#1 Intro. to Rust

@aaaadev

Apr 9, 2025

Rust 프로그래밍 언어란?

> A language empowering everyone to build reliable and efficient software.
>
> — rust-lang.org

> From startups to large corporations, from embedded devices to scalable web services, Rust is a great fit.
>
> — rust-lang.org

Rust의 철학들 (1)

- **C**는 단순히 프로그래머에게 '행운을 빕니다'하고 모든 것을 맡기는 것이였다면,
- **Rust**는 올바른 동작을 하게 만드는 것을 '쉽게'하고 올바르지 않은 동작을 하게 만드는 것을 '어렵게'하는 것이 목표.
- Ex. Ownership, Lifetime, Send / Sync traits
- 이는 Memory-safety를 보장하여 하여금 프로그래머의 런타임 디버깅 부담을 줄여줌.

Rust의 철학들 (2)

- **Rust**는 뿐만 아니라 **생산성**도 잡는 것을 목표로 함.
- Ex) `rustfmt`로 간략화된 포맷팅, `cargo`를 통한 매우 쉬운 패키지 관리, 내장 Documentation 시스템을 통한 쉬운 문서화, `clippy`를 통한 코드 스타일 관리

Rust는 만능인가?

- 명심해야할 것은 그렇다고 해서 Rust가 만능은 아님
- 언어 설계란 **trade-off** 관계
- Rust가 메모리 안전성과 성능을 잘 잡았다고 해도, GC 언어처럼 빠른 프로토타이핑은 어렵고, Python처럼 REPL 한 줄로 과학 계산을 돌리긴 힘들다.

Rust 설치 방법

Linux / macOS

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

Windows

https://win.rustup.rs/ 에 방문하여 다운로드 및 설치

OpenBSD

```
pkg_add rust
```

Rust를 구원한 패키지 매니저, Cargo

- `cargo`

  `new` · `build` · `run` · `test` · `bench` · `doc` · `publish` · `install`

  로 소프트웨어 개발에 있어 모든 cycle을 `cargo` 하나로 가능!
- `cargo`가 관리하는 단위인 `crate`는 Rust 컴파일러가 한 번에 분석/코드 생성하는 컴파일 단위. `rustc hello.rs`처럼 파일 하나만 줘도 그 파일이 곧 하나의 crate가 된다!
- Ex. `cargo`로 `hello-world` 프로젝트를 생성한다고 한다면,

```
cargo new hello-world
cd hello-world && cargo run hello-world
```

Cargo.toml 설정

```toml
[package]             # 패키지 자체 정보
name    = "myapp"
version = "0.1.0"
edition = "2021"

[dependencies]        # 실행 코드가 쓰는 external crate (crate.io)
serde = { version = "1", features = ["derive"] }
```

이런식으로 패키지 매니저를 쉽고 간편하게 설정할 수 있음!

- 참고: https://crates.io, `cargo add [PACKAGE NAME]`

Rust의 타입 시스템

- Rust는 `type-safety`[1], `null-safety`[2]를 가진다.
- 즉, 모든 변수의 타입은 컴파일 타임에 모두 올바르게 추론되며, 고정된 타입을 가진다.
- 특이한 부분으로는, Rust에서 타입은 '데이터 모양' 뿐만 아니라 '메모리 지배권'까지 서술함. (`&T`, `&mut T`, `Box<T>`)
- `Result<T, E>`, `Option<T>`로 예외와 `null`을 서술한다.

---

[1]잘못된 타입 추론으로 인한 에러로부터 자유롭다.
[2]null 값으로부터 자유롭다.

Rust의 원시 타입 - 스칼라 타입

- Rust의 원시 타입은 `usize`, `isize`를 제외하고 크기가 고정되어 프로그래머의 인텐드에 맞춘 코딩이 가능함. (C는 IdB)

타입 명	데이터 크기	설명
`i32`	4 바이트	부호 있는 32비트 정수
`i64`	8 바이트	부호 있는 64비트 정수
`u32`	4 바이트	부호 없는 32비트 정수
`u64`	8 바이트	부호 없는 64비트 정수

Rust의 원시 타입 - 스칼라 타입

타입 명	데이터 크기	설명
`isize`	포인터 크기 (플랫폼마다 다름)	부호 있는 포인터 크기의 정수
`usize`	포인터 크기 (플랫폼마다 다름)	부호 없는 포인터 크기의 정수
`char`	1 바이트	하나의 문자
`bool`	1 비트	부울 변수 (`true` or `false`)
`f32`	4 바이트	단정밀도 부동소수
`f64`	8 바이트	배정밀도 부동소수

Rust의 원시 타입 - 스칼라 타입

타입 명	데이터 크기	설명
( )	없음	값 없음

Rust의 변수와 함수

- Rust의 변수는 기본적으로 불변(immutable)

```
let x = 5; // 불변 변수
let mut y = 5; // 가변 변수 (mut 키워드 사용)
```

Rust의 변수와 함수

- 함수는 `fn` 키워드로 정의

```
fn add(a: i32, b: i32) -> i32 {
    a + b  // 세미콜론이 없으면 반환 표현식
}

fn main() {
    let sum = add(5, 3);
    println!("합계: {}", sum);
}
```

Rust의 제어 흐름

- if 표현식 (조건에 괄호가 필요 없음)

```
if number % 4 == 0 {
    println!("4로 나누어 떨어집니다");
} else if number % 3 == 0 {
    println!("3으로 나누어 떨어집니다");
}
```

- `if`는 표현식이므로 변수에 할당 가능

```
let number = if condition { 5 } else { 6 };
```

Rust의 반복문

- `loop`: 명시적으로 중단될 때까지 무한 반복

```
loop {
    counter += 1;
    if counter == 10 { break; }
}
```

Rust의 반복문

- `while`: 조건이 참인 동안 반복

```
while number != 0 {
    println!("{}!", number);
    number -= 1;
}
```

Rust의 반복문

- `for`: 컬렉션 순회

```rust
for number in 1..4 {
    println!("{}!", number);
}
```

Rust의 소유권 개념

- Rust의 핵심 기능: 메모리 안전성을 컴파일 타임에 보장
- 소유권 규칙:
  1. 각 값은 하나의 소유자만 가짐
  2. 소유자가 범위를 벗어나면, 값은 자동으로 삭제됨

```
{
    let s = String::from("hello"); // s는 여기서부터 유효
} // s는 더 이상 유효하지 않음
```

참조와 대여

- 소유권을 이전하지 않고 값을 참조할 수 있음
- &는 참조를 생성

```
fn main() {
    let s1 = String::from("hello");
    let len = calculate_length(&s1); // 참조 전달
}

fn calculate_length(s: &String) -> usize {
    s.len()
}
```

- 가변 참조는 &mut로 생성

```
let mut s = String::from("hello");
change(&mut s);

fn change(some_string: &mut String) {
    some_string.push_str(", world");
}
```

모듈 개념

ping.rs

```
pub fn ping() {
  println!("Pong!");
}
```

모듈 개념

main.rs

```
mod ping;

use ping::ping;

fn main() {
  ping();
}
```

실습

다음 타입들의 크기를 출력하시오.

- `i8`, `i128`, `u16`, `usize`, `bool`, `()`
- 참고: C의 `size_of`와 유사한 기능을 하는 `std::mem::size_of::<T>()`가 있으며, `T`에 원하는 타입을 넣으면 됨.

과제

임의의 정수 p가 소수인지 아닌지 판정하는 프로그램을 작성하시오.

- $p \leq 10^4$
- 반복문 또는 에라토스테네스의 체를 이용해도 좋다.