TD-1

Rust Introduction: Cargo, Crates, Rust project, Hello World

Pierre Cochard and Tanguy Risset

5TC-RUST 2025

A word on 5TC-RUST course

- **Course website**: https://5tc-rust-cc1d61.gitlabpages.inria.fr/
- **Course Moodle**: https://moodle.insa-lyon.fr/course/view.php?id=10386
- **Teachers**: Pierre Cochard and Tanguy Risset (from Emeraude Team at Citi https://team.inria.fr/emeraude/)
- **32h**
  - ~20h as course-labs (on your laptop)
  - ~10h for a mini-project in Rust
  - Small demo at the end of the course
- Largely inspired by information found on the web
  - https://www.rust-lang.org/learn/
  - Chat-GPT and other friends

History and Origins

- **Created:** 2006 by **Graydon Hoare** as a personal project
- **Backed by Mozilla:** since **2009**, for the **Servo** rendering engine project
- **First stable release: Rust 1.0** in **2015**
- **Today:** open source, maintained by the **Rust Foundation** (https://rustfoundation.org/, AWS, Google, Huawei, Microsoft, Mozilla, etc.)
- **Philosophy:** safety, performance, and fearless concurrency

The Philosophy Behind Rust

- **Fearless concurrency** — write concurrent code without fear of data races
- **Memory safety by design** — guaranteed at compile time
- **Zero-cost abstractions** — expressive yet as efficient as C/C++
- **Empowering developers** — the compiler is your ally, not your enemy
- **Clarity and rigor** — every rule aims for readability and robustness
- **Community values** — learning-oriented, welcoming, and focused on code quality
- **Modern ecosystem:** — Cargo, crates.io, built-in documentation

The NULL pointer reference: Tony Hoare speaking in 2009

- "I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years."

Rust in the Modern Ecosystem

- Used in:
  - **Embedded systems and operating systems** (e.g., TockOS, Redox)
  - **Browsers and rendering engines** (Servo, Firefox)
  - **Blockchain, WebAssembly, and cloud-native tools** (Parity, Deno, AWS)
- **Recent recognition:**
  - Most loved language on Stack Overflow since 2016
  - Growing adoption in both open-source and industrial projects
- **Community:** inclusive, rigorous, and highly active

Rust and the Linux Kernel

- **Rust in Linux**: officially accepted into the kernel since **version 6.1 (Dec. 2022)**
- **Goal:** enable **memory-safe kernel and driver development**
- **Maintained by:** the **Rust-for-Linux** project
- **Key motivations:**
  - Reduce memory safety bugs (use-after-free, buffer overflows)
  - Allow safer device drivers and kernel modules
  - Modern tooling (Cargo, rustfmt) integrated with kernel workflows
- **Supported architectures:** x86_64, ARM64 (others in progress https://doc.rust-lang.org/beta/rustc/platform-support.html)
- **Endorsed by** Linus Torvalds & Google (used in Android drivers)

> *Rust is now a **first-class citizen** in the Linux ecosystem.*

Outline of the course

- **TD1**: Rust Introduction: Cargo, Crates, Rust project
- **TD2**: Ownership, borrowing, mutability, heap and stack in Rust
- **TD3**: Compound and Collection types
- **TD4**: Struct, enums, Traits and Object definitions
- **TD5**: Rust as a safe language: Pattern Matching, Handling Results/Errors, Options and Simple macros
- **TD6**: Programming methodology: crates, modules and third-party libraries
- **TD7**: Closures, Threads, Async and Concurrency
- **Project** (explained in **TD8** on the web)

Rust Introduction

Setting up the environment for using Rust

- Use visual studio code as Rust IDE
  - https://code.visualstudio.com/ (Warning: *Visual Studio code* **is not** *Visual Studio*)
  - `sudo apt install code` on Linux
- Install the `cargo` command with `rustup`
- Rust should work on Windows, Linux and MacOS
- Hands-on:
  - Install Rustup and cargo as indicated on TD1

Installing Rust on Windows

- Download and run `rustup-init.exe` from https://www.rust-lang.org/tools/install
- Rust needs the **Microsoft C++ build tools** (MSVC linker) to build `.exe` files
  - if missing, choose `1) Quick install via the Visual Studio Community installer`
  - then `1) Proceed with standard installation`
- **Close and reopen** all terminals and VS Code, then check: `rustc --version`, `cargo --version`

Windows: common problems

- `linker 'link.exe' not found`
  - the C++ build tools are missing: install *Build Tools for Visual Studio* with the **Desktop development with C++** workload
- `'cargo' is not recognized`
  - the terminal was opened before the installation: close **all** terminals and VS Code, and reopen them
- Alternative: **WSL** (`wsl --install`), then follow the Linux instructions

What is cargo used for?

- Cargo is Rust’s official package manager and build system
- it handles:
  - Dependency Management
  - Project Configuration
  - Building and Compilation
  - Testing
  - Documentation
  - Publishing Packages

Rust Hello World

- Hands on: **Question 1**
  - Execute this command: `cargo new hello_world`
  - Check the generated files:
    - `Cargo.toml` is the project configuration file written in the **TOML**.
    - `src/main.rs` is the Rust *main* file (the program **entrypoint**).
  - build your project with the `cargo build` command.
  - Where is the generated executable file?
  - What is the bang (`!`) after `println` ?

A word about `println!`

- `println!` is not a function, it is a *macro*.
- Macros are called with a trailing bang
- they are a way of writing code that writes other code (*metaprogramming*)
- easy to use but complex to build

About the code examples

- **Snippets without `fn main()`**: a few statements illustrating a notion
  - copy them **inside** the `main` function of your project
- **Complete programs with `fn main()`**: items (`fn`, `struct`, `use`...) followed by `main` and
  - copy them to **replace** the contents of `src/main.rs`
  - use Rust playground: https://play.rust-lang.org/
- Pasting a complete program into a file that already has a `main`:
  "the name `main` is defined multiple times"

Types, Variables and Mutability

Rust is a *statically typed* language. *scalar* types are: *integer types*, *floating point types*, *boolean* and *characters*. Other types (i.e. *compound types*, *slices* or types defined by the user) will be introduced later.

The `let` keyword declares and initializes a variable, its type can be inferred by the compiler or stated explicitly:

```
let x: f32 = 2.0;
let y = 3;
```

Integer Types in Rust

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
Architecture-dependent	`isize`	`usize`

Literals (i.e. constant) in Rust

Number literals	Example
Decimal	`98_222`
Hex	`0xff`
Octal	`0o77`
Binary	`0b1111_0000`
Byte (`u8` only)	`b'A'`

Floats, Boolean and Characters

Floating points are `f32` or `f64`.

Boolean type (`bool`) has only two values: `true` and `false`.

Characters are indicated with simple quotes: `let c = 'z';`. they are stored on 4 bytes, as *unicode scalar values*, but in `String` or string slices (`&str`), they are stored in UTF-8.

Variables, Types and Mutability

- Variables must be declared "***mutable***" if they are changed during their lifetime
- **Scope** (i.e. portion of code associated to a variable) is a very important notion in Rust: it determines when values are freed and how long references remain valid (see TD2)
- **Question 2**:
  - What is the problem with the program below? Does it compile?

```
let x = 5;
println!("The value of x is: {x}");
x = 6;
println!("The value of x is: {x}");
```

Question 3:

- What are the values printed by the program below?

```
let x = 5;
{
    let x = x + 1;
    {
        let x = x * 2;
        println!("The value of x in the inner scope is: {x}");
    }
}
println!("The value of x is: {x}");
```

First Introduction to Character String

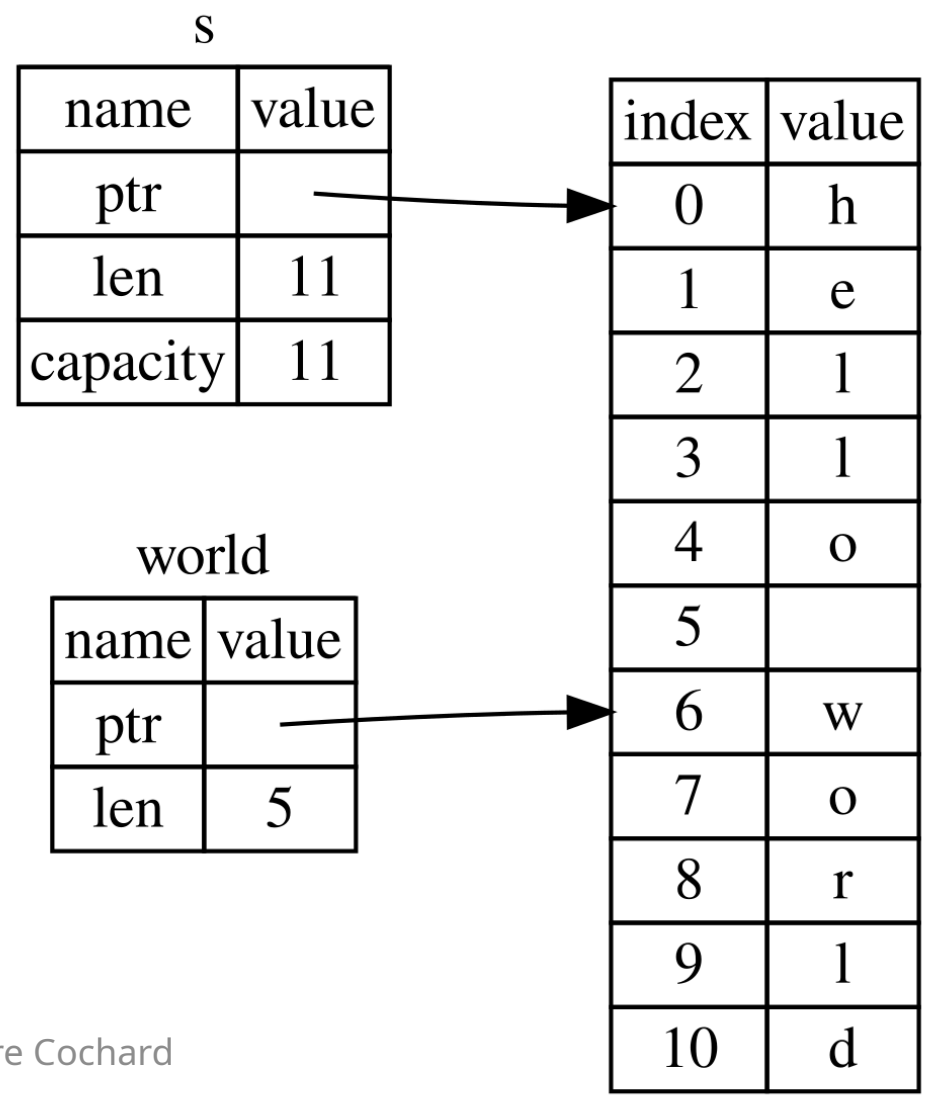
For representing strings, Rust proposes two types: `String` and *string slices* (`&str`).

- `String` is a powerful type, vector of UTF-8 bytes that can be mutated, allocated on the heap.
- `&str` is an immutable fat pointer (pointer + length) of UTF-8 bytes. It is often owned by something else: a String for instance.

```
let s = String::from("hello world");

  let hello = &s[0..5];
  let world = &s[6..11];
```

Example of String and &str



Function in Rust

- Functions in Rust are like in other languages. They are declared with the keyword `fn`.
- **Question 4**: Write a function `fibonacci: fibonacci(n: i32) -> i32`, which computes element n of the fibonacci sequence.

We will not use a recursive solution, but rather a for loop whose syntax will be: `for i in 2..n+1` (half-open range) and mutable variables.

We recall the definition of the fibonacci function fib:

```
fib(0)=1
fib(1)=1
fib(i)=fib(i-1)+fib(i-2)  for i >= 2
```

Flow Control in Rust

The most common constructs that let you control the flow of execution of Rust code are *if expressions* and *loops*.

If expressions

```
fn main() {
    let number = 3;

    if number < 5 {
        println!("condition was true");
    } else {
        println!("condition was false");
    }
}
```

The `else` part is optional, and can be replaced by `else if`

If expressions can be used on the right side of a let statement.

Loops in Rust

The `loop` keyword can be used to create infinite loops, which can be interrupted by the `break` keyword.

```
let mut counter = 0;

  let result = loop {
      counter += 1;

      if counter == 10 {
          break counter * 2;
      }
  };
  println!("result: {result}");
```

While Loops

The `while` construct is quite standard:

```
let mut number = 3;

    while number != 0 {
        println!("{number}!");

        number -= 1;
    }
```

For Loops

The `for` loop in Rust is more specific. It is designed to iterate on an object that can be transformed into an iterator with the `into_iter()` method (i.e. objects that implement the `IntoIterator` trait). For instance, for loops can iterate over the element of an array.

```
    let a = [10, 20, 30, 40, 50];

    for element in a {
        println!("the value is: {element}");
    }
```

Ranges in Rust

The most common ranges (e.g. `0..10`, `0..=10`) can be used in loops because they implement the `IntoIterator` trait). The `iter()` method can be used to iterate over references to the elements of a collection, without consuming it:

```
fn main() {
    let v1 = vec![1, 2, 3];

    let v1_iter = v1.iter();

    for element in v1_iter {
        println!("the value is: {element}");
    }
}
```

End of TD1