

Rust peliohjelmoinnissa

Victor Bankowski, Antti Karjalainen ja Janne Pulkkinen

Seminaariraportti
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 24. syyskuuta 2017

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		24. syyskuuta 2017	3
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	1
2 Historia	1
3 Perusteet	1
3.1 Omistajuus	3
4 Seuraava juttu	3
Lähteet	3

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 suorituskyykyä 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 suorituskyyvyn 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. 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.

3 Perusteet

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

Listing 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 kielten toteutuksista. Suurin ero muiden kielten 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);
}
```

Listing 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) {
    let mut result = 1;
    for i in 1..(n + 1) {
        result *= i;
    }
    result
}
```

Listing 3: Funktio joka laskee $n!$

Useimmissa tapauksissa muuttujan tyyppin voi jättää merkkeamatta, koska Rust osaa päätellä sen käännösaikana. Kuitenkin funktioiden parametrien tyypit täytyy merkata, koska Rustissa ei ole ohjelmanlaajuista tyyppipäätelyä.

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 main() {
    let x = 42;
    let mut y = 7;
    y += 1;
    if y == 8 {
        for i in 0..x {
            println!("y on 8 ja {} on pienempi kuin x", i);
        }
    }
}

```

Listing 4

3.1 Omistajuus

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