Uvod

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

O predmetu

- Asistent:
  - Branislav Ristić
    - NTP-321
    - branislav.ristic@uns.ac.rs
    - Konsultacije:
      - Dogovor putem elektronske pošte
- Sajt predmeta:
  - https://www.acs.uns.ac.rs/
    - Obaveštenja
  - https://github.com/ristic-ac/PDAJ-2024-25/tree/main
    - Repozitorijum za vežbe
  - Pratiti redovno

Pregled tehnologija

- Operativni sistem: **GNU/Linux**
- Editor koda: **VSCode**
- Kompajler: **rustc**
- Upravljač projektom i zavisnostima: **cargo**
- Ekstenzije: **rust analyzer**

Rust

- Programski jezik opšte namene
- Fokusira se na:
  - Performanse
    - Ne poseduje standardni garbage collector
      - *Ownership model*
    - Ne poseduje RTE
  - Pouzdanost
    - Detekcija grešaka prilikom kompajliranja
  - Produktivnost
    - Ekosistem bogat alatima

Primena Rust-a

- Operativni sistemi
  - Redox
- Embedded sistemi
  - IoT uređaji
- Web programiranje
  - Wasm
- Blokčejn i kriptografija
  - Solana pametni ugovori
- Video igre
  - ECS sistemi
  - Game engine

Instalacija Rust-a (Linux)

- Neophodni alati:
  - curl
- Komanda za instalaciju (rustup):
  - `$ curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh`
- U slučaju uspešne instalacije biće prikazana poruka:
  - `Rust is installed now. Great!`
- Za proveru verzije koristi komandu:
  - `$ rustc --version`
- Za kompajliranje koristiti komandu:
  - `$ rustc <naziv-izvorne-datoteke.rs>`

Cargo

- Alat za:
  - Upravljanje projektom
  - Razrešavanje zavisnosti
- Ukoliko je Rust instaliran putem rustup-a, cargo bi trebao da je već instaliran, provera putem:
  - `$ cargo --version`

Kreiranje Cargo projekta

- Komanda za kreiranje projekta:
  - `$ cargo new [--lib | --bin (default)] <project_name>`
- Razlika između tipova projekata:
  - `lib`
    - Bibliotečki projekat
    - Entrypoint tipično: `lib.rs`
    - Ne poseduje main funkciju
    - Rezultat kompajliranje `.so` ili `.rlib` datoteke
  - `bin`
    - Izvršni projekat
    - Entrypoint tipično: `main.rs`
    - Poseduje main funkciju
    - Rezultat kompajliranja binarne (izvršne) datoteke.

Kreiranje Cargo projekta

- Nakon pokretanja komande:
  - `$ cargo new hello_cargo`
- Kreira se direktorijum sa sledećom strukturom:

```
hello_cargo/
├── .gitignore
├── Cargo.toml
└── src/
    └── main.rs
```

Hello World

```rust
// main.rs
fn main() {
    println!("Hello, world!");
}
```

Cargo.toml

- TOML - Tom’s Obvious, Minimal Language
- Cargo konfiguracioni format

```
# FileName: Cargo.toml
[package]
name = "hello_cargo"    # Naziv projekta
version = "0.1.0"       # Verzija projekta
edition = "2021"        # Major release year

[dependencies]
tokio = { version = "1.32", features = ["full"] }
```

Kompajliranje projekta

- `$ cargo build [--release]`
- `Cargo.lock`
  - Fajl u okviru kog se vodi evidencija o tačnim verzijama zavisnosti u projektu
  - **Ne menjati ručno**

Pokretanje projekta

- `$ cargo run`
- Pokreće aplikaciju:
  - Provera da li postoje izmene u kodu:
    - Postoje - cargo build, potom pokretanje
    - Ne postoje - pokretanje programa

Provera projekta

- `$ cargo check`
- Proverava da li se kod kompajlira
  - Ne kreira izvršni fajl

Ostale komande

- `$ cargo help [command]`

Crate

- Najmanja jedinica komajlabilosti u Rust-u.
- Vrste crate-ova
  - Binarni (0,N)
  - Bibliotečki (0,1)
- Moguće je preuzeti već gotove crate-ove sa crates.io
- Primer:
  - `main.rs` u okviru `src/`
  - `lib.rs`

Paket

- Paket predstavlja skup jednog ili više crate-ova
  - Porencijalno više binarnih crate-ova
  - Najviše jedan bibliotečki crate
- Paket poseduje Cargo.toml koji opisuje kako se crate-ovi i njihove zavisnosti build-uju
- Paket se sam po sebi ne kompajlira, vec sluzi kako bi se upravljalo sa vise crate-ova od jednom.
- Primer:
  - U okviru paketa se nalaze:
    - `src/main.rs`
    - `src/lib.rs`
    - Dodatni bin crate-ovi u okviru `src/bin/`

Modul

- Moduli pružaju mogućnost hijerarhijske organizacije koda u okviru crate-a.
- Svaki crate ima **korenski** modul (`main.rs` ili `lib.rs`)
- Moguće je:
  - Ugnježdavati module.
  - Deliti module u različite fajlove
- Primer: *project_example*

Upotreba *use* ključne reči

- *use crate*
  - Ukoliko se modul nalazi u okviru istog crate-a
- Na primer:

```
src/
├── lib.rs          // Library crate root
└── utils/
    ├── mod.rs      // "utils" module file
    └── helper.rs   // Sub-module "helper"
```

```
use crate::utils::helper::some_function();
```

Upotreba *use* ključne reči

- *use project_name,* ukoliko se referencira:
  - Bibliotečki crate iz binarnog, u okviru istog paketa
  - Bibliotečki crate iz eksternog integracionog testa (npr. datoteke u okviru *tests* direktorijuma)
  - Drugi crate (koji predstavlja zavisnost)
- Na primer:

```
my_project/
├── Cargo.toml
└── src/
    ├── main.rs      // Binary crate root
    └── lib.rs       // Library crate root
```

```
use my_project::some_function;
```

Profili izdanja

- Unapred definisani profili
- Omogućavaju veću kontrolu nad opcijama kompajliranja:
- Glavni profili:
- dev
- release

```
# Filename: Cargo.toml

[profile.dev]
opt-level = 0

[profile.release]
opt-level = 3
```

Objavljivanje projekta na crates.io

- Crate-ovi koji se koriste u projektima se dobavljaju sa *crates.io.*
- Kako bi se objavio sopstveni crate nephodno je:
  - Napraviti (korisnu) dokumentaciju
  - Kreirati API upotrebom *pub use*
  - Podesiti nalog na crates.io
  - Dodati metapodatke u crate
  - Postaviti crate

Kreiranje (korisne) dokumentacije

- Dokumentacija je namenjena da programerima objasni kako da koriste crate, a ne kako je on implementiran.
  - Dokumentacioni komentar - ///
  - Podrška za markdown
- Primer: *crate-sample*

Testovi u okviru dokumentacije

- U komentare je moguće dodati i primere koda
  - `$ cargo test --doc` - pokretanje kreiranja dokumentacije
    - Pokreće izvršavanje testova
  - `$ cargo doc --open`
    - Generiše i otvara dokumentaciju

Korišćenje *pub use*

- Uz pomoć pub use konstrukta moguće je napraviti dostupnim
  - Konstrukte unutar biblioteke
    - Bez otkrivanja kako je biblioteka organizovana

Podešavanje naloga na crates.io

- Napraviti nalog na crates.io (login via GitHub)
- Potvrditi mail adresu
- Napraviti access token
- `$ cargo login <YOUR-KEY>`

Dodavanje metapodataka

- Neophodno je dodati ili izmeniti podatke u okviru Cargo.toml datoteke
- Crate mora imati definisano
  - Jedinstveno globalno ime
  - Kratak opis
  - Licenca

Objavljivanje crate-a

- `$ cargo publish`
- Objavljivanje je **trajno**!
  - Verzija se ne može prepisati
  - Kod se ne može obrisati
- Moguće je objaviti proizvoljan broj crate-ova.

Ažuriranje crate-a

- Izmeniti verziju nakon promene koda
- Objaviti po standardnom metodu

Yank crate-a

- Crate-ove nije moguće obrisati
  - Ali je moguće onemogućiti dalje korišćenje
    - Zabrana dodavanja u zavisnosti
    - Dosadašnji paketi su u mogućnosti da nastave korišćenje
- `$ cargo yank --vers <verzija>`

Workspace

- Moguće je “ugnježdavati” pakete
- Uz pomoć cargo workspace-a

```
# Cargo.toml (workspace
root)

[workspace]
members = [
    "lib_crate",
    "bin_crate",
]
```

```
my_workspace/
├── Cargo.toml
├── lib_crate/
│   ├── Cargo.toml
│   └── src/
│       └── lib.rs
└── bin_crate/
    ├── Cargo.toml
    └── src/
        └── main.rs
```

Cargo kao package manager

- Moguće je instalirati pakete putem cargo-a
  - `$ cargo install <naziv-paketa>`
- Neophodno je u PATH dodati putanju
  - Do direktorijuma gde se nalaze bin datoteke
- Nakon toga pokretati paket kao i obično

Izvori

- Rust Community. “The Rust Programming Language - the Rust Programming Language.” Rust-Lang.org, 2018, doc.rust-lang.org/book/.
- Rust Team. “Rust Programming Language.” Rust-Lang.org, 2018, www.rust-lang.org/.

Uvod

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