D- ja Erlang-kielten perustietotyypit ja laskennan kapselointi

Hansi Keijonen, Jari Koskinen, Eero Laine

11. helmikuuta 2013

1 Jarin D-osuus (kaikki D:stä aiheen ympärillä)

1.1 Alkeistyypit/perustyypit

D-kielessä alkeistyyppejä [ALE10] ovat: bool, byte, short, int, long, float, double ja real. Vastaavat etumerkittömät alkeistyypit ovat: ubyte, ushort, uint ja ulong. Näiden lisäksi on etumerkittömät merkkialkiot: char 8-bittiselle UTF8:lle ja wchar 16-bittiselle UTF16-merkille, jotka saavat alustuksen yhteydessä arvon 0xFF ja 0xFFFF. 32-bittinen UTF32-merkki voidaan tallettaa dchar alkeistyyppiin jonka määrittämätön alustusarvo on 0x0000FFFF. bool on aito totuusarvo ja voi saada arvon true tai false, alustuksen jälkeen määrittämätön arvo on false. byte on 8 bittiä pitkä tavu, short on 16, ja int 32 bittiä pitkä sana. Suurempia kokonaislukuja varten on 64 bitin pituinen long, joiden lisäksi kielessä on varaus cent tyypille, joka voi olla 128 bitin pituinen. Kaikki kokonaislukutyypit saavat muuttujan alustuksessa arvon 0. Reaaliluvuille varatut float, real ja double ovat alustuksen jälkeen nan, joka tarkoittaa Not A Number [DLA13]. Kompleksiluvuille on alkeistyypit cfloat, cdouble ja creal, imaginääriluvuille ifloat, idouble ja ireal jotka saavat alustuksen yhteydessä arvon nan, mikäli arvoa ei ole määritetty.

Merkkijonoja varten on varattu string, joka koostuu taulukosta char-alkeistyyppejä. Merkkijonon tiettyyn alkioon voidaan viitata suoraan indeksillä:

```
string merkkijono = "Hei Lukija!";
for(int i=0; i<merkkijono.length; i++)
  write(merkkijono[i]);
```

Koodi tulostaa:

```
Hei Lukija!
```

Arvoalueet eivät varsinaisesti ole standardoituja, mutta kaikki D-kielen kääntäjät noudattavat samoja arvoalueita. Kielessä on käytössä vahva tyypitys ja tyyppitarkistukset tehdään staattisesti käännösaikana. D-kielessä on säännöt sille, miten eri alkeistyyppien välillä voidaan suorittaa laskentaa ja kuinka niitä voidaan muuntaa tyypistä toiseen. Muuttujan tyypin voi myös jättää kääntäjän päätettäväksi, määrittämällä muuttujan tyypiksi auto.

```
void main() {
  auto sana = "merkkijono";
  auto luku = 123456;
  writeln(sana);
  writeln(luku);
}
```

D-kielessä etumerkittömät kokonaislukutyypit esiintyvät samanlaisina kuin C/C++ -kielissä. Tämä tekee alkeistyyppien käytöstä joustavampaa, verrattuna esimerkiksi Javaan, joka ei sisällä etumerkittömiä kokonaislukutyyppejä.

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

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

Toinen taulukkomuoto on avaimen ja arvon sisältävä pari. Taulukon alkio sisältää arvon ja siihen viittaava indeksi on esimerkiksi merkkijono:

```
  int[string] kuukausi;

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

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

Koodi tulostaa:

```
Tammikuu=1
```

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

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

void main() {
  palkansaaja palkollinen;
  palkollinen.palkka = 4500;
  palkollinen.titteli = "ohjelmoija";

  write(palkollinen.titteli, " tienaa ");
  writeln(palkollinen.palkka, " kuukaudessa");
}
```

Union eroaa structista siten, että yhden muuttujan arvon vaihtaminen muuttaa kaikkien saman arvotyypin muuttujien sisällön [ALE10]. Saman arvotyypin muuttujat sijoitetaan siis muistissa samalle muistialueelle. Seuraava esimerkki havainnolistaa tätä:

```
import std.stdio;

union tienaaja{
  int palkka;
  int tunnit;
  string titteli;
  string esimies;
}

void main() {
  tienaaja palkollinen;
  palkollinen.palkka = 6200;
  writeln(palkollinen.tunnit);
  writeln(palkollinen.titteli);
  palkollinen.esimies = "pomo";
  writeln(palkollinen.titteli);
}
```

Ohjelma tulostaa seuraavasti:

```
6200
(null)
pomo
```

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

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

Ohjelma tulostaa luvun 10.
Taulukko voidaan myös osoittaa tiettyyn muistialueeseen. JATKA POINTER JNE.

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 Laskennan kapselointi

Funktiot määritellään D-kielessä kuin C/C++-kielissä, funktiolle määritetään ensin sen paluuarvon tyyppi, funktion nimi ja funktion mahdolliset parametrit. Parametreille määritetään arvotyyppi. Funktion paluuarvon tyypin määrittäminen voidaan myös jättää kääntäjän tehtäväksi, asettamalla arvotyypiksi auto. Funktio, joka ei palauta mitään, asetetaan void tyyppiseksi, jolloin se vastaa Pascal-kielen procedurea. Esimerkkejä erilaisista funktioista:

```
// palauttaa int-tyypin, parametreina 2 int-tyyppiä
int erotus(int a, int b) {
  return a - b;
}

// paluuarvon tyyppi jätetään kääntäjän päätettäväksi
auto summa(int a, int b) {
  return a - b;
}

// funktio, joka ei palauta mitään
void tulosta(string teksti) {
  writeln(teksti);
}
```

Parametien välittämiseen voidaan käyttää arvo-, osoite- tai viiteparametreja. Esimerkki osoite- ja viiteparametrien välittämisestä:

```
// osoiteparametrit
int summa(int *a, int *b) {
  return *a + *b;
}

// viiteparametrit
int summa(int &a, int &b) {
  return a + b;
}
```

D tarjoaa käyttöön myös aidon funktion, joka tarkoittaa funktiota, joka ei muuta ohjelman tilaa tai käsittele globaaleja muuttujia [DLA13]. Aito funktio ei saa myöskään sisältää kutsua ei-aitoon funktioon. Tällainen funktio esitellään lisäämällä funktion eteen määritys pure. Esimerkki aidosta funktiosta:

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

D-kielessä funktioparametreille voidaan myös määrittää oletusarvot antamalla parametrien esittelyn jälkeen niiden arvot.

```
int summa(int a=1, int b=1) {
  return a + b;
}
```

1.3 Virheenkäsittely

Virheenkäsittely on D-kielessä toteutettu poikkeuksilla [ALE10], kuten Javassa tai C#-kielessäkin. D tukee try - catch - finally syntaktista rakennetta virheekäsittelyssä. Poikkeus heitetään throw-lauseella, kuten koodiesimerkissä on tehty.

```
int somethingElse() {
  throw new Exception("joku muu generoi virheen");
}

int something(int a) {
  return somethingElse();
}

void main() {
  try {
    something(10);
  }
  catch (Exception e) {
    writeln("Tapahtui virhe: ", e);
  }
  finally {
    writeln("Tulosta ainakin jotain...");
  }
}
```

catch-lauseella määritetään mitä tehdään, jos poikkeus tapahtuu, ja finally-lauseella määritetään tehtävät jotka suoritetaan lopuksi. Esimerkkinä tällaisesta on tiedostonkäsittely; finally-lauseessa voidaan huolehtia siitä, että tiedosto suljetaan asianmukaisesti.

1.4 Rinnakkaisuuden tuki

Rinnakaisuuteen D-kielessä on vahva tuki. D ei oletuksena käytä jaettua muistia säikeiden ja prosessien kesken, vaan molempia ajetaan eristettynä [DLA13];[ALE10]. Säikeiden ja prosessien välillä käytetään viestinvälitystä joka tunnetaan nimellä *message passing*. D sisältää myös tuen perinteiselle

kriittisen alueen suojaukseen lukolla ja jaetun muistin suojauksen *mutexilla*. Suosituksena on kuitenkin käyttää D-kielessä olevaa aktorimallia, jolloin säikeiden välillä hyödynnetään viestijärjestelmää. receive-lause sisältää hahmontunnistuksen, jonka ansiosta monimutkaisemmatkin viestit voidaan käsitellä niiden tyypin mukaisesti. Esimerkissä viestejä lähetetään ja vastaanotetaan säikeiden välillä.

```
import std.concurrency, std.stdio, std.exception;
void main() {
  auto tid = spawn(&kirjoittaja);
  foreach(i; 0 .. 10) {
    // lähetä viesti
    tid.send("Hei maailma!");
    // lähetä oma Tid säikeelle
    tid.send(thisTid);
    receive(
      (string msg) { writeln("Main sai viestin: ", msg); }
    );
  }
}

void kirjoittaja() {
  for(;;) {
    receive(
      // jos saatiin string, kirjoita se ruudulle
      (string msg) { writeln("Thread sai viestin: ", msg); },
      // jos saatiin Tid, lähetä viesti
      (Tid tid) { tid.send("Hei!"); }
    );
  }
}
```

Esimerkin ohjelma tulostaa ruudulle seuraavasti:

```
Thread sai viestin: Hei maailma!
Main sai viestin: Hei!
Thread sai viestin: Hei maailma!
Main sai viestin: Hei!
Thread sai viestin: Hei maailma!
Main sai viestin: Hei!
```

D-kieli sisältää myös *synchronized class* eli synkronoidun luokan, jota voi käyttää rinnakkaisohjelmoinnissa suojaukseen.

```
synchronized class PankkiTili {
  private double _tili;

  void talleta(double summa) {
    _tili += summa;
  }

  void nosta(double summa) {
    enforce(_tili >= summa);
    _tili -= summa;
  }

  double saldo( ) {
    return _tili;
  }
}
```

1.5 Tyypillisiä ohjelmistoarkkitehtuureja

D-kielellä voi toteuttaa arkkitehtuuriltaan hyvinkin erilaisia ohjelmistoja. Rinnakkaisuuden tuki mahdollistaa asiakas-palvelin-mallin, ja hajautettujen ohjelmistojen toteutuksen. Koska ohjelmoija voi valita toteutukseen D-kielen tarjoamista ohjelmointiparadigmoista imperatiivisen-, olio- tai funktionaalisen ohjelmoinnin, on D geneerisyydeltään käyttökelpoinen useimpiin arkkitehtuureihin. Kielellä tehtyjä laajempia toteutuksia ei ole kuitenkaan raportoitu julkisesti. Web-ohjelmointiin D ei sovellu yksin sellaisenaan, mutta esimerkiksi käyttöjärjestelmien ja sulautettujen järjestelmien ohjelmointi onnistuu hyvin.

1.6 Valittujen ratkaisujen vertailua

Kielessä on monipuolisia ja tehokkaita keinoja hallita rinnakkaisuutta. Rinnakkaisuudessa tuettu viestinvälitys on uudemmissa kielissä myös yleinen. Myös virheenkäsittely on toteutettu siten, että se on erittäin käyttökelpoinen. try - catch - finally rakenne vaatii jonkin verran ohjelmoijalta paneutumista asiaan, mutta on vaivan arvoinen. Funktioiden toteutus on perinteinen ja noudattelee vahvasti suunnittelun mallina olleita C/C++/C# kielten vastaavia.

2 Hansin osuus

...
...

...

Viittaukset menvät näin: [ERL99].

3 Erlangin perustietotyypit

3.1 Erlangin tukemat alkeistyypit (ja tietorakenteet?)

Erlangissa on kahdentyyppisiä numeerisia literaaleja, kokonaislukuja ja liukulukuja. Bit String -tietotyyppiä käytetään varastoimaan tyypittämätöntä muistialuetta. Fun on funktionaalinen objekti ja Port Identifier ilmaisee Erlang-portin.

Atom-tietotyyppi on yksinkertaisesti nimetty vakio. Alla on esimerkkejä atomeista.

```
heippa
ala_viiva
'Heippa'
'Hei hei'
```

Tuple on kokoomatietotyyppi (compound data type), jolla on vakiomäärä termejä. Termit ovat Erlangissa tiedon yksiköitä (pieces of data). Tuplen termejä kutsutaan elementeiksi, ja elementtien määrä määrittää tuplen koon. Tämä menettelytapa muistuttaa taulukkorakennetta Java-kielessä.

Erlangissa on useita ennltamääriteltyjä funktioita tuplejen käsittelemiseksi. Esimerkiksi:

```
Joukko = {paavi, 85}.
element(1,Joukko).
tuple_size(Joukko).
```

Tässä tapauksessa element(1,P) viittaisi siis paavi-elementtiin ja Joukko-tuplen koko olisi 2.

List on kokoomatietotyyppi, jolla on vaihteleva määrä termejä. Jokaista List-tietotyypin termiä kutsutaan elementiksi, ja elementtien määrään viitataan Listin pituutena Javan tapaan. Alla on esimerkki list-tietotyypistä.

```
Lista = [1,x,d,9]
```

String-listat eivät ole tietotyyppejä Erlangissa. Kaksoislainausmerkeissä oleva merkkijono tarkoittaa sen sijaan samaa asiaa kuin lista merkkijonon sisältämistä merkeistä. Esimerkiksi String ”heippaön sama asia kuin lista [$h, $e, $i, $p, $p, $a].

Erlangissa ei ole Boolean-tietotyyppiä. Sen sijaan atomeja true ja false käytetään ilmaisemaan totuusarvoja.

3.2 Erlangin tyypitys

Dynaaminen tyypitys viittaa kieliin, joissa suurin osa tyyppien tarkastamisesta tehdään vasta suoritusaikana käännösajan sijasta (Wikipedia). Erlangin ensimmäisillä kehittäjillä oli tausta dynaamisesti tyypitettyjen kielten parissa, joten dynaaminen tyypitys oli heille luonnollinen valinta.

On usein esitetty, että staattisesti tyypitetyt kielet löytävät koodivirheet paremmin, ja ovat siksi turvallisempia. Kuitenkin Erlangia paljon käyttäneellä Ericsson-yhtiöllä on Erlangin turvallisuudesta huomattavaa näyttöä. Erlang on rakennettu niin, että yhden komponentin epäonnistumisen ei tulisi vaikuttaa koko järjestelmään. Ohjelmointivirheisiin, laitevikoihin ja joihinkin tietoverkko-ongelmiin on varauduttu Erlangissa. Fred Hébertin mukaan Erlang mahdollistaa odottamattomien virheiden käsittelyn, ohjelman jakamisen eri solmuihin tietoverkossa ja suorituksen jatkamisen virheistä huolimatta.

Siinä missä useimmat kielet pyrkivät tekemään ohjelmista virheettömiä, Erlangin strategia on hyväksyä virheiden tapahtuminen, ja varautua niihin. Tämän nojalla Erlangin dynaaminen tyypitys ei ole este ohjelmien luotettavuudelle ja turvallisuudelle.

Erlang on myös vahvasti tyypitetty kieli. Vahvasti tyypitetyt kielet rajoittavat eri tietotyyppien yhdistelyä operaatioissa (wikipedia). Alla on esimerkki operaatiosta, joka aiheuttaisi Erlangissa virheen ohjelman suorituksessa.

```
1 + "2".
** exception error: bad argument in an arithmetic expression
in operator  +/2
called as 1 + "2"
```

Erlangissa on kuitenkin tapoja muuntaa tietoa tietotyypistä toiseen.

Viittaukset menvät näin: [HEB13].

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.

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