초보자를 위한 Rust 설치 및 활용

정보컴퓨터공학과 King

1. Rust의 탄생 배경

1.1 개요

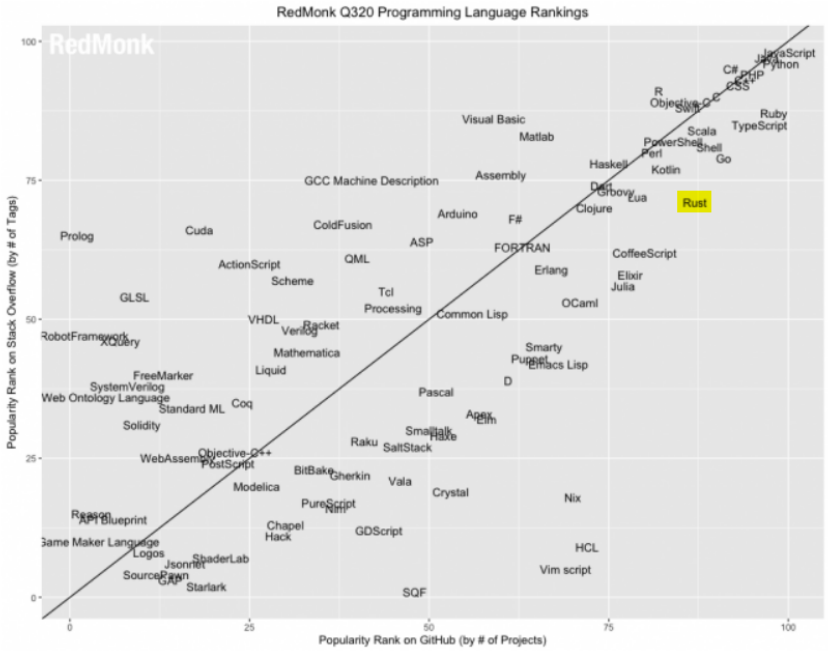
Rust는 안전하고, 병렬적이며, 실용적인 언어로 디자인되었다. 2006년에 개발자 그레이든 호아레의 개인 프로젝트로 시작되었으며, 현재에는 Mozilla Foundation(모질라 재단)에서 연구 목적으로 개발되고 있는 프로그래밍 언어이다.

C, C++, Go와 같은 컴파일 기반의 언어이자, 시스템 프로그래밍 언어에 속하며, Go보다는 늦게 나왔지만, C/C++를 대체 하려하는 점 때문에 라이벌 관계로 엮이기도 한다. GPU의 발달과 나아가 TPU의 유망성으로 인하여 멀티코어 프로세싱이 중요시되는 추세에 따라 동시, 병렬 프로그래밍에도 강점을 가진 Rust가 떠오르고 있다.

1.2 역사 및 발전 과정

2006년 개발자 그레이든 호아레의 프로젝트로 시작하여 2009년 호아레의 고용주인 모질라가 개발에 참여하였다. 2010년 7월 7일 처음으로 공개되었으며, 2015년 5월 15일에 안정 버전이 정식 발표되었다.

모질라의 정책에 따라 Rust는 전적으로 오픈 소스로 개발이 되고 있으며, 커뮤니티로부터 피드백을 받고 있다.



[그림 1] 레드 몽크 프로그래밍 언어 순위

위의 표를 통해 개발자들이 Rust를 얼마나 많이 사용하는 지와 모질라의 정책에 따라 다른 언어들과 대비해 오픈 소스로 활발하게 개발되고 있음을 알 수 있다.

1.3 특징

현대적인 시스템 프로그래밍 언어로 비슷한 때에 나온 Go와 자주 비교된다. Rust는 C/C++와 동등한 수준의 속도를 달성하면서 수명(Lifetime), 소유권(Ownership)이라는 개념을 통하여 안전성을 보장하며, 속도, 동시성을 목표로 한다. Rust는 스택 영역에서 할당되는 객체의 변수의 생성과 소멸 시기를 컴파일 타임에 모두 결정한다.

스마트 포인터와 혼동할 수 있는데 C++의 스마트 포인터와 엄연히 다른 개념이라고 볼 수 있다. 컴파일 타임에 스택 영역에 할당되는 Rust와 달리 스마트 포인터의 경우 런타임에 힙 영역에 할당하는 객체가 런타임에 결정되므로 이런 점에서 다르다는 것을 확인할 수 있다. 스마트 포인터는 순환 참조 시 메모리 누수가 여전히 발생할 수 있다.

2. 기본 설치하기

2.1 Rustup 사용하여 설치

윈도우의 경우 웹에서 https://win.rustup.rs/ 주소를 치면 rustup-init.exe 파일을 받을 수 있고, 이를 실행하면 시스템에 Rust가 설치된다.

리눅스의 경우 셸에서

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

커맨드 한 줄만 입력하면 Rust 컴파일러와 패키지 매니저가 한번에 설치된다.

2.2 환경 변수 설정

Rust 개발 환경에서 모든 도구는 %USERPROFILE%₩.cargo₩bin 디렉토리에 설치된다. (rustc, cargo, rustup 등)

설치 이후 콘솔 창에서 rustc --version을 입력하였을 때 버전이 제대로 뜨면 설치가 완료된 것인데, 위의 과정을 거쳤는데 실패한다면,

```
%USERPROFILE%₩.cargo₩bin
```

디렉토리를 환경 변수에 추가하면 된다.
위 과정을 거쳤는데도 실패한다면, 콘솔을 다시 시작하거나, 컴퓨터를 다시 시작하는 방법으로 해결할 수 있다.

2.3 간단한 출력 확인

Rust 프로젝트에서 출력하려면 Cargo라는 툴을 사용하는 방식을 이용할 수 있다. Cargo를 사용하면 코드를 포함해야 하는 여러 요소들과 빌드하는데 필요한 것들을 관리해준다. Rust를 설치했다면 Cargo도 자동으로 설치된다.

Linux 환경에서 실행하였다. Window 환경에서는 Ubuntu로 Linux를 다운받고 사용할 수 있으며, PowerShell을 다운받아서도 이와 같이 실행할 수 있다.

1) cargo 버전 확인

```
~$ cargo --version
```
: 버전 확인

버전 : `cargo 1.41.0 (626f0f40e 2019-12-03)`

2) cargo로 프로젝트 생성

```
cd hello_cargo
cargo new hello_cargo
```
: hello_cargo 프로젝트 폴더 생성

: hello_cargo 로 이동

```
~/hello_cargo$ ls
```
: hello_cargo 내의 요소 확인

```
Cargo.lock  Cargo.toml  src  target
```

3) Cargo의 구성 형식인 TOML(Tom's Obvious, Minimal Language) 형식 확인

```
~/hello_cargo$ vi Cargo.toml
```

```
[package]
name = "hello_cargo"
version = "0.1.0"
authors = ["[redacted]"]
edition = "2018"

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

[dependencies]
```

4) main.rs 에 코드 작성

```
~/hello_cargo$ vi src/main.rs
```
: main.rs 소스 열기

```
fn main() {
    println!("Hello, King!");
}
```
: 코드 작성

5) 빌드

```
cargo build
```

```
   Compiling hello_cargo v0.1.0 (/home/kang/hello_cargo)
    Finished dev [unoptimized + debuginfo] target(s) in 1.24s
```

: 빌드 확인

6) 동작 확인

```
~/hello_cargo$ ./target/debug/hello_cargo
```

```
Hello, King!
```

코드 작성 결과가 제대로 출력된 것을 확인할 수 있다.

+) 추가 사항

cargo build를 이용하는 방법 외에 cargo run을 이용하여 코드를 컴파일 한 다음 모두 하나의 명령으로 실행하는 방법이 있다.

cargo check를 통해서 코드가 컴파일 되지만 실행 파일을 생성하지 않는지 빠르게 확인한다. 종종 실행파일 생성 단계를 건너 뛰기 때문에 cargo check보다 cargo build 가 빠르다. 코드를 작성하는 동안 작업을 계속 확인하는 cargo check를 사용하면 프로세스 속도가 빨라진다. 따라서 많은 Rust 사용자들은 프로그램이 컴파일 되는지를 확인하기 위해 프로그램 작성시 cargo check를 주기적으로 실행한다.

3. 전망

마이크로소프트가 메모리 안정성이 좋고, C 및 C++를 대체할 수 있는 시스템 프로그래밍 언어라고 밝혔다. 오버헤드 없이 안전하게 메모리 관리가 가능하다는 점에서 프로그래머들에게 많이 선호된다. Firefox에서는 내부 시스템에 상당 부분 Rust를 적용하였고, Google 또한, 차기 운영체제인 퓨시아에서 Rust를 사용 중이며, Facebook 도 내부 시스템 일부에 Rust를 적용한 상태이다. 이렇듯 많은 운영체제 및 시스템에 많이 활용되는 언어이다.

Position:	**25**
Programming Language:	**Rust**
Ratings:	0.68%

[그림 2] Rust's TIOBE October 2020 Rating

긍정적인 평가에 비해 전체적으로 보았을 때 Rust의 사용률은 적은 편이며 TIOBE 순위에서도 낮은 순위에 위치한 것을 확인할 수 있다. 이러한 이유로는 첫째, C/C++ 와 같은 기존의 언어로 대부분 현재 인프라가 갖춰져 있고, 비교적 최신 언어인 Rust로 갖춰진 곳이 상대적으로 적다. 둘째, 소유권(Ownership), 수명(Lifetime) 등의 개념을 컴파일 타임 시에 프로그래머가 관리해서 프로그래밍을 해줘야 하므로 상대적으로 더 어렵고, 최근에 나온 언어이며 계속 발전중인 언어이므로 전문 인력을 찾기 어려운 점을 들 수 있다.

그러나, 2019년 StackOverflow(Stackoverflow.com) 개발자 설문조사에서 다른 언어들을 제치고 83.5%의 높은 선호도로 개발자들에게 가장 사랑받는 언어 1위를 차지했다는 점이나, 활발하고 전개되고 있는 오픈 소스 개발 등의 이유로 미래는 밝을 것으로 보인다.

4. 결론 및 느낀 점

Rust는 소유권과 수명이라는 개념을 직접 관리해줘야 하기 때문에 다른 언어에 비해 매우 어렵다. Rust는 지금 현재에도 계속 발전하고 있는 언어이다. 그렇기 때문에 다달이 변화하는 것을 확인할 수 있고, Rust에 대해서 배우게 되면 오픈 소스에 대하여 이해하며, 기여하는데 도움이 될 수 있다. 하지만, 아직 체계가 명확하지 않은 언어이다. 이 점은 장점인 동시에 단점이 될 수 있다. 오픈 소스에 기여하면서 Rust 라는 언어가 가지는 장점을 발전시키면서 단점들을 없애며 오픈 소스 공간의 다른 사람들과 협업을 통해서 발전시켜

나간다면 실력을 늘이는데 도움이 되며 한 언어의 개발에 일조를 했다는 보람을 느낄 수 있을 것이다.

출처 및 참고 자료

[그림 1] 레드몽크(https://redmonk.com/)
[그림 2] TIOBE(https://www.tiobe.com/tiobe-index/)