깐깐하게 배우는 C

차례

옮긴이의 글

여기 두 명의 운전자가 있다. 두 사람 모두 10년 넘게 매일 운전을 했고, 사고를 낸 적도 없다. 하지만 한 사람은 자동변속기가 달린 승용차만 운전하였으며, 다른 한 사람은 승용차뿐만 아니라 트럭, 버스, 트레일러, 견인차 등 안 몰아 본 차량이 없을 정도로 다양한 종류의 자동차를 운전한 경험이 있다. 그렇다면 여러분의 자동차를 몰아 줄 사람으로 누구를 선택하겠는가?

다시, 질문을 조금 바꾸어보자. 두 명의 개발자가 있다. 두 사람 모두 10년 넘게 개발을 했고, 많은 프로젝트 경험을 가지고 있다. 하지만 한 사람은 루비 언어만 사용하였고, 다른 한 사람은 C를 비롯하여 파이썬, 루비, 자바 등 다양한 프로그래밍 언어를 사용하여 개발한 경험이 있다. 그렇다면 여러분의 소프트웨어를 개발할 사람으로 누구를 선택하겠는가?

질문이 내포하는 요지는 간단하다. 별도의 정보가 없는 상태에서 여러 가지 프로그래밍 언어를 사용하여 다양한 환경에서 많은 경험을 쌓은 개발자를 선택하는 것은 어쩌면 당연한 일이다. 이 말을 뒤집어보면 결국 좋은 개발자, 선택받는 개발자가 되기 위해서는 같은 기간 동안 더 많은 경험을 해봐야 한다는 것을 알 수 있다. 그렇다면 좋은 개발자가 되기 위해서는 어떻게 해야 할까? 바로 『깐깐하게 배우는 C』를 통해 그 답을 얻게 될 것이다.

C 언어는 프로그래밍 언어사(言語史)적으로 고전 프로그래밍 언어에 분류시켜도 될 만큼 오래됐을 뿐만 아니라 프로그래밍 언어학적으로 약점이 너무 많으며, 어셈블리 언어 수준으로 시스템에 가까이 접근할 수 있기 때문에 아무리 주의를 기울여도 공격당하는 일이 허다하다. 따라서, 역설적으로 C 언어를 사용하여 견고한 프로그램을 작성할 정도의 수준이 되었다면 어떤 프로그래밍 언어를 사용하더라도 견고한 프로그램을 만들 수 있게 된다. 또한 C 언어로 인해 탄생한 수많은 레퍼런스가 경험으로 축적되기 때문에 동일한 시간을 투자하고도 더 많은 노하우를 쌓을 수 있을 것이다.

이 책은 여러분에게 많은 질문을 던진다. 그 질문에 대한 답을 고민하는 사

이 어느덧 자신도 모르게 강인한 프로그래머로 거듭나게 될 것이다. 다만 고민의 깊이가 매우 크기 때문에 그간의 다른 프로그래밍 언어, 아니 C 언어에 익숙한 개발자라 할지라도 깊은 생각에 빠지게 만들 것이다. 이 과정은 자신만의 프로그래밍 철학을 수립하는 데 굉장한 도움이 될 뿐만 아니라, 더 높은 수준의 개발자로 거듭나고자 하는 사람에게는 어디에서도 구할 수 없는 값진 경험이 될 것이다.

언제까지 스크립트, OOP 같은 자동변속기에 의존할 수만은 없는 노릇이다. 백전노장 개발자가 되기 위해서는 아스팔트길만 달리면 안된다. 이 책이 여러분의 내면에 잠자고 있던 개발자적(?) 도전 욕구를 다시 한 번 불태워줄 도화선이 되리라 믿는다.

이 책을 옮길 수 있도록 기회를 주신 인사이트 대표님께 감사드린다. 그리고 거친 원고에 생명을 불어넣는 마법 같은 일을 매일같이 하시는 정수진 에디터님께도 감사드린다. 마지막으로 항상 곁에서 응원하는 아내와 윤서, 태원에게도 깊은 감사를 전한다.

정기훈

감사의 글

지금 읽고 있는 이 책을 집필하는 데 도움을 주신 세 부류의 사람들에게 감사드린다: 비판자, 조력자, 디자이너

비판자들은 그들의 고지식함, 초기 C 언어에 대한 비뚤어진 숭배 심리, 전문 교육에 대한 부재 등을 통해 이 책이 더 강하고 견고해질 수 있도록 도와주었다. 그들의 빛나는(?) 예제들은 무엇을 하면 안 되는지 잘 알려주었을 뿐만 아니라 저자로 하여금 정말 열심히 일하지 않을 수 없도록 만들었으며, 그 결과 이 책이 더 나은 프로그래머가 되는 길로 안내해주는 지침서가 될 수 있었다.

조력자들은 데브라 윌리엄스 컬리(Debra Williams Cauley), 비키 로랜드(Vicki Rowland), 엘리자베스 라이언(Elizabeth Ryan)을 비롯한 Addison-Wesley의 모든 팀과 수정 및 제안사항을 보내준 모든 분들이다. 제작, 교정, 편집 등 책을 출간하는 과정에서 들어간 이들의 노고 덕분에 이 책이 더 전문적이고 훌륭하게 다듬어졌다.

디자이너인 브라이언(Brian), 아서(Arthur), 베스타(Vesta), 사라(Sarah)는 나를 더 잘 표현해주고, 편집자들이 정해놓은 마감일의 압박에서 벗어나게 해주었다. 디자이너들과 이들이 제공한 예술이라는 선물 덕에 삶은 더 의미있고 풍요로워질 수 있었다.

이 책을 집필할 수 있도록 도와준 모든 분들께 감사드린다. 완벽한 책이란 없다는 것을 잘 알기에 이 책 역시 완벽하지 않겠지만, 적어도 완벽을 위해 최선을 다했다고 자부할 수 있다.

서문

이 책은 사실 C에 대한 책이 아니다

속았다는 기분이 들 수도 있겠지만, 분명히 말하건대 이 책은 여러분에게 C 프로그래밍을 가르치려는 것이 아니다. 물론 이 책을 통해 C 언어로 프로그램을 작성하는 방법을 배울 것이다. 그러나 이 책을 통해 얻게 될 가장 중요한 포인트는 바로 철저한 방어적 프로그래밍(rigorous defensive programming)을 배운다는 사실이다. 요즘은 자신이 작성한 프로그램이 잘 동작할 것이라고 쉽게 단정해버리는 프로그래머가 무척 많다. 하지만 이렇게 만들어진 프로그램은 어느 날 갑자기 재앙으로 치닫게 될 것이다. 여러분을 위해 문제를 해결해주던, 다른 현대적인 언어를 공부한 사람이라면 특히 그렇다. 이 책을 읽고 책에 수록된 예제들을 학습하는 과정을 통해 여러분은 악의적인 행동이나 공격으로부터 보호하는 프로그램을 작성하는 방법을 배우게 될 것이다.

방어적 프로그래밍 학습을 위해 이 책에서는 특별히 C 언어를 선택했는데 그 이유는 C가 불완전한 언어이기 때문이다. 1970년대만 하더라도 당시의 C 언어 설계에 사용된 개념들이 충분히 의미가 있었지만 지금 시대에는 모든 의미가 사라져버렸다. 통제되지 않고 무작정 사용되는 포인터부터 심각하게 손상된 NUL 종료 문자열에 이르기까지 모든 것이 C를 공격하는 거의 모든 보안 공격에 해당되기 때문이다. C는 널리 사용되는 것에 비해서 안전한 코드를 작성하기 가장 어려운 언어라고 본다. 오히려 C보다 어셈블리 언어가 안전한 코드를 작성하기 쉽다.

그렇다면 왜 굳이 C를 설명하려고 하는 것일까? 여러분이 더 훌륭한 프로그래머가 되기를 바라기 때문이다. 프로그래머로서 더 발전하기 위해 훈련하는 용도로 C 언어는 아주 훌륭한 프로그래밍 언어이다. 그 이유는 두 가지가 있다. 첫째, C 언어에는 현대의 보안과 관련된 기능이 거의 없어서 프로그

램을 작성할 때 좀 더 긴장해야 하고 로직이 어떻게 돌아가는지를 자세히 파악해야 하기 때문이다. 만일 C 언어로 안전하고 견고한 코드를 작성할 수 있다면 어떤 프로그래밍 언어로도 견고한 코드를 작성할 수 있다. 따라서 이 책을 통해 배우는 기술은 다른 언어에 모두 적용할 수 있다. 둘째, C 언어를 공부하다 보면 많은 사람이 작성한 산더미 같은 코드를 마주할 수 있기 때문이다. 이러한 코드를 통해 C 언어 계열의 다양한 언어의 기본 문법을 익힐 수 있다. C 언어를 익히고 나면 C++, Java, 오브젝티브-C, 자바스크립트 언어를 아주 쉽게 배울 수 있을 뿐만 아니라 다른 프로그래밍 언어들도 어렵지 않게 배울 수 있을 것이다.

부디 여기까지 읽고 근심에 빠지지 않기를 바란다. 왜냐하면 이 책은 아주 재미있고 쉬운데다가 사파(邪派)의 방식으로 설명하기 때문이다! 우선, 다른 프로그래밍 언어에서는 접하지 못한 프로젝트를 통해 C 언어를 공부하는 재미를 제공한다. 그리고 C 언어의 프로그래밍과 기술을 천천히 익힐 수 있도록 만든 검증된 연습을 통해 쉽게 접근할 수 있도록 할 것이다. 마지막으로 일반적인 설명 방법에서 벗어나 프로그램이 오류를 일으키게 하는 방법을 보여주고 이를 막는 안전한 코드를 설명함으로써 오류와 관련된 이슈를 쉽게 이해할 수 있도록 할 것이다. 또한 스택 오버플로, 허용되지 않는 메모리 접근 등 프로그래머들을 괴롭게 만드는 일반적인 결함들을 몸으로 직접 부딪치며 익히게 될 것이다.

저자가 집필한 다른 책과 마찬가지로 책을 읽어갈수록 점점 도전적인 내용들이 나올 것이다. 하지만 이 책을 다 읽고 나면 훨씬 발전하여 더 경쟁력을 갖춘 프로그래머로 거듭나게 될 것이다.

UB주의자(Undefined Behaviorists)

이 책을 다 읽은 시점에는 그동안 실행했던 거의 모든 C 프로그램을 디버그하고, 읽고, 고칠 수 있게 될 것이다. 그리고 새로운 프로그램을 작성할 때 견고한 코드를 구사하게 될 것이다. 하지만 이 책에서는 소위 공식적인 C 언어를 설명하지 않을 것이다. 물론, 여러분은 프로그래밍 언어를 배우고 그것을

잘 구사하는 방법도 배우겠지만 공식적인 C 언어는 안전하지 않다. 상당수의 C 프로그래머들은 그저 단순히 안전한 C 코드를 작성하지 않으며, 그 이유로 UB(Undefined Behavior, 미정의 행동)를 든다. UB는 ANSI C에 속하는 내용으로, C 컴파일러가 무시해도 되는 방법들이 수록되어 있다. 물론 C 표준에는 UB와 관련하여 다음과 같이 프로그램을 작성하면 모든 것이 물거품이 되면서 컴파일러는 일관되게 아무것도 할 수 없다는 내용을 일부 언급하고 있다. C 프로그램이 문자열 끝을 넘어서까지 계속해서 읽는 행위가 대표적인 UB이며, C에서 아주 빈번히 발생하는 에러이다. 내용을 조금 들여다보면, 간단하게 표현해서 C 언어는 문자열을 NUL 바이트 또는 0바이트로 끝나는 메모리 블록으로 정의한다. 그런데 프로그램 외부에서 많은 문자열이 입력되면 NUL로 끝나지 않는 문자열이 들어오는 일이 종종 발생한다. 이런 경우 C 프로그램은 어쨌거나 문자열이 끝나지 않았기 때문에 계속해서 문자열을 읽으려고 시도하게 되며, 컴퓨터의 메모리 영역에 무제한으로 접근하게 되어 프로그램이 충돌을 일으키게 된다. C 언어 이후에 개발된 모든 프로그래밍 언어는 이러한 문제를 예방하고 있지만 C는 그렇지 못하다. UB에 대한 C 언어의 이러한 소극적인 대응은 모든 C 프로그래머로 하여금 UB를 다룰 필요가 없다고 생각하게 만들었다. 그 결과 C 프로그래머들은 잠재적으로 NUL 바이트를 침범할 수 있는 코드로 가득한 프로그램을 작성하면서도 이 부분에 대해 지적하면 "이것은 UB라서 막지 않아도 돼요."라고 이야기하는 일이 벌어지게 되었다. UB에 많이 의존하는 C 언어의 이러한 성향은 대부분의 C 코드가 왜 그렇게 무시무시할 정도로 위험한지를 설명해준다.

필자는 C 코드를 작성할 때 UB를 일으키지 않도록 코드를 작성하거나 UB를 방어하도록 코드를 작성하는 방식으로 UB를 피하기 위해 노력한다. 하지만 UB가 너무 많고 서로 얽혀 고르디우스의 매듭처럼 풀 수 없는 문제가 되기도 한다. 이 책을 통해 UB를 일으키는 방법과 함께 어떻게 하면 UB를 피할 수 있는지를 설명하고 다른 사람의 코드에서 UB를 일으키는 방법도 고민해 볼 것이다. 하지만 무작위로 발생하는 UB를 피하는 것은 거의 불가능하다는 것을 항상 염두에 두고, 이를 해결하기 위해서는 그저 최선을 다해야 한다는 생각을 가져야 할 것이다.

아마도 C 언어의 골수팬들이 종종 UB를 가지고 여러분을 이기려 들 것이다. 코드는 작성하지 않으면서 UB에 대한 지식만 달달 외운 부류의 사람들이 있는데, 이론만 앞세우는 이런 사람들을 초급 프로그래머가 당해내기는 쉽지 않다. 혹시라도 이런 부류의 프로그래머를 만난다면 그냥 무시하기 바란다. 종종 그들은 여러분을 도와준다고 해놓고는 막상 C 프로그램을 가르치기보다는 자신의 우월성을 증명하기 위해 오만하고 모욕적으로 끊임없이 질문할 것이다. C 코드에 대한 도움이 필요할 경우 차라리 필자의 이메일 *help@learncodethehardway.org*로 연락하라. 내가 기꺼이 도와줄 것이다.

C는 예쁘면서 까다로운 언어

UB 개념의 존재는 더 나은 프로그래머로 발전하기 위해 C 언어를 배우는 것이 훌륭한 선택인 이유 중 하나다. 여러분이 이 책에서 설명하는 방식대로 훌륭하고 견고한 C 코드를 작성할 수 있다면, 어떤 언어를 사용하더라도 살아남을 수 있을 것이다. 긍정적으로 생각하면 C 언어는 여러 면에서 정말 우아한 언어이다. C 언어 문법은 실제로 C 언어가 가진 힘을 고려할 때 매우 간결하다. 이것이 많은 프로그래밍 언어가 지난 45년 동안 C 언어의 문법을 차용한 이유이기도 하다. C 언어는 또한 최소한의 기술을 사용하고도 상당히 많은 것을 제공한다. 그래서 C 언어 학습을 마치고 나면 C 언어가 매우 우아하고 아름다우면서도 동시에 약간 까다롭다는(ugly) 것이 고맙게 느껴질 것이다. C 언어는 오래되고 아름다운 기념비 같아서 몇 미터 떨어져서 바라보면 환상적으로 보이지만 가까이 가서 보면 C 언어가 지니고 있는 균열과 결함이 모두 보일 것이다.

이러한 이유로 이 책에서는 최신의 컴파일러에서 작업할 수 있는 가장 최근 버전의 C를 설명할 것이다. 최신의 C는 실용적이고 단도직입적이며 간단하면서도 완벽하게 작동하는 C 언어의 하위 집합일 뿐만 아니라 모든 부분에서 잘 작동하며 많은 함정을 피한다. 이 버전의 C는 실제로 업무에 사용할 수 있는 것으로, 골수팬들이 사용하기를 시도하다 번번이 실패하는, 백과사전에나 나올 법한 C가 아니다.

필자는 필자가 사용하는 C 코드가 견고하다는 것을 잘 안다. 왜냐하면 대규모 작업을 고장 없이 수행하는 깨끗하고 견고한 C 코드를 작성하는 데 20

여년의 세월을 보냈기 때문이다. 필자가 작성한 C 코드는 트위터(Twitter)나 에어비앤비(Airbnb)와 같은 회사의 운영을 지원하면서 아마도 수조 건의 트랜잭션을 처리했을 것이다. 그러면서도 고장이 나거나 보안 공격을 받은 적이 거의 없었다. 이 외에도 수년간 루비 온 레일스(Ruby on Rails) 웹을 지원하는 동안 필자의 코드는 아름답게 동작했을 뿐만 아니라 심지어 보안 공격까지도 막았다. 그 사이 다른 웹 서버들은 가장 단순한 공격에도 번번이 죽고 말았다.

필자는 C 코드를 작성할 때 견고함을 추구한다. 그러나 더 중요한 것은 이렇게 필자가 C 코드를 작성할 때 지니는 마음가짐을 모든 프로그래머가 가져야 한다는 것이다. 즉, 제대로 동작하는 것은 아무것도 없다고 생각하고, 오류가 나지 않도록 최선을 다하겠다는 마음가짐으로 C를 비롯한 모든 프로그래밍 언어에 접근하는 것이다. 다른 프로그래머들은(심지어 좋은 C 프로그래머로 알려진 사람들도) 코드를 작성할 때 모든 것이 작동할 것이라고 가정하면서 UB나 운영체제에 기대는 바람에 결국 제대로 동작하지 않아 솔루션으로서의 기능을 잃는 경우가 많다. 분명히 이 책에서 가르치는 코드가 "진짜 C"가 아니라고 말하는 사람들이 있을 것이다. 만일 그들이 보여주는 내용이 이 책에서 설명하는 내용과 다르다면 아마도 여러분은 이 책에서 사용된 코드를 사용하여 그들의 코드가 왜 안전한지 않은지를 설명할 수 있을 것이다.

그렇다면 필자의 코드가 과연 완벽하다는 것인가? 절대로 그렇지 않다. 그저 C 코드일 뿐이다. 완벽한 C 코드를 작성하는 것은 불가능하며 실제로 어떠한 프로그래밍 언어로도 완벽한 코드를 작성하는 것은 불가능하다. 그것이 바로 프로그래밍이 즐거움 반 좌절 반인 이유이다. 다른 사람의 코드를 가져와서 산산조각낼 수도 있으며, 반대로 누군가가 자신의 코드를 완전히 무너뜨릴 수도 있는 것이다. 중요한 차이점은 모든 코드에는 결함이 있지만 필자는 항상 필자의 코드에 결함이 있다고 가정하고 결함을 예방하는 것이다. 부디 이 책을 완벽히 소화하여 필자가 여러분에게 주고자 하는 선물이자 20여 년간 고품질의 강력한 소프트웨어를 만들 수 있도록 해준, 프로그래밍에 임하는 방어적 마음가짐을 가질 수 있기를 바란다.

이 책을 통해 학습하게 될 내용

이 책의 목적은 여러분을 C 언어에 충분히 단련시켜 자신만의 소프트웨어를 작성하거나 다른 사람의 C 코드를 수정할 수 있도록 만드는 것이다. 이 책을 읽은 후에는 반드시 K&R C로도 불리는 C 언어의 창시자 브라이언 커니핸(Brian Kernighan)과 데니스 리치(Dennis Ritchie)의 *C Programming Language, Second Edition*(Prentice Hall, 1988)을 읽기 바란다.[1] 이 책에서 여러분이 배우게 될 내용은 다음과 같다.

- C 언어의 기본적인 문법 및 관용적 표현
- 컴파일, `make` 파일, 링커
- 버그 발견 및 방지
- 방어적인 코딩 사례
- C 프로그램 깨뜨리기
- 기본 UNIX 시스템 소프트웨어 작성

마지막 연습을 마치고 나면 기본 시스템 소프트웨어, 라이브러리 및 기타 소규모 프로젝트를 처리할 수 있는 충분한 역량을 확보하게 될 것이다.

이 책을 읽는 방법

이 책은 적어도 하나 이상의 다른 프로그래밍 언어를 배운 프로그래머를 대상으로 한다. 아직 프로그래밍 언어를 배운 적이 없다면 필자의 저서 *Learn Python the Hard Way*(Addison-Wesley, 2013)[2]를 참조하기 바란다. 초보자를 위한 책이기 때문에 프로그래밍 공부를 위한 첫 번째 서적으로 안성맞춤일 것이다. *Learn Python the Hard Way*를 모두 끝냈다면 이제 이 책을 시작할 수 있을 것이다.

이 책을 읽을 때 지켜야 하는 몇 가지 규칙이 있다.

1 (옮긴이) 이 책은 『kernighan의 C 언어 프로그래밍: 수정판 2판』(휴먼싸이언스, 2016)으로 번역 출간되었다.
2 (옮긴이) 이 책은 『깐깐하게 배우는 파이썬』(인사이트, 2014)로 번역 출간되었다.

- 모든 코드는 직접 입력할 것. 복사하여 붙여넣기하지 말 것!
- 책에 나타난 그대로 코드를 입력할 것. 심지어는 주석까지도 그대로 입력할 것
- 실행 시 동일한 결과가 나타나는지 확인할 것
- 버그가 있다면 수정할 것
- '더 해보기'를 하되, 잘 모르겠다면 일단 넘어갈 것
- 도움을 청하기 전에 항상 먼저 스스로 알아내기 위해 노력할 것

이 규칙들을 따르면서 책의 모든 내용을 수행했음에도 불구하고 C 코드를 작성하지 못할 수도 있다. 이런 경우에도 최소한 시도는 해 보아야 한다. 모든 사람에게 해당되지는 않지만, 적어도 노력하는 것만으로도 여러분은 더 나은 프로그래머로 성장할 수 있을 것이다.

핵심 역량

이 책을 읽는 여러분이 프로그래밍 언어에 대한 경험이 적다고 가정할 것이다. 엉성한 사고와 선부른 해커 흉내를 내게 만드는 파이썬이나 루비를 경험했을 수도 있고, 또는 쿠션으로 둘러싼 아기 요람과 같이, 컴퓨터가 순전히 함수로만 구성된 환상의 땅인 것처럼 가장하는 LISP과 같은 언어를 사용했을 수도 있을 것이다. 혹은 프롤로그(Prolog)를 배워 모든 세상은 데이터베이스로 되어 있고 그 안에서 단서를 찾으면 그뿐이라고 생각하고 있을 수도 있겠다. 최악의 경우로, 이미 IDE(Integrated Development Environment, 통합개발환경)에 익숙해져 버려서 더는 머리를 쓸 일이 없어진 바람에 세 글자를 입력하고 CTRL-SPACE를 치지 않으면 전체 함수의 이름을 입력하지 못하는 지경이 되어버렸을 수도 있을 것이다.

여러분의 프로그래밍 언어 배경이 무엇이든 관계없이 다음과 같은 영역에서 약간의 개선 효과를 기대할 수 있을 것이다.

읽기와 쓰기

IDE를 사용하는 경우 특히 그렇지만, 대충 넘기는 성향을 지닌 프로그래머들

을 쉽게 볼 수 있으며 이들은 자신들의 이런 성향 때문에 코드를 이해하는 데 어려움을 겪는다. 시간을 들여 자세히 이해해야 할 필요가 있는 코드임에도 불구하고 그냥 건너뛰는 것이다. 프로그래밍 언어 중에는 프로그래머가 실제로 코드를 작성하지 않아도 되는 도구를 제공하기도 하는데, 이런 것들에 익숙해진 사람들은 C와 같은 언어를 만나게 되면 바로 무너진다. 이러한 문제를 해결하기 위한 가장 간단한 방법은, 모든 사람이 자신과 동일한 문제를 겪고 있다는 것을 이해한 다음 좀 더 집중해서 꼼꼼하게 읽고 쓰도록 노력하는 것이다. 처음에는 고통스럽고 성가시겠지만 자주 휴식을 취하면서 하다 보면 마침내 읽고 쓰는 일이 쉬워질 것이다.

디테일에 집중하기

모든 사람이 이 부분에 약하며, 이것이 나쁜 소프트웨어를 만들어내는 가장 큰 이유이다. 다른 프로그래밍 언어들은 굳이 집중하지 않아도 되도록 해주지만 C 언어는 프로그래머에게 고도의 집중을 요구한다. 왜냐하면 C 언어는 컴퓨터에 가까운 언어이며, 컴퓨터는 아주 까다롭기 때문이다. C 언어에는 '유사한 종류' 또는 '충분히 가까움'이란 것은 없기 때문에 C 언어를 사용할 때는 주의를 기울여야 한다. 항상 작성한 내용을 확인하고 또 확인해야 한다. 그리고 여러분이 옳다는 것을 증명하기 전까지는 작성하는 모든 것이 잘못되었다고 가정하자.

차이점 짚어내기

다른 프로그래밍 언어에 익숙한 사람들의 핵심적인 문제는 그들의 뇌가 C 언어가 아닌 자신에게 익숙한 프로그래밍 언어를 기준으로 차이를 짚어내도록 훈련받았다는 것이다. 그래서 여러분이 작성한 코드와 연습문제의 코드를 비교하고 차이점을 찾아내면서도 이러한 차이가 크게 상관없거나 그저 친숙하지 않을 뿐이라고 생각할 것이다. 이 책은 여러분에게 자신의 실수를 볼 수 있도록 하는 전략을 제공할 것이다. 만일 자신이 작성한 코드가 이 책의 코드와 정확히 일치하지 않는다면 작성한 코드가 잘못되었다는 것을 명심하기 바란다.

설계와 디버깅

필자는 더 쉬운 프로그래밍 언어를 좋아하는데 가지고 놀기 좋기 때문이다. 인터프리터(interpreter)에 아이디어를 입력하기만 하면 즉시 그 결과를 볼 수 있다. 이런 프로그래밍 언어는 단순히 아이디어를 구현할 때는 좋지만, 동작할 때까지 계속해서 고치다 보면 결국 아무것도 동작하지 않는다. C 언어가 어려운 이유는 코딩을 하기 전에 먼저 무엇을 만들 것인지를 설계해야 하기 때문이다. 물론, 잠깐은 이리저리 해보는 것이 가능하겠지만 C 언어의 경우에는 다른 프로그래밍 언어보다 조금 더 일찍 진지해져야 한다. 이 책은 여러분에게 코딩을 시작하기 전에 프로그램의 핵심 부분을 설계하는 방법을 가르쳐 줄 것이며, 이는 동시에 여러분을 더 나은 프로그래머가 되도록 도와줄 것이다. 심지어 아주 간단한 설계만으로도 코딩하는 길이 수월해질 수 있다.

여러분은 C 언어 학습을 통해 더 나은 프로그래머로 발전할 것이다. 왜냐하면 C 언어를 공부하는 과정에서 위의 이슈들을 더 일찍, 그리고 더 자주 다루게 될 것이기 때문이다. 여러분은 엉성하게 코드를 작성할 수 없을뿐더러, 만일 그렇게 한다면 아무것도 동작하지 않을 것이다. C 언어의 장점은 혼자서도 이해할 수 있을 정도로 아주 단순한 언어라는 것이다. 그러면서도 컴퓨터를 배우고 핵심 프로그래밍 기술을 연마할 수 있도록 만든다는 점에서 C 언어는 훌륭한 프로그래밍 언어라고 할 수 있다.

연습 0

설정하기

전통적인 방식을 따라 연습 0에서는 앞으로 이 책을 공부하는 데 기본이 될 컴퓨터의 설정에 대해 설명한다. 이번 연습에서 여러분은 자신이 보유하고 있는 컴퓨터의 사양에 맞추어 패키지와 소프트웨어를 설치하게 될 것이다.

0.1 리눅스

리눅스는 C 개발 환경 구성이 가장 쉬운 시스템이다. 데비안 시스템에서는 명령 줄에서 아래와 같이 실행하면 된다.

```
$ sudo apt-get install build-essential
```

페도라, 레드햇, CentOS 7 등과 같은 RPM 기반의 리눅스의 경우에는 아래와 같이 실행한다.

```
$ sudo yum groupinstall development-tools
```

만일 위에서 언급한 것과 다른 리눅스 배포판을 가지고 있다면 "c development tools"라는 키워드로 검색한 다음 갖고 있는 배포판에서 C 언어 개발 도구를 설치하는 방법을 찾아보면 된다. 설치가 끝난 다음에는 아래와 같은 명령을 실행할 수 있을 것이다.

```
$ cc --version
```

위 명령을 통해 어떤 컴파일러가 설치되었는지 확인할 수 있다. 아마도 대부분 GNU C 컴파일러(GCC)가 설치되었을 테지만 이 책에서 사용하는 컴파일러와 다른 컴파일러가 설치되었다고 해서 걱정할 필요는 없다. Clang's

Getting Started 설명[1]에 따라 보유하고 있는 리눅스 버전에 맞는 Clang C 컴파일러를 설치하면 되기 때문이다. 이렇게 해서도 안 되는 경우에는 인터넷에서 구글링하는 수밖에 없다.

0.2 Mac OS X

Mac OS X은 훨씬 더 쉽다. Apple 사이트에서 최신 버전의 XCode를 다운받거나 설치 DVD에 있는 XCode를 설치하면 된다. 워낙 용량이 커서 다운받는 것보다는 설치 DVD를 사용하기를 권한다.[2] 인터넷에서 "xcode 설치"라고 입력하면 쉽게 설치 방법을 찾아볼 수 있다. 또는 앱스토어를 이용하여 다른 앱과 같이 XCode를 설치하고 자동으로 업데이트 소식을 받을 수도 있다.

C 컴파일러가 동작하는지 확인하기 위해서는 다음과 같이 입력하면 된다.

```
$ cc --version
```

아마도 Clang C 컴파일러 버전을 사용하고 있는 것을 확인할 수 있을 것이다. 그러나 오래된 버전의 XCode를 사용하는 경우에는 GCC가 설치되었을 것이다. 하지만 이 버전을 사용해도 무관하다.

0.3 윈도우

MS 윈도우의 경우, 표준 UNIX 소프트웨어 개발 도구의 많은 부분을 차용하고 있는 Cygwin을 사용할 것을 권장한다. Cygwin은 설치와 사용 방법이 어렵지 않다. Cygwin과 비슷한 것으로는 MinGW가 있다. MinGW는 좀 더 단순함을 추구하고 있지만 있을 것은 다 있다. MS는 C 개발 도구에 대한 지원을 점차 줄여나가는 것으로 보인다. 따라서 어쩌면 MS의 컴파일러를 이용하여 이 책에 실린 코드를 컴파일하는 데 문제가 생길 수도 있다는 점을 염두에 두자.

좀 더 고급스런 방법을 원한다면 VirtualBox를 이용하여 리눅스를 설치하

1 http://clang.llvm.org/get_started.html
2 (옮긴이) 사실 우리나라는 XCode를 다운받기에 너무 훌륭한 인터넷 환경을 갖추었기 때문에 설치 DVD를 찾을 시간이면 충분히 다운받을 수 있다.

고 윈도우상에서 완전한 리눅스를 돌릴 수 있다. 이 방법은 윈도우의 설정을 건드릴 필요 없이 리눅스 가상 머신을 없앨 수 있다는 추가적인 장점이 있다. 게다가 리눅스 사용법까지 배울 수 있어 개발 작업이 더 즐겁고 가치 있게 느껴질 것이다. 리눅스는 현재 많은 분산 컴퓨터 및 클라우드 인프라 회사에서 주 운영체제로 사용되고 있기 때문에 리눅스를 공부한다면 분명히 미래 컴퓨팅에 대한 여러분의 지식을 향상시킬 수 있을 것이다.

0.4 텍스트 편집기

프로그래머에게 있어 텍스트 편집기의 선택은 쉽지 않은 문제이다. 초급자라면 GEdit을 추천한다. GEdit는 간단하고 코딩하기에 좋다. 하지만 일부 국가 환경에서는 동작하지 않는다. 그리고 어느 정도 프로그래밍 경험이 있다면 이미 선호하는 텍스트 편집기가 있을지도 모르겠다.

이를 염두에 두고 여러분이 사용하는 플랫폼에서 프로그래머용 표준 텍스트 에디터를 몇 가지 시험해본 다음 가장 마음에 드는 텍스트 편집기를 고르기 바란다. 만일 GEdit를 사용하고 있고 이것이 마음에 든다면 그대로 사용하면 된다. 혹은 다른 텍스트 편집기를 써보고 싶다면 시험해본 다음 하나를 고르면 된다.

중요한 점은 완벽한 편집기를 찾으려고 애쓰지 말아야 한다는 것이다. 텍스트 편집기는 다음과 같은 순서로 시험하기 바란다. 우선 하나를 고르고 잠시 사용해 본 다음, 다른 것을 찾아보고 또 써보는 것이다. 절대로 한 개의 편집기만 가지고 이를 구성하고 완벽하게 만드는 데 며칠씩 소비하지 말아야 한다.

사용해 보면 좋을 만한 텍스트 편집기를 몇 가지 더 추천한다면 다음과 같은 것이 있다.

- GEdit(리눅스와 OS X)
- TextWrangler(OS X)
- Nano(거의 대부분의 터미널)
- Emacs와 OS X용 Emacs(사용하기 위해서는 약간의 학습이 필요하다)

• Vim과 MacVim

아마도 사람들마다 서로 다른 편집기를 사용할 것이다. 위에서 언급한 편집기들은 필자가 알고 있는 무료 편집기 중 일부에 지나지 않는다. 가장 마음에 드는 것이 나타날 때까지 위 편집기(혹은 상용 편집기가 될 수도 있겠다)를 시험해보기 바란다.

절대 IDE를 사용하지 말 것

프로그래밍 언어를 공부할 때는 IDE를 멀리하기 바란다. 이것들은 여러분이 필요한 일을 끝내는 데는 도움이 되겠지만, 프로그래밍 언어를 공부하는 데에는 방해 요인이 될 것이다. 필자의 경험으로 볼 때 쟁쟁한 프로그래머들은 IDE를 사용하지 않으면서도 IDE를 사용하는 프로그래머와 동일한 속도로 코드를 만든다. 또한 IDE로 생성된 코드는 품질이 낮다. 왜 그런지는 모르겠지만 프로그래밍 언어에 있어 깊이 있고 단단한 기술을 원한다면 공부하는 동안 IDE를 사용하지 말 것을 강력히 권한다.

전문 프로그래머의 텍스트 편집기 사용법을 배우는 것도 전문가적인 삶으로 가는 데 유용한 기술이다. IDE에 의존한다면 최신의 프로그래밍 언어를 배우기 전에 먼저 새로운 IDE를 기다려야 하는 일이 생긴다. 이것은 여러분의 경력에 비용이 추가됨을 의미한다. 즉, 프로그래밍 언어의 트렌드를 선도하지 못하게 한다. 그러나 일반적인 텍스트 편집기를 사용한다면, 여러분은 누군가 이 언어를 IDE에 추가할 때까지 기다릴 필요 없이 언제든지 원하는 프로그래밍 언어로 코딩할 수 있다. 일반(generic) 텍스트 편집기는 결국 자신의 의지대로 탐험하고 자신의 생각대로 경력을 관리할 수 있는 자유를 의미한다.

연습 1

간만에 써보는 컴파일러

모든 것을 설치한 후에는 컴파일러가 제대로 동작하는지 확인해야 한다. 가장 쉬운 방법은 C 프로그램을 작성하는 것이다. 적어도 하나 이상의 프로그래밍 언어를 알고 있다고 가정하고 있기 때문에 작지만 확장이 가능한 예제를 만들 수 있을 것이다.

ex1.c

```
1   #include <stdio.h>
2
3   /* 이것은 주석이다. */
4   int main(int argc, char *argv[])
5   {
6       int distance = 100;
7
8       // 이것도 주석이다.
9       printf("You are %d miles away.\n", distance);
10
11      return 0;
12  }
```

1.1 코드 분석

타이핑하면서 느꼈을지 모르지만, 이 코드 안에는 C 언어의 몇 가지 기능이 담겨 있다. 이제 코드를 줄 단위로 빠르게 분석하여 각각의 부분을 더 잘 이해하도록 하고, 이를 통해 제대로 연습할 수 있도록 할 것이다. 코드를 분석하는 동안 모든 것을 이해하지 못하더라도 걱정하지 말자. 여기에서 간략하게 언급한 개념들은 이 책의 나머지 부분을 통해 공부하게 될 것이다.

그러면 위 코드를 줄 단위로 분석해보자.

1행 `include` 구문으로, 이 소스 파일에 다른 파일의 내용을 가져오기 위

해 사용한다. C에서는 관습적으로 `.h` 확장자를 헤더(header) 파일로 사용하는데, 헤더 파일에는 프로그램에서 사용하는 함수의 리스트가 담겨있다.

3행 여러 줄의 주석을 의미한다. 주석의 시작을 뜻하는 `/*`과 끝을 나타내는 `*/` 사이에 몇 줄이든 여러분이 원하는 내용을 적을 수 있다.

4행 그동안 사용하던 것보다 조금 더 복잡한 버전의 `main` 함수이다. C 프로그램이 동작하는 방식은 운영체제가 프로그램을 로드한 다음 이름이 `main`인 함수를 실행시키는 것이다. 함수를 완전히 구현하기 위해서는 리턴 타입으로 `int`를 지정하고 두 개의 파라미터를 취하도록 해야 한다. `int` 값으로 인수의 개수를, `char *` 문자열 배열로 인수들을 받는다. 무슨 말인지 모르겠다고? 걱정하지 않아도 된다. 나중에 다시 설명할 것이다.

5행 함수의 본문을 시작하기 위해 `{` 문자를 이용하여 블록의 시작을 알린다. 파이썬의 경우 `:`을 넣고 들여쓰기만 하면 된다. 다른 프로그래밍 언어의 경우 `begin`이나 `do`를 사용하기도 한다.

6행 변수 선언과 지정을 동시에 진행한다. 이것이 `type name = value;` 구문으로 변수를 생성하는 방법이다. C에서는 구문(statement)의 마지막에 `;`(세미콜론) 글자를 붙인다(로직은 예외).

8행 주석의 다른 형태이다. 파이썬이나 루비에서의 주석과 동일하게 적용되는데, `//`으로 주석이 시작되고 문장의 끝에서 주석이 끝난다.

9행 여러분의 오랜 친구인 `printf` 함수를 호출한다. 다른 프로그래밍 언어와 마찬가지로, 함수 호출은 `name(arg1, arg2);` 구문으로 동작하며 인수는 0개 이상을 사용할 수 있다. `printf` 함수는 조금 이상한 종류의 함수로 여러 개의 인수를 받을 수 있다. 이 부분에 대해서는 나중에 다룬다.

11행 `main` 함수의 리턴(return)으로, 종료 값을 운영체제에 전해준다. 아마도 아직은 UNIX 소프트웨어에서 리턴 코드를 사용하는 방법이 익숙하지 않을 것이다. 이 부분에 대해서도 나중에 다룰 것이다.

12행 마지막으로 닫는 괄호(`}`) 문자를 사용하여 `main` 함수의 끝을 알린다.

또한 여기가 이 프로그램의 끝이기도 하다.

이번 코드 분석에는 새로운 정보가 가득하다. 그래서 줄 단위로 공부하여 최소한 어떤 일이 일어나는지는 알아야 한다. 물론 모든 것을 알 수 없겠지만 적어도 충분히 추측하고 고민한 후 다음으로 진행할 수는 있을 것이다.

1.2 실행 결과

지금까지의 코드의 내용을 ex1.c에 담고, 아래의 예제 셸 출력과 같이 입력하여 실행시켜 보자.

연습문제 1 Session

```
$ make ex1
cc -Wall -g      ex1.c    -o ex1
$ ./ex1
You are 100 miles away.
$
```

첫 번째 명령인 make는 C 프로그램(을 비롯한 많은 다른 언어들)을 빌드하는 방법을 알고 있는 툴이다. make를 실행하면서 ex1을 파라미터로 제공하는 것은 make로 하여금 ex1.c 파일을 찾고 컴파일러를 실행시켜 빌드하라고 알려주는 것이다. 이렇게 해서 생성되는 ex1 파일은 실행파일로, ./ex1과 같이 입력하여 실행시킨다. 그러면 결과가 출력된다.

1.3 프로그램 깨뜨리기

이 책에서는 (가능하다면) 각각의 프로그램을 깨뜨리는 방법을 설명할 것이다. 이를 위해 프로그램에 이상한 내용을 넣도록 하거나 이상하게 실행시키도록 할 것이다. 또는 코드를 변경하여 충돌을 일으키거나 컴파일러가 오류를 일으키도록 만들기도 할 것이다.

이 프로그램의 경우, 간단히 아무 데나 지운 후 컴파일해보기 바란다. 이때 지울 수 있는 것을 추측해보고 다시 컴파일한 다음, 어떤 오류가 발생하는지를 확인한다.

1.4 더 해보기

- `ex1` 파일을 텍스트 편집기로 연 다음 아무 부분이나 바꾸거나 삭제하고 다시 실행시켜 무슨 일이 발생하는지 확인해보자.
- "hello world"보다 더 복잡한 다섯 줄 이상의 텍스트나 무언가를 출력해보자.
- `man 3 printf`를 실행하고 이 함수에 대한 내용을 읽어본 다음 다른 내용에 대해서도 동일하게 확인해보자.
- 매 줄마다 이해하지 못하는 기호를 적고는 이것들이 무엇을 의미하는지를 추측해보자. 여러분이 추측한 내용을 노트 등에 별도로 기록해 놓으면 나중에 과연 제대로 추측했었는지를 확인할 수 있을 것이다.