An introduction to Rust

Jonathan Dönszelmann & Vivian Roest

Delft University of Technology

2024-11-18

TU Delft

Study Goals

After this course, students will be able to:

- Explain the programming language concepts followed in Rust.
- Design, implement and debug a small software system from scratch in Rust following the language standard including proper coding style.
- Set up a project and build environment, using the Rust ecosystem.
- Use Git to version and share source code contributions for collaborative development.
- Evaluate and integrate code contributions of other team members.

In simpler terms

We will teach you about:

- Programming
- Choices in programming languages
- Making safe, reliable and correct programs
- Developing software together

Software Fundamentals

- Programming
- Choices in programming languages
- Making safe, reliable and correct programs
- Developing software together

Hardware Fundamentals

- Digital Computer Systems
- Discrete Signals and Systems
- Design of Control Systems

Part 1

- Lectures (twice a week)
- Individual assignment
- Labs and Tutorials (twice a week)

Part 2

1. Group project
2. No lectures!
3. Mandatory attendance of **at least one lab a week!**

Staff

- Vivian Roest (Head TA)
- Shashwath Suresh
- Cleo Barik
- Felipe Perez
- Andre Herrera Gama

Evaluation

- Individual Assignment (50%)
- Group Project (50%)
- Git Assignment (pass/fail)

Resources

Book Recomendations:

- The Rust Programming Language Available Online
  by Steve Klabnik; Carol Nichols; The Rust Community,
- Rust for Rustaceans by Jon Gjengset

Resources

Book Recomendations:

- The Rust Programming Language Available Online
  by Steve Klabnik; Carol Nichols; The Rust Community,
- Rust for Rustaceans by Jon Gjengset

Software:

- Linux!!
- Install Rust through `rustup`, avoid Ubuntu/Debian's repository!!

Let’s start!

- Why choosing a programming language matters
- Why do we teach you Rust?
- Some basics of Rust

Tell me about you

Question:

What programming languages have you used in the past? And what for?

- work, hobby, in teams, alone?

Drones!

Question:

What properties do we care about for the software of this drone?

Programming languages for Embedded Systems

- We're teaching about Rust

Question:

What other options are there?

Programming languages for Embedded Systems

- We're teaching about Rust
- C
- C++

From the https://osdev.org wiki: people have written kernels in:

Forth, Lisp, C#, Modula-2, Ada, Bliss, Smalltalk, PL/1, Assembly, Zig, D

Programming languages for Embedded Systems

- We're teaching about Rust
- C
- C++

From the https://osdev.org wiki: people have written kernels in:

Forth, Lisp, C#, Modula-2, Ada, Bliss, Smalltalk, PL/1, Assembly, Zig, D

Question:

Can you use python on embedded systems?

Programming languages for Embedded Systems

- We're teaching about Rust
- C
- C++

From the https://osdev.org wiki: people have written kernels in:

Forth, Lisp, C#, Modula-2, Ada, Bliss, Smalltalk, PL/1, Assembly, Zig, D

Question:

Why shouldn't you use python on an embedded system?

Programming languages for Embedded Systems

- We're teaching about Rust
- C
- C++

From the https://osdev.org wiki: people have written kernels in:

Forth, Lisp, C#, Modula-2, Ada, Bliss, Smalltalk, PL/1, Assembly, Zig, D

So clearly, the features of a programming language matters.

Programming languages for Embedded Systems

Question:

What properties do we care about when we want to use a programming language for embedded systems?

Programming languages for Embedded Systems

Question:

What properties do we care about when we want to use a programming language for embedded systems?

- Compiled
- Low-level access to locations in memory
- Precise control over all program resources
- Guarantees about correctness

Programming languages for Embedded Systems

- Compiled
- Low-level access to locations in memory
- Precise control over all program resources
- Guarantees about correctness

Question:

Is there a conflict in these requirements?

What is a compiler?

Question:

Is gcc a compiler?

What is a compiler?

Question:

Is python a compiler?

What is a compiler?

Question:

Is mysql a compiler?

What is a compiler?

Question:

Is firefox a compiler?

What is a compiler?

Question:

Is Linux a compiler?

What is a compiler?

Question:

Is zip a compiler?

What is a compiler?

Question:

Is your cpu a compiler?

Problems with low level control and safety

```c
int main() {
  (int *)(address_of_peripheral) = 10;
}
```

Problems with low level control and safety

```c
int main() {
  (int *)(address_of_peripheral) = 10;
}
```

this works for any random address too:

```c
int main() {
  (int *)(0x12345678) = 10;
}
```

Problems with low level control and safety

```c
#include <...>

char *alloc_str(char *src) {
  size_t len = strlen(src);
  char *dst = malloc(len);
  memcpy(dst, src, len);
  return dst;
}

int main() {
  char *something = alloc_str("something");
  printf("%s\n", something);
  free(something);
}
```

https://godbolt.org/z/aP5cj16cT

How far can we go?

```c
int main() {
  char *arr = malloc(10);
  for (int i = 0; i < 1500; i++) {
    arr[i] = 5;
  }
}
```

https://godbolt.org/z/15qqq74oe

Undefined Behavior

```C++
int main () {
  while (1) {}
}

int unused() {
  std::cout << "unused?" << std::endl;
}
```

https://godbolt.org/z/qKMeE9xfb

Undefined Behavior

```C++
int main () {
  while (1) {}
}

int unused() {
  std::cout << "unused?" << std::endl;
}
```

https://godbolt.org/z/qKMeE9xfb

- In some compilers it's common to **not define** certain behavior.
- 2's complement in C
- The compiler is allowed to assume those cases never happen
- The programmer should simply make sure those cases never happen!

The Good Programmer Myth

- A good programmer knows to avoid undefined behavior
- If someone causes a memory safety bug, they can't have been a very good programmer
  - Look in the manual! It clearly states that this is undefined behavior!

The Good Programmer Myth

- Large projects with supposedly fine programmers still see many memory safety bugs: https://www.chromium.org/Home/chromium-security/memory-safety/
- Bugs aren't always local
- Code review misses bugs (Khoshnoud, Fatemeh, et al.)

https://steveklabnik.com/writing/memory-safety-is-a-red-herring

We're teaching you Rust

- By default, Rust does not contain any undefined behavior
- If you do want control, you can ask for it:

```Rust
unsafe {
  *(0x1234_5678usize as *const u8) = 10;
}
```

- But don’t, you don’t usually need it!

Fewer bugs in android: https://security.googleblog.com/2022/12/memory-safe-languages-in-android-13.html

Getting started in Rust

- Anatomy of a program: Items

```Rust
// functions
fn example () {}
```

Getting started in Rust

- Anatomy of a program: Items

```Rust
// constants and statics
const A: usize = 3;
static B: i32 = 5;

fn example () {}
```

Getting started in Rust

- Anatomy of a program: Items

```Rust
const A: usize = 3;
static B: i32 = 5;

// types
struct X {}

fn example () {}
```

Getting started in Rust

- Anatomy of a program: Items

```Rust
const A: usize = 3;
static B: i32 = 5;

struct Point {
  // with fields
  x: f32,
  y: f32
}

fn example () {}
```

Getting started in Rust

- Anatomy of a program: Items

```rust
const A: usize = 3;
static B: i32 = 5;

struct Point {
  x: f32,
  y: f32
}

fn example () {}

// a main function
fn main() { }
```

Getting started in Rust

- Anatomy of a program: Items

```Rust
const A: usize = 3;
static B: i32 = 5;

// modules
mod foo {
  fn example () {}
}

fn main() {}
```

Getting started in Rust

- Anatomy of a program: Items

```rust
const A: usize = 3;
static B: i32 = 5;

mod foo {
  fn example () {}
}
// imports
use foo::example;

fn main() {}
```

Getting started in Rust

- Anatomy of a program: Items

```Rust
// BUT NOT EXPRESSIONS
5 + 5;

let a = 3;

fn main() {}
```

Getting started in Rust

- Anatomy of a program: In functions: statements

```rust
fn main() {
  let a = 3;
}
```

Getting started in Rust

- Anatomy of a program: In functions: statements

```Rust
fn main() {
  let a: u64 = 3;
  let b: &str = "hello";
}
```

Getting started in Rust

- Anatomy of a program: In functions: expressions

```rust
fn main() {
  let a: i32 = 2 + 1;
}
```

Getting started in Rust

- Anatomy of a program: In functions: loops

```Rust
fn main() {
  let mut c: usize = 0;
  while c < 10 {
    println!("the counter is {c}");
    c += 1;
  }
}
```

Getting started in Rust

- Anatomy of a program: In functions: loops

```rust
fn main() {
  for c in 0..10 {
    println!("the counter is {c}");
  }
}
```

Getting started in Rust

- Anatomy of a program: In functions: conditionals

```Rust
fn main() {
  for c in 0..10 {
    if c != 3 {
      println!("the counter is {c}");
    }
  }
}
```

Getting started in Rust

- Type Inference

```Rust
// look ma, no types
let a = 3;
// still an error
let b = a + "hello";
```

Getting started in Rust

- Return is Implicit

```rust
fn square(a: i64) -> i64 {
    a * a
}
```

- Though make sure you don't put a `;` at the end

```rust
fn square(a: i64) -> i64 {
    a * a;
}
```

Getting started in Rust

- (mostly) automatic memory management

```Rust
// a string always contains a length
fn alloc_str(inp: &str) -> String {
    String::from(inp)
}

fn main() {
    let x = alloc_str("something");
    println!("{x}");

    // no free needed!
}
```

Getting started in Rust

- Mutability is explicit

```rust
fn main() {
  let a = 3;
  // error
  a = 5;

  let mut a = 3;
  // ok
  a = 5;
}
```

Getting started in Rust

- Almost everything is an expression

```Rust
fn main() {
  let x = if something() {
    4
  } else {
    3
  };

  let y = loop {
    break 3;
  };

  let double = |x| x * 2;
}
```

Getting started in Rust

- Almost everything is an expression
- Which means you can do some comical things, yes this is completely ok:

```rust
fn foo() -> bool {
  if if if true { false } else { true } { false } else { true } { false } else { true }
}
```

Assignment:

- Form pairs
- Go to https://projecteuler.net/archives
- Try one of 1, 5, or 14, or a slightly harder one: 18

Then:

- Go to https://play.rust-lang.org and program it :)
- See how far you get, I'll walk around.
- If you get stuck somewhere? Also look at: https://doc.rust-lang.org/book/

We'll discuss after