D- ja Erlang-kielten datan kapselointi

Hansi Keijonen, Jari Koskinen, Eero Laine

20. helmikuuta 2013

1 Jarin osuus

1.1 Rakenteiset tyypit

Alkeistyypeistä voidaan muodostaa taulukoita, joita on kahdenlaisia. Yksi taulukoista on yleinen perustapaus array, jollainen löytyy myös C/C++-kielistä [KRR88]. Taulukon alkioihin voidaan viitata indeksin avulla.

```
int[] lukuja;
lukuja[11] = 13;
```

Toinen taulukkomuoto on avaimen ja arvon sisältävä pari, associative array. Taulukon alkio sisältää arvon ja siihen viittaava indeksi voi olla esimerkiksi merkkijono:

```
  int[string] kuukausi;

  kuukausi["tammikuu"] = 1;
  kuukausi["helmikuu"] = 2;

  writeln("Tammikuu=", kuukausi["tammikuu"]);
```

Koodi tulostaa:

```
Tammikuu=1
```

Lisäksi taulukko voidaan jättää dynaamiseksi, määrittämällä sen pituudeksi [], jolloin sille voidaan osoittaa jokin olemassa oleva taulukko myöhemmin koodissa.

Alkeistyypeistä voidaan myös muodostaa tietueet struct tai union. Molemmat vastaavat C/C++-kielen vastaavia rakenteita [KRR88] ja ovat arvotyyppejä [DLA13]. Struct määritellään, ja sitä käytetään seuraavasti:

```
struct palkansaaja{
  int palkka;
  string titteli;
}

void main() {
  palkansaaja[5] palkolliset;
  palkolliset[1].palkka = 4500;
  palkolliset[1].titteli = "ohjelmoija";

  write(palkolliset[1].titteli, " tienaa ");
  writeln(palkolliset[1].palkka, " kuukaudessa");
}
```

Union eroaa structista siten, että muuttujien arvot on talletettu muistissa samaan kohtaan [ALE10];[KRR88], riippumatta niiden tyypistä ja pituudesta. Kääntäjän tehtävä on varata riittävä määrä muistia suurimman tyypin mukaisesti. Yhden muuttujan arvon muuttaminen vaihtaa muuttujille varatun muistin sisällön. Seuraava esimerkki havainnollistaa tätä:

```
union moniTyyppi{
  int iluku;
  uint uiluku;
  ubyte ubluku;
}

void main() {
  moniTyyppi luku;
  luku.iluku = 6200;
  writeln("int: ", luku.iluku);
  writeln("uint: ", luku.uiluku);
  writeln("ubyte: ", luku.ubluku);
  luku.iluku = -6200;
  writeln("int: ", luku.iluku);
  writeln("uint: ", luku.uiluku);
  writeln("ubyte: ", luku.ubluku);
}
```

Ohjelma tulostaa seuraavasti:

```
int: 6200
uint: 6200
byte: 56
int: -6200
uint: 4294961096
ubyte: 200
```

Lisäksi voidaan luoda dymaaninen taulukko, joka tarkoittaa sitä, että taulukkoon voidaan myöhemmin sijoittaa taulukko. Esimerkki havainnollistaa tätä toimintaa:

```
  int[] c;
  int[4] d;
  d[2] = 10;
  c = d;
  writeln(c[2]);
```

Ohjelma tulostaa luvun 10.

D-kielen tarjoama struct ja union ovat useissa tilanteissa käyttökelpoisia, varsinkin matalammalla tasolla, kuten esimerkiksi käyttöjärjestelmien toteutuksessa. Koska ne ovat arvotyyppejä, niillä saadaan aikaan tehokkaita rakenteita. D-kielen kehitykseen vaikuttaneista kielistä Java ei sisällä struct, eikä union rakennetta. C# sitä vastoin sisältää struct arvotyypin rakenteen mutta ei unionia.

1.2 Geneerisyys

D-kielessä on tuki geneerisyydelle. Tämä tarkoittaa sitä, että funktio voidaan kirjoittaa yleiseksi ilman, että sen parametreja sidotaan tiettyihin tyyppeihin. Tällöin parametrin tyypiksi määritetään T. Seuraava esimerkki on toteutettu geneerisyyttä hyödyntäen; binäärihaku, jolle voidaan antaa syötteeksi minkä tahansa tyyppinen järjestetty taulukko.

```
bool binHaku(T)(T[] input, T value) {
  while (!input.empty) {
    int i = input.length / 2;
    auto mid = input[i];
    if (mid > value)
      input = input[0 .. i];
    else
      if (mid < value)
        input = input[i + 1 .. $];
    else
      return true;
  }
  return false;
}

void main() {
  writeln(binHaku([ 1, 3, 6, 7, 9, 15 ], 6));
  writeln(binHaku([ 'a', 'b', 'c', 'd', 'e', 'g', 'i' ], 'h'));
}
```

Funktio hyväksyy syötteeksi järjestetyn taulukon ja suorittaa puolitushaun sille. Ohjelman tuloste on seuraava:

```
true
false
```

1.3 Mixin

D-kieli tarjoaa mixinin. Mixin on kuin geneerinen luokka tai template. Ero templaten ja mixinin välillä on se, että template alustetaan siihen näky-

vyysalueeseen, missä alustus tehdään kun taas mixin voidaan alustaa mihin tahansa näkyvyysalueeseen, kuten esimerkikis structiin [lähde?]. Alla on määritetty template Kirjoittaja, joka kirjoittaa metodille kirjoita annetun tiedon.

```
template Kirjoittaja(T)
{
    void kirjoita(T t)
    {
        writefln(t);
    }
}
```

Tällainen template voidaan instantioida paikalliseen näkyvyysalueeseen (ja kutsua myös metodia kirjoita) seuraavalla tavalla:

```
Kirjoittaja!(int).kirjoita(666);
```

Template Kirjoittaja voidaan alustaa mixinin avulla myös struct:in sisälle:

```
struct S
{
    mixin Kirjoittaja!(int) IntKirjoittaja;
    mixin Kirjoittaja!(char[]) StrKirjoittaja;
}
```

1.4 Luokat ja periytyminen

D-kielessä luokka voi periytyä vain yhdestä luokasta, toisin kuin C++ -kielessä, jossa moniperintä on mahdollinen. D:ssä aliluokka perii kaikki yliluokan tietueet ja funktiot.

```
class Henkilo {
  string nimi;
  int ika;
  // luokan konstruktori
  this(string nimi, int ika) {
    this.nimi = nimi;
    this.ika = ika;
  }
  ~this() {} // tyhjäksi jätetty destruktori
}

class Opiskelija : Henkilo { // perii luoka Henkilo ominaisuudet
  string opiskelijaNumero;
  int opintoPisteet;
```

```
  string opintoLinja;
  // luokan konstruktori
  this(string nimi, int ika, string opiskelijaNumero) {
    super(nimi, ika); // kutsuu yliluokan konstruktoria
    this.opiskelijaNumero = opiskelijaNumero;
  }
}

void main() {
  Opiskelija kapistelija = new Opiskelija("Kerttu Koodari", 29, "987234651");
  Opiskelija konnari = new Opiskelija("Kalle Konnari", 34, "132435467");
  kapistelija.opintoLinja = "Tietojenkasittelytiede";
  konnari.opintoLinja = "Kognitiotiede";
  kapistelija.opintoPisteet = 123;
  konnari.opintoPisteet = 87;
  writeln(kapistelija.nimi, "n opintopistekertyma on ",
kapistelija.opintoPisteet);
  writeln(konnari.nimi, "n opintopistekertyma on ", konnari.opintoPisteet);
}
```

Ohjelma tulostaa seuraavasti:

```
Kerttu Koodarin opintopistekertyma on 123
Kalle Konnarin opintopistekertyma on 87
```

Perinnän voi estää kirjoittamalla luokkamäärittelyn eteen final [DLA13]. Rajapintaluokka löytyy myös ja se määritetään luokkamäärittelyn edessä avainsanalla interface. Abstrakti luokka voidaan muodostaa avainsanalla abstract, joka kirjoitetaan luokan määrityksen eteen. Abstrakti luokka voi sisältää D-kielessä abstrakteja funktioita, joille aliluokan on annettava toteutus, ja normaaleja funktioita. Abstraktista luokasta ei voi luoda ilmentymää, vaan ainoastaan abstraktin luokan aliluokasta voidaan luoda ilmentymä. Esimerkkikoodia abstraktin luokan toteutuksesta:

```
// abstrakti luokka Tervehdys
abstract class Tervehdys {
  void tervehdi(){
    writeln("Hei!");
  }
  abstract void tervehdiNimella(string name);
}

class TervehdysNimella : Tervehdys {
  // tervehdiNimella abstraktin funktion toteutus
  void tervehdiNimella(string nimi){
```

```
    writeln("Hei ", nimi, "!");
  }
}

void main() {

  Tervehdys tervehdys = new TervehdysNimella();
  tervehdys.tervehdi();
  tervehdys.tervehdiNimella("Kerttu");
}
```

Ja ohjelma tulostaa:

```
Hei!
Hei Kerttu!
```

D-kielessä on lisäksi tuki rajapintaluokille, joita voidaan periä useampia yhdelle luokalle. Rajapintaluokassa määritellään funktiot, jotka aliluokan täytyy toteuttaa. Alla esimerkki koodista, jossa luokassa Laskenta toteutetaan rajapintaluokkien Summa ja Tulo funktiot.

```
interface Summa {
  int summa(int a, int b);
}

interface Tulo {
  int tulo(int a, int b);
}

class Laskenta : Summa, Tulo {
  int summa(int a, int b) {
    return a+b;
  }

  int tulo(int a, int b) {
    return a*b;
  }
}

void main() {
  Laskenta laskuri = new Laskenta();
  writeln("3+4=", laskuri.summa(3, 4));
  writeln("3*4=", laskuri.tulo(3, 4));
}
```

Esimerkkikoodi tulostaa:

```
3+4=7
3*4=12
```

1.5 Arvo- ja viitesemantiikka

D-kielessä tietueet, struct ja union, noudattavat arvosemantiikkaa. Luokat noudattavat viitesemantiikkaa. Arvosemantiikkaa noudattavat tietueet tallennetaan muistissa pinoon ja niiden olemassaolo riippuu näkyvyysalueesta. Viitesemantiikkaan perustuvien luokkien ilmentymille taas varataan muistia keosta; pinoon laitetaan vain osoitin keon kohtaan, jossa luotu olio sijaitsee. Funktiokutsun parametrit voidaan välittää arvoina tai viitteinä, samaan tapaan kuin C/C++ -kielissä [KRR88]. Kielessä on tätä varten varattu merkit * ja &, josta lyhyt esimerkki:

```
void main() {
  int x=10;
  int y=20;
  writeln(x,",",y);
  vaihda(&x, &y);
  writeln(x,",",y);
}

void vaihda(int *px, int *py)
{
  int temp;
  temp=*px;
  *px=*py;
  *py=temp;
}
```

Koodissa vaihdetaan muistissa x:n ja y:n arvoja keskenään. main kutsuu funktiota vaihda ja funktiolle välitetään parametrina x:n ja y:n muistiosoitteet &x ja &y. vaihda-funktion parametrien tyypeiksi on määritetty osoittimet merkillä *. Funktio tekee arvojen vaihtamisen suoria muistiosoitteita käyttäen. Tuloste on seuraava:

```
10,20
20,10
```

2 Hansin osuus

Erlangissa ei ole luokkia, joten ei ole myöskään mitään luokkiin liittyvää toiminnallisuutta kuten perintää. Myöskään geneerisiä tyyppejä tai ajonaikaista tyyppiparametrointia ei ole. Parametrinvälityksessä ja muuttujiin sijoittamisessa Erlangissa on aina käytössä arvosemantiikka.

Erlangin tietotakenne Record on hyvin samantapainen kuin c:n struct [HEB13]. Se on sopiva pienen tietorakenteen luomiseksi. Record määritellään moduulin atribuutiksi:

```
-module(piste) .
-compile(export_all) .
-record( piste, {
   x,
   y
}) .
```

Esimerkissä on yksinkertainen tietorakenne pisteen kuvaamiseksi. Recordin alustus samaisessa moduulissa tapahtuu seuraavasti:

```
piste() ->
    #piste{
        x = 20,
        y = 35
    } .
```

Moduulin käännöksen jälkeen voidaan tulostaa pisteen kaikki tiedot tai aioastaan yksittäinen tieto:

```
>c(piste) .
{ok, piste}
>piste:piste() .
#piste{x = 20, y = 35}
>piste#.piste.x .
20
```

Tärkeä joukko Erlangin tietorakenteita on avain-arvo-parit (key-value stores). Yleisin näistä on proplist, joka on tuplelista muotoa [key,value]. Muita rajoitteita ei juuri ole. Proplistin käsittelyyn (lisäys, poisto, haku jne.) löytyy moduulista proplists kaikki tarvittavat funkiot [HEB13].

Hieman formaalimmin määritelty avain-arvo -tietorakene on orddict eli järjestetty sanakirja (ordered dctionary). Siinä avain saa esiintyä ainoastaan kerran ja rakenne tarjoaa rajoitetun CRUD-toiminnallisuuden elementtien tallentamiseen, etsimiseen, lukemiseen ja poistamiseen. Elementit ovat järjestetty, joten haut ovat nopeita[HEB13]. orddict on tehokas 75 elementin säilömiseen saakka. Tätä suuremmat tietomäärät kannattaa tallettaa esimerkiksi dict:iin tai gb_tree:hin.

dict:ien toiminnallisuus on lähes sama kuin orddict:issä lisättynä muutamilla funktioilla kuten fold ja map, jotka helpottavat tiedon käsittelyä.

Erlang tarjoaa myös valmiin puurakenteen, gb_treen [HEB13]. gb_tree on tasapainotettu puu, johon on valmiiksi toteutettu yleisimmät puissa tarvittavat funktiot kuten insert, delete, lookup muutamia mainitakseni. Puurakenne on varsin tehokas pl. tilanteet, joissa tasapainotusta joudutaan tekemään.

Eräs Erlangin erikoisuus on valmiiksi toteutettu suunnattu verkko, digraph (directed graph) [HEB13]. digraph on toteutettu kahdessa moduulissa digraph ja digraph_utils, joista edellinen toteuttaa verkon ja jälkimmäinen tarjoaa palvelut verkon läpikäyntiin, renkaiden (cycle) löytämisen jne.

Erlangissa on myös valmis toteutus FIFO-jonolle, nimeltää queue [HEB13]. Luonnollisestikin jonon toteutus sisältää funktiot elementtien lisäämiseksi jonoon ja poistamiseksi jonosta.

Melko triviaali tietorakenne on taulukko (array), johon voi tallentaa ainoastaan numeerisia alkioita [HEB13]. Toisin kuin imperatiivisissa kielissä, Erlangin taulukko ei tarjoa vakioaikaisia hakufunktioita. Yleinen käytäntö esimerkiksi raskaissa matriisioperatioissa on teettää työ muilla kielillä kirjoitetuilla ohjelmilla käyttäen Erlangin siihen tajoamaa tekniikkaa port:ia

3 Eeron osuus

Erlang-kielessä set-tietorakenteet eli joukot ovat elementtien kokoelmia (collection), joissa mistään elementistä ei ole kaksoiskappaleita. Erlangissa on neljä moduulia joukkojen käsittelyyn: ordsets, sets, gb_sets ja sofs (sets of sets). Fred Hébertin mukaan suunnittelijoiden tausta-ajatuksena oli, ettei joukon esittämiseen ole yhtä, optimaalista tapaa [HEB13].

ordsets-moduulissa järjestyksessä olevaa listaa käytetään joukon elementtien tallentamiseen [ERL13]. ordsets-rakenteet ovat hyödyllisiä lähinnä pienten joukkojen esittämiseen [HEB13]. Ne ovat hitaita, mutta niiden esitystapa on kaikista Erlangin joukkorakenteista yksinkertaisin ja helppolukuisin.

ordsets-moduulin esitystapa eroaa sets-moduulin esitystavasta yhdellä tavalla. Siinä missä sets-moduuli olettaa kahden elementin olevan erilaisia, jos ne eivät ole täysin samat (=:=), ordsets-moduulissa kaksi elementtiä eroavat toisistaan ainoastaan, jos ne eivät ole yhtä suuret (==) [ERL13].

sets-moduuli on toteutettu käyttäen samankaltaista rakennetta kuin dict-tietorakenne, ja se toteuttaa saman rajapinnan kuin ordsets-moduuli. sets on kuitenkin paremmin skaalautuva, eli suurien elementtimäärien käsittely on sets-moduulin kautta toteutetuissa joukoissa tehokkaampaa [HEB13]. dict-rakenteiden tapaan sets-moduulilla toteutetut joukot ovat erityisen hyviä lukemispainotteisissa operaatioissa. Tällaisessa operaatiossa voitaisiin esimerkiksi tarkastaa, onko jokin elementti mukana joukossa vai ei.

gb_sets-moduulissa järjestetyt joukot on toteutettu soveltaen General Balanced Trees -konseptia [ERL13]. General Balanced Tree on yksinkertaisesti binääripuu, jolla on kyky korjata muotonsa tarvittaessa [AND13]. Jouk-

kojen toteuttaminen gb_sets-moduulin avulla voi olla suurien joukkojen kohdalla paljon tehokkaampaa kuin järjestettyjen listojen käyttö [ERL13]

gb_sets-moduuli on tehokas muissa operaatioissa kuin lukemisessa [HEB13]. Vaikka gb_sets toteuttaa saman rajapinnan kuin sets ja ordsets, se tarjoaa enemmän funktioita. Esimerkkeinä mainittakoon älykkäät ja naivit funktiot, iteraattorit ja nopea pienimpien ja suurimpien arvojen haku.

sofs-moduuli toteutetaan järjestetyillä listoilla, jotka ovat tuple-tyypin sisällä mukanaan metatietoa. sofs-moduuli on hyödyllinen, jos ohjelmoija haluaa täyden hallinnan joukkojen ja joukkojen muodostamien perheiden välisistä suhteista tai esimerkiksi pakottaa (?) joukkojen tyyppejä [HEB13]. sofs-moduulilla toteutetut joukot ovat ikään kuin lähellä matemaattisen joukon konseptia, ne eivät ole ainoastaan ainutkertaisten elementtien ryhmiä.

Erlangin kehittäjäryhmän jäsen Björn Gustavsson suosittelee gb_sets-moduulin käyttöä useimmissa tilanteissa [HEB13]. Gustavsson käyttäisi ordsets-moduulia silloin, kun ohjelmoija haluaa selkeän esityksen siitä, mitä haluaa koodissaan prosessoitavan. sets-moduulia Gustavsson suosittelee käytettäväksi ainostaan silloin, kun operaattoria =:= tarvitaan.

Viitteet

[ERL99] Erlang 4.7.3 Reference Manual, DRAFT (0.7), Jonas Barklund, Robert Virding, 1999.

[DLA13] http://dlang.org, noudettu 6.2.2013.

[HEB13] Learn You Some Erlang For Great Good, Fred Hébert, 2013.

[AND13] http://user.it.uu.se/ arnea/ps/gbimpl.pdf, noudettu 20.2.2013

[ERL13] http://www.erlang.org/doc, noudettu 20.2.2013

[ALE10] The D Programming Language, Andrei Alexandrescu, 2010.

[KRR88] The C Programming Language, Brian W. Kernighan, Dennis M. Ritchie, 1978/1988.