

Mikko Metsäranta
515662
19.04.2018

Ohjelmointikielet ja -paradigmat

Harjoitustyöraportti 2018

Pokeri-peli

Yleiskuvaus

Rakensin harjoitustyönäni komentorivillä toimivan yksinkertaisen pokeri-pelin. Pokeri-peli toimii täysin tekstipohjaisena ja sitä voi pelata 1-5 henkilöä. Peli ei ole täysin virheetön: esimerkiksi jos pelaajilla on sama pari niin haita (korkeinta korttia) ei tarkisteta. Kuitenkin tavallisesti korkein pari, suora, täyskäsi ynnä muut voittavat. Ohjelma on interaktiivinen ja kysyy pelaajalta käskyjä (joiden vastaukset täytyy olla syntaktisesti oikein). Minkäänlaista virheidenkäsittelyä ei ole toteutettu ja virheellisestä käskystä ohjelman suoritus päättyy.

Ohjelma on ohjelmoitu Erlang-kielellä. Kieli on toteuttu alunperin Ericssonin tietoliikennetarpeisiin. Vaatimuksia suurelle teleoperaattorille oli muun muassa korkea suoritusteho ja erittäin vakaa alusta. Erlangista tuliki Ericssonin pääkielistä heidän kehittäessä järjestelmiä, joille sallittiin maksimissaan muutaman minuutin pituisia katkoja käytettävyydessä kokonaisen vuoden aikana. Muita ominaisuuksia Erlangissa on koodin päivittäminen ilman ohjelman pysäytystä (hot code reloading), valvojat (supervisors), prosessit, asynkrooninen viestin välitys (asynchronous message passing) ja klusterointi.

Käytetty paradigma

Erlang on paradigmataan funktionaalinen kieli. Funktionaalisen ohjelmoinnin erityispiirre on sen matemaattisuus, tai tarkemmin sanottuna sen käyttämä lambda-laskento. Mikä erottaa funktionaalisen kielen muista kielistä on se, että funktio palauttaa aina saman tuloksen samalla syötteellä. Toisin sanoen funktiolla ei ole tilaa, mikä voisi vaikuttaa syötteelle saatavaan tulokseen. Funktioilla ei puhtaassa funktionaalisessa ohjelmoinnissa saa olla mitään sivuvaikutuksia. Sivuvaikutuksia voi kuitenkin käytännön takia olla, esimerkiksi I/O -operaatiot joita käytettiin tässäkin harjoitustyössä.

Toinen tärkeistä ominaisuuksista funktionaalisille kielille on datan muuttumattomuus. Tämä tarkoittaa sitä, ettei muuttujien arvot voi muuttua. Jos muttujaan yritetään asettaa uutta arvoa, aiheuttaa se virheen. Tämä luo uniikin

tarpeen miettiä, miten tietoa tallenetaan ja käsitellään. Pitkälti funktionaalisella kielellä kirjoitettu ohjelma on putkimainen rakenne, jossa funktion tuloksia ”putkitetaan” jatkuvasti seuraavalle funktiolle.

Funktionaalisia kieliä voidaan myös jakaa laiskoihin ja innokkaisiin laskijoihin. Laiskat laskija -kielet suorittavat laskentaa vasta, kun laskennan tulosta tarvitaan. Innokas laskija -kielissä laskenta suoritetaan välittömästi. Analogiana voitaisiin ottaa esimerkki vaikka kotitehtävien tekemisestä. Laiskassa laskennassa kaivaisimme repusta oppikirjan, penaa lin ja vihon pöydälle. Tämän jälkeen avaisimme oppikirjan oikeasta kohdasta, avaisimme penaa lin ja ottaisimme kynän ja viimeiseksi kirjoittaisimme vihkoon. Etsimme siis tarvittavat välineet esiin, mutta koskemme niihin vasta kun niitä tarvitaan. Innokkaassa laskennassa ottaisimme kirjan esiin ja heti avaisimme oikean tehtävisivun. Sen jälkeen nostaisimme penaa lin pöydälle, avaisimme sen ja ottaisimme kynän esiin. Lopuksi ottaisimme vihkon ja alkaisimme kirjoittamaan siihen. Suorittaisimme siis jokaisen vaiheen välittömästi ”loppuun asti”.

Yksi ominainen piirre vielä funktionaaliselle kielelle on sen kontrollirakenteet. Esimerkiksi Erlangissa ei ole muista kielistä tutuja for- ja while-silmukoita. (Toki on tietenkin mahdollista rakentaa silmukoiden toiminnallisuus itse, mutta kieli ei itsessään tarjoa näitä rakenteita.) Sen sijaan käytössä on rekursio. Silmukoiden ja muuttujien muuttumattomuus loivat ainakin minulle muutamia pysähtymisen hetkiä, kun täytyi miettiä miten onglemia täytyy nyt lähteä ratkaisemaan.

Erlangin perusteita

Erlangin on sanottu olevan vaikea kieli oppia. Tämä johtuu muun muassa siitä, ettei netissä ole tarjolla juurikaan aloittelijalle sopivaa materiaalia. Myöskin Erlangin oma dokumentaatio on todettu melko huonoksi ([video Erlang-konferenssista](#)). Käydään seuraavaksi lävitse muutamia Erlangin perusasioista.

Erlang on käännettävä kieli ja kääntäjä luo BEAM-tiedostoja. Erlang-ohjelmia suoritetaan Erlang virtuaalikoneella. Erlangille löytyy myös REPL, jolla voi kääntää tiedostoja. REPL:in suorittaminen tuo sen käyttöön kaikki BEAM-tiedostot samassa hakemistossa. REPL:in kautta voi myös navikoida muihin hakemistoihin.

Erlangin syntaksissa on muutamia erityispiirteitä. Jokainen rivi päättyy pisteeseen.

1 + 1 .

42 * 4 .

Kaikki muuttujat kirjoitetaan isolla alkukirjaimella ja ne eivät voi alkaa numeroilla.

```
Muuttuja = 1.  
Yksi = Muuttuja.  
Huomaa myös muuttujien muuttumattomuus:  
A = 1.  
A = 2.  
aiheuttaa virheen!
```

Atomit kirjoitetaan pienillä kirjaimilla ja moniosaiset atomit kirjoitetaan heittomerkkien väliin. Tuplat kirjoitetaan kaarisulkeiden sisään.

```
atomi.  
'kaksiosainen atomi'.  
{tupla, 2}.
```

Muuta huomattavaa on esimerkiksi se, että "pienempi tai yhtäsuuri kuin" ja "suurempi tai yhtäsuuri kuin" -merkkien kanssa on syytä muistaa ettei vahingossa luo "nuolia". Väärä syntaksi kahden luvun vertailuun olisi esimerkiksi:

```
1 <= 2 (vrt. oikea tapa 1 <= 2)  
tai  
1 => 2 (vrt. oikea tapa 1 >= 2)
```

Yksi tärkeistä ominaisuuksista on myös hahmonsovitus (pattern matching). Hahmonsovituksella tarkoitetaan sitä, että muuttujia voidaan verrata eri muotoihin. Tämä on hyödyllistä esimerkiksi kun halutaan tuplasta vain tiettyjä kenttiä, tai kun luodaan rekursiota. Alaviiva on "jokerimerkki", jolla voidaan jättää joitain arvoja huomiotta.

```
Luodaan tupla ja sovitetaan se muuttujille A ja B:  
Tupla = {1, 2, 3}.  
{A, B, _} = Tupla.  
Nyt A = 1 ja B = 2. Arvo 3 jätettiin nyt huomiotta.
```

Erlang-ohjelmat ohjelmat koostuvat moduuleista. Moduulit sisältävät omaa toiminnallisuutta ja koostuvat funktioista. Funktiot, joita halutaan käyttää moduulin ulkopuolella esimerkiksi muissa moduuleissa, täytyy määritellä erikseen "export"

sanan avulla. Moduuliin omaan sisäiseen käyttöön voidaan määritellä yksityisiä funktioita yksinkertaisesti niin, ettei niitä exportata. Funktion määrittely koostuu funktion nimestä, suluista, nuolesta ja funktion koodista. Koodissa peräkkäiset käskyt erotellaan pilkuilla, mutta viimeinen käsky loppuu pisteeseen.

Määritetään moduuli "esimerkki":

```
-module(esimerkki).
```

Tuodaan moduulista "esimerkki_funktio" jolle annetaan 2 parametria:

```
-export([ esimerkki_funktio/2 ]).
```

Luodaan funktio, joka laskee A:n ja B:n tulon neliön:

```
esimerkki_funktio(A, B) ->
```

```
    C = A * B,
```

```
    C * C.
```

Työn toteutus

Työni muodostuu neljästä moduulista: deck, hand, ranker ja poker_game. Jokaisella moduulilla on oma tehtävänsä. Deck-moduulissa luodaan pelissä käytettävä pakka, hand-moduuli sisältää pokerikädet, ranker-moduuli selvittää parhaimman käden ja poker_game hoitaa pelin aloituksen ja I/O:n.

Deck-moduulissa on funktiot pakan luomiseen, korttien ja pelaajien käsien jakamiseen. Pakan luominen tapahtuu näin:

```
create_deck() ->
```

```
    Suits = [hearts, diamonds, clubs, spades],
```

```
    Values = [ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, king],
```

```
    Cards = [{S, V} || S <- Suits, V <- Values],
```

```
    Cards.
```

Käytännössä yllä käytetään Erlangin listoja. Määritellään listaan korttipakan maat (Suits) ja korttien arvot (Values). Tämän jälkeen voimme kuvata kortit näiden

listojen avulla tekemällä listoista kombinaation. Lopuksi palautetaan funktiosta lista kortteja eli pakka.

Korttien jakaminen on melko suoraviivaista. Otetaan vastaan jossain tilassa oleva (sekoitettu) pakka ja otetaan pakasta rekursion avulla päällimmäisiä kortteja, kunnes kortteja on tarpeeksi. Näihin toiminnallisuuksiin on käytetty siis rekursiota sekä hahmosovitusta. Otetaan esimerkki koodista, joka käy kortteja läpi:

```
1) loop_cards({Count, Deck}) -> loop_cards(Count, Deck, []).  
  
2) loop_cards(0, Deck, Cards) ->  
    {Deck, Cards};  
  
3) loop_cards(Count, Deck, Cards) ->  
    {Top_card, Deck_after} = get_card(Deck),  
    New_cards = Cards ++ [Top_card],  
    loop_cards(Count-1, Deck_after, New_cards).
```

1) Määritetään aluksi funktiolle helpompi kutsumisasu, jossa on parametrina on yksi tupla jonka sisällä haluttavien korttien määrä ja käytettävä pakka. Tämä funktio lisää uuteen yksityisen funktion kutsuun tyhjän listan (tämä siis mahdollistaa sen, että moduulin käyttäjän ei tarvitse muistaa tyhjää listaa funktiota kutsuessaan).

2) Funktion signatuuri, jossa rekursion "lopetusehto". Jos `Count` on nolla, niin valitaan tämä funktio joka palauttaa korttipakan uuden tilan ja nostetut kortit.

3) Jos hahmonsovitusta huomasi, ettei `Count` ole 0, valitaan tämä signatuuri. Signatuurissa kutsutaan funktiota jolla saadaan pakan päällimmäinen kortti. Lisätään uusi kortti `New_cards` -listaan. Lopuksi tehdään rekursiivinen kutsu samaan funktioon, jossa `Count`:ia pienennetään yhdellä, annetaan pakan uusi tila (ts. pakka ilman nostettua korttia) ja jo nostetut kortit.

Huom: Huomaa kuinka rekursiivisen funktion eri signatuurit erotetaan toisistaan puolipilkulla. Funktion määrittely loppuu pisteeseen.

Hand-moduulin tehtävän on tunnistaa, mikä pokerikäsi annetulla kädellä on. Käytännössä apuna käytetään ideaa, että pelaajan käden ollessa korttien osalta

numerojärjestyksessä, on pokerikäsien tunnistaminen helpompaa. Käytännössä eri pokerikädet on toteutettu erilaisina hamotuksina ja näitä hahmoja koitetaan tunnistaa.

Otetaan esimerkkinä pokerikäsi "neljä samaa". Kun kortit ovat numerojärjestyksessä, on vain kaksi tapaa jolla kädessä voi olla neljä samaa korttia. Joko kortit ovat alusta neljä samaa (ja viides eri) tai neljä viimeistä samaa (ja ensimmäinen eri):

```
1)  is_four_a_kind(IntHand) ->
2)      [A,B,C,D,E] = [Value || {_ , Value} <- IntHand],
3)      if
          (A == B) and (B == C) and (C == D) ->
              {true, A};
          (B == C) and (C == D) and (D == E) ->
              {true, B};
4)      true -> {false, A}
      end.
```

1) Funktio ottaa vastaan järjestetyn käden (listan jossa 5 korttia).

2) Kuvataan kaikkien korttien arvot muuttujiin A, B, C, D ja E.

3) Käytetään Erlangin if-kontrollirakennetta. Ehtolause esitetään tässä tapauksessa and-rakenteen avulla. Ehtolauseita seuraa nuoli ja toteutettava koodi. Eri ehtolauseet erotetaan toisistaan puolipilkulla. Testataan siis onko ensimmäiset tai viimeiset neljä samoja kortteja. Palautetaan tupla, joka sisältää totuusarvon ja sen arvon, joita oli neljä.

4) Viimeiseksi napataan kaikki muut tilanteet (siis jos mikään edellisistä ehtolauseista ei evaluoitunut todeksi). Palautetaan tässä tapauksessa "false" ja ensimmäinen kortti. Tätä korttia ei oteta huomioon myöhemmin ohjelmassa, koska tuplan ensimmäinen arvon "false".

Muut moduulin funktiot ovat melko suoraviivaisia ja käyttävät samoja paradigmoja kuin aikaisemminkin. Käsi käytännössä tarkistetaan kaikkia pokerikäsiä vasten. Tämän jälkeen palautetaan funktiosta `rank_hand` tupla, jossa on syöttenä

saatu käsi, käden ratkaiseva kortti (esimerkiksi suoran korkein arvo), atomi pokerikädestä ja pokerikäden järjestysnumero (mitä korkeampi, sen parempi käsi).

Ranker-moduulissa ei sinänsä ole mitään uutta verrattuna aiempiin moduuleihin. Käytännössä sen tehtävänä on selvittää, mikä pelissä olevista käsistä on paras. Tämä tapahtuu ensin niin, että kerätään parhaat kädet yhteen (siis ne kädet, joilla on korkein pokerikäden järjestysnumero). Jos on siis useampi "voittava" pokerikäsi, kerätään ne yhteen. Näistä käsistä sitten selvitetään "ratkaisevan kortin" avulla voittaja. Esimerkiksi kahdella pelaajalla on täyskäsi. Käsiin on sisälletty tietona täyskädessä olevan kolmen saman kortin arvo (joka tässä tapauksessa määrittää kumpi täyskäsistä on parempi). Sitten vertaillaan kummalla tämä ratkaiseva kortti on suurempi ja päätetään voittaja.

Huomattavaa on se, että tässä vaiheessa ohjelmassa on bugi. Jos kahdella pelaajalla on kummallakin esimerkiksi sama pari ei ohjelma tarkista tämän lisäksi vielä haita (ts. korttia, jolla on suurin arvo).

Poker_game-moduulissa uutena asiana on vielä I/O -operaatiot. Tämä moduuli on koko ohjelman aloituskohta. Funktiolla `start` saadaan peli käyntiin. Ohjelma kysyy käyttäjältä pelaajien määrää. Näyttää pelaajan käden ja kysyy, mitkä kortit pelaaja haluaa vaihtaa. Tämän jälkeen kysely jatkuu seuraavalta pelaajalta. Kun kaikki ovat vaihtaneet haluamansa kortin, kertoo ohjelma voittaja käden.

```
start() ->
1)      io:format(os:cmd(clear)),
2)      Input = io:read("Welcome to Erlang Poker. Please enter the
                        number of players (1-5) followed by dot and
                        press enter >>"),
        Players = element(2, Input),
        {Hands, Deck} = reset_game(Players),
        {HandsAfter, _} = loop_hands({Hands, Deck}),
3)      {WinnerHand, _, Rank} = ranker:winning_hand(HandsAfter),
4)      io:format("Winning hand\n ~p \n ~p \n", [lists:keysort(2,
                        WinnerHand), Rank]).
```

1) Käytetään Erlangin tarjoamaa moduulia "io" ja "os". Näiden avulla putsataan komentoriviohjelma tyhjäksi. Tässä esimerkissä myös huomataan, miten toisen moduulin funktioita kutsutaan: "moduulin_nimi:funktion_nimi()".

2) Read-funktiolla luetaan käyttäjän syöte. Parametrina annettu merkkijono näytetään käyttäjälle.

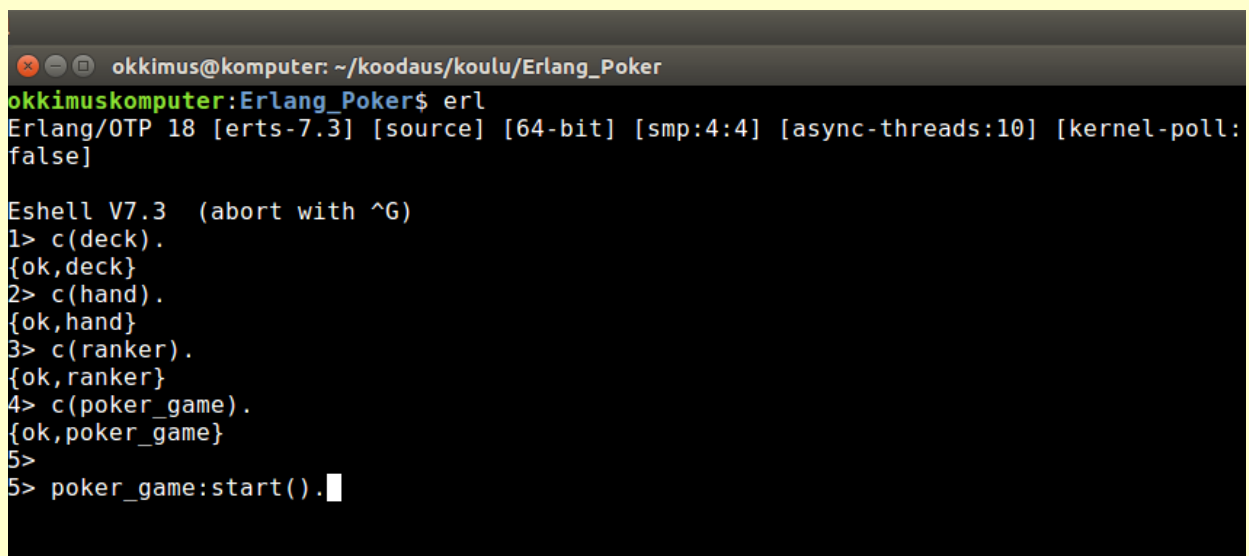
3) Selvitetään voittaja käsi.

4) Kirjotetaan ruudulle mikä on voittaja käsi ja mikä pokerikäsi se on format-funktion avulla.

Tässä oli esitelty vain pääpiirteittäin koodissa käytettyjä tekniikoita ja paradigmoja. Projektin koodin on kommentoitu kunnolla, joten sen toimintaan voi tutustua lukemalla koodia suoraan. Viimeisin versio koodista löytyy aina [Githubista](#).

Ohjelman suoritus

Käynnistä Erlang REPL komentorivillä `erl` ja käännä samassa hakemistossa olevat lähdekooditiedostot komennolla `c(moduulin_nimi)`. Aloita peli suorittamalla komento `poker_game:start()`.



```
okkimus@komputer: ~/koodaus/koulu/Erlang_Poker
okkimuskomputer:Erlang_Poker$ erl
Erlang/OTP 18 [erts-7.3] [source] [64-bit] [smp:4:4] [async-threads:10] [kernel-poll:
false]

Eshell V7.3 (abort with ^G)
1> c(deck).
{ok,deck}
2> c(hand).
{ok,hand}
3> c(ranker).
{ok,ranker}
4> c(poker_game).
{ok,poker_game}
5>
5> poker_game:start().
```


Vastaa kuinka monta pelaajaa haluat. Muista käyttää Erlangin syntaksia, siis kirjoita numero ja sen perään piste päästäksesi eteenpäin.

```
okkimus@komputer: ~/koodaus/koulu/Erlang_Poker
Welcome to Erlang Poker. Please enter the number of players (1-5) followed by dot and
press enter >>■
```

Seuraavaksi ensimmäisen pelaajan kortit tulevat näkyviin. Valitse mitä kortteja haluat vaihtaa ja kirjoita niiden korttien indeksit hakasulkeiden väliin pilkulla erotettuina ja piste perään. Näet tämän jälkeen omat uudet korttisi. Lopuksi vielä kirjoita piste ja paina enteriä.

```
okkimus@komputer: ~/koodaus/koulu/Erlang_Poker
Card: {spades,ace}
Card: {diamonds,jack}
Card: {hearts,jack}
Card: {diamonds,ace}
Card: {clubs,2}
What cards you like to change? Input the indices between '[' followed by a dot and p
ress enter >>[5].
Your hand now:
Card: {spades,ace}
Card: {diamonds,jack}
Card: {hearts,jack}
Card: {diamonds,ace}
Card: {hearts,queen}
Type a dot and press enter to continue■
```

Kun kaikki pelaajat ovat vaihtaneet kortteja, näyttää ohjelma voittavan käden ja lopettaa suorituksensa.

```
okkimus@komputer: ~/koodaus/koulu/Erlang_Poker
Winning hand
[{clubs,3},{clubs,4},{clubs,jack},{clubs,king},{clubs,queen}]
flush
ok
4>■
```

Lähteet:

<http://www.erlang.org/about> (käyty 19.04.2018)

<https://www.youtube.com/watch?v=3MvKLOecT1I> (käyty 19.04.2018)

<https://pragprog.com/articles/erlang> (käyty 19.04.2018)

<http://erlang.org/faq/introduction.html> (käyty 19.04.2018)

<http://learnyousomeerlang.com/content> (käyty 19.04.2018)