Rust

Obrada grešaka, I/O

Argumenti komandne linije

- Prosleđuju se prilikom pokretanja programa
  - `$ cargo run -- searchstring example-filename.txt`

Argumenti komandne linije

```
use std::env;


fn main() {
    let args: Vec<String> = env::args().collect();
    dbg!(args);
}
```

Argumenti komandne linije

```
use std::env;


fn main() {
    let args: Vec<String> = env::args().collect();


    let query = &args[1];
    let file_path = &args[2];


    println!("Searching for {}", query);
    println!("In file {}", file_path);
}
```

Rad sa datotekama

- Struktura *File* predstavlja datotektu koja je otvorena i omogućava pristup za čitanje i/ili pisanje
  - Sve metode za rad sa datotekama se nalaze u standardnoj biblioteci u modulu *fs*
  - Metode vraćaju *Result<T, E>*

Rad sa datotekama

create	Kreira i otvara datoteku u režimu za pisanje. Ako datoteka već postoji, stari sadržaj se uništava, a ako ne onda se kreira nova.
create_new	Kreira novu datoteku u režimu za čitanje i pisanje. Ukoliko datoteka već postoji onda vraća grešku.
open	Otvara datoteku u režimu za čitanje. Vodi računa o zatvaranju datoteke.
read_lines	Vraća iterator na sve linije iz datoteke.
read_to_string	Čita sve bajtove do EOF. Vraća broj pročitanih bajtova.
write	Upisuje podatke u datoteku.

Greške

- Dve glavne kategorije:
  - Nadoknadive
    - Korisniku samo prijavljujemo da je došlo do greške i ponovo pokušamo da izvršimo operaciju
      - *Primer*: fajl sa kojim želimo da interagujemo ne postoji
  - Nepopravljive
    - Obavestimo korisnika da je došlo do greške i prekidamo izvršavanje programa
      - *Primer*: pristupamo memoriji koja ne pripada nizu

Greške

- Mehanizmi za obradu greški
  - Makro *panic*
  - Enum *Result<T, E>*

Nepopravljive greške

- Greške koje izazivaju loše stvari u vašem kodu i ne možete ništa da uradite povodom toga.
- Upotreba *panic* makroa
  - Odštampaće poruku o neuspehu, odmotati (pravi se rezervna kopija steka i čiste se podaci iz svake funkcije na koju se naiđe) i očistiti stek i ugasiti program
- Pozivanje makroa:
  - Preduzimanje radnje koja izaziva stanje panike
  - Eksplicitno pozivanje makroa *panic*

Makro *panic*

```rust
fn main() {
    panic!("crash and burn");
}
```

```
jovana@jovana:~/Desktop/HPC-PDAJ-2022/vezbe_9/cas_9$ cargo run
   Compiling cas_9 v0.1.0 (/home/jovana/Desktop/HPC-PDAJ-2022/vezbe_9/cas_9)
    Finished dev [unoptimized + debuginfo] target(s) in 1.21s
     Running `target/debug/cas_9`
thread 'main' panicked at 'crash and burn', src/main.rs:5:5
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

Makro *panic* i *backtrace*

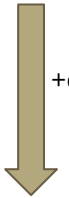
- Ako hoćemo možemo da prikažemo i šta se sve tačno desilo i gde je nastala greška
  - *RUST_BACKTRACE = 1*

```
jovana@jovana:~/Desktop/HPC-PDAJ-2022/vezbe_9/cas_9$ RUST_BACKTRACE=1 cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/cas_9`
thread 'main' panicked at 'crash and burn', src/main.rs:5:5
stack backtrace:
   0: rust_begin_unwind
             at /rustc/a8314ef7d0ec7b75c336af2c9857bfaf43002bfc/library/std/src/panicking.rs:584:5
   1: core::panicking::panic_fmt
             at /rustc/a8314ef7d0ec7b75c336af2c9857bfaf43002bfc/library/core/src/panicking.rs:142:14
   2: cas_9::main
             at ./src/main.rs:5:5
   3: core::ops::function::FnOnce::call_once
             at /rustc/a8314ef7d0ec7b75c336af2c9857bfaf43002bfc/library/core/src/ops/function.rs:248:5
note: Some details are omitted, run with `RUST_BACKTRACE=full` for a verbose backtrace.
```

Result<T, E>

```
pub enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

```rust
use std::fs::File;

fn main() {
    let greeting_file_result = File::open("hello.txt");
}
```




```rust
use std::fs::File;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => panic!("Problem opening the file: {:?}", error),
    };
}
```

```rust
use std::fs::File;
use std::io::ErrorKind;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => match error.kind() {
            ErrorKind::NotFound => match File::create("hello.txt") {
                Ok(fc) => fc,
                Err(e) => panic!("Problem creating the file: {:?}", e),
            },
            other_error => {
                panic!("Problem opening the file: {:?}", other_error);
            }
        },
    };
}
```

```rust
use std::fs::File;
use std::io::ErrorKind;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => match error.kind() {
            ErrorKind::NotFound => match File::create("hello.txt") {
                Ok(fc) => fc,
                Err(e) => panic!("Problem creating the file: {:?}", e),
            },
            other_error => {
                panic!("Problem opening the file: {:?}", other_error);
            }
        },
    };
}
```

```rust
use std::fs::File;
use std::io::ErrorKind;

fn main() {
    let greeting_file = File::open("hello.txt").unwrap_or_else(|error| {
        if error.kind() == ErrorKind::NotFound {
            File::create("hello.txt").unwrap_or_else(|error| {
                panic!("Problem creating the file: {:?}", error);
            })
        } else {
            panic!("Problem opening the file: {:?}", error);
        }
    });
}
```

Unwrap

```rust
use std::fs::File;
use std::io::ErrorKind;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => match error.kind() {
            ErrorKind::NotFound => match File::create("hello.txt") {
                Ok(fc) => fc,
                Err(e) => panic!("Problem creating the file: {:?}", e),
            },
            other_error => {
                panic!("Problem opening the file: {:?}", other_error);
            }
        },
    };
}
```

```rust
use std::fs::File;

fn main() {
    let greeting_file = File::open("hello.txt").unwrap();
}
```

Expect

```rust
use std::fs::File;
use std::io::ErrorKind;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => match error.kind() {
            ErrorKind::NotFound => match File::create("hello.txt") {
                Ok(fc) => fc,
                Err(e) => panic!("Problem creating the file: {:?}", e),
            },
            other_error => {
                panic!("Problem opening the file: {:?}", other_error);
            }
        },
    };
}
```

```rust
use std::fs::File;

fn main() {
    let greeting_file = File::open("hello.txt")
        .expect("hello.txt should be included in this project");
}
```

Propagacija grešaka

- Greška se vraća pozvanom kodu da on sam obradi grešku kako god želi

```rust
use std::fs::File;
use std::io::{self, Read};

fn read_username_from_file() -> Result<String, io::Error> {
    let username_file_result = File::open("hello.txt");

    let mut username_file = match username_file_result {
        Ok(file) => file,
        Err(e) => return Err(e),
    };

    let mut username = String::new();

    match username_file.read_to_string(&mut username) {
        Ok(_) => Ok(username),
        Err(e) => Err(e),
    }
}
```

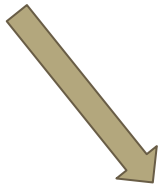
Operator ?

```rust
use std::fs::File;
use std::io::{self, Read};

fn read_username_from_file() -> Result<String, io::Error> {
    let username_file_result = File::open("hello.txt");

    let mut username_file = match username_file_result {
        Ok(file) => file,
        Err(e) => return Err(e),
    };

    let mut username = String::new();

    match username_file.read_to_string(&mut username) {
        Ok(_) => Ok(username),
        Err(e) => Err(e),
    }
}
```

```rust
use std::fs::File;
use std::io;
use std::io::Read;

fn read_username_from_file() -> Result<String, io::Error>
{
    let mut username_file = File::open("hello.txt")?;
    let mut username = String::new();
    username_file.read_to_string(&mut username)?;
    Ok(username)
}
```

```rust
use std::fs::File;
use std::io;
use std::io::Read;

fn read_username_from_file() -> Result<String, io::Error> {
    let mut username = String::new();

    File::open("hello.txt")?.read_to_string(&mut username)?;

    Ok(username)
}
```

```rust
use std::fs::File;
use std::io;
use std::io::Read;

fn read_username_from_file() -> Result<String, io::Error> {
    let mut username = String::new();

    File::open("hello.txt")?.read_to_string(&mut username)?;

    Ok(username)
}
```



```rust
use std::fs;
use std::io;

fn read_username_from_file() -> Result<String, io::Error> {
    fs::read_to_string("hello.txt")
}
```

Gde može da se koristi operator ?

- Samo u funkcijama čiji je povratni tip kompatibilan sa vrednošći ?
  - Povratna vrednost mora da bude tip za koji je definisan *trait From*
  - *Result, Option*

Panic vs Result

- Kada neko poziva vaš kod i prosledi vrednosti koje nemaju smisla, najbolje je da vratite grešku kako bi korisnik mogao sam da odluči šta će da uradi.
- Kada vaš kod može da završi u nekom lošem stanju, poželjno je da pozove makro *panic.*
- Uvek je poželjno da korisniku date informaciju o grešci, tako da je dobro iskoristiti *expect* metodu.

Zadatak

Kreirati program u kojem korisnik unosi heksadecimalne brojeve sve dok ne unese 0. Kada unese 0 unete brojeve treba sačuvati u binarnom fajlu, a zatim brojeve iščitati iz fajla i prikazati ih u decimalnom formatu.