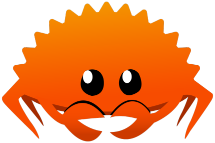
# Introduction to rust and its memory safety

Lukas Prokop

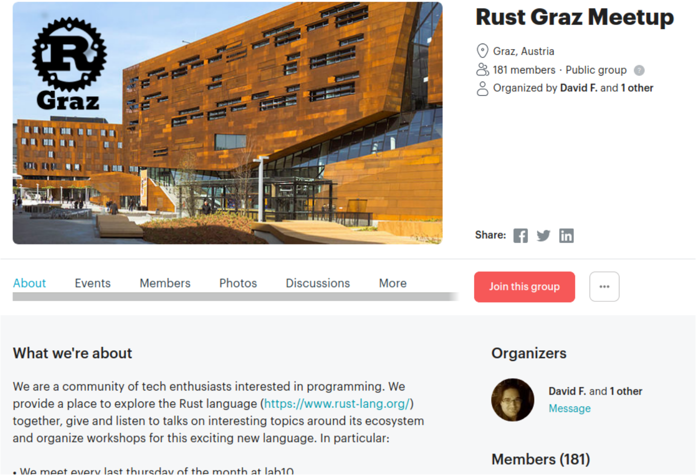
2020-09-18

for IAIK



# About me

- Software developer
- PhD student in post-quantum cryptography at IAIK
- Speaker at RustGraz (twitter @RustGraz)



## Rust Graz Meetup

Graz, Austria

181 members · Public group

Organized by **David F.** and **1 other**

Share:

### What we're about

We are a community of tech enthusiasts interested in programming. We provide a place to explore the Rust language (https://www.rust-lang.org/) together, give and listen to talks on interesting topics around its ecosystem and organize workshops for this exciting new language. In particular:

- We meet every last thursday of the month at lab10

### Organizers

**David F.** and **1 other**

Message

### Members (181)

# What is rust?

## What is rust?

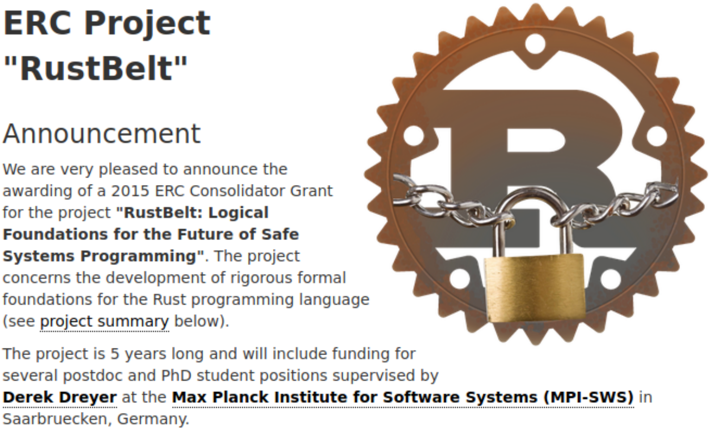
- multi-paradigmatic (imperative, functional)
- systems programming language (easy interop with C, no GC)
- focus on memory safety and concurrency
- uses the LLVM infrastructure
- syntax similar to C++
- zero-cost abstractions like C++
- Modern competitors: Nim, Crystal, D, Zig

“Most loved programming language”
(Stack Overflow Developer Survey, 2016–2020)

# Rust in academia

RustBelt[1]: 32 publications, 4 related projects.

August 2020: Ralf Jung’s PhD dissertation.



## ERC Project "RustBelt"

### Announcement

We are very pleased to announce the awarding of a 2015 ERC Consolidator Grant for the project **"RustBelt: Logical Foundations for the Future of Safe Systems Programming"**. The project concerns the development of rigorous formal foundations for the Rust programming language (see project summary below).

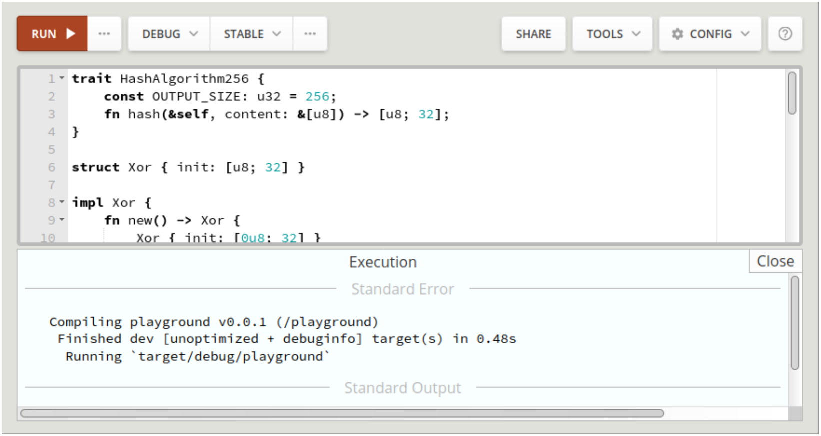
The project is 5 years long and will include funding for several postdoc and PhD student positions supervised by **Derek Dreyer** at the **Max Planck Institute for Software Systems (MPI-SWS)** in Saarbruecken, Germany.

[1] http://plv.mpi-sws.org/rustbelt/

# Tooling

# Try it! Rust Playground

Rust Playground on play.rust-lang.org



```rust
trait HashAlgorithm256 {
    const OUTPUT_SIZE: u32 = 256;
    fn hash(&self, content: &[u8]) -> [u8; 32];
}

struct Xor { init: [u8; 32] }

impl Xor {
    fn new() -> Xor {
        Xor { init: [0u8; 32] }
```

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 0.48s
     Running `target/debug/playground`
```

Also: rust on godbolt.org

# Toolchain

```
curl https://sh.rustup.rs -sSf | sh
```

| | | |
|---|---|---|
| **First release:** | 1.0 | 2015-05-16 |
| **Current release:** | 1.46 | 2020-08-27 |

Stable rust releases every 6 weeks. Beta and Nightly releases exist. Editions are done every 3 years (2015 1.0 ‘stability’, 2018 1.31 ‘productivity’, 2021 ‘maturity’?)

```
rustup install {stable,beta,nightly}
```

```
rustup default {stable,beta,nightly}
```

# Rust compiler

```
rustup doc --book
```

```
rustup update
```

```
rustup self uninstall
```

Rust compiler:

```
rustc --help
```

```
rustc --explain E0382
```

**compilation multi-passes:** HIR → MIR → LLVM-IR

# Rust compiler

```
cargo new [--bin | --lib] NAME
```

```
$ cargo new --bin iaik
     Created binary (application) `iaik` package
$ tree iaik
iaik
├── Cargo.toml
├── .git
│   …
├── .gitignore
└── src
    └── main.rs

10 directories, 18 files

$ cat iaik/Cargo.toml
[package]
name = "iaik"
version = "0.1.0"
authors = ["GIT_COMMITTER_NAME <GIT_COMMITTER_EMAIL>"]
edition = "2018"

# See more keys and their definitions
# at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
```

# Hello World

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

# Hello World

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

```
$ cargo run
   Compiling iaik v0.1.0 (/tmp/iaik)
    Finished dev [unoptimized + debuginfo] target(s) in 0.29s
     Running `target/debug/iaik`
Hello, world!
```

# Hello World

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

```
$ cargo run
   Compiling iaik v0.1.0 (/tmp/iaik)
    Finished dev [unoptimized + debuginfo] target(s) in 0.29s
     Running `target/debug/iaik`
Hello, world!
```

crates.io is rust’s package index

`--release` for optimized build

`--target TRIPLE` to specify architecture

# Hello World

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

```
$ cargo run
   Compiling iaik v0.1.0 (/tmp/iaik)
    Finished dev [unoptimized + debuginfo] target(s) in 0.29s
     Running `target/debug/iaik`
Hello, world!
```

crates.io is rust's package index

`--release` for optimized build

`--target TRIPLE` to specify architecture

```
rustc -C opt-level=3 src/main.rs
```

# Detect common mistakes

```
rustup component add clippy
cargo clippy
```

```
warning: redundant field names in struct initialization
   --> src/main.rs:114:31
    |
114 |   _ => Err(BadEncoding{ encoding: encoding }),
    |                         ^^^^^^^^^^^^^^^^^^
    |                help: replace it with: `encoding`
    |
  = note: `#[warn(clippy::redundant_field_names)]` on by default
  = help: for further information visit https://rust-lang.github.io/…
```

# Normalized code formatting

```
rustup component add rustfmt
cargo fmt
```

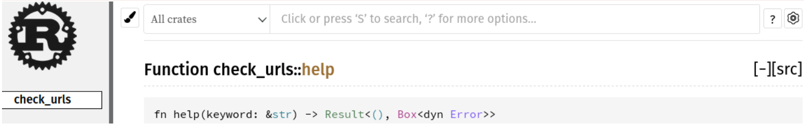
```
% grep -C1 "dst.clone()" main.rs
    let dst_encoding = lookup_encoding(
        dst.clone()
    )?;
% cargo fmt --message-format json
[{"name":"/home/meisterluk/dev/rust/encconv/src/main.rs","mism⌟
↪   atches":[{"original_begin_line":120,"original_end_line":12⌟
↪   2,"expected_begin_line":120,"expected_end_line":120,"origi⌟
↪   nal":"    let dst_encoding = lookup_encoding(\n
↪   dst.clone()\n    )?;","expected":"    let dst_encoding =
↪   lookup_encoding(dst.clone())?;"}]}]
```

Also Rust Language Server:

```
rustup component add rls rust-src          \
rust-analysis
```

# More tools

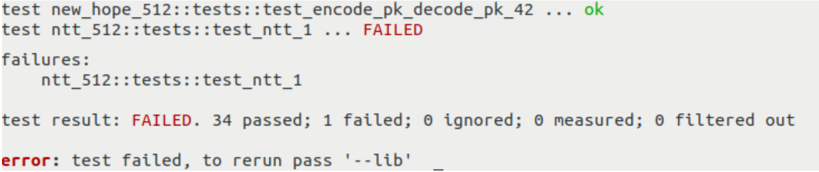
```
cargo doc
```



Function check_urls::help

```
fn help(keyword: &str) -> Result<(), Box<dyn Error>>
```

```
cargo test
```



```
test new_hope_512::tests::test_encode_pk_decode_pk_42 ... ok
test ntt_512::tests::test_ntt_1 ... FAILED

failures:
    ntt_512::tests::test_ntt_1

test result: FAILED. 34 passed; 1 failed; 0 ignored; 0 measured; 0 filtered out

error: test failed, to rerun pass '--lib'  _
```

```
cargo bench
```

# Syntax and semantics

# String formatting

```rust
fn main() {
  println!("{:09b}=000101010 {:>10}=      IAIK", 42, "IAIK");
  println!("{num:06b}=001010 {who}=rustaceans",
           who = "rustaceans", num = 10);
  let variable = 99;
  println!("{} Luftballoons", variable);
  let l: u64 = 0;
  print!("{}\n", format!("{:04x}", l));
}
```

# Immutability by default

```rust
let a: u32 = 0;
a += 1;
```

```
error[E0384]: cannot assign twice to immutable variable `a`
 --> src/main.rs:3:5
  |
2 |     let a: u32 = 0;
  |         -
  |         |
  |         first assignment to `a`
  |         help: make this binding mutable: `mut a`
3 |     a += 1;
  |     ^^^^^^ cannot assign twice to immutable variable
```

# Immutability by default

```rust
let mut a: u32 = 0;
a += 1;




dbg!(&a);
a = dbg!(&a) + 3;
```

```
[example.rs:4] &a = 1
[example.rs:5] &a = 1
```

# Primitive types

```
u8   u16   u32   u64   u128
i8   i16   i32   i64   i128
  isize   usize   f32   f64
          bool   char
```

- → type suffix notation: `42u8`
- → data type boundary value: in stdlib, e.g. `std::u32::MAX`

```
42   42_000   0xFF   0o777   0b0010_1010
1.   1e6   -4e-4f64  std::f64::INFINITY
std::f64::NAN  1usize   true   false  'c'
```

- → type inference to determine data type
- → default integer type is i32

# Strings

```
 "C escape sequences\n, Unicode scalars\u{0042}"

r"skip \backslash interpretation"

b"byte array from ASCII chars"

"multiline
 string"

"eat all \
     leading whitespace"

r#"number of balanced hashes
is arbitrary
"#
```

Two types: `&str` and `String`

# Integer semantics

- `overflow-checks`: true in debug mode, false in release mode
- integer types have method `checked_add`, `overflowing_add`, `saturating_add`, and `wrapping_add`
- `u16 as u32` for coercion
- Logical left shift. Logical right shift on unsigned integer types. Arithmetic shift on signed integer types.
- `assert_eq!(-4 % 7, -4);`

# Composite types: tuples

```rust
fn create_tuple() -> (u32, u64) {
  (4, 2)
}

fn main() {
  let (a, b) = (4, 2);

  // comparison by equality
  assert_eq!((4, 2), create_tuple());

  let pair = create_tuple();
  // access by tuple.{zero-based index}
  assert_eq!(a, pair.0);
}
```

# Composite types: array

```
let all_zero = [0u8; 32];  // type: [u8; 32]
let mut init = [9, 2, 3];  // type: [{integer}; 3]
let initial  = [1u8, 2, 3];// type: [u8; 3]

init[0] = 1;
//init[4] = 1;  // compile or runtime error
assert_eq!(initial, init);
assert_eq!(initial, initial.clone());

let first_5: &[u8] = &all_zero[0..5];
let first_5: &[u8] = &all_zero[ ..5];
let first_6: &[u8] = &all_zero[0..=5];
```

**arrays:** `[u8; 32]`, `[f64; 8]`, …

**slices:** `[u8]`, `[f64]`, …

# Composite types: Vector

`std::vec::Vec<T>` is part of the standard library.

```
let mut vec: Vec<u8> = Vec::new();
```

# Composite types: Vector

`std::vec::Vec<T>` is part of the standard library.

```
let mut vec = vec![];
```

# Composite types: Vector

`std::vec::Vec<T>` is part of the standard library.

```
let mut vec = vec![];
vec[0];
// thread 'main' panicked at
// 'index out of bounds: the len is 0 but the index is 0',
```

# Composite types: Vector

`std::vec::Vec<T>` is part of the standard library.

```
let mut vec = vec![];
```

# Composite types: Vector

`std::vec::Vec<T>` is part of the standard library.

```
let mut vec = vec![];
vec.push(5);
vec.extend(vec![3, 4]);
vec[0] = 7;
```

# Composite types: Vector

`std::vec::Vec<T>` is part of the standard library.

```
let mut vec = vec![];
vec.push(5);
vec.extend(vec![3, 4]);
vec[0] = 7;

assert_eq!(vec[0], 7);
assert_eq!(vec.len(), 3);
assert_eq!(vec.pop(), Some(4));
```

# Composite types: Vector

`std::vec::Vec<T>` is part of the standard library.

```
let mut vec = vec![];
vec.push(5);
vec.extend(vec![3, 4]);
vec[0] = 7;

assert_eq!(vec[0], 7);
assert_eq!(vec.len(), 3);
assert_eq!(vec.pop(), Some(4));

vec.sort();
vec.sort_unstable();
```

# Composite types: Vector

`std::vec::Vec<T>` is part of the standard library.

```
let mut vec = vec![];
vec.push(5);
vec.extend(vec![3, 4]);
vec[0] = 7;

assert_eq!(vec[0], 7);
assert_eq!(vec.len(), 3);
assert_eq!(vec.pop(), Some(4));

vec.sort();
vec.sort_unstable();

let elements: &[u8] = &vec[0..2];
```

# Composite types: struct

```rust
struct HashAlgorithm {
  state: [u8; 32],
  security_margin: u32,
  names: Vec<String>,
}
```

# Composite types: struct

```
struct HashAlgorithm {
  state: [u8; 32],
  security_margin: u32,
  names: Vec<String>,
}

fn main() {
  let h = HashAlgorithm{
    state: [0u8; 32],
    security_margin: 128,
    names: vec!["SHA-2".to_string(),
                "SHA-256".to_string()],
  };
  println!("aliases → {}", h.names.join(", "))
}
```

```
Output: aliases → SHA-2, SHA-256
```

# Composite types: struct must be sized

```rust
struct HashAlgorithm {
    state: [u8; 32],
    security_margin: u32,
    names: Vec<String>,
    input_bytes: [u8],
}
```

Structs must be sized:

```
error[E0277]: the size for values of type `[u8]` cannot be known at compilation time
  --> src/main.rs:12:13
   |
12 |       let h = HashAlgorithm{state: internal_state,
   |  _____________^
13 | |                           security_margin, names};
   | |_ doesn't have a size known at compile-time _____^
   |
```

# Composite types: struct: alignment

```
use std::mem::{size_of, align_of};



struct HashAlgo {
  security_margin: u32, //  4 bytes
  names: Vec<String>,   // 24 bytes
  state: [u8; 9],       //  9 bytes
}

fn main() {
  assert_eq!(size_of::<HashAlgo>(), 40);
  assert_eq!(align_of::<HashAlgo>(), 8);
}
```

# Composite types: struct: alignment

```rust
use std::mem::{size_of, align_of};

#[repr(C)]
struct HashAlgo {
  security_margin: u32, //  4 bytes
  names: Vec<String>,   // 24 bytes
  state: [u8; 9],       //  9 bytes
}

fn main() {
  assert_eq!(size_of::<HashAlgo>(), 48);
  assert_eq!(align_of::<HashAlgo>(), 8);
}
```

# Composite types: struct

```
#[derive(Debug)]   // HashAlgo { security_margin: 32,
                   // names: [], state: [0, 0 … ] }
struct HashAlgo {
  security_margin: u32,  //   4 bytes
  names: Vec<String>,    // 24 bytes
  state: [u8; 9],        //   9 bytes
}
fn main() {
  let h = HashAlgo{ security_margin: 32,
      names: vec![], state: [0u8; 9] };
  println!("{:?}", h);
}
```

`derive(Debug)` is an attribute macro implementing Debug automatically.

`{:?}` asks for *Debug* representation.

`{}` asks for *Display* representation.

# Enumerables

```
type Digest = [u8; 32];    // type alias: one type, 2 names
```

# Enumerables

```
type Digest = [u8; 32];   // type alias: one type, 2 names

enum Result {             // enumerable
  Okay(Digest),
  Error(String),
}
```

# Enumerables

```
type Digest = [u8; 32];   // type alias: one type, 2 names

enum Result {             // enumerable
  Okay(Digest),
  Error(String),
}

fn generate_digest() -> Result {
  Result::Okay([42u8; 32])
}
```

# Enumerables

```
type Digest = [u8; 32];   // type alias: one type, 2 names

enum Result {             // enumerable
  Okay(Digest),
  Error(String),
}

fn generate_digest() -> Result {
  Result::Okay([42u8; 32])
}

fn main() {
  match generate_digest() {
    Result::Okay(d) => {
      for byte in d.iter() {
        print!("{:02X}", byte);
      }
      println!("");
    },
    Result::Error(msg) => eprintln!("error: {}", msg),
  }
}
```

# Error handling in rust

`std::result::Result<T, E>`

- `Ok(T)`
- `Err(E)`

No exceptions, no error codes.

# Error handling in rust

`std::result::Result<T, E>`

- `Ok(T)`
- `Err(E)`

No exceptions, no error codes.

`std::option::Option<T>`

- `None`
- `Some(T)`

# Error handling in rust

`std::result::Result<T, E>`

- `Ok(T)`
- `Err(E)`

No exceptions, no error codes.

`std::option::Option<T>`

- `None`
- `Some(T)`

```rust
let result = Some(value);
result.unwrap();  // return Some value or panic
result.unwrap_or(default_value);  // … or default
```

# Question mark operator

The question mark operator exits early in case of `Err` or returns the value otherwise.

```rust
fn compile(src: &str) -> Result<(), Error> {
  let tokens = tokenize(&src)?;
  let ast = parse(&tokens)?;
  // …
  Ok(())
}
```

Return type of function must be a corresponding `Result`.

# Question mark operator

It is can be rewritten with a match expression:

```
fn compile(src: &str) -> Result<(), Error> {
  let tokens = match tokenize(&src) {
    Err(E) => return Err(E),
    Ok(ts) => ts,
  };
  let ast = parse(&tokens)?;
  // …
  Ok(())
}
```

# Control flow

```
if cond { … } else { … }
if let Some(val) = result { … }

loop { … }
while cond { … }
while let Some(val) = result { … }
for i in 0..1024 { … }
for elem in &vec { … }

let pair = (2, -2);
let kind = match pair {
  (0, 0)                       => "invalid",
  (x @ 1..=5, y) if x + y == 0 => "opposites",
  (x, _)         if x % 2 == 1 => "odd and something",
  _                            => "whatever",
};
println!("{:?} are {}", pair, kind);
```

# Functions, ownership and borrowing

# Function syntax

```
fn named(name1: T1, name2: T2) -> T_RETURN {}
let unnamed = |name1: T1, name2: T2| -> T_RETURN { };
let short   = |name1    , name2    |             { };
```

# Function syntax

```
fn named(name1: T1, name2: T2) -> T_RETURN {}
let unnamed = |name1: T1, name2: T2| -> T_RETURN { };
let short   = |name1    , name2    |             { };
```

Example of anonymous function usage:

```
use std::thread;
let handler = thread::spawn(|| {
    println!("Hello World!");
});
handler.join().unwrap();
```

# Function syntax

```
fn named(name1: T1, name2: T2) -> T_RETURN {}
let unnamed = |name1: T1, name2: T2| -> T_RETURN { };
let short   = |name1    , name2    |             { };
```

Example of anonymous function usage:

```
use std::thread;
let handler = thread::spawn(|| {
    println!("Hello World!");
});
handler.join().unwrap();
```

Last expression is return value (return keyword only for early exit):

```
const fn get_42() -> u32 {
    42
} // const fn = C++ constexpr
```

# Function semantics

- No variadic arguments → slices
- Multiple return values → tuples
- Functions can be nested
- Definitions order in rust does not matter
- Blocks {} define scopes
- inlining via `#[inline]`, `#[inline(always)]`, or `#[inline(never)]`

# Function semantics

- No variadic arguments → slices
- Multiple return values → tuples
- Functions can be nested
- Definitions order in rust does not matter
- Blocks `{}` define scopes
- inlining via *`#[inline]`*, *`#[inline(always)]`*, or *`#[inline(never)]`*

Memory management notes:

- Stack allocation for local variables
- Call by value or call by reference
- Arguments and return types must have known size at compilation time

# Ownership

- Each value in Rust has a variable that’s called its *owner*
- There can only be one owner at a time
- Ownership can *move* from one variable to another
- When the owner goes out of scope, the value will be “dropped”

# Ownership example

```rust
#[derive(Debug)]
struct Stats { score: u32 }

fn sub(mut s: Stats) {
  s.score += 1;
}

fn main() {
  let a = Stats { score: 8 };
  sub(a);

}
```

# Ownership example

```rust
#[derive(Debug)]
struct Stats { score: u32 }

fn sub(mut s: Stats) {
  s.score += 1;
}

fn main() {
  let a = Stats { score: 8 };
  sub(a);
  println!("{:?}", a);
}
```

# Ownership example

```
error[E0382]: borrow of moved value: `a`
  --> src/main.rs:10:20
   |
8  |     let a = Stats { score: 8 };
   |         - move occurs because `a` has type `Stats`,
   |           which does not implement the `Copy` trait
9  |     sub(a);
   |         - value moved here
10 |     println!("{}", a);
   |                    ^
   |         value borrowed here after move
```

# Ownership example

```rust
#[derive(Debug)]
struct Stats { score: u32 }

fn sub(mut s: Stats) {
  // owner of Stats instance = `s`
  s.score += 1;
  // `s` goes out of scope → Stats instance is dropped
}

fn main() {
  let a = Stats { score: 8 };
  // owner of Stats instance = `a`
  sub(a); // move Stats instance: `a` → `s`
  println!("{:?}", a); // has been dropped already
}
```

# Ownership example

Solutions:

- Use `#[derive(Debug,Copy,Clone)]`. Then sub uses copied instance. Results in `Stats { score: 8 }`
- Return `Stats` instance and assign it again in `main`.
- Use references (*borrowing* ownership)

Benefits of ownership for memory safety:

- we can pin-point when a variable is dropped (across threads!)

# References

```rust
let mut a = Stats { score: 8 };

let shared_ref = &a;
println!("{:?}", *shared_ref);

let mutable_ref = &mut a;
println!("{:?}", *mutable_ref);
```

Reference a value with &.

Dereference a reference with *.

*auto-dereferencing:* e.g. &**u32** given, **u32** required? Dereference automatically. Best practice: Dereference explicitly.

# References

Rules:

- one or more *shared* references (&T) to a resource
- exactly one *mutable* reference (&**mut** T)
- either or, not both! (“aliasing xor mutation”)

Benefits of reference limitations for memory safety:

- one writer XOR n readers in concurrent context
- prevents data races

# Ownership example with borrowing

```rust
#[derive(Debug)]
struct Stats { score: u32 }

fn sub(s: &mut Stats) {
  s.score += 1;
}

fn main() {
  let mut a = Stats { score: 8 };
  // ownership of `a` is borrowed to `s`
  sub(&mut a);
  // ownership of `s` is returned back to `a`
  println!("{:?}", a);
}
```

# Lifetimes

Basic idea:

- How long does the referenced value live?
- Where do values live?
    - scopes
    - `'static` (i.e. “lives as long as the program”)
    - …
- A lifetime is denoted `'a`, `'b` or `'c`
- *Lifetime elision*: compiler has automatic rules which derive lifetimes
- In function signatures and struct members, we sometimes need to declare the lifetime explicitly.

Benefits of lifetimes for memory safety:

- Solves the use-after-free problem

# Lifetimes example

```
struct Stats {
  score: &mut u32,
}
```

…with lifetimes becomes …

```
struct Stats<'a> {
  score: &'a mut u32,
}
```

# Methods

- methods can be associated with a `struct`
- does not depend on `self` → static methods
- `let op = Xor::new();`
  `op.name()` is syntactic sugar for `Xor::name(op)`

```
struct Xor { init: [u8; 32] }

impl Xor {
  fn new() -> Xor {
    Xor { init: [0u8; 32] }
  }

  fn name(&self) -> &'static str { "xor" }
}
```

# Traits

- Nominal type system, based on Hindley-Milner
- traits like contracts, default method implementations possible
- trait must be in scope to be used (`use` keyword)
- no subtyping, no inheritance, but method overloading
- **marker traits:** no methods to implement, but `impl Trait for Type {}` to declare some property e.g. `std::marker::Send`: Types that can be transferred across thread boundaries.
- **extension traits:** Add functionality to primitive/stdlib types

**Trait coherence:**

> *"… we can implement a trait on a type only if either the trait or the type is local to our crate"*

# Traits

```rust
trait HashAlgorithm256 {
  const OUTPUT_SIZE: u32 = 256;
  fn hash(&self, content: &[u8]) -> [u8; 32];
}

impl HashAlgorithm256 for Xor {
  fn hash(&self, content: &[u8]) -> [u8; 32] {
    let mut digest = self.init;
    for (i, byte) in content.iter().enumerate() {
      digest[i % 32] ^= byte;
    }
    digest
  }
}
```

# Generics

- T is an abstract type
- rust allows to be generic over types (but not values)
- implementation by *monomorphization*:
  optimized code for each type in executable (like C++ templates)

```
fn add<T>(a: T, b: T) -> T
  where T: std::ops::Add<Output = T>
{
  a + b
}

fn main() {
  println!("{}", add(3u8, 5));
  println!("{}", add(3f32, 5.));
}
```

# unsafe

```
#[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
fn rdtsc {
    let (mut eax, mut ebx, mut ecx) = (0, 0, 0);
    unsafe {
        asm!("rdtscp",
              out("eax") eax,
              out("ecx") ecx,
              out("edx") edx);
    }
}
```

1. Dereference a raw pointer (**`const *`**)
2. Call an **`unsafe`** function or method
3. Access or modify a mutable static variable
4. Implement an **`unsafe`** trait
5. Access fields of `unions`

# Macros

- Three kinds of macros
  1. function-like macros (`println!("hi")`)
  2. derive macros (`derive(Debug)`)
  3. attribute-like macros (`cfg(target_arch = "x86")`)

```
macro_rules! shake {
  (update $base:ident with $($elem:expr, )*)
    => { $( $base.update($elem); )* };
}
```

**Input:**

```
  shake!(update h with &data, &[b' '], &data2,);
```

**Output:**

```
  h.update(&data);
  h.update(&[b' ']);
  h.update(&data2);
```

# Stack versus Heap

- Stack requires `Sized`, opposite is `?Sized`
- Heap allocations via special types
- e.g. Box, Rc, Arc

```rust
fn main() {
  let a = Box::new(5);
  let b = Box::new(6);
  assert_eq!(11, *a + *b);
}
```

rust is awesome

# Type inference example

Assume `fn store(toml: TomlTree)`.

```
let input: &str = r#"[package]\nkey = "value" "#;
match input.parse() {
  Ok(toml) => store(toml),
  Err(error) => panic!("failed to parse TOML"),
};
```

Implementation of str.parse:

```
pub fn parse<F: FromStr>(&self)
    -> Result<F, <F as FromStr>::Err>
{
    FromStr::from_str(self)
}
```

# WebAssembly

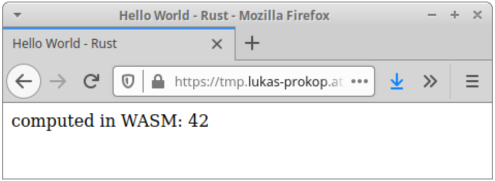
```
rustup target add wasm32-unknown-unknown
```

```
[dependencies]            # in Cargo.toml
wasm-bindgen = "0.2"
```

```
#[wasm_bindgen]           // in lib.rs
pub fn add(a: i32, b: i32) -> i32 { a + b }
```

```
wasm-pack build --target web
```

```
const wasm = await init("./pkg/webassembly_example_bg.wasm");
const sum = wasm.add(20, 22);
document.body.textContent = `computed in WASM: ${sum}`;
```


Hello World - Rust - Mozilla Firefox
Hello World - Rust
https://tmp.lukas-prokop.at

```
computed in WASM: 42
```

via wasmbyexample.dev

# Critical parts

- Long compilation times
  - compare with golang
  - see also perf.rust-lang.org
- Multi-pass compiler
  1. fix lexical errors
  2. fix types
  3. fix borrows
  4. fix lifetimes
  5. …
- “`Vec<Rc<RefCell<Box<Trait>>>>`?
  Is there a better way?”

# Resources

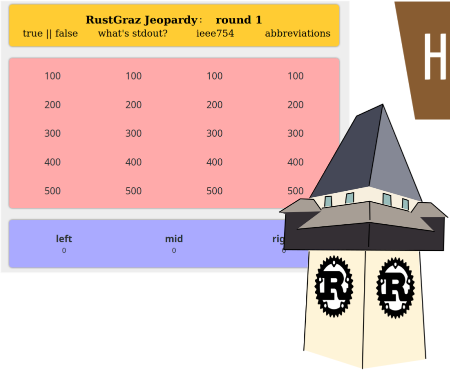
Official documentation:

- Rust book
- Rust by example
- Rust website: learn page

Event-based:

- RustFest conference (talks on youtube)
- RustGraz local meetup

Book: Rust in Action



**RustGraz Jeopardy** : **round 1**

| true \|\| false | what's stdout? | ieee754 | abbreviations |
|---|---|---|---|
| 100 | 100 | 100 | 100 |
| 200 | 200 | 200 | 200 |
| 300 | 300 | 300 | 300 |
| 400 | 400 | 400 | 400 |
| 500 | 500 | 500 | 500 |

| left | mid | rig |
|---|---|---|
| 0 | 0 | 0 |

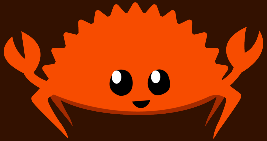
# Hacker Jeopardy

2020-09-30

TU Graz

Inffeldgasse 16b

# Thank you! Q/A?



# Concurrency

- SIMD, atomic ops, Mutex/CondVar, Once, RwLock, Barrier
- crossbeam-channel
- OS threads
- async & await since rust 1.39 requires executor/reactor/waiter like tokio/smol/async-std uses Futures
- shared memory

# Concurrency

Are we …yet?

- GUI
- (Machine) learning
- Web
- game
- audio

via Mozilla: Areweyet