러스트 프로그래밍
공식 가이드

러스트 프로그래밍
공식 가이드

1쇄 발행 2019년 11월 28일

지은이 스티브 클라브닉, 캐롤 니콜스
옮긴이 장현희
펴낸이 장성두
펴낸곳 주식회사 제이펍

출판신고 2009년 11월 10일 제406-2009-000087호
주소 경기도 파주시 회동길 159 3층 3-B호
전화 070-8201-9010 / **팩스** 02-6280-0405
홈페이지 www.jpub.kr / **원고투고** jeipub@gmail.com

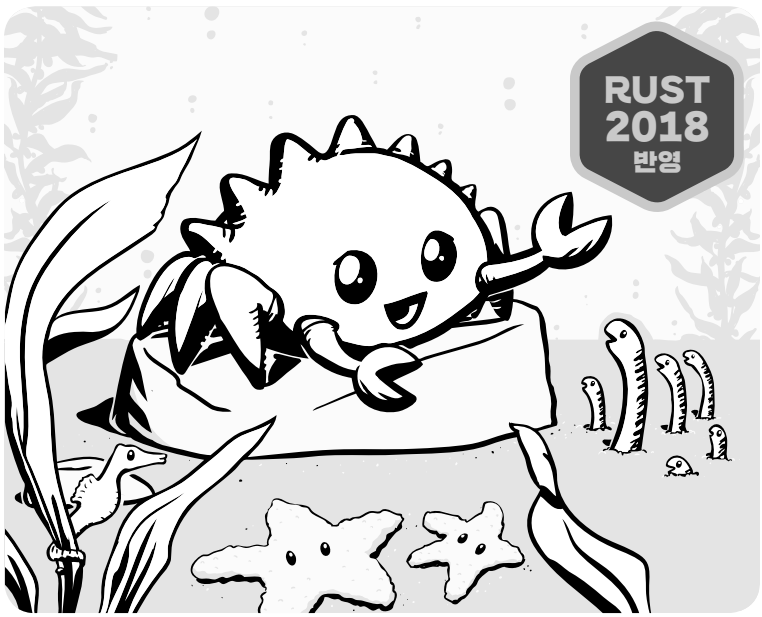
편집팀 이종무, 이민숙, 최병찬, 이 슬, 이주원 / **소통·기획팀** 민지환, 송찬수 / **회계팀** 김유미
진행 이종무 / **교정·교열** 이인호 / **내지디자인** 최병찬
용지 신승지류유통 / **인쇄** 해외정판사 / **제본** 광우제책사

ISBN 979-11-88621-72-9 (93000)
값 34,000원

THE **RUST PROGRAMMING** LANGUAGE

러스트 프로그래밍 공식 가이드

스티브 클라브닉, 캐롤 니콜스 지음 / **장현희** 옮김



no starch press

Jpub 제이펍

※ 드리는 말씀

- 이 책에 기재된 내용을 기반으로 한 운용 결과에 대해 저자, 역자, 소프트웨어 개발자 및 제공자, 제이펍 출판사는 일체의 책임을 지지 않으므로 양해 바랍니다.
- 이 책에 등장하는 각 회사명, 제품명은 일반적으로 각 회사의 등록 상표 또는 상표입니다. 본문 중에는 ™, ©, ® 마크 등이 표시되어 있지 않습니다.
- 이 책에서 사용하고 있는 제품 버전은 독자의 학습 시점이나 환경에 따라 책의 내용과 다를 수 있습니다.
- 책 내용과 관련된 문의사항은 옮긴이나 출판사로 연락해 주시기 바랍니다.
  - 옮긴이: aspnetmvp@gmail.com
  - 출판사: jeipub@gmail.com

차례

옮긴이 머리말 x
추천사 xii
감사의 말 xiv
이 책에 대하여 xv
베타리더 후기 xx

CHAPTER 1 시작하기 1
1.1 설치하기 1
1.2 첫 번째 러스트 프로그램 작성하기 4
1.3 카고 알아보기 9
요약 14

CHAPTER 2 숫자 맞히기 게임의 구현 15
2.1 새 프로젝트 셋업하기 16
2.2 플레이어가 예측한 값 처리하기 17
2.3 난수 생성하기 23
2.4 난수와 사용자의 입력 비교하기 28
2.5 반복문을 이용해 다중 입력 지원하기 32
요약 37

CHAPTER 3 일반 프로그래밍 개념 39
3.1 변수와 가변성 40
3.2 데이터 타입 45
3.3 함수 53
3.4 주석 60
3.5 흐름 제어 61
요약 71

CHAPTER 4 소유권 73
4.1 소유권이란? 73
4.2 참조와 대여 86

4.3 슬라이스 타입 93
요약 100

CHAPTER 5 구조체를 활용한 관련 데이터의 구조화 101
5.1 구조체 정의와 인스턴스 생성 102
5.2 구조체를 사용하는 예제 프로그램 107
5.3 메서드 문법 112
요약 117

CHAPTER 6 열거자와 패턴 매칭 119
6.1 열거자 정의하기 120
6.2 match 흐름 제어 연산자 127
6.3 if let을 이용한 간결한 흐름 제어 133
요약 135

CHAPTER 7 패키지, 크레이트, 모듈로 프로젝트 관리하기 137
7.1 패키지와 크레이트 138
7.2 모듈을 이용한 범위와 접근성 제어 140
7.3 경로를 이용해 모듈 트리의 아이템 참조하기 142
7.4 use 키워드로 경로를 범위로 가져오기 150
7.5 모듈을 다른 파일로 분리하기 157
요약 159

CHAPTER 8 범용 컬렉션 161
8.1 벡터에 일련의 값 저장하기 162
8.2 String 타입에 UTF-8 형식의 텍스트 저장하기 168
8.3 키와 값을 저장하는 해시 맵 178
요약 184

CHAPTER 9 에러 처리 185
9.1 panic! 매크로를 이용한 회복 불가능한 에러 처리 186
9.2 Result 타입으로 에러 처리하기 189
9.3 패닉에 빠질 것인가? 말 것인가? 200
요약 205

CHAPTER 10 제네릭 타입, 트레이트 그리고 수명 207
10.1 함수로부터 중복 제거하기 208
10.2 제네릭 데이터 타입 211
10.3 트레이트: 공유 가능한 행위를 정의하는 방법 220
10.4 수명을 이용해 참조 유효성 검사하기 233
10.5 제네릭 타입 매개변수, 트레이트 경계, 그리고 수명 248
요약 248

CHAPTER 11 자동화 테스트 작성하기 251
11.1 테스트의 작성 252
11.2 테스트 실행 제어하기 268
11.3 테스트의 조직화 274
요약 281

CHAPTER 12 I/O 프로젝트: 명령줄 프로그램 작성하기 283
12.1 명령줄 인수 처리하기 284
12.2 파일 읽기 287
12.3 모듈화와 에러 처리 향상을 위한 리팩토링 289
12.4 테스트 주도 방법으로 라이브러리의 기능 개발하기 303
12.5 환경 변수 다루기 310
12.6 stderr을 이용해 에러 메시지 출력하기 316
요약 319

CHAPTER 13 함수형 언어의 기능: 반복자와 클로저 321
13.1 클로저: 주변 환경을 캡처하는 익명 함수 322
13.2 반복자를 이용해 일련의 아이템 처리하기 337
13.3 입출력 프로젝트의 개선 346
요약 352

CHAPTER 14 카고와 crates.io 353
14.1 릴리즈 프로필을 이용한 빌드 커스터마이징 354
14.2 crates.io 사이트에 크레이트 발행하기 355
14.3 카고 작업공간 367
14.4 cargo install 명령을 이용해 crates.io에서 바이너리 설치하기 374
14.5 사용자 정의 명령을 이용해 카고 확장하기 375
요약 375

CHAPTER 15 스마트한 포인터 377
15.1 Box<T>를 이용해 힙 메모리의 데이터 참조하기 379
15.2 Deref 트레이트를 이용해 스마트 포인터를 참조처럼 취급하기 384
15.3 Drop 트레이트를 구현해서 메모리를 해제할 때 코드 실행하기 391
15.4 Rc<T>, 참조 카운터 스마트 포인터 395
15.5 RefCell<T> 타입과 내부 가변성 패턴 400
15.6 메모리 누수의 원인이 되는 순환 참조 410
요약 419

CHAPTER 16 자신 있는 동시성 421
16.1 코드를 동시에 실행하기 위한 스레드 422
16.2 공유 상태 동시성 438
16.3 Sync와 Send 트레이트로 동시성 확장하기 446
요약 448

CHAPTER 17 러스트의 객체지향 프로그래밍 기능 449
17.1 객체지향 언어의 특징 450
17.2 다른 타입의 값을 허용하는 트레이트 객체 454
17.3 객체지향 디자인 패턴 구현 462
요약 476

CHAPTER 18 패턴과 매칭 477
18.1 패턴을 활용할 수 있는 위치 478
18.2 부인 가능성: 패턴이 일치할 수도 있고 그렇지 않을 수도 있는 경우 484
18.3 패턴 문법 486
요약 503

CHAPTER 19 러스트의 고급 기능 505
19.1 안전하지 않은 러스트 506
19.2 고급 트레이트 517
19.3 고급 타입 시스템 529
19.4 고급 함수와 클로저 537
19.5 매크로 541
요약 553

CHAPTER 20 최종 프로젝트: 다중 스레드 웹서버 구축 555
20.1 단일 스레드 웹서버 구현하기 556
20.2 다중 스레드 서버로 전환하기 568
20.3 우아한 종료와 해제 591
요약 599

APPENDIX A 키워드 601
현재 사용 중인 키워드 601
향후에 사용하기 위해 예약한 키워드 603
원시 식별자 603

APPENDIX B 연산자와 심볼 605
연산자 605
비연산자 심볼 607

APPENDIX C 상속 가능한 트레이트 612
프로그래머용 출력을 위한 Debug 613
일치 비교를 위한 PartialEq와 Eq 613
순서를 비교하는 PartialOrd와 Ord 614
값을 복제하기 위한 Clone과 Copy 615
어떤 값을 고정된 크기의 값에 매핑하는 Hash 616
기본값을 제공하는 Default 616

APPENDIX D 유용한 개발 도구 617
rustfmt를 이용한 자동 포매팅 617
rustfix 도구로 코드 수정하기 618
Clippy 린트 619
러스트 언어 서버를 이용한 IDE 통합 621

APPENDIX E 에디션 622

찾아보기 625

옮긴이

머리말

러스트는 모질라가 파이어폭스 브라우저의 렌더링 엔진을 새로 작성하기 위해 함께 개발되었으며, 기존의 C/C++ 언어가 지니는 메모리 관리의 어려움을 언어 차원에서 해소하면서 최신의 멀티 코어 프로세스를 활용한 동시성에 대한 지원, 비용 없는 추상화 등 강력한 기능을 탑재한 새로운 시스템 프로그래밍 언어다.

최근 마이크로소프트의 발표에 따르면 지금까지 윈도우 운영체제에서 발견된 버그의 70%가량이 메모리 관련 버그이며, 이 문제를 해결하려는 목적으로 러스트의 사용을 심각하게 고려하고 있다고 한다. 또한, 전 세계 사용자들이 사용하는 파이어폭스 브라우저를 통해 이미 러스트의 안정성과 성능은 검증을 마쳤다.

이 책은 러스트를 처음 시작하는 개발자를 위해 쓰인 책이다. 여느 프로그래밍 언어 입문서와 마찬가지로 러스트의 기본적인 타입과 문법에 대한 소개로 시작해서 러스트만의 고유한 개념인 소유권과 수명, 더 나아가 러스트의 고급 기능에 이르기까지 친절하고 자세하게 설명한다.

러스트는 시스템 프로그래밍 언어를 표방하고 있다. 따라서 일반적인 애플리케이션이나 웹 서버 같은 다중 스레드 애플리케이션은 물론 심지어 운영체제도 개발할 수 있다. 게다가 가장 먼저 웹어셈블리(WebAssembly)의 개발을 지원한 언어이기도 한 만큼 그 활용도는 무궁무진하다.

자바나 파이썬, C# 같은 고수준 언어를 주로 다루는 개발자에게 러스트는 다소 접근이 어려운 언어일 수도 있다. 고수준 언어에서는 대부분의 개발자가 굳이 신경 쓰지 않아도 되는 많은 부분이 러스트 코드를 작성할 때는 상당히 중요한 부분이기 때문이다. 게다가 깐깐한 러스트 컴파일러가 아무리 친절하게 에러 메시지를 출력해 준다고 해도 그 에러의 원인이 정확히 무엇인지 파악하기도 처음에는 쉽지 않을 수 있다.

그럼에도 러스트는 시간과 노력을 들여 학습해 볼 가치가 충분한 언어다. 깃허브를 비롯한 각종 프로그래밍 언어 인기 순위에서도 실제 사용 빈도는 높지 않지만 개발자 사이에서 가장 사랑받는 언어로 손꼽히는 것만 보더라도 러스트가 현재 소프트웨어 개발 분야의 여러 문제를 효과적으로 해소하는 좋은 언어라는 점을 알 수 있다.

모쪼록 이 책이 한국의 많은 개발자가 러스트 언어를 학습하는 데 도움이 되기를 바라며, 번역을 통해 러스트를 학습할 좋은 기회를 제공해 주신 제이펍의 장성두 대표님, 탈고 후 러스트 2018 에디션의 출시로 인해 편집과 교정에 더 많은 시간과 노력을 할애해 주신 이종무 팀장님, 이인호 실장님, 최병찬 과장님을 비롯한 제이펍 식구들에게 감사의 인사를 전한다. 마지막으로 이 책을 번역하는 내내 아낌없는 사랑과 지원을 보내준 사랑하는 아내 지영과 딸 예린, 아들 은혁에게 고마움을 전한다.

깊어가는 2019년의 어느 가을 날

__ 장현희

추천사

항상 명확하게 드러나지는 않지만, 러스트 프로그래밍 언어의 기본은 증진(empowerment)에 있다. 지금 작성하고 있는 코드가 무엇이든지 간에 러스트는 더 많은 것을 이룰 수 있도록 개발자를 증진하며, 이전에 이미 경험했던 것보다 훨씬 다양한 도메인 관련 프로그램을 작성하는 데 더 큰 자신감을 불어넣어 준다.

예를 들어, 저수준의 상세한 메모리 관리, 데이터 표현, 그리고 동시성 등을 고려해야 하는 '시스템 수준'의 작업을 생각해 보자. 전통적으로 이런 프로그래밍의 범위는 매우 어려우며, 그 악명 높은 어려움을 피해 가는 방법을 수년간 공부한 몇몇 소수만이 해낼 수 있는 것이었다. 또한, 아무리 주의 깊게 개발하는 것이 몸에 밴 사람이라도 그들이 작성한 코드는 여전히 잘못 실행되거나, 충돌이 발생하거나, 혹은 오염될 가능성을 내포하고 있다.

러스트는 이러한 어려움을 제거하고 같은 작업을 수행할 수 있는 더 친근하면서도 적절한 도구들을 제공함으로써 그 장벽을 허물고 있다. 저수준의 제어를 해야 하는 프로그래머들은 통상적인 충돌이나 보안 허점의 위협 없이, 그리고 불안정한 도구들을 새롭게 배우지 않고도 러스트를 이용해 목적을 달성할 수 있다. 더 좋은 점은 본질적으로 언어 자체가 속도와 메모리 사용량 관점에서 안정적인 코드를 작성할 수 있도록 디자인되어 있다는 점이다.

이미 저수준의 코드를 다루는 프로그래머라면 야심 차게 러스트를 시작할 수 있다. 예를 들어, 러스트의 병렬성(parallelism)은 그 위험도가 훨씬 낮은 작업이다. 컴파일러가 대부분의 실수를 미리 감지해 주기 때문이다. 또한, 실수로 충돌이나 자원의 부적절한 사용을 유발하지 않는다는 자신감을 바탕으로 더욱 공격적인 최적화를 수행할 수도 있다.

하지만 러스트는 단지 저수준 시스템 프로그래밍을 위한 언어가 아니다. 러스트는 표현이 풍

부하며, CLI 앱, 웹 서버를 비롯해 다양한 종류의 코드를 즐겁게 작성할 수 있는 인간공학적인 언어다. 이 두 가지 특성에 대한 예시는 머지않아 이 책에서 찾아볼 수 있을 것이다. 러스트를 이용하면 한 도메인에서 다른 도메인으로의 전환을 가능케 하는 기술을 습득할 수 있다. 즉, 웹 앱을 작성하면서 배운 러스트 기술을 라즈베리 파이(Raspberry Pi) 개발에도 적용할 수 있다.

이 책은 사용자에게 막강한 힘을 불어넣는 러스트의 잠재력을 모두 설명하고 있다. 친근하면서도 읽기 쉬운 문체로 러스트에 대한 지식뿐만 아니라 프로그래머로서의 목표 달성과 자신감을 획득하는 데 도움을 줄 것이다. 이제 러스트의 세계로 뛰어들어 그 경이로움을 학습해 보자. 그리고 러스트 커뮤니티의 일원이 된 것을 환영한다.

__ 니콜라스 마츠사키스, 애런 튜론

감사의 말

우리 저자진은 책으로 집필할 가치가 있는 놀라운 언어인 러스트 개발에 참여한 모든 이에게 고마움을 전한다. 또한, 저자진을 환영해 주고 더 많은 사람이 참여할 가치가 있는 환경을 마련하는 데 물심양면으로 노력해 준 러스트 커뮤니티의 모든 구성원에게도 감사의 인사를 드린다.

온라인을 통해 이 책의 이전 버전을 읽고 피드백과 버그 보고, 그리고 풀 리퀘스트(Pull Request)를 보내준 모든 이에게 특별히 더 고맙다고 전하고 싶다. 특히, 기술 감수를 맡아준 에드워드-미하이 버테스쿠(Eduard-Mihai Burtescu)와 알렉스 크리쉬튼(Alex Crishton), 그리고 표지 그림을 그려준 카렌 러스타드 토르바(Karen Rustad Tölva)에게 감사한다. 빌 폴락(Bill Pollock), 리즈 채드윅(Liz Chadwick) 그리고 야넬 르도와이즈(Janelle Ludowise)를 비롯해 이 책의 집필과 출간에 도움을 준 노스타치(No Starch)의 모든 임직원 여러분에게 감사한다.

저자 스티브는 공저자인 캐롤에게 고마움을 전한다. 그녀의 도움이 없었다면 이 책은 이만큼의 품질을 확보할 수도 없었을뿐더러 더 많은 시간을 할애해야 했을 것이다. 더불어 이 책을 진행하는 내내 이루 말할 수 없는 도움을 준 애슐리 윌리엄스(Ashley Willams)에게도 감사한다.

저자 캐럴은 러스트에 흥미를 느낄 수 있는 동기를 제공하고 이 책을 함께 집필할 기회를 제공해 준 스티브에게 고마움을 표한다. 또한, 헌신적인 사랑과 지원을 아끼지 않은 가족들, 특히 남편 제이크 굴딩(Jake Goulding)과 딸 비비안(Vivian)에게 사랑을 담아 감사함을 전한다.

__ 스티브 클라브닉, 캐롤 니콜스

이 책에 대하여

러스트 언어를 소개하는 이 책을 구입해 준 독자 여러분을 환영한다. 이 책은 더 빠르고 더 신뢰할 수 있는 소프트웨어를 작성하는 데 도움을 주고자 집필한 책이다. 사람에게 더 친숙한 고수준의 프로그래밍과 저수준의 제어는 프로그래밍 언어 디자인에서 서로 상충하는 경향이 있다. 러스트는 이러한 상충관계에 도전장을 내민 언어다. 러스트는 강력한 기술적 기능과 훌륭한 개발자 경험 사이의 균형을 적절히 갖추고 있으므로, 예전 같으면 (메모리 사용 같은) 저수준 제어 과정에서 어김없이 따라왔던 여러 불편을 모두 해소하고 있다.

러스트의 대상 사용자

러스트는 여러 이유로 많은 사람에게 이상적인 언어다. 가장 중요한 사용자층을 살펴보자.

개발자로 구성된 팀

러스트는 다양한 수준의 시스템 프로그래밍 지식을 갖춘 대규모 개발팀 간의 협업을 위한 생산적 도구로 자리매김하고 있다. 대부분의 다른 언어 환경에서 저수준의 코드는 경험이 풍부한 개발자에 의한 광범위한 테스트와 코드 리뷰를 통해서만 잡아낼 수 있는 미묘한 버그가 발생하기 쉽다. 하지만 러스트 환경에서는 컴파일러가 동시성 관련 버그를 포함해 사람이 찾아내기 힘든 버그를 유발하는 코드의 컴파일 자체를 거부함으로써 일종의 문지기 역할을 수행한다. 컴파일러의 든든한 지원 덕분에 팀은 버그를 추적하는 데 소비하는 시간을 오로지 프로그램 로직 작성에 활용할 수 있다.

또한, 러스트는 시스템 프로그래밍 세계에서 활용할 수 있는 최신의 개발자 도구를 제공한다.

- **카고(Cargo):** 내장된 의존성 관리 및 빌드 도구인 카고는 러스트 생태계를 통틀어 편리하면서도 일관된 컴파일 및 의존성 관리 기능을 제공한다.

- **Rustfmt:** 이 도구는 모든 개발자가 일관된 코딩 스타일을 유지할 수 있도록 돕는다.
- **러스트 언어 서버(The Rust Language Server):** 통합개발환경(Integrated Development Environment, IDE)을 위한 코드 완성 및 인라인 에러 메시지 등의 기능을 제공한다.

이 도구를 비롯해 러스트 생태계를 구성하는 다양한 도구를 활용하면 개발자는 시스템 수준의 코드를 작성하면서도 충분한 생산성을 발휘할 수 있다.

학생

러스트는 시스템의 개념을 학습하는 데 관심이 있는 학생에게도 좋은 도구다. 많은 사람이 러스트를 통해 운영체제 개발이라는 주제를 학습하고 있다. 커뮤니티 또한 학생을 환영하며 그들의 질문에 기꺼이 답해 주고 있다. 이 책의 집필 노력을 통해 러스트 팀은 시스템의 개념을 더 많은 사람, 특히 프로그래밍을 처음 시작하는 사람에게 전파하고자 한다.

기업

크고 작은 수백 곳의 기업이 다양한 업무를 실제 환경에서 처리하기 위해 러스트를 사용하고 있다. 러스트는 명령줄(command line) 도구, 웹 서비스, 데브옵스(DevOps) 도구, 임베디드 장치, 오디오 및 비디오의 분석과 변환, 암호학, 생물정보학, 검색 엔진, 사물인터넷 애플리케이션, 머신러닝을 비롯해 파이어폭스 웹 브라우저의 주요 기능 구현에 이르기까지 폭넓게 사용되고 있다.

오픈 소스 개발자

러스트는 러스트 프로그래밍 언어 자체를 비롯해 커뮤니티, 개발자 도구 및 라이브러리 등을 개발하고자 하는 사람도 활용하고 있다. 러스트 언어에 대한 여러분의 기여는 언제든지 환영한다.

속도와 안전성을 중요시하는 모든 사람들

러스트는 언어 측면에서의 속도와 안전성을 최우선으로 꼽는 사람을 위한 언어다. 여기서 속도란, 러스트로 작성된 프로그램이 동작하는 속도는 물론 러스트를 이용해 프로그램을 작성할 수 있는 속도도 의미한다. 러스트 컴파일러는 기능의 추가와 리팩토링 과정에서 안전성을 확보하기 위한 검사를 수행한다. 이러한 검사를 수행하지 않는 기존의 언어로 작성된 레거시 코드는 개발자가 수정하는 것조차 꺼리는 것과는 매우 상반된다. 무비용 추상화(zero-cost abstractions), 손으로 직접 작성한 코드만큼 빠르게 실행되는 저수준 코드로 컴파일되는 고수준의 기능 등 러스트는 안전한 코드를 빠르게 동작하도록 하는 데 주안점을 두고 있다.

러스트 언어는 그 외에 다른 사용자를 지원하기 위한 노력도 아끼지 않는다. 지금까지는 그저 가장 활발하게 러스트를 사용하는 사용자층을 언급했을 뿐이다. 전반적으로 러스트의 가장 큰 목표는 안전성과 생산성, 속도와 적합성을 제공함으로써 지난 수십 년간 개발자가 어쩔 수 없이 받아들였던 트레이드오프를 제거하는 것에 있다. 바라건대, 이 책을 통해 러스트를 경험해 보고 러스트가 선택한 방식이 여러분에게 적합한 방법인지 직접 확인해 보기 바란다.

누구를 위한 책인가?

이 책은 독자 여러분이 이미 다른 프로그래밍 언어를 이용해 코딩을 해본 경험이 있다고 가정하지만, 특정 언어에 대한 경험을 가정하지는 않는다. 저자진은 이 책을 다양한 프로그래밍 경험을 갖춘 많은 독자를 대상으로 집필했다. 따라서 프로그래밍이란 무엇인지 혹은 어떻게 사고해야 하는지 등에 대해 많은 시간을 할애하지는 않는다. 만일 프로그래밍이 처음이라면 프로그래밍에 대한 개요를 설명하는 데 초점을 맞춘 책을 먼저 읽어볼 것을 권한다.

이 책을 읽는 방법

전반적으로 이 책은 독자들이 처음부터 끝까지 순서대로 읽는 것을 가정하고 있다. 나중에 다루는 내용은 앞서 다룬 개념을 토대로 구성되며, 비교적 앞쪽에 있는 내용은 특정 주제를 깊이 있게 다루지는 않는다. 통상적으로는 앞에서 소개한 주제를 뒤에서 더 깊이 살펴보는 경우가 많다.

이 책을 읽다 보면 각 장의 특성을 크게 개념을 학습하기 위한 것과 프로젝트를 수행하는 것의 두 가지로 구분할 수 있을 것이다. 개념 학습을 위한 장에서는 러스트의 개발 방식에 대해 학습한다. 반면, 프로젝트 수행 장에서는 앞서 학습한 내용을 토대로 작은 규모의 프로그램을 개발해 본다. 제2장, 제12장, 그리고 제20장은 프로젝트를 수행하는 장이며, 나머지는 개념을 학습하기 위한 장이다.

제1장은 러스트를 설치하는 방법을 먼저 소개하고, 개발 언어 서적이라면 빠질 수 없는 'Hello, World!' 프로그램을 작성하면서 러스트의 패키지 관리자이자 빌드 도구인 카고의 사용법을 설명한다. 제2장은 러스트 언어의 실질적인 소개를 담당한다. 여기서는 러스트의 개념을 개략적으로 살펴보며, 그 이후의 장에서 더 자세한 내용을 학습할 것이다. 바로 코딩을 시작해 보고 싶다면 제2장부터 시작해도 좋다. 처음에는 다른 프로그래밍 언어와 유사한 러스트의 기능을 소개하는 제3장으로 곧바로 넘어간 후 제4장을 통해 러스트의 오너십(ownership) 시스템을 공부

하고 싶을 수도 있다. 하지만 새로운 것을 세심하게 배우는 편이라서 모든 세부사항을 다 학습한 후 진도를 나가는 것을 선호하는 독자라면, 제2장을 건너뛰고 제3장을 먼저 학습한 후 다시 제2장으로 돌아와 자신이 학습한 내용을 곱씹으며 프로젝트를 진행해도 좋다.

제5장은 구조체와 메서드를 소개하며, 제6장은 열거자(enums), match 표현식, 그리고 if let 흐름 제어 구문을 설명한다. 구조체와 열거자는 러스트에서 사용자 지정 타입을 선언할 때 활용한다.

제7장에서는 러스트의 모듈 시스템과 코드의 구조를 위한 비공개 규칙 및 공개 애플리케이션 프로그래밍 인터페이스(Application Programming Interface, API)에 대해 알아본다. 그리고 제8장에서는 표준 라이브러리가 제공하는 벡터(vector), 문자열(string), 해시 맵(hash map) 같은 공통 컬렉션(collection) 데이터 구조에 대해 살펴본다. 이후 제9장에서는 러스트의 에러 처리 철학과 기법에 대해 학습한다.

제10장은 여러 타입에 적용할 수 있는 코드를 정의하기 위한 강력한 개념인 제네릭(generics), 트레이트(traits) 및 수명(lifetimes)에 대해 알아본다. 제11장은 러스트의 안전성 보장과 더불어 프로그램의 로직이 올바른지 확인하기 위한 테스팅 기법을 집중적으로 소개한다. 제12장에서는 파일 내에서 원하는 텍스트를 검색하는 grep 명령줄 도구의 일부 기능을 직접 구현해 본다. 물론, 이 과정에서 앞서 학습한 개념의 상당 부분을 활용하게 될 것이다.

제13장은 함수형 프로그래밍 언어로부터 도입한 러스트의 기능인 클로저(closures)와 반복자(iterators)를 소개한다. 제14장은 카고를 더 깊이 살펴보며, 직접 작성한 라이브러리를 다른 개발자들과 공유하기 위한 모범 사례(best practices)를 알아본다. 제15장에서는 표준 라이브러리가 제공하는 스마트 포인터(smart pointers)와 이 기능을 활성화하는 트레이트를 소개한다.

제16장에서는 다양한 종류의 동시성 프로그래밍 모델을 살펴보고 다중 스레드 프로그램 작성에 도움을 주는 러스트의 언어적 기능에 대해 알아본다. 제17장에서는 러스트의 구문을 이미 익숙한 객체지향 프로그래밍(Object-Oriented Programming) 원리에 비추어 비교해 본다.

제18장은 러스트 프로그램을 이용해 아이디어를 표현할 수 있는 강력한 방법인 여러 패턴 및 패턴 매칭을 설명한다. 제19장에서는 러스트의 안전하지 않은(unsafe) 요소와 생명주기, 트레이트, 타입, 함수 및 클로저 등 다양한 고급 주제들을 모두 섭렵한다.

제20장에서는 저수준의 다중 스레드 기반 웹 서버 프로젝트를 마무리하게 될 것이다.

이 책의 마지막에 수록된 몇 개의 부록은 참고 자료 형식으로 언어에 대한 유용한 정보를 소개한다. 부록 A는 러스트의 키워드를 다루며, 부록 B는 러스트의 연산자와 심볼을 소개한다. 부록 C는 표준 라이브러리가 제공하는 상속 가능한 트레이트를 소개하며, 부록 D는 몇 가지 유용한 개발 도구를, 부록 E는 러스트 에디션을 설명한다.

이 책을 읽는 순서는 딱히 정해져 있지 않다. 건너뛰고 싶은 부분이 있다면 얼마든지 건너뛰어도 무방하며, 책을 읽다가 혼란스러운 부분이 있으면 언제든지 그 전의 내용으로 되돌아가면 된다. 어떤 방법이든 독자가 편한 방법으로 이 책을 즐기기를 바란다.

러스트를 학습하는 과정에서 가장 중요한 부분은 컴파일러가 표시하는 에러 메시지를 읽는 방법을 터득하는 것이다. 이 에러 메시지는 코드를 계속 작성하기 위한 중요한 가이드가 된다. 그래서 이 책에서는 실제로 컴파일되지 않는 다양한 예제를 수록하고 있으며, 컴파일러가 표시하는 에러 메시지도 함께 설명하고 있다. 책의 예제를 임의로 선택해서 작성해 보면 일부 코드는 컴파일되지 않을 수도 있다는 점을 기억하자. 반드시 예제 주변의 설명을 잘 읽어서 해당 예제가 일부러 에러를 유발하도록 작성된 것인지를 확인하자. 대부분은 예제가 컴파일되지 않으면 해당 에러를 수정한 올바른 버전의 코드를 작성하기 위한 내용이 설명되어 있을 것이다.

기타 리소스 및 이 책에 기여하는 방법

이 책은 오픈 소스다. 이 책을 읽으면서 잘못된 부분을 발견한다면 깃허브(https://github.com/rust-lang/book/)에 이슈를 생성하거나 풀 리퀘스트를 보내주기 바란다. 이 책에 기여하는 방법에 대한 보다 자세한 내용은 https://github.com/rust-lang/book/blob/master/CONTRIBUTING.md 파일의 내용을 참고하기 바란다.

이 책의 예제 코드와 정오표, 그리고 기타 자료는 https://www.nostarch.com/Rust2018에서 찾을 수 있다.

베타리더 후기

김태근(연세대학교 대학원 물리학과)

그동안 러스트를 사용해 오면서 한글로 된 책이 있다면 참 좋겠다고 생각했는데 드디어 친절하게 번역된 책이 나와서 재미있게 읽었습니다. 코드를 모두 분해하여 하나하나 알려줄 뿐 아니라, 오류가 어떤 경우에 왜 발생하는지 상세히 설명되어 있어서 매우 좋았습니다.

박종영(쿠팡)

러스트는 다른 언어에서 찾기 어려운 소유권 개념 때문에 비교적 진입 장벽이 낮지 않은 편입니다. 그에 비해 해당 부분에 대해 이해하기 쉽게 잘 쓰인 책이라 생각합니다. 러스트 한국어판 도서가 아직 없는 상황에서 충실한 설명과 분량을 갖춘 양질의 입문서가 등장한 것 같아서 앞으로가 더욱 기대됩니다.

송헌(규슈대학교 대학원)

이 책은 결코 프로그래밍을 처음 접하는 사람을 위한 책이 아닙니다. 하지만 만약 여러분이 기본적인 C 언어에 대한 지식이 있다면 러스트를 처음 접하는 사람부터 좀 더 높은 단계로 가기 원하는 전문가 모두에게 추천할 만한 책입니다. 기본기부터 시작하여 좀 더 러스트다운 코드를 작성하기까지 친절한 내용으로 설명해 주며, 번역도 굉장히 깔끔하여 이해하기 쉬웠습니다.

심백선(컴투스연구소)

러스트는 입문하기에 결코 쉬운 언어가 아니며, 멀티 패러다임, 메모리 안전성과 같은 여러 특성은 많은 분에게 생소할 수 있습니다. 이 책에 풍부하게 서술된 기본 개념은 그런 분의 이해를 돕기에 충분하리라 생각합니다. 이번 베타리딩으로 러스트 언어 입문서 한국어판에 이바지하게 되어 기쁘게 생각하며, 제이펍과 한국 러스트 사용자 그룹에 감사드립니다.

이봉준(네이버)

러스트 언어를 다룬 최초의 한국어 번역 도서가 나온 것을 반갑게 생각합니다. C 언어는 빠르고 훌륭하지만 개발자가 구현해야 할 부분이 너무 많으며, 최근의 C++ 언어는 많은 기능이 추가되어 편리해졌지만, 언어 자체가 갈수록 복잡해지는 느낌이 있었습니다. 러스트는 그 중간 어디쯤으로, 함수형 언어의 장점을 도입하여 우아하고, 깔끔하고, 효율적인 언어라고 생각합니다. 러스트 커뮤니티에서 'the book'이라 불리는 이 책은 러스트 언어를 시작하는 데 가장 좋은 선택지라 생각합니다.

정욱재(스캐터랩)

러스트를 배운다면 한 번쯤 보는 책인 《The Rust Programming Language》의 번역본입니다. 러스트를 배우기 위해 필요한 모든 개념과 표현을 다루고 있습니다. 패키징이나 C 코드와의 통신까지 다루면서 예제의 양도 많고, 예제 코드도 보기 쉽고, 따라 하기까지 쉬워서 러스트를 시작하기에 완벽한 책입니다.

조경민(KAIST)

우리 연구실의 주력 언어인 러스트 공식 가이드 한국어판을 베타리딩하게 되어 매우 영광스럽게 생각합니다. 확실히 러스트는 처음 접할 때 인내심이 필요한 언어입니다. 그러나 이 책으로 인해 한국의 프로그래머들(특히, 제 친구들)이 쓰고 있던 다른 언어를 잠시 내려놓고 더욱 안전한 프로그래밍을 위해 러스트를 즐겨 사용하길 기원합니다.

제이펍은 책에 대한 애정과 기술에 대한 열정이 뜨거운 베타리더들로 하여금
출간되는 모든 서적에 사전 검증을 시행하고 있습니다.

CHAPTER

1

시작하기

지금부터 러스트 세계로의 여행을 떠나보자. 학습할 내용이 많기는 하지만 모든 여행은 그 출발점이 있게 마련이다. 이번 장에서 학습할 내용은 다음과 같다.

- 리눅스, macOS 및 윈도우 운영체제에서 러스트 설치하기
- Hello, world! 문자열을 출력하는 프로그램 작성하기
- 러스트의 패키지 관리자이자 빌드 시스템인 카고의 기본적인 사용법 익히기

1.1 설치하기

가장 먼저 해야 할 일은 당연히 러스트를 설치하는 것이다. 이 책에서는 러스트의 버전과 관련된 도구들을 관리하는 명령줄 도구인 rustup을 이용해 러스트를 내려받을 것이다. 물론, 이 도구를 설치하려면 인터넷 연결이 필요하다.

NOTE 어떤 이유로든 rustup을 사용하기 원하지 않는다면 https://www.rust-lang.org/tools/install/에서 다른 설치 방법을 살펴보기 바란다.

지금부터 가장 최신의 안정 버전 러스트 컴파일러를 설치해 보자. 러스트의 안정성 보장 덕분에 이 책의 모든 예제는 이보다 최신 버전의 러스트 컴파일러에서도 올바르게 컴파일이 될 것이다.[1] 하지만 러스트는 에러와 경고 메시지를 계속해서 개선하기 때문에 버전에 따라 출력 결과가 조금 다를 수는 있다. 다시 말하자면, 이 방법을 이용해 설치한 러스트가 이 책의 집필에 사용된 버전보다 더 높다 하더라도 예제 코드들은 아무런 문제 없이 동작한다는 뜻이다.

명령줄 표현

이번 장을 비롯해 이 책의 전반에서는 터미널 환경에서의 명령을 일부 사용한다. 터미널에 입력해야 하는 코드는 모두 $ 기호로 시작한다. $ 기호를 직접 입력할 필요는 없으며, 단지 각 명령의 시작을 표시하는 역할을 할 뿐이다. $ 기호로 시작하지 않는 코드는 주로 이전 명령의 출력 결과다. 또한, 파워셸(Powershell) 전용 예제는 $ 기호 대신 > 기호를 사용한다는 것도 알아두자.

1.1.1 리눅스나 macOS에 rustup 설치하기

리눅스나 macOS를 사용하는 독자라면 터미널을 열고 다음의 명령을 입력하자.

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

이 명령은 스크립트 파일을 하나 내려받은 후 rustup 도구와 함께 최신 안정 버전의 러스트를 시스템에 설치한다. 간혹 비밀번호를 입력해야 할 수도 있다. 성공적으로 설치가 완료되면 다음과 같은 결과가 나타난다.

```
Rust is installed now.. Great!
```

원한다면 내려받은 스크립트를 실행하기 전에 작성된 코드를 살펴봐도 무방하다.

설치 스크립트는 사용자가 터미널에 다시 로그인하는 시점에 러스트 컴파일러를 시스템의 PATH 환경 변수에 자동으로 등록한다. 만일 러스트를 설치한 후 터미널을 종료하지 않고 곧바로 사용해 보고 싶다면, 다음 명령을 이용해 러스트를 시스템 PATH 변수에 수동으로 등록하자.

1 옮긴이 참고로 역자는 이 책을 번역하는 시점(2019년 07월)의 최신 안정 버전인 1.36.0을 사용하고 있다.

```
$ source $HOME/.cargo/env
```

또는 ~/.bash_profile 파일에 다음의 코드를 추가해도 된다.

```
export PATH="$HOME/.cargo/bin:$PATH"
```

rustup 도구 및 컴파일러와 더불어 일종의 링커(linker)도 설치해야 한다. 대부분은 링커도 미리 설치되어 있지만, 러스트 프로그램을 컴파일할 때 링커를 실행할 수 없다는 에러가 발생한다면 시스템에 링커가 설치되어 있지 않기 때문이다. 이때는 링커를 직접 설치해 주어야 한다. 통상 C 컴파일러는 올바른 링커를 포함하고 있다. 사용 중인 플랫폼의 관련 문서를 확인해서 C 컴파일러를 설치하자. 또한, 일부 공통 러스트 패키지들은 C 코드에 의존하고 있으므로 C 컴파일러가 필요하다. 따라서 지금 바로 설치할 것을 권한다.

1.1.2 윈도우에 rustup 설치하기

윈도우 사용자라면 https://www.rust-lang.org/tools/install 문서에 설명된 단계를 따라 러스트를 설치하자. 설치 과정에서 비주얼 스튜디오(Visual Studio) 2013 혹은 그 이후 버전을 위한 C++ 빌드 도구들이 필요하다는 메시지를 보게 될 것이다. 이 빌드 도구를 설치하기 위한 가장 쉬운 방법은 https://visualstudio.microsoft.com/ko/downloads/에서 비주얼 스튜디오 2019를 다운로드해서 설치하는 것이다. C++ 빌드 도구는 '기타 도구 및 프레임워크' 절에서 찾을 수 있다. 앞으로 이 책에서는 cmd.exe와 파워셸에서 모두 동작하는 명령을 사용할 것이다. 혹시 둘 사이에 차이점이 있을 때는 반드시 관련된 설명을 제공한다.

1.1.3 업데이트와 제거

rustup을 이용해 러스트를 설치했다면 최신 버전으로의 업데이트는 너무나 간단하다. 셸에서 다음 업데이트 스크립트를 실행하자.

```
$ rustup update
```

rustup 도구와 러스트를 제거하려면 셸에서 다음의 제거 스크립트를 실행하면 된다.

```
$ rustup self uninstall
```

1.1.4 문제 해결

러스트가 제대로 설치되었는지 확인하려면 셸을 열고 다음 명령을 입력한다.

```
$ rustc --version
```

그러면 버전 번호와 커밋 해시, 가장 최신의 안정 버전을 커밋한 날짜 등이 다음과 같이 출력된다.

```
rustc x.y.z (abcabcabc yyyy-mm-dd)
```

이 정보가 제대로 출력된다면 러스트를 성공적으로 설치한 것이다. 만일, 윈도우를 사용하고 있는데 이 정보를 볼 수 없다면 러스트 실행 파일이 %PATH% 시스템 변수에 등록되어 있는지를 확인하자. 시스템 변수에 문제가 없는데 여전히 러스트가 실행되지 않아도 걱정 말자. 이 문제를 해결할 방법이 몇 가지 있다. 가장 쉬운 방법은 러스트 공식 디스코드(Discord) 채널(https://discord.gg/rust-lang)을 이용하는 것이다. 이 채널을 통해 다른 러스타시안(Rustaceans, 러스트 사용자를 일컫는 다소 민망한 별칭이다)들과 채팅을 시도할 수 있다. 또 다른 방법은 러스트 사용자 포럼(https://users.rust-lang.org/)이나 스택 오버플로(http://stackoverflow.com/questions/tagged/rust/)를 방문하는 것이다.

1.1.5 로컬 문서 서버

인스톨러는 로컬에 러스트 문서 전체의 복사본도 포함하고 있으므로 이 문서를 오프라인 상태에서도 읽을 수 있다. rustup doc 명령을 실행하면 웹 브라우저에서 로컬 문서를 읽을 수 있다.

표준 라이브러리가 제공하는 타입이나 함수에 대해 궁금하다면 언제든지 이 명령을 실행해 애플리케이션 프로그래밍 인터페이스(API) 문서를 살펴보기 바란다.

1.2 첫 번째 러스트 프로그램 작성하기

이제 러스트의 설치를 마쳤으므로 최초의 러스트 프로그램을 작성하자. 새 프로그래밍 언어를 배울 때는 Hello, world!라는 문자열을 화면에 출력하는 프로그램을 작성해 보는 것이 전통이지 않은가!

NOTE 이 책은 독자들이 명령줄 환경에 이미 익숙하다고 가정한다. 러스트는 편집 도구나 코드 위치에 대해 특별히 제약을 두지는 않기 때문에 명령줄 환경보다는 통합개발환경(IDE, Integrated Development Environment)을 선호한다면 얼마든지 선호하는 IDE를 사용해도 무방하다. 러스트를 지원하는 IDE는 제법 많이 존재하므로 자세한 내용은 각 IDE의 문서를 참고하기 바란다. 최근 러스트 개발팀은 IDE 지원을 강화하려고 노력하고 있으며, 덕분에 상당히 빠르게 개선되고 있다.

1.2.1 프로젝트 디렉터리 만들기

그러면 러스트 코드를 저장할 디렉터리를 하나 만들자. 러스트는 코드 위치에 대해서는 전혀 개의치 않지만, 이 책에서 제공하는 예제와 프로젝트를 위해 홈 디렉터리에 projects 디렉터리를 만들어 코드를 보관할 것을 권한다.

먼저, 터미널을 열고 projects 디렉터리를 생성한 후 이 디렉터리 아래에 다시 Hello, World! 프로젝트를 위한 디렉터리를 생성하자.

리눅스나 macOS, 윈도우 파워셸 사용자라면 다음 명령을 실행하자.

```
$ mkdir ~/projects
$ cd ~/projects
$ mkdir hello_world
$ cd hello_world
```

윈도우 환경에서 CMD 도구를 이용한다면 다음의 명령을 실행하자.

```
> mkdir "%USERPROFILE%\projects"
> cd "%USERPROFILE%\projects"
> mkdir hello_world
> cd hello_world
```

1.2.2 러스트 프로그램의 작성과 실행

다음으로, main.rs라는 이름으로 새로운 소스 파일을 생성하자. 러스트 파일은 모두 .rs 확장자를 갖는다. 만일, 파일명에 하나 이상의 단어를 사용하려면 밑줄(_)을 이용해 단어를 구분한다. 예를 들어, helloworld.rs보다는 hello_world.rs와 같은 파일명을 사용하자.

방금 생성한 main.rs 파일을 열고 [예제 1-1]의 코드를 입력하자.

예제 1-1 **Hello, world! 문자열을 출력하는 프로그램**

main.rs

```
fn main() {
    println!("Hello, world!");
}
```

코드를 입력했으면 파일을 저장한 후 터미널 창으로 돌아가자. 리눅스나 macOS라면 다음 명령을 이용해 파일을 컴파일하고 실행할 수 있다.

```
$ rustc main.rs
$ ./main
Hello, world!
```

윈도우 환경이라면 ./main 대신 .\main.exe 명령으로 실행하자.

```
> rustc main.rs
> .\main.exe
Hello, world!
```

사용하는 운영체제와 관계없이 'Hello, world!'라는 문자열이 터미널에 출력될 것이다. 만일, 같은 결과를 보지 못한다면 4페이지의 '문제 해결' 절을 참고해 도움을 받자.

Hello, world!가 제대로 출력됐다면 축하한다! 이것으로 첫 번째 공식 러스트 프로그램을 작성한 것이다. 이제 독자 여러분도 엄연한 러스트 프로그래머다. 러스트의 세계에 온 것을 환영한다.

1.2.3 러스트 프로그램 자세히 살펴보기

방금 작성한 Hello, world! 프로그램을 더 자세히 살펴보기로 하자. 먼저, 퍼즐의 첫 번째 조각은 다음과 같다.

```
fn main() {

}
```

이 코드는 러스트에서 함수를 정의하는 코드다. 이 main 함수는 특별한 의미가 있다. 이 함수는 실행 가능한 모든 러스트 프로그램에서 가장 첫 번째로 실행된다. 첫 번째 줄은 매개변수

도 없고 리턴값도 없는 main이라는 이름의 함수를 선언한다. 함수에 매개변수가 필요하면 괄호 안에 매개변수를 나열하면 된다.

또한, 함수의 본문은 중괄호({})로 둘러싸여 있다. 러스트에서는 모든 함수의 본문 코드를 중괄호로 둘러싸야 한다. 여는 중괄호는 함수 선언과 같은 줄에 입력하며, 함수 이름 다음에 공백을 이용해 띄어준다.

이 책을 집필하는 시점에 러스트의 자동 포매팅(formatting) 도구인 rustfmt는 아직 개발 중이다.[2] 모든 러스트 프로젝트에서 동일한 코딩 스타일을 유지하고 싶다면 rustfmt를 이용해 일정한 스타일로 코드를 작성할 수 있다. 궁극적으로 러스트 팀은 이 도구를 rustc 같은 다른 도구와 함께 표준 러스트 배포판에 포함할 계획이다. 따라서 여러분이 이 책을 읽는 시점에는 이미 이 도구가 설치되어 있을 수도 있다. 더 자세한 내용은 온라인 문서를 참고하기 바란다.

main 함수 안에는 다음과 같은 코드가 작성되어 있다.

```
println!("Hello, world!");
```

이 작고 귀여운 프로그램이 수행하는 작업은 화면에 텍스트를 출력하는 이 한 줄의 코드가 전부다. 하지만 이 한 줄에는 네 가지 중요한 요소가 숨어 있다. 먼저, 러스트에서의 들여쓰기는 탭이 아니라 공백 문자 4개를 이용한다.

둘째로, println!은 '러스트 매크로'라고 부르는 것이다. 매크로 대신 함수를 사용했다면 (! 없이) println만 사용했을 것이다. 러스트 매크로에 대한 더 자세한 내용은 제19장을 확인하기 바란다. 지금은 ! 기호를 보면 함수가 아닌 매크로를 호출한다는 것만 알아두자.

세 번째는 'Hello, world!'라는 문자열이다. 이 문자열은 println! 매크로의 인수로 전달되어 화면에 출력된다.

네 번째로, 각 구문은 세미콜론(;)으로 끝난다. 이 기호는 표현식이 완료되었으며, 다음 표현식이 시작한다는 것을 의미한다. 러스트 코드의 대부분 줄은 세미콜론으로 끝난다.

1.2.4 컴파일과 실행의 분리

방금 작성한 프로그램을 실행까지 해봤으므로 실행까지의 과정을 개별적으로 살펴보자.

2 옮긴이 2018년 11월 19일 rustfmt 1.0.0이 배포되었다)

러스트 프로그램을 실행하려면 러스트 컴파일러로 프로그램을 먼저 컴파일해야 한다. 컴파일을 실행하려면 다음과 같이 rustc 명령에 컴파일할 소스 파일의 이름을 전달하면 된다.

```
$ rustc main.rs
```

C나 C++ 언어 개발 경험이 있다면 gcc나 clang 컴파일러를 이용하는 것과 유사하다는 것을 눈치챘을 것이다. 컴파일이 성공적으로 완료되면 러스트는 실행 가능한 바이너리 파일을 생성한다.

이렇게 생성된 바이너리 파일은 리눅스, macOS 또는 윈도우의 파워셸 환경에서 'ls' 명령을 입력해 확인할 수 있다. 리눅스와 macOS 환경에서는 두 개의 파일을 보게 된다. 윈도우의 파워셸 환경이라면 CMD 도구를 사용할 때와 마찬가지로 세 개의 파일을 보게 된다

```
$ ls
main main.rs
```

윈도우 환경에서 CMD 도구를 이용하는 경우라면 다음의 명령을 입력하자.

```
> dir /B      /B 옵션은 파일의 이름만 출력하는 옵션이다.
main.exe
main.pdb
main.rs
```

이 명령을 실행하면 .rs 확장자가 부여된 소스 코드 파일, 실행 파일(윈도우는 main.exe, 다른 플랫폼은 main), 그리고 윈도우용 디버깅 정보를 가지고 있는 .pdb 확장자를 가진 파일까지 확인할 수 있다. 이제 다음의 명령을 이용하면 main 혹은 main.exe 파일을 실행할 수 있다.

```
$ ./main      윈도우라면 .\main.exe
```

조금 전에 구현한 Hello, world! 프로그램이 main.rs 파일에 작성했다고 가정하면, 이 파일을 실행하면 터미널에 'Hello, world!'가 출력되는 것을 확인할 수 있다.

루비나 파이썬 혹은 자바스크립트 같은 동적 언어에 익숙하다면 프로그램의 컴파일과 실행을 독립적인 단계로 나눈 것이 꽤 어색할 것이다. 러스트는 미리(ahead-of-time) 컴파일하는 언어다. 즉, 프로그램을 컴파일해서 생성된 바이너리를 다른 사람에게 전달하면, 그 사람은 러스트를

설치하지 않고도 해당 프로그램을 실행할 수 있다. 누군가에게 .rb, .py 혹은 .js 파일을 전달하면 당사자는 루비, 파이썬 혹은 자바스크립트 언어를 자신의 머신에 설치해야 비로소 프로그램을 실행할 수 있다. 하지만 이런 동적 언어들은 프로그램의 컴파일과 실행을 하나의 명령으로 처리한다. 각 언어의 디자인에 따라 이런 트레이드오프(trade-off)가 존재하기 마련이다.

rustc를 이용해 간단한 프로그램을 컴파일하는 것은 아무런 문제가 없지만, 프로젝트의 규모가 커질수록 소스 코드를 관리하고 공유하기 위한 다양한 옵션이 필요하다. 그래서 지금부터는 실제 환경에서 러스트 프로그램을 작성하는 데 유용한 도구인 카고(Cargo)에 대해 알아보자.

1.3 카고 알아보기

카고(Cargo)는 러스트의 빌드 시스템이자 패키지 관리자다. 대부분의 러스타시안은 이 도구를 이용해 러스트 프로젝트를 관리한다. 카고는 코드의 빌드, 코드가 의존하는 라이브러리의 다운로드, 그리고 이런 라이브러리의 빌드 등 다양한 작업을 대신 처리해 주기 때문이다(코드가 의존하는 라이브러리들을 '의존 라이브러리(dependencies)'라고 부른다).

조금 전에 작성했던 것처럼 간단한 러스트 프로그램은 의존 라이브러리가 필요하지 않다. 그래서 카고를 이용해 Hello, world! 프로그램을 빌드하면 코드를 빌드하는 카고의 기능만을 사용하게 된다. 하지만 더 복잡한 러스트 프로그램을 작성하게 되면 의존 라이브러리가 필요할 것이며, 카고를 이용해 프로젝트를 생성하면 이런 의존성을 추가하기 훨씬 쉬워진다.

대부분의 러스트 프로젝트들이 카고를 이용하고 있으므로 앞으로 이 책에서도 카고를 활용할 것이다. 앞에서 살펴본 방법에 따라 러스트를 설치했다면 카고 역시 함께 설치되어 있다. 만일, 다른 방법으로 러스트를 설치했다면 다음의 명령을 터미널에 입력해 카고가 설치되었는지를 확인하자.

```
$ cargo --version
```

화면에 버전 번호가 출력된다면 카고가 잘 설치된 것이다. 반면, command not found 같은 에러를 보게 된다면 러스트를 설치할 때 참고했던 문서를 통해 카고를 별도로 설치하는 방법을 확인하자.

1.3.1 카고를 이용해 프로젝트 생성하기

이제 카고를 이용해 새로운 프로젝트를 생성하고 처음 생성했던 Hello, world! 프로젝트와의 차이점을 알아보자. projects 디렉터리(혹은 앞서 작성한 소스 코드를 저장해 둔 위치)로 돌아간 후, 운영체제와 관계없이 다음의 명령을 실행하자.

```
$ cargo new hello_cargo
    Created binary (application) 'hello_cargo' package
$ cd hello_cargo
```

첫 번째 명령은 'hello_cargo'라는 새로운 디렉터리를 생성한다. 프로젝트의 이름을 'hello_cargo'라고 지정했으므로 카고는 필요한 파일을 프로젝트의 이름과 같은 디렉터리에 생성한다.

hello_cargo 디렉터리로 들어가 파일의 목록을 살펴보자. 그러면 카고가 main.rs 파일이 보관된 src 디렉터리와 Cargo.toml 파일을 생성해 준 것을 확인할 수 있다. 또한, .gitignore 파일과 함께 해당 디렉터리가 새로운 깃(Git) 저장소로 초기화된 것도 알 수 있다.

> **NOTE** 깃은 매우 광범위하게 사용되는 버전 관리 시스템이다. cargo new 명령을 사용할 때 --vcs 플래그를 이용하여 깃 외의 다른 버전 관리 시스템을 사용하거나 버전 관리 시스템을 아예 사용하지 않도록 설정할 수도 있다. 사용 가능한 옵션에 대한 더 자세한 내용을 확인하려면 cargo new –help 명령을 사용해 보자.

Cargo.toml 파일을 평소 자주 사용하는 텍스트 편집기를 이용해 열어보자. 그러면 [예제 1-2]와 비슷한 코드가 작성되어 있을 것이다.

예제 1-2 cargo new 명령이 생성한 Cargo.toml 파일의 코드

Cargo.toml

```
[package]
name = "hello_cargo"
version = "0.1.0"
authors = ["작성자 이름 <메일 주소>"]
edition = "2018"

[dependencies]
```

이 파일은 TOML(Tom's Obvious, Minimal Language) 형식으로 작성되어 있다. 이 형식은 카고의 설정 파일 형식이다.

첫 번째 줄의 [package]는 패키지의 설정을 관리하기 위한 구문들이 시작됨을 의미하는 섹션

의 제목이다. 이 파일에 더 많은 정보를 추가할수록 다른 섹션들이 더 추가된다.

섹션 제목 다음의 네 줄은 카고가 프로그램을 컴파일할 때 필요한 이름, 버전 및 작성자, 그리고 러스트가 사용할 에디션 등의 설정 정보를 지정하는 코드다. 카고는 현재 시스템 환경에서 작성자의 이름과 메일 주소를 읽기 때문에 이 정보가 올바르지 않다면 해당 정보를 지금 수정하고 파일을 저장하도록 하자. `edition` 키에 대한 더 자세한 내용은 부록 E에서 설명한다.

마지막 줄의 `[dependencies]`는 프로젝트의 의존 라이브러리 목록을 관리하는 섹션이 시작하는 부분이다. 러스트에서 코드의 패키지는 '크레이트(crate)'라고 부른다. 이 프로젝트는 별도의 크레이트가 필요하지 않지만, 제2장에서 구현할 프로젝트에서는 의존 패키지가 필요하므로 이 섹션에 의존 패키지 정보를 추가할 것이다. 이제 src/main.rs 파일을 열어서 확인해 보자.

src/main.rs

```
fn main() {
    println!("Hello, world!");
}
```

보다시피 카고가 [예제 1-1]과 같은 Hello, world! 프로그램을 미리 만들어 둔 것을 확인할 수 있다. 다만, 앞서 우리가 직접 만들었던 프로젝트와 다른 점은 카고 프로젝트에서는 코드가 src 디렉터리에 생성되었다는 점과 상위 디렉터리에 Cargo.toml 설정 파일이 생성되었다는 점이다.

카고 프로젝트는 소스 코드를 src 디렉터리에 보관한다. 최상위 프로젝트 디렉터리는 README 파일, 라이선스 정보, 설정 파일 등을 비롯해 코드와는 무관한 파일들을 위한 공간이다. 카고를 이용하면 프로젝트를 일관되게 정돈할 수 있다. 모든 파일은 종류별로 보관할 위치가 정해져 있으므로 적절한 위치에 보관하면 된다.

앞서 작성했던 Hello, world! 프로젝트처럼 카고를 이용하지 않고 프로젝트를 시작했더라도 얼마든지 해당 프로젝트를 카고 프로젝트로 변환할 수 있다. 프로젝트 코드를 src 디렉터리로 옮긴 후 Cargo.toml 파일을 적절하게 생성해 주면 된다.

1.3.2 카고 프로젝트의 빌드 및 실행

이제 카고 프로젝트를 빌드하고 실행하는 방법을 Hello, world! 프로그램과 비교해 보자. 먼저, 프로젝트를 빌드하려면 hello_cargo 디렉터리에서 다음 명령을 실행하면 된다.

```
$ cargo build
```

```
   Compiling hello_cargo v0.1.0 (/Users/rust/projects/hello_cargo)
    Finished dev [unoptimized + debuginfo] target(s) in 2.85 secs
```

이 명령을 실행하면 현재 디렉터리가 아니라 target/debug/hello_cargo(윈도우는 target\debug\hello_cargo.exe) 경로에 실행 파일이 생성된다. 이 실행 파일은 다음의 명령으로 실행할 수 있다.

```
$ ./target/debug/hello_cargo
Hello, world!
```

아무런 문제가 없이 빌드와 실행이 완료된다면 터미널에 'Hello, world!'라는 문자열이 출력된다. cargo build 명령을 처음 실행하면 카고는 최상위 디렉터리에 'Cargo.lock'이라는 파일을 생성한다. 이 파일은 프로젝트에 필요한 의존 패키지의 정확한 버전을 추적하기 위한 파일이다. 현재 프로젝트는 의존 패키지가 필요하지 않기 때문에 이 파일은 거의 비어 있다. 또한, 이 파일은 우리가 직접 관리하는 파일이 아니라 카고가 알아서 관리한다.

지금까지 cargo build 명령을 이용해 프로젝트를 빌드한 후 ./target/debug/hello_cargo 파일을 실행했지만, 카고를 이용하면 한 줄의 명령으로 코드를 컴파일하고 결과 파일을 실행할 수 있다.

```
$ cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.0 secs
     Running 'target/debug/hello_cargo'
Hello, world!
```

그런데 이번에는 카고가 hello_cargo 프로젝트의 컴파일을 실행했다는 결과가 출력되지 않았다. 그 이유는 카고가 소스 파일이 변경되지 않았음을 탐지하고 곧바로 바이너리를 실행했기 때문이다. 만일, 소스 코드를 수정하면 카고는 바이너리를 실행하기 전에 프로젝트를 다시 빌드하고, 이때 다음과 같은 결과가 나타난다.

```
$ cargo run
   Compiling hello_cargo v0.1.0 (/Users/rust/projects/hello_cargo)
    Finished dev [unoptimized + debuginfo] target(s) in 0.33 secs
     Running 'target/debug/hello_cargo'
Hello, world!
```

카고는 cargo check라는 명령 또한 지원한다. 이 명령은 코드의 컴파일 여부를 신속하게 검사하지만 실행 파일은 생성하지 않는다.

```
$ cargo check
   Compiling hello_cargo v0.1.0 (/Users/rust/projects/hello_cargo)
    Finished dev [unoptimized + debuginfo] target(s) in 0.32 secs
```

왜 실행 파일을 생성하지 않는 기능이 필요할까? 통상적으로 cargo check 명령은 실행 파일을 생성하는 과정을 생략하므로 cargo build 명령보다 훨씬 빠르게 실행된다. 따라서 코드를 작성하는 동안 작업 결과에 오류가 없는지를 계속 확인하려면 cargo check 명령을 이용하는 편이 훨씬 빠르다. 그래서 러스타시안들은 수시로 cargo check 명령을 실행해서 자신들의 코드가 제대로 컴파일되는지를 확인한다. 그리고 실행 파일이 필요하면 그때 cargo build 명령을 실행한다.

지금까지 카고에 대해 학습했던 내용을 정리해 보자.

- cargo build와 cargo check 명령을 이용하면 프로젝트를 빌드할 수 있다.
- cargo run 명령을 이용하면 프로젝트의 빌드와 실행을 한번에 해결할 수 있다.
- 카고는 빌드 결과물을 소스와 같은 디렉터리가 아니라 target/debug 디렉터리에 따로 저장한다.

카고를 사용하면서 얻을 수 있는 또 다른 장점은 사용 중인 운영체제와 무관하게 같은 명령을 사용할 수 있다는 점이다. 그래서 지금부터는 리눅스와 macOS용 명령과 윈도우용 명령을 별도로 구분할 필요가 없다.

1.3.3 릴리즈를 위한 빌드

프로젝트를 릴리즈할 준비가 되면 cargo build --release를 이용해서 최적화된 컴파일을 실행할 수 있다. 이 명령은 target/debug가 아닌 target/release 경로에 실행 파일을 생성한다. 이때 실행되는 최적화 덕분에 러스트 코드는 더 빠르게 실행되지만, 프로그램을 컴파일하는 시간은 더 길어진다. 더 빨리 자주 컴파일하기 위한 개발용 프로필과, 최종 완성된 프로그램을 사용자에게 제공하기 위해 최대한 빠르게 실행될 수 있도록 컴파일하기 위한 프로필이 별개로 분리된 이유는 바로 이 때문이다. 코드의 실행 시간을 벤치마킹해 보려면 cargo build --release 명령을 이용해 빌드한 후 target/release 경로의 실행 파일을 이용해 벤치마킹을 실행하기 바란다.

1.3.4 카고는 일종의 규칙이다

간단한 프로젝트라면 카고는 rustc 컴파일러를 사용했을 때와 비교해 그다지 큰 장점은 없다. 하지만 프로그램이 점점 더 복잡해지면 그 유용함을 충분히 느낄 수 있다. 여러 개의 의존 크레이트를 활용하는 복잡한 프로젝트라면 카고를 이용해 빌드를 조율하기가 훨씬 쉽다.

지금 만든 hello_cargo는 비록 간단한 프로젝트지만, 앞으로 러스트 경력을 쌓아가는 동안 사용하게 될 실제 도구들을 충분히 활용하고 있다. 사실, 기존의 어떤 프로젝트에 참여하더라도 깃을 이용해 코드를 내려받고 프로젝트의 디렉터리로 이동한 후 빌드를 수행하는 다음 명령을 공통으로 활용할 수 있다.

```
$ git clone someurl.com/someproject
$ cd someproject
$ cargo build
```

카고에 대한 더 자세한 정보는 https://doc.rust-lang.org/cargo/ 문서에서 확인하기 바란다.

요약

러스트 세계로의 첫발을 멋지게 내디딘 것을 축하한다. 이번 장에서 학습했던 내용을 요약하면 다음과 같다.

- rustup 스크립트를 이용해 최신 버전의 러스트 설치하기
- 더 최신 버전의 러스트로 업데이트하기
- 로컬에 설치된 문서 열어보기
- rustc 컴파일러를 직접 이용해서 Hello, world! 프로그램을 작성하고 실행하기
- 카고의 규칙을 이용해 새 프로젝트를 생성하고 실행하기

이제는 러스트 코드를 읽고 쓰는 것에 더 익숙해지기 위해 더욱 실용적인 프로그램을 구현해 볼 시간이다. 따라서 제2장에서는 '숫자 맞히기 게임(guessing game)' 프로그램을 만들어 볼 것이다. 만일, 러스트에서 동작하는 일반적인 프로그래밍 개념을 먼저 학습하고 싶다면 제3장을 읽은 후 다시 제2장으로 돌아와도 좋다.