Wrox Programmer to Programmer™

Beginning

Rust® Programming

Ric Messier

BEGINNING RUST PROGRAMMING

BEGINNING
Rust® Programming

BEGINNING

Rust® Programming

Ric Messier

wrox™
A Wiley Brand

Beginning Rust® Programming

Copyright © 2021 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-119-71297-8
ISBN: 978-1-119-71298-5 (ebk)
ISBN: 978-1-119-71287-9 (ebk)

Manufactured in the United States of America

Library of Congress Control Number: 2020937958

ABOUT THE AUTHOR

RIC MESSIER is an author, consultant, and educator who holds CCSP, GCIH, GSEC, CEH, and CISSP certifications and has published several books on information security and digital forensics. With decades of experience in information technology and information security, Ric has held the varied roles of programmer, system administrator, network engineer, security engineering manager, VoIP engineer, consultant, and professor. He is currently a Principal Consultant with FireEye Mandiant.

ABOUT THE TECHNICAL EDITOR

JESSICA ROCCHIO has been in the information technology industry for over a decade and is currently an incident response consultant at Mandiant. Over the last few years, she has worked with various programming languages. She has spent most of her career in incident response, forensics, intelligence, insider threats, and vulnerability management. Jessica has worked on a wide range of incidents, including espionage, cybercrime, fraud, data theft, and insider threats.

ACKNOWLEDGMENTS

Without my tech editor, Jessica, and my project editor, Kim, I might not have made it through this book, so many thanks to them!

CONTENTS

INTRODUCTION

Save me from another "hello, world" book. Don't make me have to skim or skip through a half dozen chapters before I can get to something that's going to be useful to me. Or you, in this case. I can't tell you the number of programming books I've purchased over the decades, hoping to actually learn the language, only to end up just not using the book because it wasn't presented in a way that made a lot of sense to me. Instead of a dry explanation of how the language is constructed so you can try to put it all together in meaningful ways yourself, the purpose of this book is to jump straight into writing hopefully interesting or useful programs. Once we have the program, we can take a look at how it's constructed. You'll be learning by doing—or learning by example, if you prefer. I hope you'll find this a more useful and practical way of learning Rust.

Rust is an interesting language, as it turns out. Like so many other languages, it claims a C-like syntax, which is roughly correct but misses out on many important elements. Where Rust really shines is where C has introduced bad behavior in programming practices. This is more apparent as more have been using C as a language. Where C provides you with the gun and the bullets to shoot yourself in the foot, Rust provides you with necessary protections to keep you from injuring yourself or, from the perspective of the application, keeps the application from crashing. Rust is focused on protecting the memory space of the program, in part to provide a better ability for concurrent programming. After all, Rust is considered to be a systems programming language, meaning it is intended for applications that are lower level than those that a user directly interacts with.

In addition to protections provided to the programmer, Rust has a reasonably active community that can be used not only for support but also to get additional functionality for your programs. There are a lot of third-party libraries. These libraries can make your life easier by introducing you to functionality without you needing to write it yourself.

The idea behind this book is to introduce you to Rust in context, rather than via snippets that, by themselves, don't work. You need all the surround to fully understand what is happening in the program. You'll find this out when you are looking at example code sometimes. This is true with the Rust documentation: it's like you need to fully understand the language to understand the examples you are looking at. This book doesn't take that approach. It assumes that you don't know the language, so every line in every program is explained in as much detail as is necessary to pull it all apart, since Rust can be a dense language in some ways. This means single lines can pack a lot of meaning and functionality.

The one thing this book does not assume, though, is that you are coming to programming completely fresh. You will see examples for the programs written in Rust also presented in other programming languages. This may be helpful if you come from another language like C or Python, for instance, but want to learn Rust. Seeing the approach in a language you know before translating it into Rust may be beneficial. If you don't know those other languages, you can skip through those examples and jump to the explanation of how to write a program for the problem under discussion in Rust. You can still compare the other languages to Rust as you are going through so you can better understand Rust and how it is different from other languages.

OBTAINING RUST

Rust is a collection of programs that you will use. While a big part of it is the compiler, that's not the only program that will get installed. First, of course, is the compiler, `rustc`. This program will compile any Rust source code file, but more than that, it will compile complete executables. With some compiler programs, you have to compile source code files individually and then perform a step called *linking,* where you link all the source code files together along with any needed libraries to create the executable. If there is a reference to another source code file you have written as a module, the Rust compiler will compile all the modules and generate an executable without any additional intervention.

In practice, though, you probably won't use the Rust compiler directly. Instead, you'll use the `cargo` program. You'll want to get used to using `cargo` because it not only compiles your source code but also will manage any external dependencies. You will probably have libraries that are not part of the standard library. With languages like C and Python, you'd typically need to go get the library yourself and get it installed. You'd need to make sure it was installed in the right place, and then, in the case of C, you'd probably need to call the compiler in a way that made it clear you wanted to link in the external library so all the external references could get resolved and put into the resulting executable.

Rust is also a newer program, which means there are changes being made to it. You'll generally want to keep up-to-date on the newest Rust compiler. Your third-party libraries may be keeping up with the latest Rust changes, and if you aren't up-to-date, your program won't compile. You'll want the `rustup` utility to help manage your Rust installation.

If you are working on a Linux distribution, you may be inclined to use whatever package manager you have to install Rust. There's a better-than-good chance that your distribution has the Rust language in it. The problem is, once you install using the package manager, you may be held back by the package manager. The latest Rust software may not be available to you. It's easier to just install Rust without the Linux package manager. With operating systems like macOS and Windows, you don't even have a built-in package manager, so installing that way wouldn't be an option anyway.

The best approach is to go to the Rust website (`www.rust-lang.org`). For Unix-like operating systems, including Linux and macOS, there is a command-line string you will probably use to install. Because there is a chance this approach may change, it's best to just go to the website to get the right way. As of the writing of this book, the command used to install Rust on those operating systems follows. If you are on Windows, you can download an installer from the Rust website:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Once you have the Rust toolchain installed, you can keep it updated by using the command `rustup update`. This will always get the latest version of the Rust toolchain and make sure it is installed. You will also need to use a good source code editor. There are several available that will support Rust extensions, including Visual Studio Code, Atom, and Sublime. You should make sure you have installed the Rust extensions, which will help you with syntax highlighting and other features.

GETTING THE SOURCE CODE

As you work your way through this book, you will see primarily complete programs that are explained in context. You can certainly retype the programs from the book, and most are not that long. There is some value in retyping because it helps to ingrain the code and approach to programming used by Rust. However, it can be tedious to stare at a program and try to retype it. You may want to just start with the source code. It's all available on GitHub. GitHub is a source code repository site using the `git` source code management software. It was originally written to be used with the Linux kernel, as previous source code management software was not considered to be feature-rich enough. While there is other software available, `git` is most widely used today because public repositories like GitHub use `git`. To get the source code for this book, you can use the following command:

```
git clone https://github.com/securitykilroy/rust.git
```

If you have a `git` client that you prefer to the command line, you can certainly use it. The command line is going to be the most common approach to grabbing source code from a `git` server.

NOTE *The files are also available at* `www.wiley.com/go/beginningrust`.

WHAT YOU WILL LEARN

The approach in this book is to write complete programs that are useful in some way, even if they are very simple starting points to more interesting programs. The idea is not to try to deconstruct enormous programs, so each chapter will tackle important ideas, but the programs presented may be limited. You will get important building blocks but maybe not large, complex programs. Each chapter will present some essential ideas in Rust and, sometimes, programming in general. Many chapters build on ideas from previous chapters. You can certainly read individual chapters since, in most cases, the program is still explained in detail, not always assuming you have read previous chapters.

The book doesn't exclusively cover the Rust programming language. Programming is about far more than language syntax. There is much more to programming than just how a language is constructed. This is especially true if you ever want to write software on a team—working with an open source project or being employed as a programmer. You need to be aware of how larger programs are constructed and ways to write software in a way that is readable and maintainable, as well as ways to write tests of your software. You can see the topics covered in each chapter here.

Chapter 1

We get started with a partially functional implementation of Conway's Game of Life, a classic computer science program. Along the way, you will learn how to use `cargo` to create a new program with all the files and directories needed for `cargo` to build the program for you. You'll also learn about data types and some initial control structures, as well as creating functions in Rust.

Chapter 2

The reason for making the program in Chapter 1, "Game of Life: The Basics," only partly functional is that the complete program is larger, and there are a lot of concepts to introduce to implement everything. By the end of this chapter, you will have a fully functional program that will implement Conway's Game of Life. You will also learn about the use of a collection data type that is good for dynamically sized collections. You will also learn about performing input/output to interact with the user. One of the most important concepts in Rust is introduced in this chapter, and it will keep recurring in several subsequent chapters. Ownership is foundational to Rust and is part of what makes it a good language for systems programming. Rust is designed to be a safe language, unlike a language like C.

Chapter 3

This chapter works with another essential concept in Rust—the struct. This is a complex data structure, defined entirely by the programmer. It underpins data abstraction in Rust, so it will be covered across multiple chapters in different ways. You'll also be working with writing to files as well as working with JavaScript Object Notation (JSON), a common approach to store and transmit complex data structures in a way that is self-describing. We'll also extend the idea of ownership by talking about lifetimes.

Chapter 4

The struct is an important concept in Rust because it provides a way to abstract data. *Data abstraction* is hiding the data behind a data structure and a set of functionality that acts on the data. This is done using traits in Rust, and this chapter introduces those traits. We'll spend a lot of time in subsequent chapters looking at traits in more detail. We'll also talk about error handling, which is another dense and important topic that will be covered in unfolding detail across several chapters. Additionally, we'll cover another control structure that allows you to make different decisions based on the contents of an identifier. Identifiers in Rust are similar to variables in other languages, although there are some subtle nuances, which is why it's easier to refer to them as identifiers. We'll also look at how to take input from a user.

Chapter 5

This chapter covers concurrent programming, sometimes called parallel programming. This is where a program ends up breaking into multiple, simultaneous execution paths. There are a lot of challenges with concurrent programming, not least of which is the way the different execution paths communicate with one another to keep data and timing synchronized. We'll also look at how to interact with the operating system to get information from the filesystem. And we'll take an initial pass at encryption, although this is not the last time encryption will be covered.

Chapter 6

We'll start on network programming, although this will also be spread across additional chapters. There are a lot of different ways to write programs for network communication because there are so many protocols that are used over networks. We'll look at some additional interactions with the operating system in this chapter as well. This is the first of a pair of chapters that are linked. In this chapter, we implement a network server that requires a client to talk to it. This chapter also talks about different ways to design your program so you'll have thought about all the elements and features the program needs before you start writing it.

Chapter 7

This is the chapter that covers the client that communicates with the server from the previous chapter. We will also cover using encryption to communicate over the network. Additionally, we'll use regular expressions, which can be a powerful pattern-matching system. While they have a lot of other uses, we're going to use regular expressions in this chapter to help us make sure we have the right input from the user.

Chapter 8

This is the first chapter that talks about database communications. This chapter covers the use of relational databases, which are traditional ways to store structured information. If you've seen the use of MySQL, PostgreSQL, Microsoft SQL Server, Oracle, SQLite, or other databases, you've seen relational databases in action. You may be working with a database server or an embedded database. This chapter will cover those two techniques so you will be able to talk to a server or store data in a searchable way in a local file.

Chapter 9

Relational databases have been around for decades; but the way forward is using other database types, since data isn't always so well structured that you know exactly what properties will be associated with it. Additionally, there may be documents involved that need to be dealt with. This chapter covers the use of NoSQL databases, which are databases that use something other than traditional relational techniques to store and retrieve data. This chapter also covers assertions, which are ways to ensure that data is in the state it is expected to be in before being handled by a function. This is a way of protecting the program, allowing it to fail gracefully.

Chapter 10

Many applications are moving to the web. This means you need to be able to write programs that can communicate over web-based technologies, including the HTTP protocol. This chapter will cover not only how to write web client programs but also extracting data from web pages and asynchronous communication, where you may send a request and not wait for the response but still be able to handle the response when it comes back. This chapter also covers how to use style guides to make your programs more consistent and readable.

Chapter 11

Where the last chapter talked about writing web-based clients, this program presents a couple of different ways to write a web server. This is useful if you want to write an application programming interface (API) that can be consumed by clients remotely. This gives Rust the ability to be on the server end of a multitier web application as well as on the client side. Additionally, this chapter will talk about considering offensive and defensive programming practices to make your programs more resilient and more resistant to attack. This includes the idea of design by contract, guaranteeing that a program acts exactly the way it is expected to.

Chapter 12

Rust is considered a systems programming language, so we will investigate how to interact with the system. We'll start by writing programs to extend data structures, including some built-in data structures. We'll also take a look at how to interact with the Windows Registry to store and retrieve information. Finally, we'll introduce functionality to get information about the system, including process listings.

Chapter 13

We're going to take the systems programming idea and talk about an essential aspect of programming that is often overlooked; whether you are writing a system service or something that is user-focused, you should always be generating logs. We'll take a look at how to write to both syslog as well as the Windows Event Log. On top of that, we'll take a look at how to write directly to hardware on a Raspberry Pi using the General Purpose Input Output (GPIO) header on the single-board computer.

Chapter 14

Early in the book, we covered data collections in the form of arrays and vectors. Data collections are such a useful feature, though, that we spend this chapter on different types of data collections, including linked lists, queues, stacks, and binary search trees.

Chapter 15

There are some fun and useful ideas that are left over and covered in this chapter. First, recursion is a common way to tackle programming problems, so we take a look at how to address some problems using recursion. We'll also look at how to use Rust to write machine learning programs using third-party libraries. Finally, we will be writing unit tests in Rust, which are ways to ensure that a function does what it is meant to do. This can also be a way to try to break a function. A library included in Rust makes it easy to write tests, which should be a practice always used when writing programs.

PROVIDING FEEDBACK

We hope that *Beginning Rust Programming* will be of benefit to you and that you create some amazing programs with Rust. We've done our best to eliminate errors, but sometimes they do slip through. If you find an error, please let our publisher know. Visit the book's web page, `www.wiley.com/go/beginningrust`, and click the Errata link to find a form to use to identify the problem.

Thanks for choosing *Beginning Rust Programming*.

1

Game of Life: The Basics

IN THIS CHAPTER, YOU WILL LEARN THE FOLLOWING:

- How to create a new project using Cargo
- How to use variables in Rust
- How to use basic functions in Rust, including returning values and passing parameters
- How basic control mechanisms work

In 1970, British mathematician John Horton Conway devised a game using cellular automata. In October of that year, Martin Gardner wrote about the game in his monthly column Mathematical Games in *Scientific American*. It's a game with simple rules, which can be played on paper, but honestly, it's more fun to write programs that implement the game. We're going to start the dive into Rust by writing a simple implementation of *Conway's Game of Life*. First we'll talk about the rules so that when we get to implementing it, you'll know what you are looking at.

Imagine a two-dimensional space that consists of cells on both the horizontal and vertical axes. Maybe it's just easier to think about graph paper—row upon row and column upon column of little boxes. Each of these little boxes contains, or at least has the potential to contain, a living creature—a single-celled organism living in a single cell. The game is evolutionary, meaning we cycle through one generation after another, determining whether each cell lives or dies based on the rules of the game. Speaking of those rules, they are as follows:

- If a cell is currently alive but it has fewer than two neighbors, it will die because of lack of support.
- If a cell is currently alive and has two or three neighbors, it will survive to the next generation.

- If a cell is currently alive and has more than three neighbors, it dies from overpopulation (lack of resources).
- If a cell is currently dead but has exactly three neighbors, it will come back to life.

To turn this game into code, we need to do a couple of things. First, we need a game grid where all of our little cells are going to live. Second, we need a way to populate the game grid with some living cells. An empty game board won't lead to anything good. Once we have a game board, we can run generations using these rules.

The following is the complete program that will create the game board and also run the checks for whether different cells live or die. Don't worry—you don't have to take it all in at once. We'll go through it step-by-step as we introduce you to Rust.

GAME OF LIFE: THE PROGRAM

The program in this section will create the game board for *Conway's Game of Life* and populate it with an initial generation. This portion of this program will be more than enough to get us started talking about how to begin a Rust program. However, this is not a complete program in the sense that it won't fully implement a useful *Conway's Game of Life*. It's primarily missing the output and generational functions.

```
extern crate rand;
use std::{thread, time};

fn census(_world: [[u8; 75]; 75]) -> u16
{
    let mut count = 0;

    for i in 0..74 {
        for j in 0..74 {
            if _world[i][j] == 1
            {
                count += 1;
            }
        }
    }
    count
}
fn generation(_world: [[u8; 75]; 75]) -> [[u8; 75]; 75]
{
    let mut newworld = [[0u8; 75]; 75];

    for i in 0..74 {
        for j in 0..74 {
            let mut count = 0;
            if i>0 {
                count = count + _world[i-1][j];
            }
```