RUST

Osnovni koncepti - 1. deo

Unos podataka

```
use std::io;
fn main() {
    println!("Guess the number!");
    println!("Please input your guess.");
    let mut guess = String::new();
    io::stdin()
        .read_line(&mut guess)
        .expect("Failed to read line");
    println!("You guessed: {guess}");
}
```

Promenljive

Promenljive,

Konstante,

Preklapanje promenljivih

Promenljive

- Podrazumevano su nepromenljive
  - Sigurnost i konkurentnost
- Definisanje promenljive:
  - ***let*** *naziv_promenljive*

Promenljive

- Definisanje promenljive i štampanje njene vrednosti.

```
fn main(){
    //int promenljiva
    let x = 15;
    println!("int -> {x}");

    //char promenljiva
    let c = 'R';
    println!("char -> {c}");

    //string promenljiva
    let s = "Hello";
    println!("string -> {}", s);
}
```

Promenljive - zadatak

Napraviti mini program koji zadovoljava sledeće uslove:

a. Definisati promenljivu *x* koja na početku ima vrednost 4.
b. Na konzolu ispisati: *“x = 4”.*
c. Vrednost promenljive *x* uvećati za neku proizvoljnu vrednost.
d. Na konzolu ispisati novu vrednost promenljive *x (”Nova vrednost promenljive x je: {x}”)*

PROBLEMI?

Šta da radimo ako promenljivoj hoćemo da dodelimo novu vrednost?

Promenljive

- Rezime prethodnog zadatka:
  - Promenljivoj koja je definisana sa ključnom rečju ***let*** ne možete da menjate vrednost.
- Bitno:
  - Greška će biti prijavljena u vreme kompajliranja
  - Kompajler vam garantuje da kada navedete da se vrednost neći promeniti, onda se zaista neće promeniti, tako da ne morate sami da vodite računa.
- Rešenje:
  - ***let mut*** *naziv_promenljive*
    - Drugi delovi programa mogu da menjaju vrednost promenljive

Promenljive

```rust
fn main(){
    let a = 3;
    println!("The value of a is {a}");

    a = a + 3;
    println!("The new value of a is {a}");
}
```

```
error[E0384]: cannot assign twice to immutable variable `a`
 --> src/main.rs:5:5
  |
2 |     let a = 3;
  |         -
  |         |
  |         first assignment to `a`
  |         help: consider making this binding mutable: `mut a`
...
5 |     a = a + 3;
  |     ^^^^^^^^^ cannot assign twice to immutable variable
```

```rust
fn main(){
    let mut a = 3;
    println!("The value of a is {a}");

    a = a + 3;
    println!("The new value of a is {a}");
}
```

```
The value of a is 3
The new value of a is 6
```

Konstante

- Promenljive čija vrednost ne može da se menja.
- Definisanje konstante:
  - ***const*** *naziv_konstante: tip = vrednost*

Konstante vs *mut* promenljive

- ***mut*** ne može da se koristi sa konstantama
- Definišu se sa ključnom rečju ***const***, a ne sa ***let***
- Uvek mora biti označen tip vrednosti
- Mogu da se definišu u bilo kom opsegu → korisne za vrednosti o kojima mnogi delovi koda moraju da znaju
- Vrednost mora da bude konstantni izraz, ne može da bude izraz koji se evaluira u toku izvršavanja
- *Preporuka:*
  - Ime konstante se navodi velikim slovima
  - Ako se sastoji od više reči onda su one odvojene “_”

Preklapanje promenljivih

- Dozvoljeno je definisanje promenljive sa istim imenom više puta
- Uvek je važeća promenljiva koja se poslednja definiše, a važiće dok se ne definiše nova promenljiva sa istim imenom ili dok se ne izađe iz opsega u kojem je definisana

Preklapanje promenljivih

```
fn main(){
    let x = 5;
    let x = x + 5;
    {
        let x = x - 2;
        println!("The value of x in the inner scope is: {x}");
    }
    println!("The value of x is: {x}");
}
```

Uzima se vrednost promenljive *x* koja je prethodno definisana i ta promenljiva prestaje da važi, pošto je definisana promenljiva sa istim imenom.

x = 10 → zato što je prva *x* promenljiva preklopljena drugom, koja ima vrednost koja je za 5 veća od prve promenljive *x*, a promenljiva *x* koja je definisana u bloku ne važi više

x = 8 → zato što je promenljiva **x** definisana u novom bloku i ona će da važi sve dok ne izađemo iz opsega u kojem je definisana

Preklapanje promenljive vs *mut* promenljiva

- Menjate vrednost promenljivoj ali ona ostaje nepromenljiva nakon što se te transformacije završe.
- Kreirate novu promenljivu što znači da možete da joj promenite tip vrednosti, a da ponovo koristite isto ime. Ovo sa upotrebom *mut* ne može.

Preklapanje promenljive vs *mut* promenljiva

```rust
fn main(){
    let x = 5;
    println!("The value of x is: {x}");

    let x = 'Z';
    println!("The value of x is: {x}");

}
```

```rust
fn main(){
    let mut x = 5;
    println!("The value of x is: {x}");

    x = 'Z';
    println!("The value of x is: {x}");

}
```

```
error[E0308]: mismatched types
 --> src/main.rs:7:9
  |
3 |     let mut x = 4;
  |                 - expected due to this value
...
7 |     x = 'Z';
  |         ^^^ expected integer, found `char`
```

Tipovi podataka

Prosti tipovi,

Složeni tipovi podataka

Tipovi podataka

- Svaka vrednost je određenog tipa i na osnovu njega Rust zna kako da radi sa njom
- Dva tipa podataka:
  - Prosti,
  - Složeni.
- Rust je statički tipiziran jezik → u vreme kompajliranja mora da zna tipove svih promenljivih
  - Obično tip podatka može da se zaključi na osnovu vrednosti koju dodelimo nekoj promenljivoj, ali
  - Postoje i mesta gde jasno moramo da naznačimo kog je tipa neki podatak
    - Pr. želite vrednost koju je korisnik uneo da pretvorite u neki drugi tip, morate tačno da naznačite kog tipa želite da bude ta vrednost

```
let guess: u32 = "42".parse().expect("Not a number!");
```

Prosti tipovi podataka

- Predstavljaju jednu vrednost

tip podatka	opis	vrednost/oznaka/primer		
Integer	Ceo broj	dužina	označeni	neoznačeni
		8-bita	*i8*	*u8*
		16-bita	*i16*	*u16*
		32-bita	*i32*	*u32*
		64-bita	*i64*	*u64*
		128-bita	*i128*	*u128*
Floating point number	Decimalni broj	*f32*		Jednostruka preciznost
		f64		Dvostruka preciznost
Boolean	Vrednost relacionog ili logičkog izraza	*bool*		True
				False

Character	Osnovni abecedni tip	*char*		`let c = 'z';`
				`let z: char = 'ℤ'; // with explicit type annotation`
				`let heart_eyed_cat = '😻';`

Osnovni tipovi podataka

```rust
use std::any::type_name;

fn type_of<T>(_: T) -> &'static str {
    type_name::<T>()
}


fn main() {
    let x:i8 = 34;
    let y:i16 = 278;
    let w:i32 = 1_000;
    let f0 = 2.0;
    let f1: f32 = 1.0;
    let t: bool = true;
    let f: bool = false;
    let z: char = 'Z';
    let smile: char = '😻';
    println!("{x} -> {}", type_of(x));
    println!("{y} -> {}", type_of(y));
    println!("{w} -> {}", type_of(w));
    println!("{f0} -> {}", type_of(f0));
    println!("{f1} -> {}", type_of(f1));
    println!("{t} -> {}", type_of(t));
    println!("{f} -> {}", type_of(f));
    println!("{z} -> {}", type_of(z));
    println!("{smile} -> {}", type_of(smile));

}
```

```
34 -> i8
278 -> i16
1000 -> i32
2 -> f64
1 -> f32
true -> bool
false -> bool
Z -> char
😻 -> char
```

Operator Symbol	Operator Name	Description
+	Arithmetic Addition	Returns the sum of two or more operands
–	Arithmetic Subtraction	Return the difference between two or more operands.
*	Arithmetic multiplication	Returns the product of two or more operands
/	Arithmetic division	Returns the quotient of the left operand dividend by the right operand
%	Arithmetic remainder.	Returns the remainder from the division of the left operand by the right operand.

Operator Symbol	Operator Name	Description
&&	Short-circuiting logical AND	Returns true if all the specified conditions evaluate to be true.
||	Short-circuiting logical OR	Returns true if at least one of the specified conditions is true.
!	Logical NOT	Negates the result of a Boolean expression. If the condition is true, the not operator returns false.

Operator Symbol	Operator Name	Description
>	Greater than	Returns true if the operand on the left is greater than the right operand.
<	Less than	Returns true if the left operand is less than the right operand.
>=	Greater than or equal to	Returns true if the left operand is greater than or equal to the right operand.
<=	Less than or equal to	Returns true if the left operand is less than or equal to the right operand.
==	Equal to	Return true if the left operand is equal right operand.
!=	Not Equal to	Returns true if the left operand is not equal right operand.

Složeni tipovi podataka

- Mogu da grupišu više vrednosti u jedan tip
  - *Tuple*
  - *Array*

Tuple

- Grupisanje više vrednosti različitih tipova u jedan složen tip.
- Fiksne dužine
  - Kada se jednom definišu, ne mogu da rastu ili da se smanjuju
- Definisanje torke:
  - *let tup: ([tip1, tip2, ...]) = ([vrednost1, vrednost2, ...])*
  - *Primer:*

```
let tup: (i32, char, bool) = (8, 'Z', true)
```

Tuple – preuzimanje vrednosti

- Destrukturiranje vrednosti
  - Izdvajanje pojedinačnih vrednost iz *tuple-a* upotrebom šablona

```
let (x, y, z) = tup;
```

- Preuzimanje jedne vrednosti sa određenog mesta

```
let x = tup.0;

let y = tup.1;

let z = tup.2;
```

Array

- Kolekcija elemenata istog tipa.
- Fiksne dužine

Array

- Definisanje:

```
let a = [1, 2, 3, 5, 6, 6];
let b: [i32; 4] = [1, 2, 3, 4];
let c = [2; 4]; //[4, 4]
```

- Pristup elementima niza:

```
let x = a[2];
let y = b[0];
```

Array – zadataka pristup elementima niza

- Napisati Rust mini program koji ispiše element koji se nalazi u nizu na indeksu koji korisnik želi.

Array

- Nevalidan pristup elementima niza
  - Ako pokušate da pristupite elementu na indeksu koji je veći od dužine niza onda će se Rust uspaničiti.
  - *Bezbednost* - zabranjen pristup memoriji koja ne pripada nizu

Funkcije

Funkcije

- Svaki program ima *main* funkcija
- Deklarisanje funkcije:

```
fn naziv_funkcije([param1, param2,...]) -> tip_povrtne_vrednosti{

    Statements;

}
```

Funkcije

```
fn main(){
    let a = 5;
    let b = 9;

    let sum = suma(a, b);

    println!("a + b = {sum}");
}

fn sum(a:i32, b:i32) -> i32{
    a + b
}
```

Iskaz i izraz

- *Statement (iskaz)* – instrukcije koje izvršavaju neku radnju i ne vraćaju vrednost.
  - Kreiranje promenljive i dodeljivanje vrednosti pomoću *let*
  - Definisanje funkcije
- *Expression (izraz)* – izračunava se do rezultujuće vrednosti

Povratna vrednost

- Ne imenuju se, ali morate deklarisati tip povratne vrednosti

```
fn sum(a:i32, b:i32) -> i32{
    a + b
}
fn sum(a:i32, b:i32) -> i32{
    return a + b;
}
```

Kontrole toka

Kontrole toka

Petlje

IF

```
let number = 6;


if number % 3 == 0 {

    println!("number is divisible by 3");

} else if number % 2 == 0 {

    println!("number is divisible by 2");

} else {

    println!("number is not divisible by 3, or 2");

}
//_____________________________________________________________
let number = if 6 > 8 {2} else {3};
```

Zadatak

Napraviti mini kalkulator. Korisnik unosi operaciju i 2 cela broja. Operacija može da bude:

- “PLUS”
- “MINUS”
- “MULTIPLY”
- “DIVIDE”

Na osnovu prosleđene operacije, izvršiti potrebnu aritmetičku operaciju i prikazati njen rezultat. *Sprečiti potencijalne greške.*

LOOP

- Izvršava deo koda beskonačno puta ili dok je eksplicitno ne zaustavite (*CTRL+C*)

```
loop {
    println!("hello");
}
```

- *break* ili *continue*

LOOP

- Iz petlje možete da vratite vrednost

```
let mut counter = 0;

let result = loop {

    counter += 1;

    if counter == 3 {

        break counter * 2;

    }

};

println!("The result is {result}");
```

WHILE

```
let mut num = 0;

while num <= 0 {

    num += 1;

}

println!("The result is {num}");
```

FOR

```
let a = [10, 20, 30, 40, 50]

for e in a {

    println!("{e}");

}
```

Zadatak

Napisati program koji ispisuje prvih 25 prostih brojeva.

Kolekcije

Vector

HashMap

Vector

- Omogućavaju da sačuvate više od jedne vrednosti u jednoj strukturi koja sve vrednosti čuva jednu pored druge u memoriji.
- Čuvaju samo vrednosti istog tipa.

```
let vec: Vec<i32> = Vec::new();
 let vec = vec![1, 2, 3, 4, 5]
```

Vector

- Manipulacije nad vektorom:
  - ***push*** – ubacivanje novog elementa u vektor
  - ***get*** – dobavljanje elementa na određenoj poziciji

Vector

```
let mut vector_nums = Vec::new();

vector_nums.push(1);

vector_nums.push(2);

vector_nums.push(3);


let second = vector_nums[1];

println!("The second value is {}", second);


let second: Option<&i32> = vector_nums.get(1);

match second {

    Some(second) => println!("The second element is {}", second),

    None => println!("There is no second element"),

}
```

HashMap

- Skladišti elemente u vidu para (ključ, vrednost).

```
use std::collections::HashMap;
let mut map_example = HashMap::new();
```

HashMap

- Manipulacije sa HashMap-om
  - ***insert*** – unos novog elementa u mapu
  - ***get*** – preuzimanje vrednosti elementa iz mape na osnovu ključa
  - Izmene vrednosti u mapi
    - Element sa istim ključem već postoji
      - Preklapanje vrednosti koja je asocirana prosleđenim ključem
      - Izmena vrednosti na osnovu postojeće
    - Upis elementa u bazu samo ako ne postoji element sa prosleđenim ključem
      - ***entry***

HashMap

```rust
use std::collections::HashMap;
let mut map_example = HashMap::new();
map_example.insert("first", 1);
map_example.insert("second", 2);

for (key, value) in &map_example{
    println!("{} : {}", key, value);
}

let elem = map_example.get("first");
match elem{
    Some(elem) => println!("Element with key first is {}", elem),
    None => println!("Not exist element with key first"),
}
```

HashMap

```
use std::collections::HashMap;

let mut map_example = HashMap::new();

map_example.insert("first", 1);

map_example.insert("second", 2);

map_example.insert("first", 10);

map_example.entry("second").or_insert(13);

map_example.entry("third").or_insert(100);

let order_nums = ["first", "second", "third", "first", "fifth"];

let mut order_nums_count = HashMap::new();

for num in order_nums {

    let count = order_nums_count.entry(num).or_insert(0);]

    *count +=1;

}

println!("{:?}", order_nums_count);
```

Preklapanje postojeće vrednosti

Upis novog elementa ukoliko ne postoji element sa prosleđenim ključem.

Izmena vrednosti elementa na osnovu postojeće vrednosti

Zadaci

Zadatak 1.

Napraviti program koji omogućava korisniku da unese veličnu niza, a zatim:

- Dinamički popuniti niz.
- Ispisati sve elemente niza.
- Ispisati elemente niza u obrnutom redosledu.
- Ispisati element na indeksu koji je korisnik uneo. Ako nema elementa na traženom nizu onda korisnika obavestiti o tome.
- Ispisati sve elemente koji se nalaze na indeksu koji je deljiv sa 3.
- Prebrojati koliko se elemenata nalazi na parnim, a koliko na neparnim indeksima. *Napomena: koristiti HashMap-u.*

Zadatak 2.

Implementirati biblioteku za sortiranje niza. Biblioteka treba da podrži sledeće algoritme za sortiranje:

- Bubble sort
- *... biblioteka će biti proširena na narednim vežbama ...*

Zadatak 3.

Napraviti program za administraciju korisnicima koristeći *HashMap-u.* Program treba da obezbedi:

- Dodavanje novog korisnika,
- Prikaz postojećih korisnika,
- Logovanje postojećih korisnika.

Napomena: Za sada je dovoljno da korisnike predstavite u string formatu:

“Name Surname username password”