use After Free

hello_world — bkircher@tabaqui:~ — ssh — 80x24

```
[bkircher@tabaqui:~]$ g++ -Wall -Wextra -pedantic use-after-free.cpp
```

```
[bkircher@tabaqui:~]$ ./a.out
```

```
operation aborted before commit: 0
```

```
[bkircher@tabaqui:~]$ g++ --version
```

```
g++ (GCC) 4.8.3 20140911 (Red Hat 4.8.3-9)
```

```
Copyright (C) 2013 Free Software Foundation, Inc.
```

```
This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

```
[bkircher@tabaqui:~]$
```

```
[bkircher@tabaqui:~]$ g++ -Wall -Wextra -pedantic -fsanitize=address use-after-free.cpp
```

```
[bkircher@tabaqui:~]$ ./a.out
```

```
=====  
==2514== ERROR: AddressSanitizer: heap-use-after-free on address 0x60420001f400  
at pc 0x400937 bp 0x7fffc7bc5b80 sp 0x7fffc7bc5b70
```

```
READ of size 4 at 0x60420001f400 thread T0
```

```
#0 0x400936 (/home/bkircher/a.out+0x400936)
```

```
#1 0x4009b6 (/home/bkircher/a.out+0x4009b6)
```

```
#2 0x7f097a19eaf4 (/usr/lib64/libc-2.17.so+0x21af4)
```

```
#3 0x400828 (/home/bkircher/a.out+0x400828)
```

```
0x60420001f400 is located 0 bytes inside of 1024-byte region [0x60420001f400,0x60420001f800)
```

```
freed by thread T0 here:
```

```
#0 0x7f097ad73009 (/usr/lib64/libasan.so.0.0.0+0x16009)
```

```
#1 0x40099f (/home/bkircher/a.out+0x40099f)
```

```
#2 0x7f097a19eaf4 (/usr/lib64/libc-2.17.so+0x21af4)
```

```
previously allocated by thread T0 here:
```

```
#0 0x7f097ad73129 (/usr/lib64/libasan.so.0.0.0+0x16129)
```

```
#1 0x40097b (/home/bkircher/a.out+0x40097b)
```

```
#2 0x7f097a19eaf4 (/usr/lib64/libc-2.17.so+0x21af4)
```

```
Shadow bytes around the buggy address:
```

```
0xc08bffffbe30: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
```

```
0xc08bffffbe40: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
```

To catch this in C or C++ you need

- Something like ASAN and/or Valgrind's Memcheck
- Good test coverage
- And you need to run those tests
- If code is in one TU: static code analysis *might* help

In Rust, lifetime is part of an object's type

```
fn main() {  
    let x;  
  
    {  
        let y = 5;  
        x = &y;  
    }  
  
    println!("x's value is {}", x);  
}
```

And thus, checked at compile time

```
static — bash — 80x24
[bkircher@sherekhan:static]$ rustc lifetime.rs
lifetime.rs:6:14: 6:15 error: `y` does not live long enough
lifetime.rs:6
           x = &y;
           ^
lifetime.rs:2:11: 10:2 note: reference must be valid for the block suffix following statement 0 at 2:10...
lifetime.rs:2   let x;
lifetime.rs:3
lifetime.rs:4   {
lifetime.rs:5       let y = 5;
lifetime.rs:6       x = &y;
lifetime.rs:7   }
           ...
lifetime.rs:5:19: 7:6 note: ...but borrowed value is only valid for the block suffix following statement 0 at 5:18
lifetime.rs:5       let y = 5;
lifetime.rs:6       x = &y;
lifetime.rs:7   }
error: aborting due to previous error
[bkircher@sherekhan:static]$
```

OWNERSHIP

Move by default: variables are *moved* to new locations, preventing previous locations from using it.

There is only one owner of data!

WHO IS USING RUST?


- rustc
- Mozilla Servo browser engine
- Skylight, a profiler for Rails apps

LIBRARIES?

<https://crates.io/>

Cargo x +

https://crates.io Search




CARGO Search


Browse All Crates | Log in with GitHub







The Rust community's crate host

[Install Cargo](#) [Getting Started](#)

Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.

 **1,912,115** Downloads

 **2,041** Crates in stock

New Crates	Most Downloaded	Just Updated
syr (0.1.0) 	libc (0.1.7) 	ieee754 (0.1.1) 
static_assert (0.1.2) 	rustc-serialize (0.3.14) 	cassandra (0.2.5) 

CRATES & MODULES

A CRATE IS THE RUST EQUIVALENT TO A C LIBRARY OR BINARY

```
rustc hello_world.rs  
./hello_world  
Hello, World!
```

A SINGLE SOURCE FILE DEFINES ONE OR MORE MODULES.

foo.rs:

```
mod foo {  
    fn hello_world() {  
        println!("Hello, World!");  
    }  
  
    mod bar {  
        fn goodbye_world() {  
            println!("Goodbye!");  
        }  
    }  
}
```

```
foo::bar::goodbye_world();
```

A SOURCE FILE CAN REFERENCE OTHER MODULES

main.rs:

```
mod foo;  
  
fn main() {  
    foo::hello_world();  
}
```

```
rustc main.rs
```

Compiles main.rs and foo.rs into main executable.

A SOURCE FILE CAN REFERENCE EXTERNAL MODULES

```
rustc --crate-type dylib foo.rs
```

main.rs:

```
extern crate foo;  
  
fn main() {  
    foo::hello_world();  
}
```

```
rustc main.rs
```

Compiles main.rs and links foo at run time.

CARGO

```
cargo new --bin program_name
```

```
[package]
name = "program_name"
version = "0.0.1"
authors = ["Kai Michaelis <kai.michaelis@rub.de>"]

[dependencies]
num = "~0.0.4"
slow_primes = "~0.1.4"
```

Downloads all dependencies and builds everything

```
cargo build
```

FFI

Calling C from Rust and vice versa

FFI

Call Rust code directly from C or C++

```
#[no_mangle]
pub extern fn hello_rust() -> *const u8 {
    "Hello, world!\0".as_ptr()
}
```

FFI

Call C code from Rust also pretty easy

```
int sqlite3_open(  
    const char *filename,    /* Database filename (UTF-8) */  
    sqlite3 **ppDb         /* OUT: SQLite db handle */  
);
```

translates basically to

```
extern crate libc;  
use libc::c_int,c_char;  
  
#[link(name = "sqlite3")]  
extern {  
    pub fn sqlite3_open(filename: *const c_char,  
                        ppDb: *mut *mut sqlite3) -> c_int;  
}
```

FFI

You wouldn't do this yourself all the way: there is `bindgen`

FFI

And of course, you wouldn't do this with `sqlite3` because there is already crates.io/crates/rusqlite

PATTERN MATCHING & DECONSTRUCTION

TYPES IN RUST

Rust	C++
<code>bool</code>	<code>bool</code>
<code>u8, u16, u32, u64</code>	<code>uint8_t, uint16_t, uint32_t, uint64_t</code>
<code>i8, i16, i32, i64</code>	<code>int8_t, int16_t, int32_t, int64_t</code>
<code>usize, isize</code>	<code>uintptr_t, intptr_t</code>
<code>f32, f64</code>	<code>float, double</code>
<code>char</code>	32 bit Unicode code point

TYPES IN RUST (CONT.)

Rust	C++
<code>str</code>	<code>std::string</code> , UTF-8 encoded
<code>(A, B, ...)</code>	<code>std::tuple<A,B,...></code>
<code>[T; N]</code>	<code>std::array<T,N></code>
<code>&[T]</code>	Pair of iterators
<code>type A = B</code>	<code>using A = B</code>
<code>struct A { ... }</code>	<code>class A { ... }</code>
<code>enum A { ... }</code>	<code>boost::variant</code>

DECONSTRUCTION

```
struct Employee {  
    name: String,  
    age: u8,  
    department: String  
}
```

```
let e1 = Employee {  
    name: "Kai Michaelis".to_string(),  
    age: 29,  
    department: "Engineering".to_string()  
};
```

```
let Employee{ name: n, .. } = e1;
```

```
// Prints "Hello, I'm Kai Michaelis"  
println!("Hello, I'm {}", n);
```



```
let e1 = Employee {
  name:      "Kai Michaelis".to_string(),
  age:       29,
  department: "Engineering".to_string()
};

if let Employee{ age: 67, ..} = e1 {
  println!("Time to retire!");
} else {
  println!("You still got {} years to go", 67 - e1.age);
}
```

ENUMERATIONS

```
#[derive(PartialEq)]
enum Fruit {
    Apple = 1,
    Banana = 2,
    Kiwi,
    Pineapple
}
```

```
fn say_it(fruit: Fruit) {
    if fruit == Fruit::Apple {
        println!("Apple");
    } else if fruit == Fruit::Banana {
        println!("Banana");
    } else if fruit == Fruit::Kiwi {
        println!("Kiwi");
    } else if fruit == Fruit::Pineapple {
        println!("Pineapple");
    }
}
```

```
enum NumberOrText {  
    Number(i32),  
    Text(String)  
}
```

```
fn print_number_or_text(nt: NumberOrText) {  
    match nt {  
        NumberOrText::Number(i) => println!("Number: {}", i),  
        NumberOrText::Text(t) => println!("Text: {}", t)  
    }  
}
```

```
let a: NumberOrText = Number(42);  
let b: NumberOrText = Text("Hello, World".to_string());  
  
// Prints "Number: 42"  
print_number_or_text(a);  
  
// Prints "Text: Hello, World"  
print_number_or_text(b);
```

SIMPLE TREE WALKING

```
use std::boxed::Box;
use std::ops::Deref;

enum Tree {
    Leaf(char),
    Node(Box<Tree>,Box<Tree>)
}

fn depth_first_search(root: &Tree) {
    match root {
        &Tree::Leaf(s) => println!("{}",s),
        &Tree::Node(ref left,ref right) => {
            depth_first_search(left.deref());
            depth_first_search(right.deref())
        }
    }
}
```

SIMPLE TREE WALKING (CONT.)

```
fn main() {
    let tree =
        Box::new(Tree::Node(
            Box::new(Tree::Node(
                Box::new(Tree::Leaf('H')),
                Box::new(Tree::Node(
                    Box::new(Tree::Leaf('e')),
                    Box::new(Tree::Leaf('l'))))))),
            Box::new(Tree::Node(
                Box::new(Tree::Node(
                    Box::new(Tree::Leaf('l')),
                    Box::new(Tree::Leaf('o')))),
                Box::new(Tree::Leaf('!')))))));

    // Prints "Hello!"
    depth_first_search(&tree);
}
```

SMALL PEEK INTO ERROR HANDLING

panic! unwinds the thread

```
fn guess(n: i32) -> bool {
    if n < 1 || n > 10 {
        panic!("Invalid number: {}", n);
    }
    n == 5
}

fn main() {
    guess(11);
}
```

std::option::Option

```
enum Option<T> {  
    None,  
    Some(T),  
}
```


Option<T>

```
fn find(haystack: &str, needle: char) -> Option<usize> {
    for (offset, c) in haystack.char_indices() {
        if c == needle {
            return Some(offset);
        }
    }
    None
}

fn main() {
    let filename = "foobar.txt";
    match find(filename, '.') {
        Some(i) => println!("Filename extension: {}", &filename[i+1..]),
        None => println!("No extension found!"),
    }
}
```

THOUGHTS ON ERROR HANDLING IN RUST

Somewhat mixed...

1. Way better than arbitrary return values
2. Seems harder to use than exceptions
3. Anyone?