Ohjelmointikielten periaatteet 2013

Jari Koskinen, Hansi Keijonen, Eero Laine

1. helmikuuta 2013

Sisältö

1 D-ohjelmointikieli

D-ohjelmointikieli on C/C++-kielten pohjalta kehitetty, ja sen pääasiallisena kehittäjänä on alusta lähtien toiminut ohjelmointikielten kääntäjiin erikoistunut Walter Bright. Kehitystyö on alkanut vuonna 1999 ja ensimmäinen julkaisu kääntäjästä on tehty vuonna 2001 Brightin omistaman yhtiön Digital Marsin toimesta. Ensimmäinen vakiintunut versio 1.0 julkaistiin kuitenkin vasta vuonna 2007. D-kielen nykyinen versio on 2.0, ja kielen kehitys jatkuu edelleen. Mukaan on myös liittynyt useita kehittäjiä Brightin rinnalle. Version 2.0 pohjalta on julkaistu vuonna 2010 Andrei Alexandrescun kirjoittama kirja The D Programming Language, jonka myötä version voidaan katsoa vakiintuneen.

Versio 1.0 kattoi imperatiivisen paradigman, oliopohjaisen paradigman ja metaohjelmointiparadgiman. Versio 2.0 toi mukanaan tuen funktionaaliselle ohjelmoinnille ja aktorimallin rinnakkaisohjelmointiin. Tässä kirjoituksessa käsitellään kielen versiota 2.0. Kieli sisältää paljon uudistuksia, jotka ovat parannuksia C/C++-kieliin tai muiden kielten, kuten C#:n ja Javan hyödylliseksi havaittuja ominaisuuksia. Samalla kielestä on pyritty jättämään pois C/C++- kielten taaksepäin yhteensopivuuden asettamia rajoitteita.

D on moniparadigmainen, oliopohjainen ja imperatiivinen ohjelmointikieli. D tarjoaa tuen myös funktionaaliselle ja metaohjelmoinnille. D:llä kirjoitetut ohjelmat käännetään käännös- ja linkitysvaiheen kautta kohdeympäristön konekielelle. Ohjelmoijalla on täysi pääsy laitetasolle, joten D soveltuu myös käyttöjärjestelmien ohjelmointiin. Muistinhallinta voidaan jättää automatiikan huoleksi, koska D sisältää C#:n ja Javan tapaan roskienkeruun, mutta sallii muistinkäsittelyn myös C/C++:lle tyypillisen tapaan. Perintä on toteutettu yksinkertaisella perinnällä, rajapinnoilla ja mixineillä. Kielen syntaksi muistuttaa hyvin paljon C++-kielen syntaksia.

Alla on esimerkki ohjelmasta, joka palauttaa annettua lukua suuremman Fibonaccin luvun.

```
1.  import std.stdio;
2.  import std.conv;
3.
4.  void main(string[] args) {
5.  uint f;
6.   try {
7.    f = parse!uint(args[1]);
8.   }
9.   catch {
10.   writeln("Error in arguments!");
11.   return;
12.  }
```

```
13.  writeln(args[1]);
14.  writeln(fib(f));
15. }
16.
17. ulong fib(uint n) {
18.  ulong iter(ulong fib_1, ulong fib_2) {
19.   return fib_2 > n ? fib_2 : iter(fib_1 + fib_2, fib_1);
20.  }
21.  return iter(1, 0);
22. }
```

Ohjelmakoodin toiminta on seuraavanlainen: riveillä 1 ja 2 ladataan käyttöön kirjastot syötön ja tulostuksen käsittelyä (std.stdio) ja tyyppimuunnoksia (std.conv) varten. Rivillä 7 muunnetaan saatu tekstimuotoinen parametri etumerkittömäksi kokonaisluvuksi. Rivit 9 - 11 ovat virheenkäsittelyä varten, ja virheen sattuessa ohjelmasta poistutaan. Rivillä 14 kutsutaan fib-funktiota. Fib-funktiossa on riveillä 18 ja 19 oma sisäfunktio, joka suorittaa varsinaisen laskennan. Sisäfunktiota kutsutaan ulomman funktion viimeisellä rivillä 21. Kyseessä on tällöin häntärekursio, jota varten D-kielessä on optimoija.

2 Erlang-ohjelmointikieli

Erlang on funktionaalinen kieli, joka on tarkoitettu hajautettujen järjestelmien ohjelmointiin. Erlang soveltuu tosiaikajärjestelmien ja laajasti skaalautuvien järjestelmien, kuten puhelinkeskusten, pankkijärjestelmien ja sähköisen liiketoiminnan kehittämiseen. Kieli sisältää laajan tuen rinnakkaisuudelle ja mahdollistaa erittäin vikasietoisten järjestelmien kehittämisen. Erlangin kehittämisen aloitti Joe Armstrong vuonna 1986 Ericsson-yhtiössä. Alunperin kieli on ollut suljetussa käytössä, mutta se julkaistiin avoimena vuonna 1998. Erlangin tietotyypit ovat dynaamisesti ja vahvasti tyypitettyjä.

Ohjelma, joka tulostaa ensimmäisen syöttölukua suuremman Fibonaccin luvun, näyttää Erlangilla seuraavalta. Kommentit alkavat %-merkillä. Ohjelman toimintaa selventävät numeroidut kommentit on selitetty koodin alla.

```
1.  -module(fibo).
2.  -export([fibo/1, compfibo/2, fiboplus/1]).
3.       fiboplus(N) -> compfibo(N,0).
4.       compfibo(N,S) -> Curr = fibo(S),
5.            if Curr > N -> Curr;
6.            true -> compfibo(N,S+1)
7.            end.
8.  fibo(0) -> 0 ;
```

```
9.  fibo(1) -> 1 ;
10. fibo(N) when N > 1 -> fibo(N-1) + fibo(N-2) .
11. fibo(0) -> 0 ;
12. fibo(1) -> 1 ;
13. fibo(N) when N > 1 -> fibo(N-1) + fibo(N-2) .
```

Ensimmäisenä tulee määritellä moduulin nimi, tässä tapauksessa 'fibo', rivillä 1. Jotta moduulissa olevia metodeita pystytään kutsumaan ohjelman ulkopuolelta, tulee ne määritellä export-lauseella, kuten rivillä 2 on tehty. Jokaisen määrityksen yhteydessä ilmoitetaan kutsussa tarvittavien parametrien määrä. Rivillä 3 on varsinainen päämetodi, jota kutsutaan halutulla parametrilla. Parametrina annetaan siis luku, jota suuremman (ensimmäisen) Fibonaccin luvun ohjelma laskee. Kyseessä oleva metodi kutsuu ainoastaan apumetodia compfibo, lisäten mukaan apuparametrin, joka pitää kirjaa siitä, monettako Fibonaccin lukua kulloinkin haetaan. Rivillä 4 compfibo-metodi tallettaa ensimmäisenä muuttujaan Curr Fibonaccin sarjan S:nen luvun. Jos tämä luku on suurempi kuin alkuperäinen annettu luku N, se palautetaan. Muussa tapauksessa compfibo-metodia kutsutaan uudelleen siten, että parametrin S arvoa kasvatetaan yhdellä. Näin seuraavalla kutsukerralla tutkitaan järjestyksessä seuraavaa Fibonaccin lukua. Rivillä 5 fibo-metodi laskee Fibonaccin N:nen luvun. Toimintatapa metodissa on hahmonsovitus, joka on yleinen myös Haskell-ohjelmointikielessä.

3 Kielten alkiorakenteen vertailu

3.1 D-kielen alkiorakenteen vertailua

D-kielen tunnuksissa voi olla kirjaimia, alaviivoja tai ISO/IEC 9899:1999(E)-standardissa määriteltyjä merkkejä (universal alphas), ja ensimmäistä merkkiä lukuunottamatta myös numerot ovat käytössä. Tunnusta ei voi myöskään aloittaa kahdella alaviivalla. D-kieli on "case sensitive", eli isot ja pienet kirjaimet on eroteltu toisistaan. Varattuja sanoja on noin sata, esimerkiksi Javastakin löytyvät termit abstract, final, switch ja class. Rivinvaihdoilla ja sisennyksillä ei D-kielessä ole kielen suorituksen kannalta merkitystä.

Merkkijonoliteraali on D:ssä joko merkkijono kaksinkertaisissa lainausmerkeissä, wysiwyg-merkkijono, ohjausmerkki (escape sequence), rajattu merkkijono (delimited string), järjestetty merkkijono tai heksamerkkijono. Rivin loppumista kuvaava \n lasketaan yhdeksi merkiksi kaikissa merkkijonoliteraaleissa.

Wysiwyg on lyhennys sanoista "what you see is what you get", ja wysiwyg-merkkijonoissa kaikki merkit toistetaan uskollisesti. Wysiwyg-merkkijonot voidaan erotella joko r- tai '-merkein. Merkkijonot r"c:\root\foo.exe"ja 'c:\root\foo.exe' ovat esimerkkejä wysiwyg-merkkijonoista. Merkkijonot kaksinkertaisissa lainausmerkeissä ovat -merkkien ympäröimiä. Ohjausmerkke-

jä voi sulauttaa näihin merkkijonoihin monista ohjelmointikielistä tutulla \-merkintätavalla, kuten esimerkiksi merkkijonossa "c:\\root\\foo.exe". Heksamerkkijonot mahdollistavat merkkijonoliteraalien luomisen heksadatasta. Heksadatan ei tarvitse muodostaa hyväksyttyjä UTF-merkkejä. Valinnainen StringPostfix-merkki antaa merkkijonolle määritellyn tyypin.

Rajoitetut merkkijonot käyttävät useanlaisia erottimia, esimerkiksi merkkejä ja tunnuksia. Erottimen täytyy saumattomasti jatkaa -merkistä ilman välilyöntejä, ja päättävän erottimen täytyy saumattomasti edeltää merkkijonon päättävää -merkkiä ilman välilyöntejä.

Tekstialkiot alkavat merkeillä q{ ja päättyvät merkkiin }. Välissä tulee olla valideja D-kielen tekstialkioita. Merkkiliteraalit ovat yksittäisiä merkkejä tai ohjausmerkkejä, jotka on ympäröity yksinkertaisilla lainausmerkeillä, ’ ’.

Kokonaisluvut voidaan määritellä useissa eri kantalukujärjestelmissä. Kymmenjärjestelmän luvut ovat yksinkertaisesti kymmenjärjestelmän numeroiden sarjoja, mutta binäärijärjestelmän luvut koostuvat binäärijärjestelmän numeroista, joita edeltää merkkojono 0b. Liukuluvut voidaan esittää sekä kymmenluku- että heksadesimaalijärjestelmässä.

3.1.1 D-kielen avainsanat

abstract	alias	align	asm	assert
auto	body	bool	break	byte
case	cast	catch	cdouble	cent
cfloat	char	class	const	continue
creal	dchar	debug	delegate	delete
default	deprecated	do	double	else
enum	export	extern	false	final
finally	float	for	foreach	foreach_reverse
function	goto	idouble	if	ifloat
immutable	import	in	inout	int
interface	invariant	ireal	is	lazy
long	macro	mixin	module	new
nothrow	null	out	override	package
pragma	private	protected	public	pure
real	ref	return	scope	shared
short	static	struct	super	switch
synchronized	template	this	throw	truetry
typedef	typeid	typeof	ubyte	ucent
uintn	ulong	union	unittest	ushort
version	void	volatile	wchar	while
with	___FILE___	___LINE___	___gshared	___traits
___vector	___parameters			

3.2 Erlangin alkiorakenne

Erlangin erottimet ovat:

```
( ) { } [ ] .  : | || ; , ? -> #
```

Erlangin avainsanat ovat:

```
after cond let when
begin end  of
case  fun  query
catch if   receive
```

Kaikki avainsanat ovat varattuja sanoja. Niiden lisäksi varattuja sanoja ovat:

```
band   bnot bor
bsl    bsr  bxor
div    not  or
```

```
orelse rem  try
when  xor
```

Operaattorit:

```
 +   -    *   /    div rem
 or  xor  bor bxor bsl bsr and band
 ==  /=   =:= =/=  =<  >   >=
 not bnot ++  --   =   !   <-
```

Operaattoreita ei voi käyttää atomiliteraaleina.

Ohjausmerkit (escape sequences):

```
\ b     backspace BS
\ d     delete DEL
\ e     escape ESC
\ f     form feed FF
\ n     linefeed LF
\ r     carriage return CR
\ s     space SPC
\ t     horizontal tab HT
\ v     vertical tab VT
\\      backslash \
\'      single quote '
\"      double quote "
```

\-merkillä voi kuvata myös oktaalinumeroita sekä ASCII-järjestelmän erikoismerkkejä.

Kokonaislukuliteraali voidaan ilmaista perinteiseen tapaan:

```
1 2 3 666
```

Tai merkitsemällä ennen #-merkkiä luvun kantaluku:

```
2#100100 16#FA66
```

Kääntäjä ymmärtää em. tapauksissa tulkita luvut binääri- ja heksadesimaaliluvuiksi.

Liukulukuliteraalit ilmaistaan mahdollisella etumerkillä, kokonaislukuosalla, desimaaliosalla ja mahdollisella eksponenttiosalla:

```
1.0 -5.1e6 0.346e-20
```

Merkkiliteraali ilmaistaan $ -merkin perään kirjoitetulla merkillä tai ohjausmerkillä (escape secuence) :

```
$R $s $$ $\\
```

Merkkijonoliteraali ilmaistaan merkeillä, jotka ovat "merkkien sisällä:

```
"Pekka Järvelä" "\n"
```

Atomiliteraalit ovat Erlangin erikoisuus. Ne ovat merkkijonoliteraalivakioita, joiden arvo on atomi itse. Atomeihin ei voi sitoa mitään arvoa. Esimerkiksi atomi

```
auto
```

tarkoittaa autoa, eikä mitään muuta. Atomi alkaa pienellä kirjaimella, _:lla tai @ -merkillä. Jos atomin haluaa aloittaa muulla tavoin, tulee se ympäröidä 'merkeillä.

Muuttujan nimen tulee alkaa isolla kirjaimella, @ :llä, tai _:lla:

```
Data I
```

Koska mitkään varatut sanat, operaattorit tai avainsanat eivät ala isolla kirjaimella, tulkintavirhe ei ole mahdollinen.

Universaalimerkkinä käytetään '_' -merkkiä. Sen voidaan ajatella olevan mikä tahansa merkki. Hahmonsovituksessa tähän merkkiin vertautuu mikä tahansa arvo totena.

Merkkiliteraali ilmaistaan $ -merkin perään kirjoitetulla merkillä tai ohjausmerkillä (escape secuence) :

```
$R $s $$ $\\
```

Merkkijonoliteraali ilmaistaan merkeillä, jotka ovat "merkkien sisällä:

```
"Pekka Järvelä" "\n"
```

Atomiliteraalit ovat Erlangin erikoisuus. Ne ovat merkkijonoliteraalivakioita, joiden arvo on atomi itse. Atomeihin ei voi sitoa mitään arvoa. Esimerkiksi atomi

```
auto
```

tarkoittaa autoa, eikä mitään muuta. Atomi alkaa pienellä kirjaimella, _:lla tai @ -merkillä. Jos atomin haluaa aloittaa muulla tavoin, tulee se ympäröidä 'merkeillä.

Muuttujan nimen tulee alkaa isolla kirjaimella, @ :llä, tai _:lla:

```
Data I
```

Koska mitkään varatut sanat, operaattorit tai avainsanat eivät ala isolla kirjaimella, tulkintavirhe ei ole mahdollinen.

Universaalimerkkinä käytetään '_' -merkkiä. Sen voidaan ajatella olevan mikä tahansa merkki. Hahmonsovituksessa tähän merkkiin vertautuu mikä tahansa arvo totena.

Kommentit ilmaistaan seuraavasti:

```
/* lohkokommentti */
//rivin perään tuleva kommentti
/+ alisteinen lohkokommentti +/
```

3.3 Esimerkkejä ohjelmakoodista

Alla on esimerkki D-kielen muuttujien näkyvyysalueista ja sisäfunktioista.

```
import std.stdio;

void main() {
 writeln("uloin a=", uloin());
}

int uloin() {
  int a=1;
  int keski() {
    int a=2;
    int sisin() {
      int a=3;
      return a;
    }
    writeln("sisin a=", sisin);
    return a;
  }
  writeln("keski a=", keski());
  return a;
}
```

Pääohjelmassa kutsutaan funktiota uloin(), joka kutsuu funktiota keski(), josta kutsutaan funktiota sisin(). Kaikissa funktioissa muuttujalla a on eri arvot, jonka kukin palauttaa kutsuvalle funktiolle. Ohjelman tuloste on seuraavanlainen:

```
sisin a=3
keski a=2
uloin a=1
```

D-kielessä voidaan muuttuja määritellä myös auto, jolloin muuttujan tyyppi päätellään käänösvaiheessa. Alla on esimerkki tällaisesta.

```
import std.stdio;

void main() {
  auto str = "merkkijonoliteraali";
  auto i = 123456789098776543;
  writeln(str, " ja ulong ", i);
}
```

Ohjelma tulostaa seuraavasti:

```
merkkijonoliteraali ja ulong 123456789098776543
```

3.4 Ratkaisujen vertailua

D-kieli sisältää paljon varattuja sanoja, joka on mahdollisesti osoitus paisuneesta kielestä. Toisaalta kieli on melko rikas ja mahdollistaa erilaisia ohjelmointitapoja. Erlangissa varattuja sanoja on vähän. Molemmat kielet ovat vahvasti tyypitettyjä.

4 Lähteet

D-kieli, http://dlang.org
The D Programming Language, Andrei Alexandrescu, 2010
Erlang, http://www.erlang.org
Learn you some Erlang, Fred Hébert