Rust

Box<T>, vežbanje

Pametni pokazivači

- Pokazivač - osnovni koncept za promenljive koje sadrže adresu u memoriji.
- Pametni pokazivači - strukture podataka slične pokazivačima koje imaju dodatne metapodatke i mogućnosti.
- Referenca - pozajmljuje podatke
- Pametni pokazivač - poseduje podatke na koje upućuje

Pametni pokazivač

- Implementiraju se pomoću strukture i za razliku od obične strukture implementiraju karakteristike:
  - *Deref* - omogućava da se instanca strukture pametnog pokazivača ponaša kao referenca tako da možete napisati kod da radi sa pametnim pokazivačima ili sa referencama
  - *Drop* - omogućava da prilagodite kod koji se pokreće kada instanca pametnog pokazivača izađe van opsega

Box<T>

- Najjednostavnija vrsta pametnih pokazivača, koja omogućava da čuvate podatke na heap-u, a ne na stack-u.
  - Na stack-u ostaje samo pokazivač na podatke koji se nalaze na heap-u.
- Koriste se u sledećim situacijama:
  - Kada imate tip čija veličina nije poznata u vreme kompajliranja i želite da koristite vrednost tog tipa u kontekstu koji zahteva tačnu veličinu.
  - Kada imate velike količine podataka i želite da prenesete vlasništvo, ali i da osigurate da se podaci neće kopirati kada to uradite.
  - Kada želite da posedujete vrednost i brinete samo da je top tip koji primenjuje određenu osobinu, a ne da je određenog tipa.

Box<T>

```
fn main() {
    let b = Box::new(5);
    println!("b = {}", b);
}
```

Box i rekurzivni tipovi

- *Problem: Rust u vreme kompajliranja treba da zna koliko prostora zauzima tip. Ugnježdene vrednosti rekurzivnih tipova teoretski se mogu nastaviti u beskonačnost.*
- ***Rešenje: Box ima poznatu veličinu.***

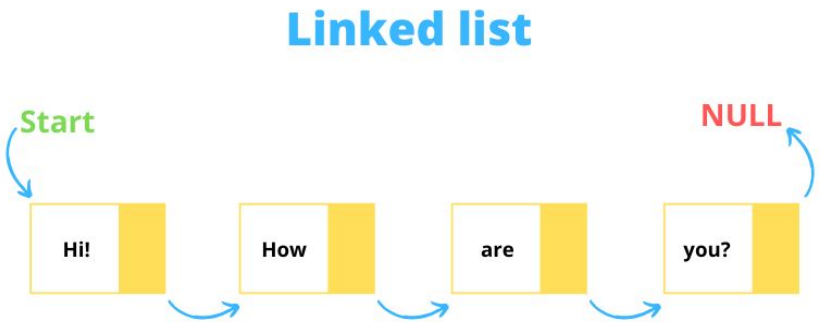
```
enum List {
    Cons(i32, List),
    Nil,
}
use crate::List::{Cons, Nil};
fn main() {
    let list = Cons(1, Cons(2, Cons(3, Nil)));
}
```

```
enum List {
    Cons(i32, Box<List>),
    Nil,
}
use crate::List::{Cons, Nil};
fn main() {
    let list = Cons(1, Box::new(Cons(2, Box::new(Cons(3, Box::new(Nil))))));
}
```

Zadatak 1.

Implementirati jednostruku spregnutu listu. Neophodno je implementirati sledeće funkcije:

- *Append* - novi element se dodaje na kraj liste
- *Pop* - uklanja element sa početka liste
- *Delete* - briše traženi element
- *Size_of* - računa koliko elemenata ima u listi
- *Print* - ispisuje elemente koji se nalaze u listi



Zadatak 2.

Implementirati binarno stablo. Neophodno je implementirati sledeće funkcije:

- *Add* - dodaje novi element u stablo
- *Pop* - vraća element koji se nalazi na poslednjem mestu u stablu
- *Get_root* - vraća koren stabla
- *Delete* - uklanja poslednji element iz stabla
- *Print* - ispisuje elemente stabla
- *Search* - pronalazi određeni element