C1

Funcții și aritmetică

Despre: câteva întrebări pe care vi le puneți atunci când abordați un nou limbaj, *tipurile de date (arhi)cunoscute din alte limbaje*, *Tipurile compuse*, *liste și perechi*, *operatorii (operațiile)*, *Funcțiile*, *funcția increment*, *Rularea unui program*, *variabilele folosite la signaturile funcțiilor și semnul ->*

Răspundem în acest capitol la câteva întrebări pe care vi le puneți atunci când abordați un nou limbaj. Haskell 98 este un limbaj funcț ional pur, dar în care se poate programna și în stil imperativ (vom vedea că se folosește o noțiune care a făcut multă vâlvă în lumea matematicienilor, monada - de fapt o anumită structură algebrică). În Haskell 98 există de asemenea o serie de noțiuni specifice programării orientate obiect cum ar fi clasă, instanță a unei clase etc. Funcțiile, vom vedea, sunt polimorfice. Se pot descrie ușor de tot liste și arbori, reuniuni de tipuri ba chiar mulțimi ordonate și liste infinite. (Interesant, nu-i așa ?) Și totul se scrie rapid și compact (un algoritm Quicksort încape în doar patru-cinci rânduri de program !) și fără aglomerarea

aceea de paranteze cu care ne-au forțat să ne obișnuim limbaje ca Lisp (poreclit de unii "Limbajul InSuportabilelor Paranteze") sau ML. Toate aceste concepte le vom prezenta rând pe rând, răspunzând una câte una la niște întrebări pe care și le pune cel care învață un limbaj de programare. Începem cu cele mai simple lucruri, cum ar fi întrebarea dintâi:

- Cum se scriu în Haskell tipurile de date (arhi)cunoscute din alte limbaje: întregii, numerele reale și alte tipuri comune ?

Răspunsurile vor fi detaliate, fiecare subcapitol dintr-un capitol fiind centrat unei întrebări sau tratând, exhaustiv ori uneori tangențial, câte un subiect, caz în care subiectul este ulterior reluat într-un alt capitol. Subiectul este anunțat de titlul subcapitolului în cauză.

De exemplu, în capitolul I vom răspunde printre altele la întrebări cum ar fi :

- Cum se scriu în Haskell operatorii ?
- Cum se scrie o funcție simplă ?
- Cum se rulează un program ?

Ulterior vom trece la o serie de exemple, scriind mai întâi simple funcții în Haskell, și trecând gradat la exemple mai complicate. În primul capitol ne vom ocupa printre altele de niște exemple care ilustrează:

- Felul cum se scriu funcțiile
- Funcția increment
- Funcția factorial

Vom explica și cum veți rula aceste exemple pe computerul dumneavoastră dotat cu sistem de operare Linux. La realizarea acestei cărți am folosit Mandrake Linux 8.2 (iar ulterior am trecut la Mandrake Linux 10). Puteți folosi orice distribuție cu condiția să aveți

inclus în distribuție interpretorul Hugs sau să-l instalați ulterior. Hugs este disponibil atât ca arhivă zip cât și ca pachet .rpm. Teoretic vorbind, aceste exemple pot fi rulate și pe o mașină Windows pentru care dețineți software-ul necesar, o implementare de Haskell aliniată la standardul Haskell 98. Utilizatorii unor versiuni mai vechi sunt invitați să treacă la Haskell 98, versiunile mai vechi fiind actualmente depășite.

1.1. Tipurile Integer, Bool, Char și altele; Tipurile compuse, liste și perechi

Atunci când începem învățarea unui limbaj, deoarece limbajul este ceva destinat pentru a programa un calculator să prelucreze niște date, unul dintre primele lucruri care ne-ar interesa este ce fel de date putem prelucra cu acel limbaj și cum se descriu ele. De obicei ne intreresează să prelucrăm date din niște tipuri clasice: întregii, numerele reale, valorile logice (numite uneori valori booleene), simbolurile tipografice cunoscute sub denumitrea de "caractere" și citatele între ghilimele, numite stringuri. Iată cum se numesc aceste mulțimi de date - tipurile citate mai sus, în Haskell 98:

Numerele întregi se numesc *Integer* sau *Int.*
Numerele reale sunt de mai multe feluri,rețineți deocamdată tipul *Float*.
Valorile logice, booleene, două la număr (True și False) formează tipul *Bool*.
Caracterele tipografice sunt elemente din tipul (care include și simbolurile), *Char*.

Stringurile sunt de fapt **liste** de caractere iar tipul string se notează în consecință ca un tip compus: ***[Char].*** Parantezele pătrate avertizează că este vorba de listă de elemente de tipul descris în paranteză, acționând deci ca un constructor de tipuri (cam cum era perechea RECORD ... END în Pascal și Oberon numai că aici se declară liste nu structuri).

Notați și faptul că în biblioteca ce se încarcă la pornirea interpretorului (pe care v-o puteți imagina deocamdată ca un fel de <stdio.h> din C, cu toate că în realitate ea oferă mult mai multe lucruri), bibliotecă numită "Standard Prelude" sau mai pe scurt "Prelude" veți găsi și alte tipuri fie ele simple fie veritabile clase . Vă invităm să le descoperiți. Căutați cuvinte ca Rational, Double...

Rețineți: Cele mai simple tipuri de sunt:

Integer (sau ***Int***), ***Bool***, ***Float***, ***Char***

iar stringurile formează tipul ***[Char]*** denumit cu aliasul ***String***.

Deocamdată nu discutăm amănunte subtile despre tipurile de date . Acestea însă există: de exemplu despre diferența între Int și Integer programatorii hârșîiți v-ar spune că funcțile care folosesc pe unul sunt puțin mai mai rapide decât cele care îl folosesc pe celălalt.) Asemenea diferențe nu vor fi vizibile în exemplele noastre. De asemenea, nu punem în discuție aici pe () un fel de tip cu un singur element notat chiar cu (). Această valoare () va fi folosită la operații de intrare/ieșire.
Atenție: În Haskell numele tipurilor de date **încep întotdeauna cu**

majusculă. Regula este valabilă și pentru tipurile utilizator, motiv pentru care veți scrie tipul **Arbore** cu **A** majusculă, atunci când veți avea nevoie de un asemenea tip. În schimb funcțiile se scriu cu minusculă, de exemplu o funcție care incrementează o variabilă o veți numi **inc**. Haskell 98 este un limbaj din categoria celor în care minusculele și majusculele au sens diferit, fiind asemenea din acest punct de vedere cu limbajele C, Java și Oberon .

Tipuri compuse: Pe lângă modurile de a construi tipuri definite de utilizator (pe care le vom prezenta în alt capitol, Haskell permite construirea de *liste* și *perechi. Aceleași semne (paranteza pătrată respectiv cea rotundă și virgula care separă elementele) se folosesc și pentru a construi tipul compus din tipuri mai simple dar și pentru a construi valoarea compusă din valori mai simple.*

Exemple:
[Integer] - listă de întregi iar [1,2,3] - lista formată din 1 2 și 3
(Char,Int) - perechi de Char și Int ('b',4) - 'b' și 4 împerecheate

Atenție: Caracterele se scriu între ghilimele simple, obținute cu tasta de lângă Enter. Celălalt fel de ghilimele simple, inverse, ce se obțin cu tasta de sub Escape, sunt folosite în alt scop în Haskell (la definirea operatorilor infixați). Nu le confundați ! Vom remarca și faptul că stringurile admit o scriere dublă, atât ca liste de caractere cât și ca texte între ghilimele duble. Ex. ['a','l','f','a'] versus ”alfa”.

Observație: Elementele unei liste pot fi: - oricât de multe
- dar de același tip, unic

Dar: Elementele unei perechi sunt: - exact două
- dar pot fi din două tipuri diferite dinainte precizate

Vom vedea în capitolele următoare că utilizatorul poate :
- să introducă noi tipuri și noi *constructori de tip*
- să redenumească tipuri (de exemplu [Char] se poate redenumi String și în acest nume, String chiar se folosește ca atare)
- să inventeze noi feluri de date, definind *constructorii de date* (deoarece nu este obligatoriu să se folosesc același nume de funcție pentru a descrie cum se compun atât tipurile cât și valorile lor - dar despre acestea discutăm în alt capitol)
- există și alți constructori de tip. Pe lîngă (), [] , exemplele următoare îl vor folosi intensiv pe "->" ,cel care construiește tipul unei funcții.

1.2. Operatorii (operațiile)

Operatorii sunt grupați în 10 clase de prioritate. Un operator, chiar și unul definit de utilizator (Da există așa o posibilitate în Haskell !) trebuie plasat într-una din cele 10 clase de prioritate. În acest fel, Haskell poate stabili corect ordinea operațiilor dintr-o expresie, indiferent ce operatori sunt implicați. Unii operatori asociază implicit la stânga, alții la dreapta. Operatorii infixați care au un nume format din litere apar în programe cu acest nume încadrat de ghilimele simple inverse (de pe tasta cu semnul tilda). Regula aceasta este valabilă și pentru operatorii definiți de programator. Dacă veți deschide cu un editor de text fișierul

/usr/share/hugs/lib/Prelude.hs (Standard Prelude) veți găsi acolo, deja declarați, o serie întreagă de operatori. Îi vom comenta imediat pe cei mai mulți dintre ei, precizând pentru fiecare printr-o pereche literă - cifră dacă e asociativ la dreapta (R) sau stânga (L) și ce nivel de prioritate are.

Iată o mică listă de operatori declarați în Standard Prelude :

R9 **.**
L9 !!
R8 **^ , ^^ , ****
L7 *** , / , `quot` , `rem` , `div` , `mod` , :% , %**
L6 **+ , -**
R5 **:**
4 **==, /=, <=, <, >=,>, `elem`, `notElem`**
R3 &&
R2 **||**
L1 **>> , >>=**
R1 **=<<**
R0 $, $! , **`seq`**

Recunoașteți imediat aici: pe nivelul 6 sunt adunarea și scăderea iar acești operatori asociază la stânga. În Haskell **a+b+c** este deci înț eles ca **(a+b)+c** . Pe nivelul 7 găsiți înmulțirea, împărțirea și operatorii care calculează câtul și restul. Operatorii `div` și `mod` sunt similari cu omologii lor din Pascal. **div** face împărți-rea întreagă și obține câtul iar **mod** obține restul.

Remarcați că operatorul care implementează noțiunea de "diferit" este

scris în Haskell “/=” , spre deosebire de alte limbaje (Pascal, C, Oberon). Observați că relațiile de apartenență care exprimă noțiuni ca "a fi element" și "a nu fi element" au denumiri formate din litere: **`elem`** respectiv **`notElem`.**

Operațiile logice, conjuncția și disjuncția se notează ca în C și le găsiți pe nivelele de prioritate 2 și 3.

Operatorii de pe nivelul 1 servesc la scrierea operațiilor dintr-o monadă. Aceste structuri algebrice, monadele, provin din teoria categoriilor și se poate lucra (nativ) cu ele în Haskell 98 deorece clasele corespunzătoare sunt predefinite în Prelude. Un exemplu găsiți în capitolul 6. Poate n-ați știut că, de exemplu, un model de sistem care face calcule, model în care calculele dau valori și modifică o memorie (ca atribuirile) se implementează imediat în Haskell ca o monadă.

Nu uitați că Haskell este un limbaj funcțional, care operează cu funcții de la date la rezultate (acestea putând fi din tipuri diferite). Un calcul făcut de un program este până la urmă o asemenea funcție. Vom vedea că operatorul >>= este compunerea în succesiune a două calcule. Ceea ce se obține depinde evident de cele două calcule. Operatorul >>= , (pronunțat "bind") exprimă doar felul cum se combină, cum se succed ele.

.	**- este compunerea funcțiilor**
+	**- este ridicarea la putere**
`quot`	**- similară ca nume cu o funcție din Lisp**
`div`	**- împărțirea întreagă**
`mod`	**- restul dat de împărțirea întreagă**
+ , -	**- adunarea și scăderea, supraîncărcate**

:	**- cons, adăugarea unui elment în capul unei liste**
==, /=, <=, <, >=,>-	**relațiile, comparațiile, supraîncărcate**
`elem`, `notElem`	**- apartenența și neapartenența la o listă (mulțime)**
&&	**- conjuncția**
||	**- disjuncția, (negația se notează cu not)**
>> , >>=	**- operatori pentru monade (de obicei înlănțuiesc calculele fiind cumva similari lui ";" din limbajele imperative. Pot fi redefiniți în instanțele claselor monadice pentru a obține diverse efecte)**
=<<	**- la fel dar cu operanzii în ordine inversa**

Cu ajutorul lor sunt definiți și alți operatori, de exemplu folosindu-se operatorul "**:**" (pronunțat "cons" ca în Lisp) este definită concatenarea a două liste sau a două stringuri, notată cu **++** . Veți folosi frecvent acest operator.

1.3. Funcțiile. Exemplul Nr.1, funcția increment.

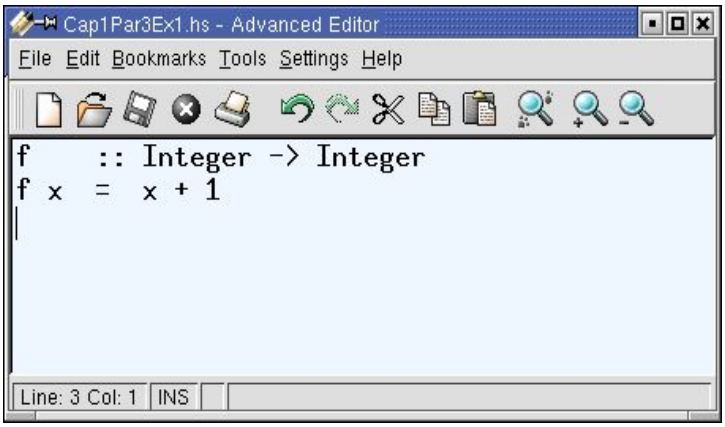
Matematicianul care face o declarație de funcție va scrie întâi semnătura acesteia (adică schema de tipuri căreia funcția i se conformează). În Haskell ea se nume-ște chiar tipul funcției, deoarece "->" este și el un constructor de tipuri, capabil să îmbine două tipuri și să obțină altul. (Obținându-se astfel o "funcție de la ... la...").

Exemplu: **f : Z -> Z** prin relația **f(x) = x + 1** se scrie în Haskell:

```
f   :: Integer -> Integer
f x  = x + 1
```

Cap1Par3Ex1.hs

Pentru a-l testa, scrieți acest mic program cu un editor de text :



Cap1Par3Ex1.hs

Puteți alege orice editor doriți din cele disponibile (kwrite, gedit, emacs, xemacs, vi, pico). Salvați programul sub numele **Cap1Par3Ex1.hs.**

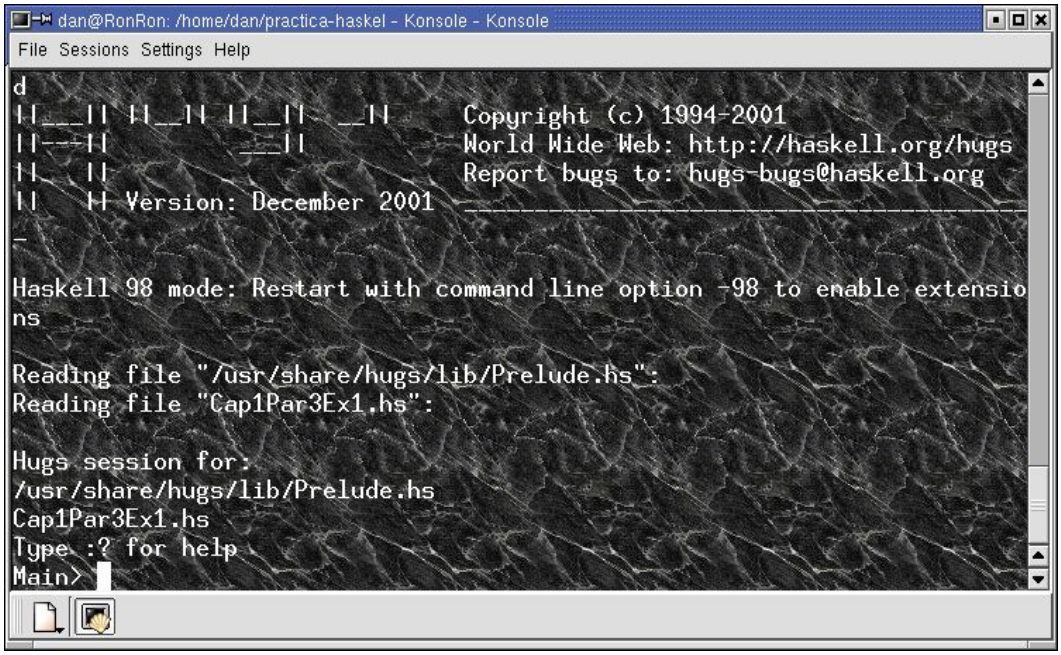
Câteva explicații sunt aici și acum necesare:

Alinierea pe verticală este extrem de importantă. În limbaje ca Pascal, C sau Oberon și multe altele, un programator putea scrie tot programul pe un singur rând, fără ca aceasta să deranjeze compilatorul sau interpretorul. Acest lucru nu se poate face în Haskell deoarece Haskell, pentru a elimina acele paranteze care ne copleșesc în Lisp sau ML, folosește un concept revoluționar: *sintaxa bidimensională.* Iată ce trebuie să știți deocamdată despre ea:

Trecerea pe rândul următor, dar mai la stânga decât pe precedentul

rând echivalează cu o închidere de structură sau paranteză. E ca și cum s-ar pune automat în Lisp paranteză închisă. (Rețineți că Haskell este un limbaj cu sintaxă bidimensională!). Această aranjare în pagină se numește cu termenul englezesc “layout”. Este chiar posibil ca interpretorul să semnaleze prezența incorectă a unei paranteze pe care dumneavoastră n-ați scris-o, din cauza unei asemenea erori de layout. În asemenea cazuri verificați alinierea pe verticală și aveți grijă ca nu cumva unele rânduri să înceapă mult prea din stânga. Pare un dezavantaj dar lipsa acelor paranteze din Lisp merită prețul acesta, după părerea mea. Practic, este esențial să aliniați corect pe verticală tot ce este în interiorul unei substructuri sintactice, sau, dacă sunteți începător, să păstrați alinierile pe verticală din exemplele nostre și să programați cam în același stil (sau unul mai bun). Notați și faptul că, urmare a sintaxei bidimensionale, în Haskell este important să puneți spațiile în anumite locuri. Un tab este considerat egal cu 8 spații, motiv pentru care trecerea la un editor de text pentru care TAB-ul are alt număr de spații poate creea unele probleme. Sfatul meu pentru începători: Lucrați cu tasta SPACE nu cu tasta TAB când aveți nevoie de spații.

Extensia fișierelor conținând surse de program Haskell este .hs . Mai există o extensie .lhs pentru "literate - Haskell" acele texte de lucrări în care totul este comentariu cu excepția liniilor care încep cu > și formează ele însele programul. (În Cap 6. se folosește acest stil de scriere.)



Pornirea interpretorului Hugs.

Hugs este numele interpretorului Haskell 98 livrat împreună cu distribuțiile de Linux. Nu este însă inclus în toate distribuțiile oferite spre download. Eu l-am găsit inclus în distribuțiile Mandrake 8.2 și Mandrake 10. Îl puteti lansa cu una din comenzile:

hugs

hugs <numefișier>

hugs “<numefișier>”

Ghilimele sunt necesare dacă numele de fișier conține spații. Observati în imaginea de mai sus că fiind lansat cu comanda

hugs Cap1Par3Ex1.hs

interpretorul deschide și citeste programe din două fisiere: Standard Prelude și fi-șierul dat de noi în linia de comandă.

În paragraful următor vom explica cum să rulați exemplele incluse în această carte.