Dr Dinu Dragan

PARALELNE I DISTRIBUIRANE ARHITEKTURE I JEZICI (AKA RUST)

KO IZVODI NASTAVU

Nastavnik:

Van. prof. dr Dinu Dragan

Kabinet: NTP 330

Telefon: ---

E-mejl: dinud@uns.ac.rs

Asistent:

MSc Jovana Jovanović

Kabinet: NTP 328

Telefon: ---

E-mejl: jovana.jovanovic@uns.ac.rs

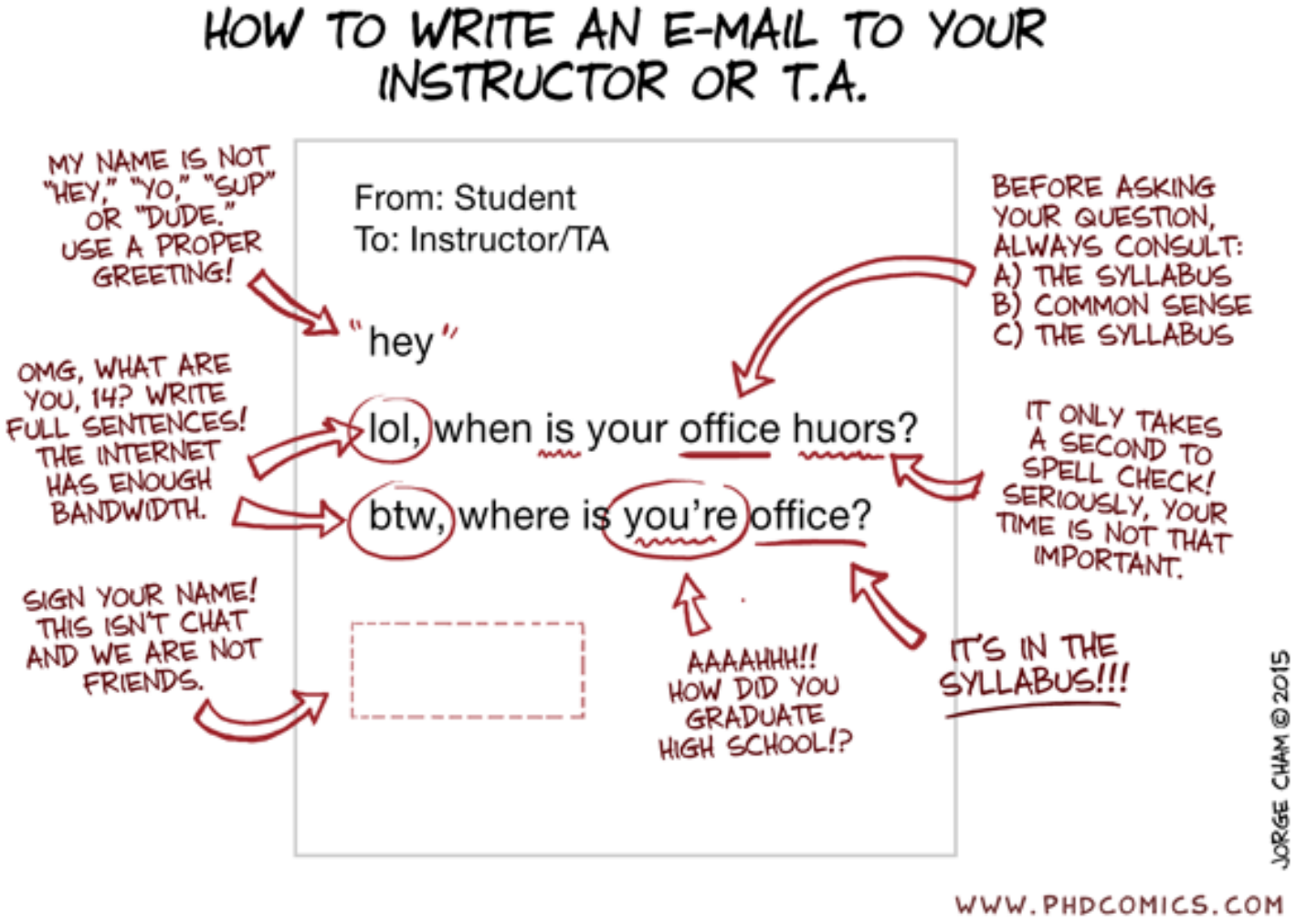
Canvas (http://acs.uns.ac.rs/)

Ako postoje pitanja, iskoristiti za to:

- Slobodno vreme na predavanjima i vežbama
- Individualne konsultacije (dogovor putem mejla)
- E-mejl

Asistent nije dibager !

O PISANJU E-MEJLOVA



O NASTAVNIKU - *Dr Dinu Dragan*

- 1998. - 2003. – Računarska nauka i informatika, FTN, Novi Sad
- 2003. – dipl. ing. “Primena OMR algoritama u DMS sistemima,”
- 2008. – mr – “Enkapsulacija JPEG2000 kompresione tehnike u DICOM standard,”
- 2013. – dr – “Metrika prihvatljivosti kompresione tehnike mirne slike u implementaciji PACS sistema,”
- 2014. docent, 2020. vanr. prof. na PRN, FTN
- Više od 15 godina iskustva na praktičnim projektima: sve od kupovanja bureka do rukovođenja R&D centara i projekata
- **Projekti iz Računarske vizije, Računarske grafike, Veštačke inteligencije, Blokčejna (od konsultacija do proizvoda)**
- **Radio sa Panasonikom, SAPom, Džim Henson Kompanijom, i gomilom startapa za koje niste još čuli …**

NAČIN POLAGANJA

1. Predispitne obaveze **70 poena** (**75** za izuzetan trud)
   - 2 zadatka: **50 poena**
     - **Mini zadatak – 15 poena**
     - **Završni projekat – 35 poena**
   - Seminarski – **20 poena**
   - aktivnost na predavanjima – **5 poena**

2. Usmeni ispit (možda i pismeni), ukupno **30 poena**

- Za potpis potrebno minimum **36 poena** iz predispitnih obaveza

Broj bodova	Ocena
51 – 60	Šest
61 – 70	Sedam
71 – 80	Osam
81 – 90	Devet
91 – 100	Deset

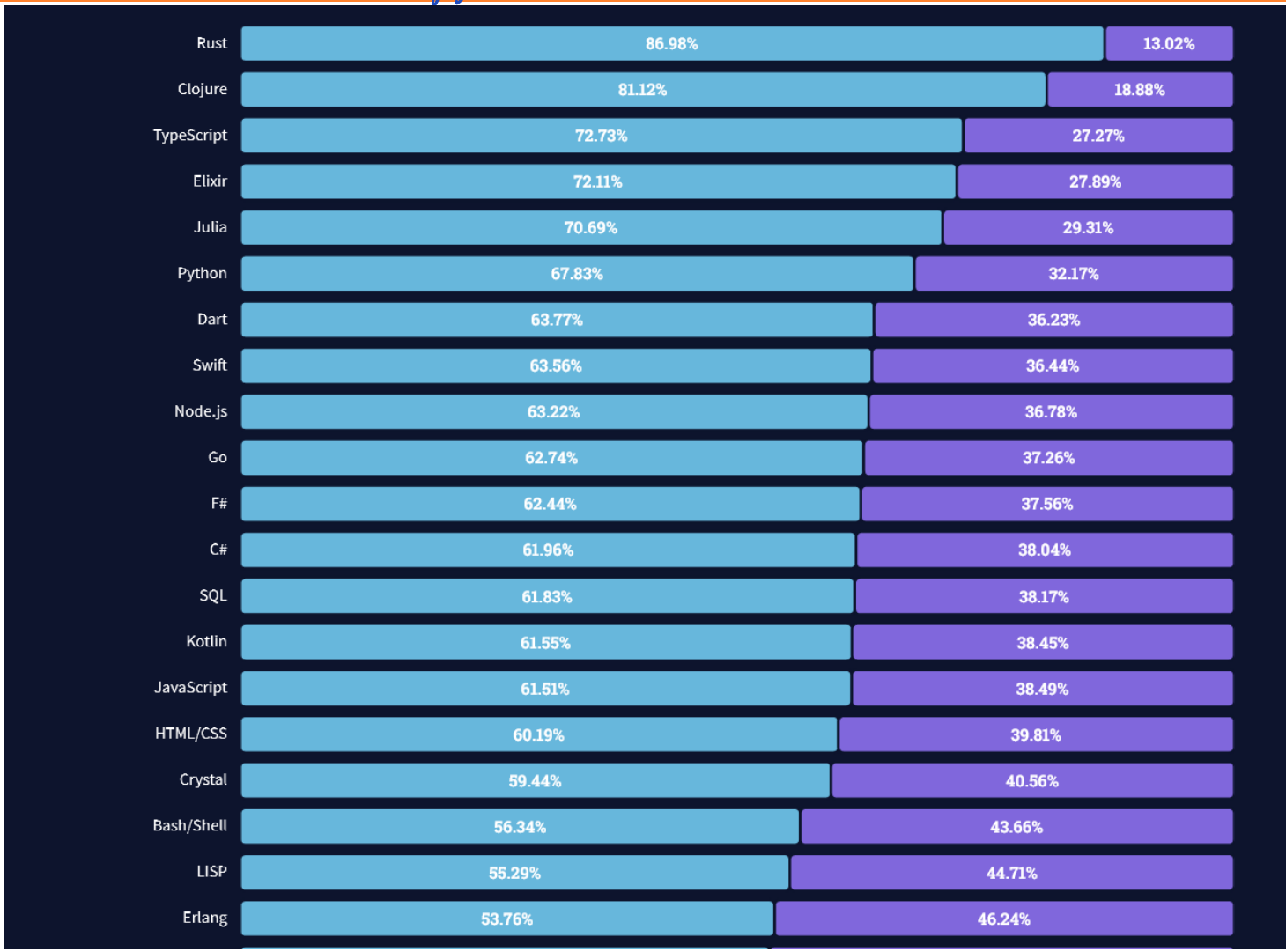
LITERATURA

1. SKRIPTA + BELEŠKE SA PREDAVANJA + Internet
2. Steve Klabnik and Carol Nichols, with contributions from the Rust Community, THE RUST PROGRAMMING LANGUAGE or THE RUST BOOK, https://doc.rust-lang.org/book/.
3. Jim Blandy, Jason Orendorff, and Leonora F.S. Tindall, PROGRAMMING RUST, O'Reilly Media, Inc. 2021, 978-1-492-05259-3.
4. The Rust Community, GETTING STARTED - RUST COMPILER DEVELOPMENT GUIDE, https://rustc-dev-guide.rust-lang.org/

ZAŠTO RUST?

VOLJEN JE

- 6. godina za redom na Stack Overflow-u



https://insights.stackoverflow.com/survey/2021#technology-most-loved-dreaded-and-wanted

RELATIVNO MLAD

- Rust je izrastao iz ličnog projekta koji je Graydon Hoare, zaposlen u Mozili, započeo 2006. godine
- Mozila je počela da sponzoriše projekat 2009. godine i projekat je zvanično objavljen 2010. godine
- Novi Rust kompajler, pod nazivom **rustc**, uspešno je izkompajlirao samog sebe 2011. godine
- Tokom iste godine, rad se prebacio sa inicijalnog kompajlera napisanog u OCaml na kompajler (a self-hosting compiler) baziran na Rust jeziku
  - **Ekstremno važno!** Samo se prva verzija kompajlira u OCaml kompajleru, sve ostale su se kompajlira u **Rust kompajleru**!
  - U čemu se inače kompajliraju druge verzije programskih jezika?!
- Prva numerisana pre-alfa verzija kompajlera, Rust 0.1, objavljena je u januaru 2012. godine

PODRŠKA I ZAJEDNICA

- 8. februara 2021. godine, formiranje Rust fondacije objavilo je pet kompanija osnivača (AVS, Huavei, Google, Microsoft i Mozilla)
- Google je najavio podršku za Rust u okviru svoj Android Open Source Project kao alternativu za C/C++
- Prvi jezik koji je dodat (decembar 2022.) u Linux karnel od kada je napisan u C programskom jeziku (1991.)!
- Vrlo aktivna zajednica gde je ciklus dodavanja novih funkcija/ispravki bagova u rasponu do 6 nedelje
  - Ali što je važnije, izmene ne kvare interoperabilnost sa starim kodom (pričaćemo kad budemo pričali o kompajleru i zašto)
- Koriste ga veliki: Firefox, Dropbox, Cloudflare, Discord, Microsoft Azure, Polkadot, Solana, Facebook, Google
- Tokom prošle godine i kroz ovu bilo je malo problema oko upravljanja fondacijom

ŠTA JE RUST?

PROGRAMSKI JEZIK

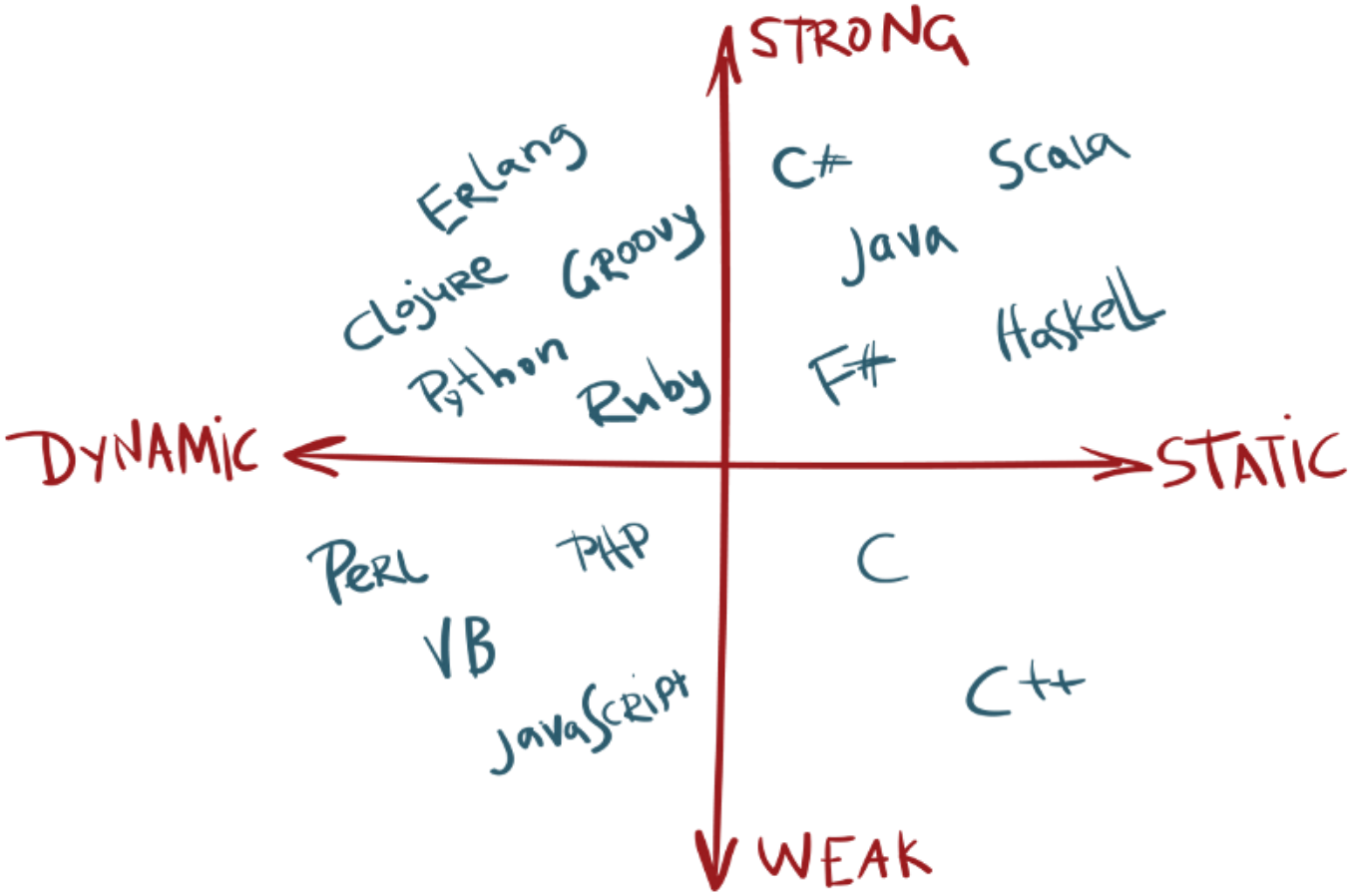
- Treća generacija, očigledno, programski jezik opšte namene
- Podržava više programskih paradigmi
  - (zapamtiti ovo, jer će pitanje na kraju da bude koje?)
- Akcenat na je na performansama (**performance**), sigurnosti napisanog koda (**type safety**) i konkurentnosti (**concurrency**), kao i na upravljanju memorijom (**memory management**)
  - Poseban osvrt na sigurnoj konkurentnosti (**safe concurrency**)
- Kompajlerski programski jezik strogo tipiziran sa statičkim tipovima
  - Šta to znači?
- Iako treća generacija, on je ipak jezik niskog nivoa apstrakcije sa direktnim pristupom hardveru i memoriji
  - Šta to znači?
- Izrazito brz

Kompajlerski vs. interpreterski programski jezici

- Diskusija, sve bi ovo trebali da znate

Jako vs. slabo tipizirani programski jezici

- Diskusija, sve bi ovo trebali da znate



Statički vs. dinamički tipizirani programski jezici

- Diskusija, sve bi ovo trebali da znate

Statically Typed Languages
Type checking is completed at compile time
Explicit type declarations are usually required
Errors are detected earlier
Variable assignments are static and cannot be changed
Produces more optimized code

Dynamically Typed Languages
Type checking is completed during runtime
Explicit declarations are not required
Type errors are detected later during execution
Variable assignments are dynamic and can be altered
Produces less optimized code, runtime errors are possible

ZAŠTO JE OVO SVE BITNO?

- Pomoć od strane kompajlera i njegove poruke (**Compiler feedback**)
- Ako imate jednostavan sistem tipova, kompajler ne može da uhvati mnogo različitih grešaka umesto vas
- Rust greške se dešavaju u vreme kompajliranja, na samoj mašini programera, a ne negde na serveru ili na mašini korisnika
- Rust kompajler je poput instruktora vožnje, nežno vas podučava kako da se krećete opasnim autoputem
- Ako ste napravili kompajler koji iscrpno razume vašu memoriju, onda ste napravili kompajler koji iscrpno razume vaš kod

ZAŠTO JE UPRAVLJANJE MEMORIJOM BITNO?

- Važnost pravilnog upravljanja memorijom postaje odmah očigledna kada se uzme u obzir da su u poslednjih 12 godina oko 70% svih bezbednosnih grešaka u Microsoft proizvodima bili problemi vezani za bezbednost memorije (**memory safety issues**) https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/
- Isti broj je prijavljen i za Google Chrome
- Šta nas briga kod programa sa garbage collector-om?
  - Šta mislite u čemu su pisani njihovi kompajleri?
- Šta je ‘use after free errors?’
  - dešava se kada program nastavi da koristi pokazivač nakon što je memorija na koju pokazuje oslobođena, npr. poziva se lambda funkcija nakon što se oslobode sve reference na dati objekat
- Safe Rust and Unsafe Rust (2 načina za pisanje Rust programa)

ZAŠTO JE UPRAVLJANJE MEMORIJOM BITNO?

- Rust ima proveru pozajmljivanja (**borrow checker**), deo kompajlera koji obezbeđuje da reference ne nadžive podatke na koje se odnose
- Ova funkcija pomaže eliminisanje grešaka nastalih usled lošeg upravljanja memorijom i nedozvoljenog pristupa memoriji (**memory violation bugs**)
- Takvi problemi se otkrivaju u vreme kompajliranja i zato je prikupljanje smeća nepotrebno
- U Rustu svaka referenca ima životni vek, gde možete podesiti doseg za koji je ta referenca važeća, ovo rešava problem sa referencama koje više nisu važeće, a takođe razlikuje Rust od C i C++.
- Safe Rust and Unsafe Rust (2 načina za pisanje Rust programa)

  1. Nameću se dodatna ograničenja programeru (npr. upravljanje vlasništvom nad objektima), čime se osigurava da kod ispravno funkcioniše.
  2. Daje programeru veću autonomiju (npr. može da radi na sirovim pokazivačima nalik C progr. jeziku), ali kod može da se pokvari.

NEDEFINISANO PONAŠANJE – C/C++

- *Behaviour, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements*
- Nedefinisano ponašanje (undefined behavior) nema samo nepredvidiv rezultat: standard eksplicitno dozvoljava programu da uradi bilo šta
- C i C++ 100 imaju preporuke za izbegavanje nedefinisanog ponašanja. Uglavnom zdrav razum: ne pristupajte memoriji kojoj ne bi trebalo, ne dozvolite da aritmetičke operacije rezultuju izrazom van opsega, ne delite sa nulom itd. Ali kompajler ne primenjuje ova pravila!
- Odgovornost za izbegavanje nedefinisanog ponašanja u potpunosti pada na programera – **zato nema šanse da to izbegnemo**
- Skoro svi C/C++ programi izvršavaju određene oblike nedefinisanog ponašanja (Univerzitet Juta, istraživač Peng Li je modifikovao C i C++ kompajlere kako bi pratio ponašanje implementiranog koda)

NEDEFINISANO PONAŠANJE – RUST?

- Povremena čudna poruka ili rušenje mogu biti problem kvaliteta, ali nenamerno nedefinisano ponašanje je takođe bio glavni uzrok bezbednosnih propusta još od Morris Crva iz 1988. godine
- Rust programski jezik daje jednostavno obećanje:
  - ako vaš program prođe proveru kompajlera, nema u sebi nedefinisanog ponašanja
  - Viseći pokazivači, dvostruko oslobađanje memorije i dereferenciranje nula pokazivača se detektuju tokom kompajliranja
  - Reference na niz su obezbeđene mešavinom provera tokom kompajliranja i izvršavanja, tako da nema prekoračenja bafera ili prepisivanja preko memorije (Rust detektuje te situacije i izlazi se iz programa bezbedno sa porukom o grešci)
- Rust nameće više ograničenja kodu nego C i C++, a ova ograničenja zahtevaju praksu i iskustvo da biste se navikli

PODRŠKA KONKURENTNOM PROGRAMIRANJU

- Konkurentnost je notorno teško pravilno koristiti u C i C++
- Programeri se obično okreću paralelnom programiranju samo kada se pokaže da jedna niz izvršavanja nije u stanju da postigne performanse koje su potrebne
- Ali paralelizam je previše važan za savremene računare da bi ga tretirali kao krajnje sredstvo
- Rust takođe olakšava pisanje konkurentnih programa sprečavanjem trke u podacima (**data races**) tokom samog kompajliranja
  - Podaci se trkaju? Šta? Šta je to?
  - *Ova situacija se dešava kada najmanje dve različite instrukcije iz različitih niti pokušavaju da istovremeno pristupe istoj memorijskoj lokaciji, dok bar jedna od njih pokušava da napiše nešto i ne postoji sinhronizacija koja bi mogla da postavi bilo kakav poseban red među različitim pristupima*
  - *Pristup memoriji bez sinhronizacije je nedefinisan*

PODRŠKA KONKURENTNOM PROGRAMIRANJU

- Ista ograničenja koja obezbeđuju bezbednost memorije u Rustu takođe obezbeđuju da Rust programi nemaju trke u podacima
- Možete slobodno deliti podatke između niti, sve dok se podaci ne menjaju
- Podacima koji se menjaju može se pristupiti samo pomoću primitiva za sinhronizaciju
- **Dostupni su svi tradicionalni alati za konkurentnost:**
  - mutexes,
  - condition variables,
  - channels,
  - atomics, itd.
  - **Rust jednostavno proverava da li ih pravilno koristite**

PODRŠKA KONKURENTNOM PROGRAMIRANJU

- **U Rustu se otkrivaju trke u podacima!**
  - Ako pristup datom objektu ne podržava mnogo niti (tj. nije označen odgovarajućom osobinom), treba ga sinhronizovati pomoću muteksa koji će zaključati pristup ovom konkretnom objektu za druge niti
  - Kako bi se osiguralo da operacije izvršene na objektu neće nešto pokvariti, samo jedna nit ima pristup objektu
  - Iz perspektive drugih niti, operacije na ovom objektu su atomične (atomic)
- Uočeno stanje objekta je uvek ispravno i ne može se posmatrati nikakvo međustanje koje je rezultat operacije koju je na ovom objektu izvršila druga nit
- **Rust jezik može da proveri da li vršimo bilo kakve pogrešne operacije na takvim objektima i obavestiti nas u vreme kompajliranja**

ŠTA JE TO ATOMIČNI OBJEKAT/OPERACIJA?

- Atomične operacije su od kritične važnosti kada se radi sa deljenim resursima
- Primer: Mika i Žika pokušavaju da kupe artikal na Internetu koristeći isti bankovni račun sa saldom od 1000 RSD:
  1. Mika klikne „Kupi sada“ na Lego kompletu od 1000 RSD.
  2. Bankarski softver proverava da li se u bazi podataka računa nalazi najmanje 1000 RSD i potvrđuje da ima dovoljno za kupovinu.
  3. Žika klikne „kupi sada“ na video igrici od 500 RSD.
  4. Bankarski softver proverava da li se u bazi podataka računa nalazi najmanje 500 RSD i potvrđuje da ima dovoljno za kupovinu igre.
  5. Banka povlači 1000 RSD da plati Mikin Lego set. (Stanje == 0 RSD)
  6. Banka povlači 500 RSD da plati Žikinu video igricu. (Stanje == -500 RSD)
- Da bi se ispravio ovaj problem, bankarskom softveru bi bio potreban neki oblik međusobnog isključivanja kako bi se osiguralo da je višestepeni proces provere sredstava i povlačenja sredstava nedeljiv, npr. **zaključavanje** (**lock** mehanizam).

ŠTA JE TO ATOMIC OBJECT/OPERATIONS?

- Postavljamo **lock** (zaključamo resurs) pre pristupa deljenom resursu, a zatim ga otpuštamo kada završimo, druge niti moraju da čekaju dok se zaključavanje ne oslobodi, kod unutar bloka se naziva kritična sekcija

```
acquire lock
  /* Entering critical section */
  if $100 is availabile
    withdraw $100
  /* Leaving critical section */
release lock
```

- Atomičnost je suštinska karakteristika mnogih računarskih sistema:
  - kod sa više niti,
  - baze podataka,
  - paralelna obrada,
  - upravljanje memorijom itd.

ŠTA JE TO ATOMIC OBJECT/OPERATIONS?

- Kada bi softver banke koristio gornji algoritam:

1. Mika klikne „kupi sada“ na Lego kompletu od 1000 RSD.
2. Mikin zahtev zaključa račun, proverava da li se u bazi podataka naloga nalazi najmanje 1000 RSD i potvrđuje da ima dovoljno za kupovinu.
3. Žika klikne „kupi sada“ na video igrici od 500 RSD.
4. Softver za bankarstvo čeka na mogućnost da zaključa račun—trenutno se zaključavanje koristi na Mikin zahtev.
5. Banka povlači 1000 RSD da plati Mikin Lego set (Stanje == 0 RSD) i otpušta zaključavanje (tj. bravu).
6. Žikin zahtev preuzima zaključavanje (zaključa račun), proverava da li je u bazi podataka naloga najmanje 500 RSD i vidi da nema dovoljno.
7. Žika nije u mogućnosti da kupi video igricu.

KOLABORACIJA, ALATI

- **Cargo**, Rustov menadžer paketa (**packet manager**) i alat za pravljenje (**build tool**), kao i **crates.io**, Rust-ovo javno skladište paketa, su Rust-ov odgovor na NPM
- Jednostavno dodajte ime biblioteke i potreban broj verzije u datoteku, a Cargo se brine o preuzimanju biblioteke, zajedno sa bilo kojim drugim biblioteke koje koristi zauzvrat, i povezujući čitav niz zajedno
- Rust-ove osobine (**traits**) i generici (**generics**) vam omogućavaju da kreirate biblioteke sa fleksibilnim interfejsima tako da mogu da služe u mnogo različitih konteksta
- Rust-ova standardna biblioteka pruža osnovni skup osnovnih tipova (a core set of fundamental types) koji uspostavljaju zajedničke konvencije za uobičajene slučajeve, čineći različite biblioteke lakšim za zajedničko korišćenje

NIJE SVE TAKO RUŽIČASTO

- Rust je relativno nova tehnologija
- Može biti neugodno dobijati mnogo poruka o grešci i razvoj koda nije tako brz kao u popularnijim programskim jezicima
- Međutim, Rust programeri daju sve od sebe da ove poruke o grešci učine što informativnijim i efikasnijim
- Kompajliranje zna da traje jako dugo

RUST vs. C++

- Odličan pregled može se naći na https://www.apriorit.com/dev-blog/520-rust-vs-c-comparison

ZAŠTO RUST?

- Na kraju, treba shvatiti da Rust ne izmišljava ništa novo
- Uzima 50+ godina iskustva u razvoju sistemskog softvera i konkurentnih programa u C/C++ jezicima i primenjuje sve ono što se naučilo do sada u jedan savremen programski jezik koji je siguran i lagan za korišćenje
- Ipak, on je ujedno i vrlo fleksibilan i ekspresivan, što znači da se može lako proširivati i prilagoditi različitim potrebama a da pri tome ne žrtvuje ništa od svoje sigurnosti
- Takođe, ima tzv. zero-cost abstraction
  - *Zero Cost Abstractions means adding higher-level programming concepts, like generics, collections and so on do not come with a run-time cost, only compile time cost (the code will be slower to compile)*

PRIMER


```rust
use std::io;
use rand::Rng;
use std::cmp::Ordering;

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1..=100);

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

        io::stdin()
            .read_line(&mut guess)
            .expect("Failed to read line");

        let guess: u32 = match guess.trim().parse() {
            Ok(num) => num,
            Err(_) => continue,
        };

        println!("You guessed: {guess}");

        match guess.cmp(&secret_number) {
            Ordering::Less => println!("Too small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal => {
                println!("You win!");
                break;
            }
        }
    }
}
```

CARGO I INSTALACIJA

- Rust se instralira najčešće kroz **rustup**, https://rustup.rs/

  $ rustc --version

  $ rustdoc --version

- Cargo je Rustov sistem za bildovanje i menadžer paketa
- Koristi se za bildovanje koda, preuzimanje biblioteka od kojih kod zavisi i pravljenje tih biblioteka (dependencies)

  $ cargo --version

  $ cargo new <ime_projekta>

  $ cd <ime_projekta>

  $ cargo build

  $ cargo run

  $ cargo check

- Visual Studio Code