Rust peliohjelmoinnissa

Victor Bankowski, Antti Karjalainen ja Janne Pulkkinen

Seminaariraportti
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 8. lokakuuta 2017

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Victor Bankowski, Antti Karjalainen ja Janne Pulkkinen

Työn nimi — Arbetets titel — Title

Rust peliohjelmoinnissa

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Seminaariraportti	8. lokakuuta 2017	4

Tiivistelmä — Referat — Abstract

Tiivistelmä.

Avainsanat — Nyckelord — Keywords
avainsana 1, avainsana 2, avainsana 3

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Rust on käännettävä ohjelmointikieli, jonka kehitystä tukee Mozilla-säätiö (Anderson et al., 2016). Mozilla käyttää kieltä uuden rinnakkaisuutta hyödyntävän Servo -internet-selainmoottorin ohjelmointiin [ja lisäksi käytetään missä?]. Käännettävänä ohjelmointikielenä C ja C++ -kielten tavoin Rust mahdollistaa suorituskykyä ja hallittua muistin käyttöä vaativien sovellusten kehittämisen esimerkiksi sulautetuissa järjestelmissä. Edellä mainituista kielistä poiketen Rust kuitenkin estää yleisiä C-kielissä esiintyviä muistinhallintaa ja kilpatilanteita koskevia ongelmia, mahdollistaen kuitenkin vastaavan suorituskyvyn ajettavassa ohjelmassa. Rust ratkaisee nämä ongelmat käyttämällä muistinhallinnassa omistajuuden ("ownership") ja lainaamisen ("borrowing") käsitteitä. Tämä estää mahdolliset virhetilanteet jo ohjelman käännösvaiheessa vaatimatta virtuaalikoneen, kääntäjän tai tulkin käyttöä ohjelman suorituksen aikana.

2 Historia

Rust-kielen kehitys alkoi vuonna 2006 Graydon Hoaren sivuprojektina, jollaisena se jatkui yli kolmen vuoden ajan[1]. Mozilla-säätiö osallistui kehitykseen ensimmäisen kerran vuonna 2009 ja on tukenut ohjelmointikielen kehitystä siitä lähtien. Nykyisin kieltä kehittävä ryhmä – *The Rust Team* – jakautuu osaryhmiin, jotka vastaavat kielen eri osa-alueista. Osa-alueisiin kuuluvat esimerkiksi kääntäjän kehittäminen, kielen ominaisuuksien suunnittelu ja dokumentaatio.

Suurimpiin Rustia käyttäviin projekteihin kuuluu Mozillan kehittämä Servo -web-selainmoottori. Sen tavoitteisiin kuuluu sivun piirtämisen, HTML-datan parsimisen ja muiden web-selaimen piiriin kuuluvien tehtävien rinnakkaistaminen[2]. Servo-projektiin kuuluva CSS-moottori Stylo on otettu käyttöön Mozilla Firefox -selaimen uusissa kehitysversioissa[3].

Muihin Rust-kieltä hyödyntäviin organisaatioihin kuuluu muun muassa Dropbox ja Canonical[4].

[1] https://www.rust-lang.org/en-US/faq.html

[2] https://hacks.mozilla.org/2017/08/inside-a-super-fast-css-engine-quantum-css-aka-stylo/

[3] https://blog.mozilla.org/blog/2017/09/26/firefox-quantum-beta-developer-edition/

[4] https://www.rust-lang.org/en-US/friends.html

3 Perusteet

```
fn main() {
    println!("Hei maailma");
}
```

```
$ cargo run
Hei maailma
```

Koodi 1: Tulostaa "Hei maailma"

Ohjelmointikieleen tutustuessa on tapana kirjoittaa klassinen "Hei maailma-ohjelma joka tulostaa kyseisen lauseen. Koodi 1 on kyseisen ohjelman Rust toteutus. Kyseinen toteutus ei poikkea juurikaan muiden proseduraalisten kielien toteutuksista. Suurin ero muiden kielien toteutuksiin on se että tulostuskomento on makro. Tulostuskomento on toteutettu makrolla tekstin muotoilun helpottamiseksi. Koodissa 2 on esimerkki tästä.

```
fn main() {
    let s_luku: i32 = 4; // Valittu reilulla nopanheitolla.
    println!("Satunnaislukusi on {}", s_luku);
}
```

Koodi 2: Tulostaa "Hei maailma"

Rustissa muuttujat määritellään käyttäen **let** avainsanaa ja muuttujan tyyppi erotellaan kaksoipistellä. Koodissa 2 muuttuja nimeltä *s_luku* on 32-bittinen etumerkillinen kokonaisluku. Muuttujat oletusarvoisesti eivät ole muokattavissa. Muokattavat muuttujat määritellään käyttäen avainsanayhdistelmää **let mut**.

```
fn factorial(n: u64) -> u64 {
    let mut result = 1;
    for i in 1..(n + 1) {
        result *= i;
    }
    return result;
}
```

Koodi 3: Funktio joka laskee $n!$ iteratiivisesti.

Useimmissa tapauksissa muuttujan tyypin voi jättää merkkaamatta, koska Rust osaa päätellä sen käännösaikana. Kuitenkin funktioiden parametrien

ja palautusarvon tyypit täytyy merkata, koska Rustissa ei ole ohjelmanlaajuista tyyppipäättelyä. Funktion palautusarvon tyyppi määritellään nuolen -> jälkeen. Koodissa 3 funktion parametri n ja palautusarvo ovat 64-bittisiä etumerkittömiä kokonaislukuja.

Rustin **for**-silmukat ovat *for–each*-tyyppisiä, jossa käydään iteraattorin kaikki alkiot läpi. Esimerkiksi koodissa 3 käydään kaikki välin $[1, n + 1)$ kokonaislukuarvot läpi.

Rustissa ei tarvitse käyttää **return** avainsanaa, jos arvo palautetaan funktion lopussa. Tällöin ei myöskään merkata puolipistettä.

```
fn factorial(n: u64) -> u64 {
    if n == 0 {
        1
    } else {
        n * factorial(n-1)
    }
}
```

Koodi 4: Funktio joka laskee $n!$ rekursiivisesti.

3.1 Omistajuus

Ohjelmointikielet vaativat muutt(Klabnik & Nichols, 2018)ujien luomisen lisäksi niiden poistamisen. Käännetyissä kielissä tämä voi vaatia ohjelmoijalta muuttujien manuaalisen poistamista, kun niitä ei enää käytetä. Esimerkiksi C käyttää manuaalista muistinhallintaa. Useissa tulkkia käyttävissä kielissä muuttujaan kohdistuvia viittauksia lasketaan suorituksen aikana; kun muuttujaan ei enää viitata, automaattinen roskienkeräys ennen pitkää löytää käyttämättömän muuttujan ja poistaa sen. Tämä ei vaadi ohjelmoijalta yleensä lisätöitä, mutta roskankerääjä lisää ohjelman resurssivaatimuksia. Roskienkeräystä käyttäviin kieliin kuuluvat muun muassa Python ja Java.

Rust varmistaa muuttujien muistinhallinnan käyttämällä omistamisen käsitettä, joka ei vaadi suorituksen aikaisen roskankeruun käyttöä, eikä myöskään muuttujien manuaalista poistamista. Kun muuttuja luodaan missä tahansa osassa koodia, sille määritellään omistaja muuttujan vaikutusalueen perusteella. Esimerkiksi metodissa main luodut muuttujat saavat omistajakseen kyseisen metodin. Kun kyseisen main-metodin suoritus on päättynyt, kyseisen metodin omistamat muuttujat poistetaan välittömästi ja niiden varaama muistialue palautetaan järjestelmän käyttöön. Tätä periaatetta on havainnollistettu esimerkissä 2.

3.2 Lainaminen ja elinajat

4 Seuraava juttu

Lähteet

Anderson, B., Bergstrom, L., Goregaokar, M., Matthews, J., McAllister, K., Moffitt, J., & Sapin, S. (2016). Engineering the servo web browser engine using rust. Teoksessa *Proceedings of the 38th international conference on software engineering companion* (s. 81–89). New York, NY, USA: ACM. doi: 10.1145/2889160.2889229

Klabnik, S., & Nichols, C. (2018). *The rust programming language*. No Starch Press. Lainattu saatavilla `https://doc.rust-lang.org/book/second-edition`