

Spis treści

PRZEDMOWA	17
WSTĘP	19
PODZIĘKOWANIA	20
WPROWADZENIE	21
1	
PIERWSZE KROKI	27
Instalacja	27
Instalowanie narzędzia rustup w systemie Linux lub macOS	28
Instalowanie narzędzia rustup w systemie Windows	29
Rozwiązywanie problemów	29
Aktualizowanie i odinstalowywanie	30
Lokalna dokumentacja	30
Witaj, świecie!	30
Tworzenie katalogu projektu	31
Pisanie i uruchamianie programu Rusta	31
Anatomia programu Rusta	32
Kompilowanie i uruchamianie to oddzielne kroki	33
Witaj, Cargo!	34
Tworzenie projektu za pomocą Cargo	35
Kompilowanie i uruchamianie projektu Cargo	37
Kompilowanie w celu wydania projektu	38
Cargo jako konwencja	39
Podsumowanie	39
2	
PROGRAMOWANIE GRY W ZGADYWANIE	40
Konfigurowanie nowego projektu	41
Przetwarzanie zgadywania	42
Przechowywanie wartości za pomocą zmiennych	43
Odbieranie danych wprowadzanych przez użytkownika	44

Obsługa ewentualnego niepowodzenia za pomocą typu Result	44
Wypisywanie wartości za pomocą symboli zastępczych println	46
Testowanie pierwszej części	46
Generowanie sekretnej liczby	47
Korzystanie ze skrzynki w celu uzyskania większej funkcjonalności	47
Generowanie liczby losowej	50
Porównywanie próby odgadnięcia z sekretną liczbą	51
Umożliwianie wielokrotnego zgadywania dzięki zastosowaniu pętli	55
Zakończenie gry po odgadnięciu prawidłowej liczby	56
Obsługa nieprawidłowych danych wejściowych	56
Podsumowanie	59

3 POWSZECHNE KONCEPCJE PROGRAMOWANIA60

Zmienne i mutowalność	61
Stałe	63
Przysłanianie	63
Typy danych	65
Typy skalarne	66
Typy złożone	70
Funkcje	74
Parametry	75
Instrukcje i wyrażenia	76
Funkcje z wartościami zwracanymi	78
Komentarze	80
Przepływ sterowania	81
Wyrażenia if	81
Powtarzanie za pomocą pętli	85
Podsumowanie	90

4 WŁASNOŚĆ91

Czym jest własność?	91
Reguły własności	93
Zakres zmiennych	93
Typ String	94
Pamięć i alokacja	95
Własność i funkcje	101
Wartości zwracane i zakres	101
Referencje i pożyczanie	103
Referencje mutowalne	105
Referencje wiszące	108
Reguły referencji	109

Typ wycinka	110
Wycinki łańcucha znaków	111
Inne wycinki	116
Podsumowanie	116

5 WYKORZYSTANIE STRUKTUR DO ORGANIZOWANIA POWIĄZANYCH ZE SOBĄ DANYCH

Definiowanie struktur i tworzenie ich instancji	117
Użycie skrótu inicjowania pól	119
Tworzenie instancji z innych instancji za pomocą składni aktualizacji struktury	120
Tworzenie różnych typów za pomocą struktur krotkowych bez nazwanych pól	121
Struktury jednostkowe bez żadnych pól	122
Przykładowy program wykorzystujący struktury	124
Refaktoryzacja z wykorzystaniem krotek	125
Refaktoryzacja z wykorzystaniem struktur: dodanie znaczenia	126
Dodanie użytecznej funkcjonalności za pomocą cech pochodnych	127
Składnia metod	130
Definiowanie metod	130
Metody z większą liczbą parametrów	133
Funkcje powiązane	134
Wiele bloków impl	135
Podsumowanie	136

6 TYPY WYLICZENIOWE I DOPASOWYWANIE WZORCÓW

Definiowanie typu wyliczeniowego	138
Wartości wyliczenia	138
Enumeracja Option i jej zalety w porównaniu z wartościami zerowymi	142
Konstrukcja match przepływu sterowania	145
Wzorce, które powodują powiązanie z wartościami	146
Dopasowywania za pomocą Option<T>	147
Dopasowywania są wyczerpujące	149
Wzorce catch-all i symbol zastępczy _	149
Związły przepływ sterowania z wykorzystaniem składni if let	151
Podsumowanie	152

ZARZĄDZANIE ROZWIJAJĄCYMI SIĘ PROJEKTAMI ZA POMOCĄ PAKIETÓW, SKRZYNEK I MODUŁÓW	154
Pakiety i skrzynki	155
Definiowanie modułów w celu kontrolowania zakresu i prywatności	159
Ścieżki odwoływania się do elementów w drzewie modułów	160
Udostępnianie ścieżek za pomocą słowa kluczowego <code>pub</code>	163
Rozpoczynanie ścieżek względnych od słowa kluczowego <code>super</code>	166
Upublicznianie struktur i wyliczeń	167
Wprowadzanie ścieżek do zakresu za pomocą słowa kluczowego <code>use</code>	168
Tworzenie idiomatycznych ścieżek <code>use</code>	170
Wprowadzanie nowych nazw za pomocą słowa kluczowego <code>as</code>	171
Ponowne eksportowanie nazw za pomocą <code>pub use</code>	172
Korzystanie z pakietów zewnętrznych	173
Skracanie długich list instrukcji <code>use</code> za pomocą zagnieżdżonych ścieżek	174
Operator <code>glob</code>	175
Rozdzielanie modułów na różne pliki	175
Podsumowanie	177

POWSZECHNIE UŻYWANE KOLEKCJE	179
Przechowywanie list wartości za pomocą wektorów	180
Tworzenie nowego wektora	180
Aktualizowanie wektora	181
Odczytywanie elementów wektorów	181
Iterowanie przez wartości w wektorze	183
Przechowywanie wielu typów za pomocą wyliczenia	184
Porzucenie wektora powoduje porzucenie jego elementów	185
Wykorzystywanie typu <code>String</code> do przechowywania tekstu zakodowanego w UTF-8	185
Czym jest <code>String</code> ?	186
Tworzenie nowej instancji <code>String</code>	186
Aktualizowanie instancji <code>String</code>	187
Indeksowanie wartości <code>String</code>	190
Tworzenie wycinków wartości <code>String</code>	192
Metody iterowania przez wartości <code>String</code>	192
Typy <code>String</code> nie są takie proste	193
Przechowywanie kluczy z powiązаныmi wartościami w mapach mieszających	193
Tworzenie nowej instancji <code>HashMap</code>	194
Uzyskiwanie dostępu do wartości z mapy mieszającej	194
Mapy mieszające i własność	195
Aktualizowanie instancji <code>HashMap</code>	196
Funkcje mieszające	198
Podsumowanie	198

OBŚŁUGA BŁĘDÓW	200
Obsługa błędów nieodwracalnych za pomocą wywołania panic!	201
Obsługa błędów odwracalnych z wykorzystaniem typu Result	204
Dopasowywanie na podstawie różnych błędów	206
Propagacja błędów	209
Panikować czy nie panikować?	215
Przykłady, kod prototypów i testy	216
Przypadki, w których mamy więcej informacji niż kompilator	216
Wskazówki dotyczące obsługi błędów	217
Tworzenie typów niestandardowych do celów walidacji	218
Podsumowanie	220
 10	
TYPY SPARAMETRYZOWANE, CECZY I CZASY ŻYCIA	221
Redukowanie duplikacji przez wyodrębnianie funkcji	222
Sparametryzowane typy danych	225
Typy sparametryzowane w definicjach funkcji	225
Typy sparametryzowane w definicjach struktur	227
Typy sparametryzowane w definicjach wyliczeń	229
Typy sparametryzowane w definicjach metod	230
Wydajność kodu wykorzystującego typy sparametryzowane	232
Cechy — definiowanie współdzielonego zachowania	233
Definiowanie cechy	233
Implementowanie cechy w typie	234
Implementacje domyślne	236
Cechy jako parametry	238
Zwracanie typów, które implementują cechy	240
Używanie wiązań cech do warunkowego implementowania metod	241
Walidacja referencji za pomocą czasów życia	243
Używanie czasów życia do zapobiegania powstawaniu wiszących referencji	243
Kontrola pożyczania	244
Sparametryzowane czasy życia w funkcjach	245
Składnia adnotacji czasów życia	247
Adnotacje czasów życia w sygnaturach funkcji	247
Myślenie w kategoriach czasów życia	250
Adnotacje czasów życia w definicjach struktur	251
Elizja czasów życia	252
Adnotacje czasów życia w definicjach metod	254
Statyczny czas życia	255
Parametry typów sparametryzowanych, wiązania cech i czasy życia razem wzięte	256
Podsumowanie	256

11	
PISANIE ZAUTOMATYZOWANYCH TESTÓW	258
Jak pisać testy?	259
Anatomia funkcji testowej	259
Sprawdzenie wyników za pomocą makra assert!	263
Testowanie równości za pomocą makr assert_eq! i assert_ne!	266
Dodawanie niestandardowych komunikatów o błędach	268
Sprawdzanie paniki za pomocą atrybutu should_panic	270
Wykorzystanie w testach typu Result<T, E>	273
Kontrolowanie sposobu uruchamiania testów	274
Uruchamianie testów równoległe lub po kolei	274
Wyświetlanie danych wyjściowych funkcji	275
Uruchamianie podzbioru testów według nazwy	277
Ignorowanie niektórych testów i uruchamianie ich na specjalne żądanie	279
Organizowanie testów	280
Testy jednostkowe	280
Testy integracyjne	282
Podsumowanie	286
12	
PROJEKT Z WYKORZYSTANIEM OPERACJI WE-WY — BUDOWANIE PROGRAMU WIERZSA POLECEŃ	287
Przyjmowanie argumentów wiersza poleceń	288
Odczytywanie wartości argumentów	289
Zapisywanie wartości argumentów w zmiennych	290
Odczyt pliku	291
Refaktoryzacja w celu poprawy modułowości i obsługi błędów	293
Separacja zagadnień projektów binarnych	293
Naprawianie obsługi błędów	297
Wyodrębnianie logiki z funkcji main	301
Wydzielanie kodu do skrzynki biblioteki	303
Rozwijanie funkcjonalności biblioteki za pomocą programowania opartego na testach	305
Pisanie testu kończącego się niepowodzeniem	305
Pisanie kodu w celu zapewnienia pozytywnego wyniku testu	308
Praca ze zmiennymi środowiskowymi	311
Pisanie niepomysłnego testu dla funkcji search nieuwzględniającej wielkości liter ...	311
Implementacja funkcji search_case_insensitive	313
Wypisywanie komunikatów o błędach do standardowego strumienia błędów, a nie do standardowego strumienia wyjściowego	317
Sprawdzanie, gdzie są zapisywane błędy	317
Wypisywanie błędów do standardowego strumienia błędów	318
Podsumowanie	319

Cechy języka funkcyjnego — iteratory i domknięcia	320
Domknięcia — funkcje anonimowe, które przechwytyują swoje środowisko	321
Przechwytywanie środowiska za pomocą domknięć	321
Inferowanie i adnotowanie typu domknięcia	323
Przechwytywanie referencji lub przenoszenie własności	325
Przenoszenie przechwyconych wartości z domknięć i cechy Fn	327
Przetwarzanie serii elementów za pomocą iteratorów	331
Cecha Iterator i metoda next	332
Metody, które konsumują iterator	333
Metody, które wytwarzają inne iteratory	334
Używanie domknięć, które przechwytyują swoje środowisko	335
Ulepszenie projektu operacji we-wy	337
Usuwanie clone przy użyciu iteratora	337
Zwiększanie czytelności kodu dzięki adapterom iteratora	340
Wybór pomiędzy pętlami i iteratorami	341
Porównanie wydajności pętli i iteratorów	341
Podsumowanie	343
 14	
NARZĘDZIE CARGO I REJESTR CRATES.IO	344
Dostosowywanie kompilacji za pomocą profili wydania	345
Publikowanie skrzynki w rejestrze Crates.io	346
Tworzenie przydatnych komentarzy dokumentujących	346
Eksportowanie wygodnego publicznego API za pomocą pub use	350
Zakładanie konta w rejestrze Crates.io	353
Dodawanie metadanych do nowej skrzynki	354
Publikowanie w rejestrze Crates.io	355
Publikowanie nowej wersji istniejącej skrzynki	356
Dezaktualizowanie wersji z Crates.io za pomocą polecenia cargo yank	356
Obszary robocze Cargo	357
Tworzenie obszaru roboczego	357
Tworzenie drugiego pakietu w obszarze roboczym	358
Instalowanie plików binarnych za pomocą polecenia cargo install	363
Rozszerzanie Cargo przy użyciu niestandardowych poleceń	364
Podsumowanie	364
 15	
INTELGENTNE WSKAŹNIKI	365
Używanie Box<T> do wskazywania danych na stercie	366
Używanie Box<T> do przechowywania danych na stercie	367
Zastosowanie boksów do używania typów rekurencyjnych	367

Wykorzystanie cechy Deref do traktowania inteligentnych wskaźników	
jak regularnych referencji	372
Podążanie za wskaźnikiem do wartości	372
Używanie Box<T> jak referencji	373
Definiowanie własnego inteligentnego wskaźnika	374
Implementacja cechy Deref	375
Domyślne wymuszenie wytuskania w funkcjach i metodach	376
Jak wymuszenie wytuskania współdziela z mutowalnością?	378
Wykorzystanie cechy Drop do uruchamiania określonego kodu przy czyszczeniu	378
Rc<T> — inteligentny wskaźnik zliczania referencji	382
Używanie Rc<T> do udostępniania danych	382
Klonowanie Rc<T> zwiększa liczbę referencji	385
RefCell<T> i wzorzec mutowalności wewnętrznej	386
Egzekwowanie reguł pożyczania w czasie wykonywania za pomocą RefCell<T>	386
Mutowalność wewnętrzna — mutowalne pożyczanie niemutowalnej wartości	388
Wykorzystanie typów Rc<T> i RefCell<T>, aby mutowalne dane mogły mieć wielu właścicieli	394
Cykle referencji mogą powodować wycieki pamięci	395
Tworzenie cyklu referencji	396
Zapobieganie cyklom referencji za pomocą typu Weak<T>	399
Podsumowanie	404

16

NIEUSTRASZONA WSPÓLBIEŻNOŚĆ	405
Używanie wątków do jednoczesnego uruchomienia kodu	406
Tworzenie nowego wątku za pomocą funkcji spawn	407
Wykorzystanie uchwytów metody join w celu zaczekania na zakończenie wszystkich wątków	408
Używanie domknięć move z wątkami	410
Używanie przekazywania komunikatów do przesyłania danych między wątkami	413
Kanały i przeniesienie własności	416
Wysyłanie wielu wartości i obserwowanie czekającego odbiornika	417
Tworzenie wielu producentów przez klonowanie nadajnika	418
Współbieżność stanu współdzielonego	420
Używanie muteksów w celu umożliwienia dostępu do danych z jednego wątku na raz	420
Podobieństwa między typami RefCell<T> i Rc<T> a Mutex<T> i Arc<T>	426
Rozszerzanie funkcjonalności współbieżności za pomocą cech Send i Sync	426
Umożliwienie przenoszenia własności między wątkami za pomocą cechy Send	427
Umożliwianie dostępu z wielu wątków za pomocą cechy Sync	427
Ręczne implementowanie cech Send i Sync jest niezabezpieczone	428
Podsumowanie	428

17	
FUNKCJONALNOŚCI PROGRAMOWANIA OBIEKTOWEGO	429
Cechy języków obiektowych	430
Obiekty zawierają dane i zachowania	430
Hermetyzacja, która ukrywa szczegóły implementacji	430
Dziedziczenie jako system typów i współdzielenie kodu	432
Używanie obiektów cech, które umożliwiają stosowanie wartości różnych typów	433
Definiowanie cech typowego zachowania	434
Implementowanie cechy	436
Obiekty cech wykonują dynamiczną dyspozycję	439
Implementacja obiektowego wzorca projektowego	439
Definiowanie struktury Post i tworzenie nowej instancji w stanie Draft	441
Przechowywanie tekstu treści wpisu	442
Upewnianie się, że treść wersji roboczej wpisu jest pusta	442
Żądanie korekty zmienia stan wpisu	443
Dodanie metody approve w celu zmiany zachowania metody content	445
Kompromisy wzorca stanu	447
Podsumowanie	452
18	
WZORCE I DOPASOWYWANIE	453
Wszystkie miejsca, w których można używać wzorców	454
Ramiona wyrażenia match	454
Wyrażenia warunkowe if let	455
Pętle warunkowe while let	456
Pętle for	457
Instrukcje let	457
Parametry funkcji	458
Odrzucalność — czy dopasowywanie wzorca może się nie powieść?	459
Składnia wzorców	461
Dopasowywanie literałów	462
Dopasowywanie zmiennych nazwanych	462
Wiele wzorców	463
Dopasowywanie przedziałów wartości za pomocą ..=	464
Destrukturyzacja w celu rozkładania wartości na poszczególne elementy	464
Ignorowanie wartości we wzorcu	469
Dodatkowe wyrażenia warunkowe ze strażnikami dopasowywania	473
Wiązanie za pomocą operatora @	476
Podsumowanie	477

FUNKCJONALNOŚCI ZAAWANSOWANE	478
Niezabezpieczony Rust	479
Niebezpieczne supermoce	479
Wyłuskiwanie surowego wskaźnika	480
Wywoływanie niezabezpieczonej funkcji lub metody	482
Uzyskiwanie dostępu do mutowalnej zmiennej statycznej lub modyfikowanie jej	487
Implementowanie niezabezpieczonej cechy	488
Uzyskiwanie dostępu do pól typu unii	489
Kiedy korzystać z niezabezpieczonego kodu?	489
Cechy zaawansowane	489
Typy powiązane	489
Domyślne parametry typu sparametryzowanego i przeciążanie operatora	491
Rozróżnianie metod o tej samej nazwie	493
Korzystanie z supercech	497
Używanie wzorca newtype do implementowania cech zewnętrznych	499
Typy zaawansowane	500
Używanie wzorca newtype	
dla zapewnienia bezpieczeństwa typów i warstwy abstrakcji	500
Tworzenie synonimów typów za pomocą aliasów typów	501
Typ never, który nigdy nie zwraca wartości	503
Typy o dynamicznie ustalanych rozmiarach i cecha Sized	505
Zaawansowane funkcje i domknięcia	507
Wskaźniki funkcji	507
Zwracanie domknięć	509
Makra	510
Różnica między makrami i funkcjami	510
Makra deklaratywne z konstrukcją macro_rules!	
dla metaprogramowania ogólnego	511
Makra proceduralne do generowania kodu z atrybutów	513
Jak napisać niestandardowe makro derive?	514
Makra atrybutowe	519
Makra funkcyjne	519
Podsumowanie	520
20	
PROJEKT KOŃCOWY — BUDOWA WIELOWĄTKOWEGO SERWERA WWW	521
Tworzenie jednowątkowego serwera WWW	522
Następowanie połączenia TCP	523
Odczytywanie żądania	525
Analiza żądania HTTP	527
Pisanie odpowiedzi	527
Zwracanie prawdziwego HTML-a	529

Walidacja żądania i selektywne odpowiadanie	530
Odrobina refaktoryzacji	532
Przekształcenie serwera jednowątkowego w wielowątkowy	533
Symulowanie powolnego żądania	533
Zwiększenie przepustowości przy użyciu puli wątków	534
Eleganckie zamykanie i czyszczenie	551
Implementacja cechy Drop w typie ThreadPool	551
Sygnalizowanie wątkom, aby przestały nasłuchiwać zadań	554
Podsumowanie	557
A	
SŁOWA KLUCZOWE	559
Obecnie używane słowa kluczowe	559
Słowa kluczowe zarezerwowane do wykorzystania w przyszłości	561
Surowe identyfikatory	561
B	
OPERATORY I SYMBOLE	563
Operatory	563
Symbole niezwiązane z operatorami	566
C	
CECHY POCHODNE (ZAPOŻYCZONE)	571
Cecha Debug do uzyskiwania programistycznych danych wyjściowych	572
Cechy PartialEq i Eq do porównania równości	572
Cechy PartialOrd i Ord do porównywania kolejności	573
Cechy Clone i Copy do duplikowania wartości	573
Cecha Hash do mapowania wartości na wartość o stałym rozmiarze	574
Cecha Default dla wartości domyślnych	575
D	
PRZYDATNE NARZĘDZIA PROGRAMISTYCZNE	577
Automatyczne formatowanie za pomocą narzędzia rustfmt	577
Naprawianie kodu za pomocą narzędzia rustfix	578
Lintery narzędzia Clippy	579
Integracja z IDE przy użyciu narzędzia rust-analyzer	580
E	
EDYCJE	581