

APPRENDRE À PROGRAMMER EN RUST

APPRENDRE À PROGRAMMER EN RUST

Guillaume Gomez

DUNOD

NOUS NOUS ENGAGEONS EN FAVEUR DE L'ENVIRONNEMENT :



Nos livres sont imprimés sur des papiers certifiés pour réduire notre impact sur l'environnement.



Le format de nos ouvrages est pensé afin d'optimiser l'utilisation du papier.



Depuis plus de 30 ans, nous imprimons 70 % de nos livres en France et 25 % en Europe et nous mettons tout en œuvre pour augmenter cet engagement auprès des imprimeurs français.



Nous limitons l'utilisation du plastique sur nos ouvrages (film sur les couvertures et les livres).

AVANT-PROPOS

Rust est un langage moderne et puissant qui est de plus en plus populaire auprès des développeurs, notamment en raison de ses performances élevées, de sa sécurité, de sa concurrence et de sa prise en charge des architectures multiplateformes. Cet ouvrage a pour objectif de vous aider à apprendre Rust.

Cependant, je tiens à souligner qu'il est préférable de connaître au moins un autre langage de programmation avant de plonger dans Rust. Ce n'est pas parce que Rust est un langage particulièrement difficile à apprendre, mais plutôt parce que les concepts abordés dans cet ouvrage supposent une certaine familiarité avec la programmation. En d'autres termes, si vous êtes totalement novice en programmation, vous pouvez trouver certains passages de cet ouvrage difficiles à suivre.

Si vous avez déjà une expérience de la programmation dans un autre langage (tel que C, C++, Java, Python, Javascript, Haskell...), vous aurez une longueur d'avance pour apprendre Rust. Vous serez plus en mesure de comprendre les concepts de programmation avancée, et vous pourrez vous concentrer sur les aspects spécifiques de Rust plutôt que de devoir assimiler à la fois les notions de base de la programmation et de Rust.

Néanmoins, même si vous n'avez pas d'expérience préalable en programmation, vous pouvez toujours vous plonger dans cet ouvrage avec enthousiasme et motivation. Vous y trouverez de nombreux exemples de code, d'explications claires et détaillées, ainsi que des astuces et des conseils pour vous aider à avancer. Vous apprendrez non seulement Rust, mais vous améliorerez également vos compétences en programmation en général.

TABLE DES MATIÈRES

Avant-propos	5
Introduction	13
 Les bases de la programmation en Rust	
1 Mise en place des outils	17
L'éditeur de code	17
Les outils de Rust	17
2 Premier programme	18
3 Les variables	19
Variables constantes et mutables	19
Les types	19
Les slices	21
4 Conditions et pattern matching	22
If/else if/else	22
Comparaison de booléens	23
Pattern matching	23
Toujours plus loin	25
5 Les fonctions	27
Créer une fonction	27
Utiliser une fonction	28
Les macros en Rust	29
6 Les expressions	30
Différence entre expressions et déclarations	30
Mise en pratique	31
7 Les boucles	33
While	33
Loop	33
For	35
Énumération	36
Les boucles nommées	37

8	Les enums	38
	Utilisation	38
	Implémenter des méthodes sur une enum	40
9	Les structures	41
	À quoi cela ressemble ?	41
	Déstructuration	43
	Les méthodes	44
	Syntaxe de mise à jour (ou « update syntax »)	46
	Destructeur	47
10	If let/while let	48
	Qu'est-ce que le if let ?	48
	While let	49
11	Gestion des erreurs	51
	Result	51
	Option	52
	Panic!	54
	Possibles améliorations	54
12	Cargo	56
	Commencer un projet	56
	Gérer les dépendances	57
	Publier une crate sur crates.io	60
	Utiliser des bibliothèques externes	62
13	Jeu du plus ou moins	63
	La règle du jeu	63
	La solution	64
	Améliorations	67

Les spécificités de Rust

14	Le formatage des flux	71
	Exemple de print! et println!	71
	Format!	72
	Toujours plus loin	72

15	Les traits	74
	Définition.....	74
	Créer un trait.....	75
	Les supertraits.....	77
	Les derive traits.....	78
	Utilisations de traits.....	79
16	Généricité	80
	La généricité en Rust.....	80
	Where	85
17	Propriété (ou ownership)	87
	Intérêt de l'ownership.....	87
	Clone et Copy.....	88
	Les références.....	89
18	Durée de vie (ou lifetime)	92
	Les durées de vies statiques.....	92
	Les durées de vie temporaires.....	93
	Types avec une référence comme champ.....	94
	Contraintes sur les durées de vie.....	96
19	Déréférencement	97
	Implémentation.....	98
	Auto-déréférencement.....	98
20	Sized et String vs str	100
	Str.....	100
	Le trait Sized	100
	String.....	101
	Vec vs slice	102
21	Les unions	103
	Définition et propriété.....	103
	Mise en pratique	104
	Pattern machting	105
22	Closure	106
	Définition et utilité.....	106
	Fn.....	107
	Fnmult.....	108
	Fnonce.....	108

23	Projet multi-fichiers	110
	Lib.rs	112
	Un_fichier.rs	112
	Module1/mod.rs	113
	Module1/file1.rs	113
	Module1/module2/mod.rs	113
	Module1/module2/file1.rs	113
24	Les macros	114
	Fonctionnement	114
	Les arguments (ou flux de tokens)	115
	Les différents types « d'arguments »	116
	Répétition	116
	Pattern matching encore plus poussé	118
	Scope et exportation d'une macro	119
	Quelques macros utiles	119
	Petite macro mais grande économie de lignes	120
25	Box	122
	Structure récursive	122
	Liste chaînée	123
26	Les itérateurs	125
	Les itérateurs sur/liés à un type	125
	Les générateurs	127
	Conclusion	128

Pour aller plus loin

27	Les macros procédurales (ou proc-macros)	131
	Définition	131
	Function-like macro	132
	Derive macro	133
	Macro attribut	139
28	Utiliser du code compilé en C avec les FFI	141
	Les bases	141
	Interfaçage avec une bibliothèque C	142
	Interfacier les fonctions	143

29	Documentation et rustdoc	146
	Génération de la documentation.....	146
	Ajouter de la documentation.....	147
	Documenter un module.....	148
30	Ajouter des tests	149
	L'attribut #[test].....	149
	L'attribut #[should_panic].....	150
	Mettre les tests dans un dossier à part.....	151
	Écrire des suites de tests.....	151
	Tests dans la documentation?.....	152
	Options de test.....	152
	Cacher des lignes.....	153
31	Rc et RefCell	155
	RefCell.....	155
	Rc.....	158
32	Les threads	160
	Un exemple tout simple.....	160
	Mutex.....	161
	Arc.....	162
	Les channels.....	163
	Utilisation détournée.....	164
	Empoisonnement de Mutex.....	165
	Autres façons d'utiliser les threads.....	165
33	Le réseau	166
	Le client.....	166
	Le serveur.....	167
	Multi-client.....	168
	Gérer la perte de connexion.....	169
	Exemple d'échange de message entre un serveur et un client.....	170
34	Codes annexes	174
	Écrire des nombres différemment.....	174
	Toujours plus de parenthèses.....	174
	Toujours plus vers le fonctionnel avec le slice pattern.....	175
	Une autre façon de faire des boucles infinies.....	175
	Calculer des factorielles avec un itérateur.....	175

INTRODUCTION

Le développement de Rust a été initié par Graydon Hoare en 2006, notamment dans le but de résoudre les failles de sécurité dans Firefox sans que cela impacte négativement les performances. Sa première version stable, la 1.0, est sortie le 15 mai 2015. En août 2020, Mozilla a arrêté de soutenir le développement du langage, conduisant à la création de la fondation Rust le 8 février 2021, le but de cette fondation n'étant pas de diriger le développement du langage mais de le soutenir financièrement.

Depuis sa première version stable, Rust a été adopté par toutes les plus grosses entreprises de l'informatique telle que Google qui s'en sert pour Android ainsi que son cloud, Microsoft qui s'en sert dans Windows, Amazon, Facebook, Discord, Huawei, Dropbox, Mozilla...

Du côté des projets open source, c'est devenu le troisième langage de programmation utilisé dans le développement du kernel Linux après le C et l'assembleur en 2022. Le projet GNOME a de plus en plus de projets internes utilisant Rust et a déjà réécrit certaines de ses bibliothèques telles que libsvg.

Rust est un langage de programmation système, compilé et multi-paradigme. C'est un croisement entre langage impératif (C), objet (C++), fonctionnel (Ocaml) et concurrent (Erlang). Il s'inspire des recherches en théorie des langages de ces dernières années afin d'atteindre trois objectifs : rapidité, sécurité (en gestion mémoire notamment) et concurrent (partage des données sécurisé entre tâches). Il est notamment utilisé pour de la programmation système, écrire des serveurs webs, faire des applications en ligne de commandes, des applications graphiques et des jeux vidéo.

Ses points forts sont :

- ✓ la gestion de « propriété » (**ownership**) des variables ;
- ✓ la gestion de la mémoire ;
- ✓ le typage statique ;
- ✓ l'inférence de type ;
- ✓ le filtrage par motif (**pattern matching**) ;
- ✓ la généricité.

En premier lieu, nous aborderons la syntaxe de Rust ainsi que ses types et les outils à utiliser pour se faciliter la vie.

Ensuite, nous verrons plus en détail les spécificités de ce langage et notamment les systèmes de « propriété » et de « durée de vie ».

Enfin, nous verrons les bases de concepts et fonctionnalités avancés telles que l'utilisation de bibliothèques écrites en C directement dans votre code Rust ou comment écrire des macros procédurales (proc-macro).

Avant de poursuivre, vous trouverez ci-dessous les principaux sites internet traitant de Rust, notamment son site web officiel, sa documentation ainsi que son dépôt sur github :

- ✓ le site internet : rust-lang.org
- ✓ la documentation officielle : doc.rust-lang.org/stable/std/
- ✓ le dépôt Github du langage : github.com/rust-lang/rust

Il est maintenant temps de commencer.