Kloužeri, iteratori, pametni pokazivači

Paralelne i distribuirane arhitekture i jezici
Računarstvo visokih performansi
Zimski semestar, školska 2024/25.
Branislav Ristić

Anonimne funkcije

- Anonimne funkcije nemaju ime.
- Koriste se za kratke, lokalne zadatke.
- Često se definišu unutar drugih funkcija.
- Pogodne su za inline ili jednokratnu upotrebu.
- Omogućavaju koncizan kod.

Kloužeri

- Kloužeri hvataju vrednosti iz okruženja.
- Omogućavaju prilagodljiv kod prema kontekstu.
- Omogućavaju fleksibilnost.
- Podržavaju proizvoljno ponašanje pomoću `unwrap_or_else`.
- Primer:
  - *01_unwrap_or_else.rs*

Kloužeri - Definicije

```
fn  add_one_v1   (x: u32) -> u32 { x + 1 }

let add_one_v2 = |x: u32| -> u32 { x + 1 };

let add_one_v3 = |x|             { x + 1 };

let add_one_v4 = |x|               x + 1  ;
```

Kloužeri

- Kloužeri hvataju vrednosti na tri načina
  - nepromenljivo
  - promenljivo
  - premeštanje
- Kloužer bira način prema telu funkcije
- Više nepromenljivih referenci dozvoljeno istovremeno
- Promenljiva referenca onemogućava druge reference
- `move` ključna reč forsira preuzimanje vlasništva
  - Korisno za prenos podataka u novi niti
- Primer:
  - *02_closures.rs*

Kloužeri - Osobine

- Kloužeri implementiraju neke od sledećih osobina:
  - `FnOnce`: Jednom se poziva, premešta varijable u capture-u.
  - `FnMut`: Može biti pozvan više puta, menja varijable u capture-u.
  - `Fn`: Može biti pozvan više puta, menja i ne premešta varijable u capture-u.
- Kada kloužer uhvati vrednost kroz vlasništvo, ta vrednost se premesti u kloužer.
- Nakon što je vrednost premeštena, ona više nije dostupna u spoljnim funkcijama.
- Primer:
  - *03_closures.rs*
  - *04_closures.rs*

Kloužeri

- Kloužeri moraju znati koliko dugo žive varijable koje koriste.
- Rust automatski prepoznaje životne vekove u jednostavnim slučajevima.
- Primer:
  - *05_closure_lifetimes.rs*

Iteratori

- Iteratori omogućavaju iteriranje kroz kolekciju elemenata.
- Sakrivaju implementaciju od korisnika.
- Iteratori u Rust-u su lenji, što znači da nemaju efekta dok se ne pozove metoda koja ih konzumira.
- Primer:
  - *06_iterator.rs*

Iteratori

- Svi iteratori implementiraju osobinu `Iterator`.
- Iterator trait zahteva asocirani tip `Item` i metodu `next`.
- Metoda `next` vraća stavku obavijenu u `Some` ili `None` kada iteracija završi.
- Metoda `next` konzumira stavke, menjajući unutrašnje stanje iteratora.
- Iteratori moraju biti mutabilni jer pozivanje next menja njihovo stanje.
- Neke kolekcije imaju metode za kreiranje iteratora nad kolekcijama:
  - `iter()`
  - `iter_mut()`
  - `into_iter()`
- Primer:
  - *07_iterator_impl.rs*
  - *08_iterator_usage.rs*

Iteratori

- Metode koje pozivaju next se nazivaju potrošači, jer iskoriste iterator.
- Metode koje iskoriste iterator, ali takođe naprave i novi, se nazivaju adapteri.
- Tri glavne operacije nad iteratorima:
  - `map`
  - `filter`
  - `fold`
- Primer:
  - *09_iterator_ops.rs*

Box

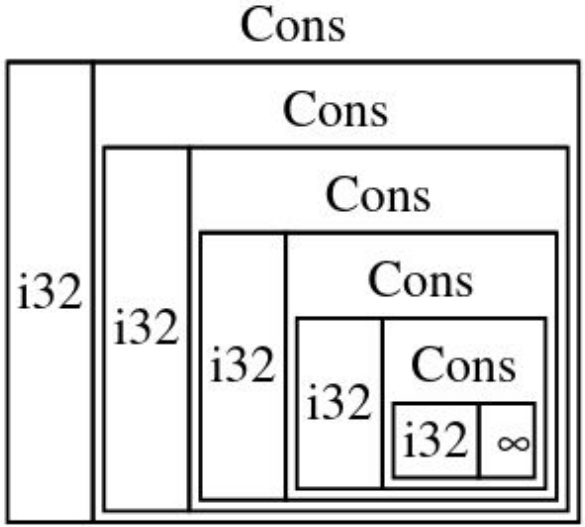
- `Box` predstavlja pametni pokazivač.
- Njegova vrednost (koja predstavlja adresu na heap-u) se nalazi na stack-u.
- Koristi se u slučajevima:
  - Nije poznata veličina podatka u vreme kompajliranja.
  - Pri prenosu vlasništva bez kopiranja podataka.
  - Dynamic dispatch.
- Primer:
  - *10_box.rs*

Box - Rekurzivni tipovi

- Rekurzivni tipovi sadrže vrednosti istog tipa.
- Problem: Rust mora da zna koliko prostora tip zauzima.
- Rekurzija može trajati beskonačno, pa Rust ne može odrediti veličinu.
- `Box`-ovi imaju poznatu veličinu, omogućavajući rekurzivne tipove.

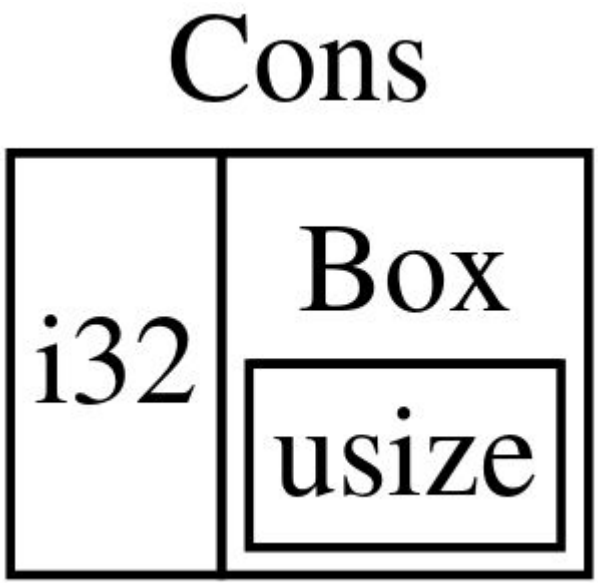
Cons lista

- Struktura podataka iz Lispa, slična spregnutim listama.
- Sastoji se od parova: (vrednost, sledeći element).
- Poslednji element je "Nil", koji označava kraj liste.
- Problem:
  - Direkta rekurzija dovodi do beskonačne veličine.
  - Rust ne može da izračuna
    - Potrebnu memoriju
      - Za rekurzivne tipove.
        - Kako se računa veličina enuma?
- Primer:
  - 11_cons.rs



Cons lista

- Rešenje:
  - Korišćenje `Box<T>`
- Primer:
  - *12_cons.rs*



Deref osobina

- Implementiranjem `Deref` osobine moguće je obezbediti proizvoljno ponašanje operatora dereferenciranja `*`.
- Time se obezbeđuje da se pametni pokazivač suštinski posmatra kao referenca.
- Primer:
  - *13_ref.rs*
  - *14_box.rs*
  - *15_my_box.rs*
- Bez `Deref` osobine kompajler je u mogućnosti da dereferencira samo `&` reference.

Implicitno dereferenciranje - Funkcije i metode

- Implicitna `Deref` prinuda predstavlja koncept pretvaranja reference iz jednog tipa u drugi
- Na primer:
  - Omogućeno je automatsko pretvaranje iz `&String` u `&str`
    - Usled postojanja `Deref` osobine.
- Izbacuje potrebu za prekomernim eksplicitnim referenciranjem i dereferenciranjem.
- Primer:
  - *16_deref_coercion.rs*

Deref prinuda - mutabilnost

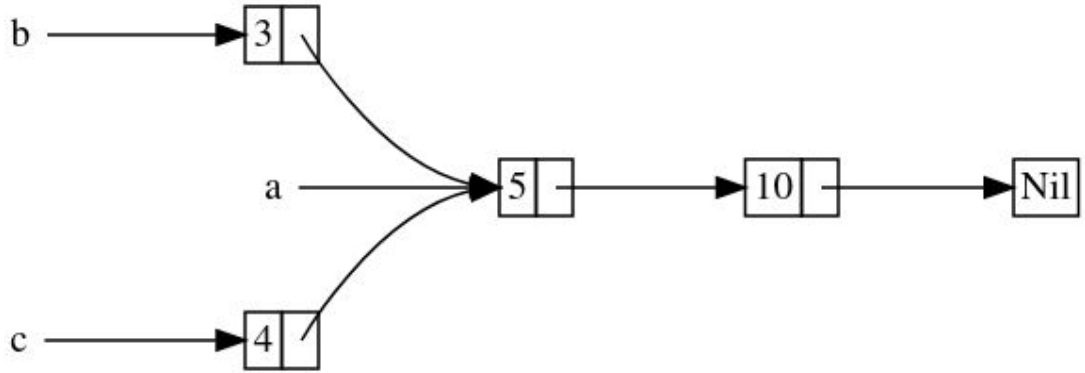
- Rast vrši `Deref` prinude kada pronađe tipove i implementacije osobina u tri slučaja:
  - Iz `&T` u `&U` kada `T: Deref<Target=U>`
  - Iz `&mut T` u `&mut U` kada `T: DerefMut<Target=U>`
  - Iz `&mut T` u `&U` kada `T: Deref<Target=U>`

Drop

- `Drop` trait u Rust-u omogućava programerima da definišu kako se resursi (npr. fajlovi, konekcije) oslobađaju kada vrednosti izlaze iz opsega.
- Rust automatski poziva metodu drop kada objekat izađe iz opsega, sprečavajući curenje resursa.
- ***non-lexical lifetimes***
- Ne možete direktno pozvati metodu `drop`
- Možete koristiti funkciju1 `std::mem::drop` za ručno otpuštanje resursa pre izlaska iz opsega.
- Primer:
  - *17_drop.rs*

Rc

- `Rc<T>` omogućava višestruko vlasništvo nad podacima.
- Praćenje broja referenci na podatke.
- Kada broj referenci padne na nulu, podaci se oslobađaju.
- Koristi se u jednonitnim aplikacijama.
- Omogućava deljenje podataka između više delova programa.
- Primer:
  - *18_a_rc.rs*
  - *18_b_rc.rs*



RefCell

- `Rc`
  - Upravlja vlasništvom između više strana
  - Beleži broj referenci
  - Ne podržava mutabilnost out of the box
- `RefCell`
  - **Omogućava unutrašnju mutabilnost**
  - Čak i iza nemutabilne reference
  - Borrow check prilikom runtime-a
- `RefCell` se kombinuje sa `Rc` i drugim pametnim pokazivačima
  - Na primer, `Box`
- Primer:
  - *19_refcell.rs*
  - *20_refcell.rs*

Zadatak 1.

- Implementirati jednostruko spregnutu listu.

Zadatak 2.

- Implementirati dvostruko spregnutu listu.

Zadatak 3.

- Implementirati binarno stablo.

Izvori

- Rust Community. “The Rust Programming Language - the Rust Programming Language.” Rust-Lang.org, 2018, doc.rust-lang.org/book/.
- Crichton, Will. “Experiment Introduction - the Rust Programming Language.” Brown.edu, rust-book.cs.brown.edu/.
- Rust Community. “Tour of Rust - Let’s Go on an Adventure!” Tourofrust.com, tourofrust.com/.
- Rust Team. “Rust Programming Language.” Rust-Lang.org, 2018, www.rust-lang.org/.

Kloužeri, iteratori, pametni pokazivači

Paralelne i distribuirane arhitekture i jezici
Računarstvo visokih performansi
Zimski semestar, školska 2024/25.
Branislav Ristić