

UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET



Jovan Dmitrović

**MASTER IZ MATEMATIKE ILI
RAČUNARSTVA ČIJI JE NASLOV JAKO
DUGAČAK**

master rad

Beograd, 2021.

Mentor:

dr Mika MIKIĆ, redovan profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Ana ANIĆ, vanredni profesor
University of Disneyland, Nedodija

dr Laza LAZIĆ, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Mami, tati i dedi

Naslov master rada: Master iz matematike ili računarstva čiji je naslov jako dugačak

Rezime: Apstrakt ide ovde

Ključne reči: programiranje, programski jezici

Sadržaj

1	Uvod	1
2	Programski jezik <i>Rust</i>	2
2.1	Istorijat	2
2.2	Instalacija	3
2.3	Korišćenje sistema <i>Cargo</i>	4
2.4	Osnovne karakteristike	5
3	Razrada	9
4	Zaključak	10
	Literatura	11

Glava 1

Uvod

Glava 2

Programski jezik *Rust*

Rust je statički tipiziran jezik koji ima podršku za više programskih paradigmi, fokusiran na bezbednost i performanse. Od svog nastanka, ovaj jezik je dobio veliku pažnju u svetu programiranja, čemu svedoči i činjenica da je *Rust* bio proglašen za „omiljeni programski jezik” već petu godinu za redom u anketi koju je sproveo popularni veb-sajt *Stack Overflow*.

Danas se *Rust* koristi na sve većem broju ozbiljnih projekata, na primer:

- AWS servisima firme Amazon, poput *Lambda*, *EC2* i *Cloudfront*,
- U okviru operativnog sistema kompanije Gugl (engl. *Google*) *ChromeOS*,
- Određenim komponentama Majkrosoftove platforme *Azure*, uključujući i IoT sigurnosni servis *edgelet*,
- Registru *JavaScript* paketa *npm*, kod procedura koje prouzrokuju veliko CPU opterećenje,
- Mozilinom veb-brauzeru Fajerfoks (engl. *Firefox*).

2.1 Istorijat

Programski jezik *Rust* je dizajnirao Grejdon Hor (engl. *Graydon Hoare*) koji je, u to vreme, bio zaposlen u kompaniji Mozila (engl. *Mozilla*). Hor je rad na ovom jeziku započeo 2006. godine kao svoj lični projekat, na kojem je samostalno radio naredne tri godine. Kada je *Rust* počeo da zreli, tada se u projekat uključila i sama Mozila, koja i dan-danas sponzorise njegov razvoj. Pored zaposlenih Mozile,

pošto je u pitanju programski jezik otvorenog koda, svoj doprinos je dalo i preko 5000 dobrovoljaca.

Pre nego što se Mozilla priključila projektu, *Rust* je izgledao dosta drugačije nego danas. U ovoj, početnoj, fazi, *Rust* je bio čist jezik, tj. nije imao bočne efekte; takođe, postojala je i analiza stanja tipa (engl. *typestate analysis*), koja je omogućavala proveru operacija koje se mogu izvoditi nad specifičnim tipom podataka pri kompajliranju. Za razliku od ove dve osobine, neka dizajnerska rešenja su ostala do danas, kao što je imutabilnost i kontrola pristupa memoriji.

Nedugo nakon priključivanja Mozile 2010. godine, Grejdon Hor napušta projekat 2012. godine, što dovodi do određenih novina; *Rust* dobija svoj menadžer paketa *Cargo*, kao i sakupljač otpadaka. Proces *RFD*, inspirisan procesom *PEP* programskog jezika *Python*, se osniva 2014. godine u svrhu strogog kontrolisanja novina u samom jeziku.

Rust 1.0, prva „stabilna” verzija *Rust*-a je distribuirana 2015. godine. Od tad, *Rust* ima politiku izbacivanja novih verzija gde se one distribuiraju svakih 6 nedelja, što je „agresivniji” pristup u odnosu na većinu programskih jezika gde je taj period minimalno godinu dana. Ovom odlukom se stavlja akcenat na stabilnost jezika time što će svaka nova verzija biti slična svom prethodniku, dok se kod jezika sa dugim periodom između verzija očekuju velike promene, što može da šteti kompatibilnosti.

2.2 Instalacija

Ukoliko se zvaničan veb-sajt programskog jezika *Rust* poseti na klijentu koji ima *Windows* operativni sistem, biće ponuđeni instalacioni fajlovi za 32-bitne i 64-bitne sisteme.

Na *GNU/Linux* operativnim sistemima potrebno je uneti sledeću komandu u terminal:

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

Potvrdu da li se instalacija izvršila uspešno može se dobiti komandom:

```
rustc --version
```


Pored samog *Rust* kompajlera, u instalaciju su uključeni i *Cargo* i *rustup*; *rustup* daje mogućnost dobavljanja nove verzije *Rust*-a sa veba, kao i mogućnost deinstalacije komandama:

```
rustup update
rustup self uninstall
```

2.3 Korišćenje sistema *Cargo*

Pored toga što je menadžer paketa, *Cargo* vrši i automatizaciju kompajliranja. Kreiranje novog projekta uz pomoć ovog sistema izvršava se komandom:

```
cargo new novi_projekat
```

Komandom iznad se pravi novi direktorijum `novi_projekat` koji sadrži fajl `Cargo.toml` i direktorijum `src` u kome postoji fajl `main.rs`. Takođe, sa novim direktorijumom se inicijalizuje i novi *Git* repozitorijum.

Generisani `Cargo.toml` fajl izgleda ovako:

```
1 [package]
2 name = "novi_projekat"
3 version = "0.1.0"
4 authors = ["jovan <jdmitrovic@gmail.com>"]
5 edition = "2018"
6
7 [dependencies]
```

U prvoj liniji koda, `[package]` označava sekciju koja opisuje paket koji je napravljen; informacije koje se ovde nalaze su dobijene iz varijabli okruženja. Posle oznake `[dependencies]` se popisuju svi paketi koji su neophodni za rad sa novim paketom, tako da ih *Cargo* može dopremiti.

Po osnovnim podešavanjima, u `main.rs` fajlu se nalazi *Hello World* program.

```
1 fn main() {
2     println!("Hello, world!");
3 }
```

Cargo takođe može kompajlirati projekat komandom `cargo build` ili ga kompajlirati i pokrenuti sa `cargo run`. Korišćenjem komande `cargo check` može se proveriti da li se kod kompajlira, bez generisanja izvršnog fajla, što je korisno jer

je ova opcija efikasnija od korišćenja pomenute `build` komande. Kompajliranjem projekta se pravi nova putanja `target/debug`, gde će se generisati izvršni fajlovi.

2.4 Osnovne karakteristike

Imutabilnost

U programskom jeziku *Rust*, kada se nova promenljiva definiše ključnom rečju `let`, podrazumevano ponašanje je da je ta promenljiva imutabilna, tj. ona se ne može menjati. Razlog ovakvog ponašanja leži u tome što *Rust* teži tome da kompajler može prepoznati eventualne greške u kodu, koje bi se teško mogle uočiti ukoliko bi se one dešavale tokom izvršavanja programa. I pored eventualnih problema, mutabilne promenljive se mogu definisati sa `let mut`.

Moguće je definisati i konstante ključnom rečju `const`. Konstante se razlikuju od imutabilnih promenljivih po tome što se mogu definisati u bilo kom opsegu, uključujući i globalni, i po tome što konstante samo mogu imati vrednost konstantnog izraza, ali ne i vrednost izvršavanja funkcije.

Još jedna od opcija je i tzv. sakrivanje (engl. *shadowing*). Sakrivanje je ponovno definisanje promenljivih. Za razliku od korišćenja ključne reči `mut`, prilikom sakrivanja je moguće promeniti tip promenljive koja se ponovo definiše.

Tipovi

Programski jezik *Rust* je statički tipiziran jezik, ali ne zaheva pisanje tipa uz svaku promenljivu, osim ako je to neophodno; primer je korišćenje funkcije `parse`:

```
1 fn main() {
2     let num = "10".parse().expect("Nije unet broj!");
3     println!("Unet broj je: {}", num);
4 }
```

Ukoliko se pokuša kompajliranje ovog koda, Cargo će pokazati grešku:

```
error[E0282]: type annotations needed
--> src/main.rs:2:9
|
2 |     let num = "10".parse().expect("Nije unet broj!");
|           ^^^ consider giving `num` a type
```

Ova greška se javlja zbog toga što funkcija `parse` prima generičke parametre, te je kompajleru neophodna informacija kojeg je tipa promenljiva `num`.

Rust sadrži četiri vrste prostih tipova: cele brojeve, brojeve zapisane sa pokretnim zarezom, karaktere i Bulove konstante *true* i *false*. Celi brojevi mogu biti označeni ili neoznačeni; označeni brojevi su predstavljeni tipovima `i8`, `i16`, `i32`, `i64` i `i128`, gde broj posle karaktera `i` predstavlja veličinu tipa u bitovima. Neoznačeni brojevi su predstavljeni analogno označenim, s tim da oni počinju karakterom `u`. Postoje i tipovi `isize` i `usize` čija veličina zavisi od arhitekture. Analogon tipovima `float` i `double` iz programskog jezika *C* su tipovi `f32` i `f64`. Tip `char` je veličine 4 bajta, gde su karakteri predstavljeni *Unicode* vrednostima.

Osnovni složeni tipovi u programskom jeziku *Rust* su torke i nizovi. Torke se predstavljaju na sledeći način:

```
1 fn main() {  
2     let koordinate: (i32, f32, i32) = (1, 2.0, 3);  
3     let (x, y, z) = koordinate;  
4  
5     println!("x: {}, y: {}", x, koordinate.1);  
6 }
```

Torke mogu sadržati vrednosti različitog tipa, a može im se pristupiti ili korišćenjem novih promenljivih, ili korišćenjem `.` i sintakse, čime se pristupa elementu na *i*-toj poziciji, gde se elementi torke broje počevši od 0.

Nizovi predstavljaju kolekciju vrednosti istog tipa i, poput torki, fiksne su veličine. Obeležavaju se sa uglastim zagradama, unutra kojih su elementi razdvojeni zarezima.

Funkcije

Funkcije se definišu na sledeći način:

```
1 fn zbir_kvadrata(x: i32, y: i32) -> i32 {  
2     x*x + y*y  
3 }
```

Pored korišćenja ključne reči `return`, funkcija će vratiti vredost poslednje naredbe koja se ne završava delimiterom.

Kontrola toka

Rust podržava klasične načine kontrole toka sa `if`, `while` i `for` komandama; postoji par dodatnih mogućosti koje *Rust* dovoljava:

- `if` se može koristiti u `let` naredbama; ali tipovi moraju biti unifikabilni,
- I `while` i `for` se mogu naći u okviru `let` naredbe, s tim da se u telu petlje mora naći `break` komanda, uz koju se dopisuje povratna vrednosti, analogno korišćenju `return` komande u funkcijama,
- Ukoliko je potrebno napisati beskonačnu petlju, nije neophodno koristiti `while` naredbu sa uvek netačnim uslovom, već se može koristiti ključna reč `loop`,
- Iteriranje kroz kolekciju pomoću `for` petlje se može vršiti ili preko indeksa trenutnog elementa, ili pomoću `iter` funkcije; za iteriranje od kraja kolekcije, na `iter` se nadovezuje funkcija `rev` kao u listingu 2.1.

```
1 fn main() {  
2     let a = [1, 2, 3];  
3  
4     for elem in a.iter().rev() {  
5         println!("{}", elem);  
6     }  
7 }
```

Listing 2.1: Korišćenje `iter` i `rev` funkcija

Vlasništvo

Koncept **vlasništva** je novina u odnosu na druge programske jezike; ova osobina dozvoljava programskom jeziku *Rust* da funkcioniše bez sakupljača otpadaka. Za razliku od sakupljača otpadaka, sistem vlasništva ne utiče ni na koji način na izvršavanje programa, jer se postupanje prema skupu pravila vlasništva proverava prilikom kompajliranja.

Sistem vlasništva se može svesti na tri pravila:

1. Svaka vrednost u programskom jeziku *Rust* ima promenljivu koja je poseduje.

2. U jednom trenutku, za svaku vrednost, postoji tačno jedan vlasnik.
3. Kada promenljiva završi svoj životni vek, tada se vrednost koju ta promenljiva poseduje automatski briše iz memorije.

Glava 3

Razrada

Glava 4

Zaključak

Literatura

- [1] Yuri Gurevich and Saharon Shelah. Expected computation time for Hamiltonian path problem. *SIAM Journal on Computing*, 16:486–502, 1987.
- [2] Petar Petrović and Mika Mikić. Naučni rad. In Miloje Milojević, editor, *Konferencija iz matematike i računarstva*, 2015.

Biografija autora

Jovan Dmitrović (*Gornji Milanovac, 17.11.1995.*)