Ohjelmointikielet ja -paradigmat 2017

Sakari Snäll 507267
Antti Vainikka

April 10, 2017

1 Johdanto

Toteutimme harjoitustyön kolmannen tehtävän. Valitsimme toteutettavaksi peliksi pokerin koska järkevän toteutuksen tekeminen tutustuttaisi meidät kielen tapaan luoda struktuureita, lukea syötettä standardista syöteputkesta ja joutuisimme toteuttamaan pelijatkumon funktionaalisesti. Alkuperäisen tietämyksemme perusteella pelin ohjelmoiminen on funktionaalisesti vaikeampaa verrattuna imperatiiviseen tapaan ohjelmoida koska peleissä on tila, jota pitää muuttaa usein. Useat pelienohjelmointiin liittyvät kirjat käyttävät objektiorientoitunutta paradigmaa, joten ohjelmointityö antaa meille myös tuntumaa miksi funktionaalista paradigma ei ole suosittu ainakaan pelien ohjelmoinnissa.

2 Funktionaalinen paradigma

Funktionaalinen paradigma on ohjelmointiparadigma. Funktionaalisen paradigman omainaisuuksia ovat esimerkiksi muuttumattomat tietorakenteet ja laiskalaskenta. On vaikea määrittää onko ohjelmoinkieli funktionaalinen vai ei. Useimmissa ohjelmointikielissä on olemassa mahdollisuus luoda anonyymejafunktioita ja sen pohjalta on vedetty johtopäätös, että kieli olisi funktionaalinen. Esimerkiksi Java8 kutsutaan funktionaaliseksi koska siihen lisättiin lyhenne paikallaan luodulle objektille, joka toteuttaa jonkin rajapinnan. Funktionaaliset ohjelmointikielet voidaan jakaa kuitenkin puhtaasti funktionaalisiin ja epäpuhtaisiin. Puhtaus funktionaalisissa ohjelmointikielissä viittaa siihen

voidaan kielessä toteuttaa sivuvaikutuksia. Täysin funktionaalisia ohjelmointikieliä on esimerkiksi Haskell ja Clear. ML pohjaiset kielet ovat funktionaalisia mutta esimerkiksi standardissa ML:ssä ja Ocamlissa voidaan asettaa muuttujalle arvo. Jos halutaan olla tarkkoja niin myös Haskelissa tapahtuu sivuvaikutuksia koska esimerkiksi tekstin tulostaminen sisältää sivuvaikutuksen. Täydellisesti sivuvaikutuksellisessa ohjelmassa ei kuitenkaan ole järkeä koska tarpeeksi tehokas kääntäjä voisi tuottaa ohjelman, joka ei suorita mitään koska sivuvaikutuksia ei tehdä.

```
(let loop ((n 0))
   (cond
      ((= n 10))
      (#t
        ...
        (loop (+ n 1)))))
```

```
for(int n = 0; n < 10; n++) {
        ...
}
```

2.1 Sulkeumat objektien sijaan

Ohjelmointikielissä joissa funktioon voidaan sisällyttää ympäristö(closure), pystytään triviaalisti toteuttamaan objektisysteemi. Yksi mahdollinen tapa toteuttaa objektisysteemi olisi tehdä objektin luova funktio, joka palauttaisi funktion, joka ottaa listan parametreja. Listan ensimmäinen parametri voisi olla metodi ja loput metodin vaatimia parametreja.

2.2 Mahdolliset hyödyt

2.3 Mahdolliset vaikeudet

3 Lisp

Lispejä pidetään yleensä funktionaalisina mutta ne ovat todellisuudessa moniparadigmaisia. Common Lisp on tunnetuista Lispeistä tulkintamme mukaan vähiten funktionaalinen koska sen standardi ei edellytä häntärekursion optimointia. Common Lispissä on myös useita sivuvaikutuksellisia funktioita.

Schemessä on myös sivuvaikutuksellisia funktioita mutta ainakin standardin määrittävät sivuvaikutukselliset funktiot päättyvät aina huutomerkkiin. Schemen standardi myös määrittää, että toteutuksen täytyy tukea häntärekursion optimointia. Schemen keskustelufoorumeilla myös kokemuksen perusteella suositaan enemmän rekursion käyttöä, toisin kuin Common Lispin ohjelmoijat, jotka tuntuvat hyödyntävän enemmän imperatiivisia silmukka-rakennetta.

3.1 Clojure

Clojure on ohjelmointikieli, joka on saanut paljon vaikutteita Common Lispistä. Clojure ei vältämättä tue häntärekursion optimointia. Clojure kuitenkin eroaa Schemestä ja Common Lispistä siinä, että Clojuressa perus datarakenteet ovat muuttumattomia.

Muut Lisp-ohjelmoijat ovat usein kriittisiä Clojuren suhteen koska siinä käytetään kaikkia eri tyyppisiä sulkeita pelkkien tavallisten sulkeiden sijaan.

3.2 Funktionaalinen Clojure

4 Harjoitustyö

Tehtävänannossa mainittiin, että ohjelmakoodin tulisi olla idiomaattista valitulle ohjelmointikielelle. Emme ole täysin varmoja onko ohjelmakoodi täysin idiomaaattista mutta koska työn oli tarkoitus olla harjoitus funktionaaliseen paradigmaan, toteutimme koodin niin, että siinä ei muuteta muuttujia ollenkaan.

4.1 Funktionaalisuudesta saadut hyödyt

4.2 Funktionaalisuuden haitat