TD-1

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

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: Closures, Threads, Channels and Concurrency
- Project (explained in TD7 on the web)

Rust Introduction

Setting up the environnement for using Rust

- Use visual studio code as Rust IDE
  - https://visualstudio.microsoft.com/fr/downloads/
  - `sudo apt install code` on Linux
  - Install the `cargo` command with `rustup`
  - should work on Windows, Linux and MacOS
- Hands-on:
  - Install Rustup and cargo as indicated on TD1

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 world about `println!`

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

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 an important notion in Rust that can be manipulated as an object
- 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? </question>

```
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}");
```

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
```

Struct in Rust

- As a class in C++, types can be defined and can implement different methods.
- *e.g.* a `Complex` struct:

```
struct Complex {
   re: f32,
   im: f32,
}

fn build_complex(re: f32,im:f32)->  Complex {
   Complex {re,im}
}
```

Generic Types

- As *templates* in C++, Rust enables the use of *genericctype* in function or struct, enums or methods definitions, *e.g.*:

```
struct Point<T> {
    x: T,
    y: T,
}

let intP = Point { x: 5, y: 10 };
let floatP = Point { x: 1.0, y: 4.0 };
```

- Question 5: Modify the program of `Complex` creation above by using a type `struct Complex<T>` which uses a generic type. Test it with `println!`

Traits: Defining common behaviour

- A **trait** defines a functionality that a particular type has and can share with other types.
- Traits are similar to a feature often called *interfaces* in other languages.
- For instance the `copy` or `clone` are methods traits implemented in traits `Copy` and `Clone`
- the `fmt` method (of the trait `std::fmt::Display`) enables the use of `{}` format specifier in `println!`

- *e.g.:* implementation of `Clone` trait for `Complex` :

```
impl Clone for Complex {
  // Self refers to the type that is implementing the trait:
    fn clone(&self) -> Self {
        Complex{re: self.re, im: self.im}
    }
}
```

- Question 6: Implement the Clone trait for the `struct Complex<T>` type defined before. You will have to use `impl<T: Clone>` to ensure that the `T` generic type implements the `Clone` trait.

Deriving traits

- For certain traits, the compiler is capable of providing basic implementations for some
  traits via the **#[derive] attribute**
- e.g.: derive `Clone` for `Complex`

```
#[derive(Clone)]
struct Complex {
    re: f32,
    im: f32,
}
```

Quesion 7

- How could we use `println!` to display `Complex` variables? Test the two following methods:

  1. Implement the `std::fmt::Display` trait for type `Complex`.

  This will require to:

  - use `std::fmt`
  - search for the prototype of the `Display` trait
  - use the macro `write!` to print fields

  2. Derive the `Debug` trait that includes the `std::fmt::Display` trait and use the `"{:?}` format.

Move semantics

- In Rust, **move semantics** refers to the ownership transfer of data from one variable to another.
- By default an assignement such as `let y = x` implies a transfer of ownership of the content of `x` to `y`.
- In order to dupplicate a variable (as it would be done in any language), one has to clone it or to implement the `Copy` trait.
- Deriving the `Copy` trait for a type changes the semantic of the assignement: the assignement is now a copy, not a move.

Question 8

- Write a Rust program that defines a structure `Cube` and prints its size

Question 9

Can you print the cube itself? Can you write:

```
println!("My cube: {}", Cube{c:0.5});
```

If not, how can you make the cube printable?

Question 10:

- Define a variable `x` assigned to a given cube, print it and then define> a second variable `y` defined by `let y = x;` . Then print `x` again, what is the problem?

End of TD1