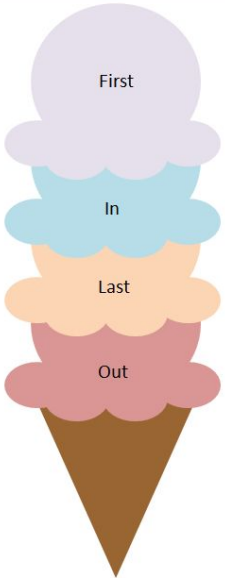
RUST

Osnovni koncept – 2. deo

Ownership

- Jedinstvena osobina
- Omogućava *Rust-u* da garantuje bezbednost memorije bez potrebe za sakupljačima smeća
- Predstavlja skup pravila koja regulišu kako Rust upravlja memorijom:
  - Svaka vrednost ima vlasnika
  - Istovremeno može da postoji samo 1 vlasnik
  - Kada vlasnik izađe iz opsega vrednost će biti ispuštena

Stack, heap



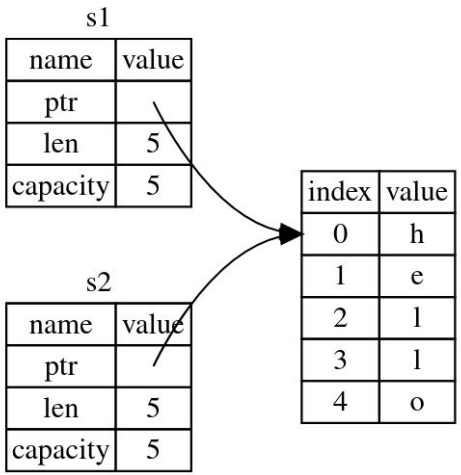
STACKS

Heaps

Interakcija više promenljivih

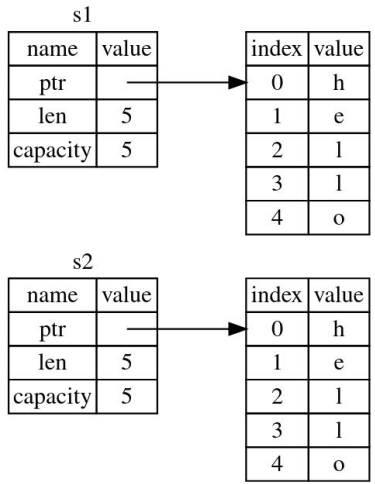
Move

```
let s1 = String::from("hello");

let s2 = s1;
```



Clone

```
let s1 = String::from("hello");

let s2 = s1.clone();
```



Interakcija više promenljivih

Copy

- Implementirana samo za vrednosti koje se čuvaju na stack-u
- Tipovi koji je implementiraju ovu metodu rade trivijalno kopiranje, čineći ih i dalje validnim nakon dodeljivanja drugoj promenljivoj.
- Nije dozvoljeno implementirati *Copy* metodu ako tip ima implementiranu *drop* metodu i ako treba nešto posebno da se desi izlaskom iz opsega → kompajler će prijaviti grešku

Funkcije i vlasništvo

- Mehanizam sličan kao i kod dodeljivanja vrednosti promenljivoj, biće urađeno ili kopiranje ili premeštanje vrednosti

```rust
fn main() {
    let text = "text";
    let s1 = String::from("hello"); // s1 je ovde kreiran i tu počinje da važi

    takes_ownership(s1); // s1 se prosledjuje funkciji i vise ne vazi u main-u
                         // vlasnistvo se prebacuje na funkciju
    println!("s1 u main-u posle poziva f-je: {}", s1);
    let x = 1; //kreirana nova promenljiva x; počinje da važi ovde
    makes_copy(x); // x se prosledjuje funkciji, ali posto je prostog tipa koji se cuva na stack-u
                   // onda ce doci do kopiranja i bice validna i posle prosledjivanja funkciji
    println!("x u main-u: {}", x);
}
fn takes_ownership(some_string: String){
    println!("{some_string}");
}
fn makes_copy(some_integer: i32){
    println!("{}", some_integer);
}
```

Povratna vrednost i opseg

- Povratne vrednosti mogu da prenesu vlasništvo

```
fn main() {
   let s1 = gives_ownership(); // funkcija ce vratiti neku vrednost koju cemo dodeliti promenljivoj s1

   let s2 = String::from("hello"); //kreirana promenljiva s2 i ona počinje ovde da važi

   let s3 = takes_and_gives_back(s2); // s2 se prosledjuje funkciji, koja ce vratiti vrednost u s3

   println!("s1: {},  s3: {}", s1, s3);
} // s1 i s3 ce ovde biti izbrisane odnosno dropovane
 // s2 je premestena tako da se nista nece desiti


fn takes_and_gives_back(some_string: String) -> String{
   some_string // promenljiva je vracena i premestena u opseg u kojem je funkcija pozvana
}

fn gives_ownership() -> String{
   let some_string = String::from("yours");  //kreirana promenljiva i odavde pocinje da vazi

   some_string // promenljiva je vracena i premestena tamo gde se poziva funkcija

}
```

Reference

- Koristi se za korišćenje vrednosti bez prenosa vlasništva
- Ona je poput pokazivača zato što predstavlja adresu memorijske lokacije na osnovu koje možete da pristupite podacima koji su uskladišteni na toj adresi
- Za razliku od pokazivača garantuje da ukazuje na važeću vrednost određenog tipa tokom trajanja reference
- Prosleđivanje parametara po referenci se naziva pozajmljivanje. → Funkcija neće dobiti vlasništvo nad promenljivom, već je samo dobija na korišćenje.

Reference

```
fn main() {
    let s1 = String::from("hello");

    let len = calculate_length(&s1); //promenljiva s1 prosledjena kao referenca

    println!("The length of '{}' is {}. ", s1, len);
}

fn calculate_length(s: &String) -> usize {
    s.len()
}
```

Zadatak

- Napraviti funkciju koja na prosleđeni string dodaje string “function”, a zatim u *main* funkciji ispisati rezultat.

PROBLEM: Vrednost promenljive ne može da se promeni. Referenca je nepromenljiva isto kao i promenljive.

REŠENJE: Mut reference

Promenljive reference

```
fn main() {
    let s = String::from("hello");
    change(&s);

    println!("{}", s);
}

fn change(some_string: &String) {
    some_string.push_str(", world");
}
```

- Rešenje promenljive reference

```
fn main() {
    let mut s = String::from("hello");
    change(&mut s);

    println!("{}", s);
}
fn change(some_string: &mut String) {
    some_string.push_str(", world");
}
```

Promenljive reference

- ***Ograničenje: ako imate promenljivu reference na vrednost, ne možete imati druge reference na tu vrednost.***
  - Omogućava mutaciju, ali na veoma kontrolisan način
  - Prednost:
    - Rust može da spreči trku oko podataka u vreme kompajliranja.
- Zaobilaženje ograničenja
  - Upotreba blokova

Promenljive reference

```rust
fn main() {
    let mut s = String::from("hello");

    let r1 = &mut s;
    let r2 = &mut s;

    println!("{}, {}", r1, r2);
}
```

```
error[E0499]: cannot borrow `s` as mutable more than once at a time
 --> src/main.rs:5:14
  |
4 |     let r1 = &mut s;
  |              ------ first mutable borrow occurs here
5 |     let r2 = &mut s;
  |              ^^^^^^ second mutable borrow occurs here
6 |
7 |     println!("{}, {}", r1, r2);
  |                        -- first borrow later used here

For more information about this error, try `rustc --explain E0499`.
error: could not compile `ownership` due to previous error
```

Reference - promenljive i nepromenljive

- Ne možete imati promenljivu i nepromenljivu reference na istu vrednost.
- Višestruke nepromenljive reference su dozvoljene jer niko ko samo čita podatke nema mogućnost da utiče na čitanje podataka nekog drugog.
- Opseg reference počinje od mesta gde je uvedena i nastavlja se do poslednjeg puta kada se ta referenca koristi.

```
let mut s = String::from("hello");

let r1 = &s; // no problem

let r2 = &s; // no problem

let r3 = &mut s; // BIG PROBLEM


println!("{}, {}, and {}", r1, r2, r3);
```

```
let mut s = String::from("hello");

let r1 = &s; // no problem

let r2 = &s; // no problem

println!("{} and {}", r1, r2);


let r3 = &mut s; // no problem

println!("{}", r3);
```

Slice tip

- Omogućava da referencirate neprekidni niz elemenata u kolekciji, a ne celu kolekciju.
- Predstavlja neku vrstu reference, tako da nema vlasništvo.

String Slices

- Referenca na deo stringa (&str)
- *string[start_index, end_index]*
  - *start_index* - prva pozicija
  - *end_index* - jedan više od poslednje pozicije u preseku

Zadatak

Napisati funkciju koja uzima niz reči razdvojenih razmacima i vraća prvu reč koju pronađe u tom nizu. Ako funkcija ne pronađe razmak u stringu, ceo niz mora biti jedna reč, tako da ceo string treba da bude vraćen.