Cap 3. Cât mai multe despre tipuri și crearea lor

3.1. Un vechi exemplu: Triunghiul . Declarația *data*

Cuvântul triunghi are mai multe înțelesuri, cel puțin în vorbirea curentă. Pentru cel care consemnează un raționament geometric, triunghiul este un *triplet de litere* (ex: ABC) din figura în discuție. Tot în plan, dar pentru cel care lucrează cu mijloacele geometriei analitice, un triunghi este desemnat de *trei perechi de coordonate*. În geometria analitică a spațiului cu trei dimensiuni triunghiul este dat de *trei triplete de coordonate*. Triunghiul mai poate fi un triunghi amoros adică un *triplet de nume de persoane* (practic notate prin *trei stringuri* cum ar fi "Ion", "Popescu", "Ioana") așa cum este în unele romane polițiste.

Pentru a introduce în programele Haskell acest tip cu toate accepțiunile lui simultane trebuie să putem numi pe de o parte *tipul compus triunghi* generat de alte trei tipuri identice (matematicienii l-ar numi produsul cartezian) cât și *tripleta* de valori care va forma o valoare a noului tip.

Declarațiile acestea se fac în Haskell folosind cuvântul **data**, cu care se introduc atât constructorul de tipuri "Triunghi" cât și constructorul de valori "Tripleta". Ambii constructori se scriu cu majuscule. Tipul comun al celor trei elemente care formează o tripletă a tipului triunghi poate fi un tip oarecare, îl notăm printr-o variabilă de tip, fie ea "a". Astfel vom putea declara folosi cu aceeași declarație fele de fel de triunghiuri. Declarația este:

```
data Triunghi a = Tripleta a a a
```

Cap3Par1Ex1.hs

În urma declarației de mai sus tipul triunghi care rezultă este un tip polimorfic, putând exista în programul care-l folosește fel de fel de triunghiuri. De exemplu:

Tripleta 'A' 'B' 'C'	este un triunghi de caractere, din tipul Triunghi Char
Tripleta (1.2,2.3) (1.5,2.7) (1.9,3.0)	este un triunghi de perechi de numere reale din tipul Triunghi (Float , Float)
Tripleta ”Ion” ”Popescu” ”Ioana”	este un triunghi de nume (stringuri) din tipul Triunghi String

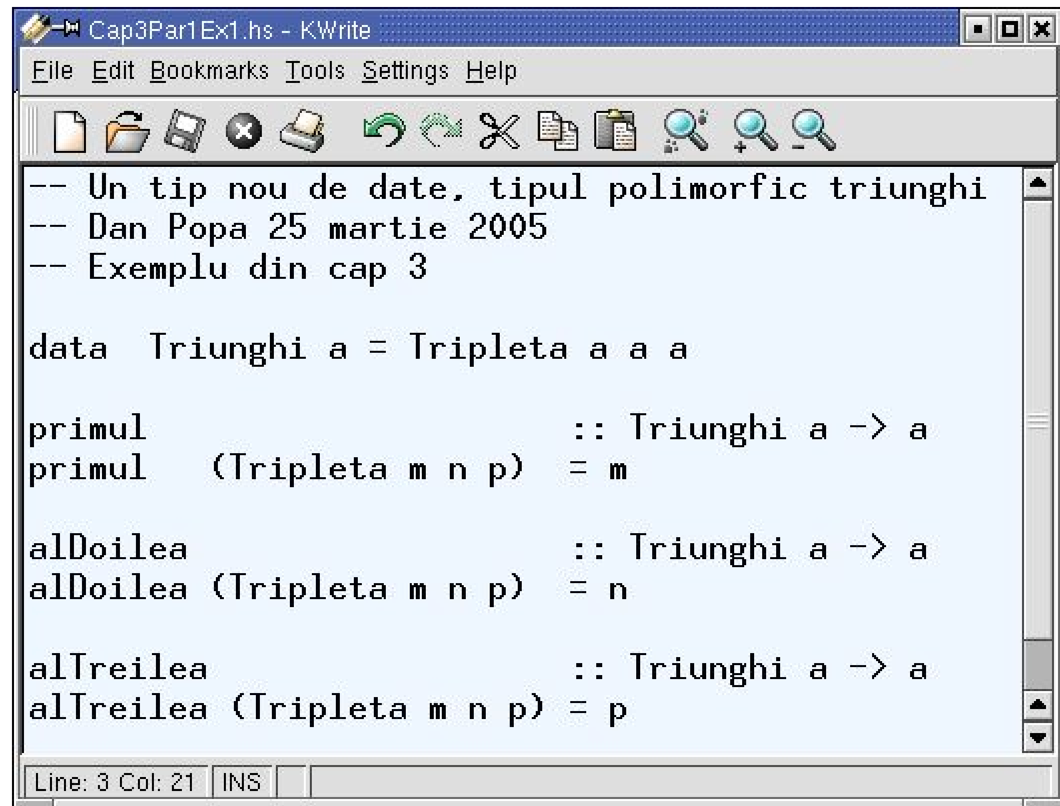
Observați că prin înlocuirea variabilei de tip a cu un tip concret , Char, String,(Float , Float) obținem din tipul cel mai general al triunghiurilor tipuri particulare. Aceasta este o proprietate a sistemului de tipuri din Haskell, unei entități i se pot asocia mai multe tipuri dintre care unul este cel *mai general tip* al ei și este unic, bine determinat.
Rețineți: Constructorii de tipuri ca Triunghi și constructorii de date ca Tripleta se scriu în Haskell cu majuscule pentru a-i deosebi de funcțiile care se scriu cu minuscule.

Să trecem la partea practică. Se pot scrie imediat funcțiile care selectează dintr-un triunghi unul sau altul dintre elemente:

```
primul                  :: Triunghi a  -> a
primul (Tripleta m n p) = m
```

Cap3Par1Ex1.hs

Și similar se scriu funcțiile alDoilea și alTreilea, care extrag din Tripleta elementul al doilea și respectiv al treilea.



Ca și tipul triunghi, aceste funcții *sunt polimorfice* așa că are sens să puneți interpretorul să calculeze valoarea lor pentru mai multe tipuri (concrete) de argumente. Aceste funcții vor extrage elementul dorit din orice fel de triplete. Evaluați:

> primul (Tripleta (1.5,3.0) (1.4,2.0) (7,9)

(1.5,3.0)

Am extras primul element dintr-o tripletă de puncte din spațiul bidimensional.

> primul (Tripleta [1,2,3] [10,20,30] [40,50,60])

[1,2,3]

Am extras primul element dintr-o tripletă de liste de întregi.

> alDoilea (Tripleta ('a','b') ('c','d') ('e','g'))

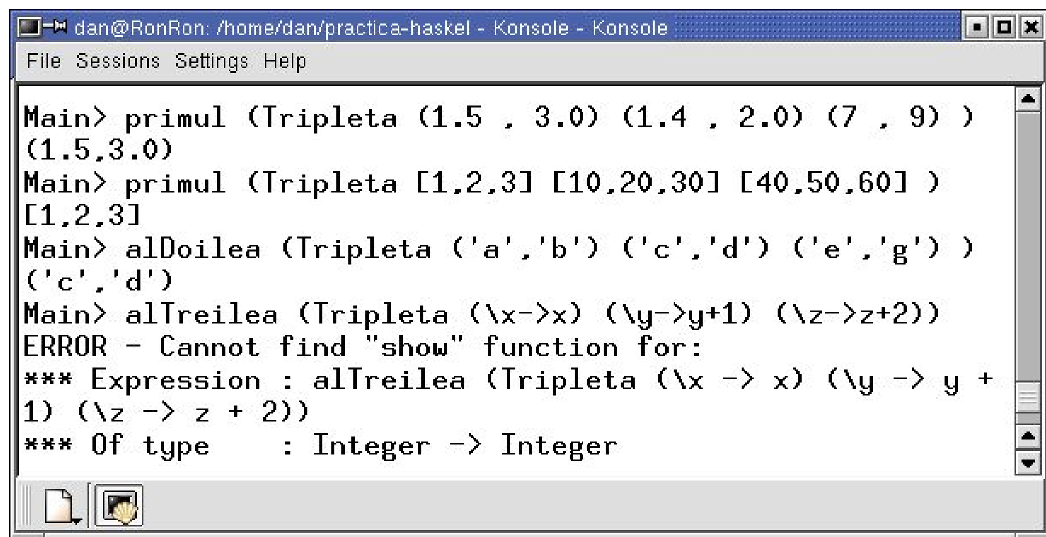
('c','d')

Am extras al doilea element dintr-o tripletă de perechi de caractere.

Încercând să evaluați:

> alTreilea (Tripleta (\x->x) (\y->y+1) (\z->z+2))

obțineți cunoscutul mesaj de eroare care ne spune că interpretorul nu poate afișa rezultatul evaluării.



Citind mesajul constatăm că interpretorul Haskell 98 a stabilit corect că expresia dată are tipul **Integer->Integer** dar nu dispune de o funcț ie “show” pentru a afișa asemenea valori. În schimb puteți încerca să evaluați:

> alTreilea (Tripleta (\x->x) (\y->y+1) (\z->z+2)) 19

21

Și obțineți răspunsul corect ! Interpretorul a aplicat a treia funcție din tripletă argumentului dat, 19. Același rezultat l-ați fi obținut evaluînd:

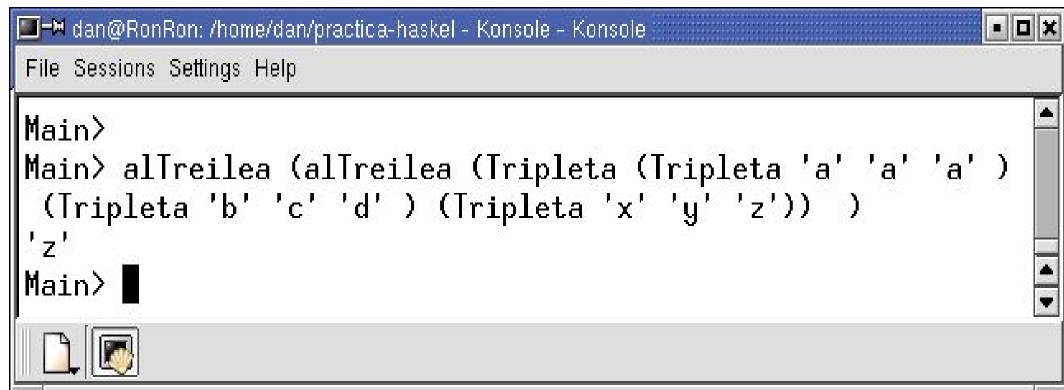
```
> (\z->z+2) 19
21
```

Supliment: Pentru Haskell funcțiile și constructorii de date sunt cam același lucru. Practic un constructor de date este o funcție care construiește data compusă, dar rămâne o funcție. Prin urmare ei se pot folosi în compuneri ca și funcțiile.

Exemplu: Al treilea din a treia dintr-o tripletă de triplete.

```
> alTreilea ( alTreilea (Tripleta   (Tripleta 'a' 'a' 'a')
                                    (Tripleta 'b' 'c' 'd')
                                    (Tripleta 'x' 'y' 'z') )
```

Rezultatul va fi: 'z'



Rețineți: *Constructorii de tipuri pot fi* [ ] , (,) , -> *dar și constructori definiți de utilizator dați printr-un cuvânt scris cu majusculă.*

Pentru fiecare tip astfel construit – care poate fi și polimorfic – trebuie

să indicați măcar un constructor de valori,numit *constructor de date* , a șa cum era Tripleta pentru Triunghi. (În următorul subcapitol vom vedea că pot fi chiar mai mulți constructori de date, dacă tipul dorit este o reuniune de tipuri.)

Constructorul de date nu este obligat să folosească aceeași notație, același nume ca și constructorul de tip asociat lui. În cazul tipurilor listă, perche, funcție, constructorul de date și cel de tip au aceeași notație, respectiv: [ ] , (,) , -> Am fi putut declara tipul Triunghi cu constructor de date numit tot Triunghi în loc de Tripleta și exemplele anterioare ar fi funcționat perfect. Haskell deduce după contextul folosirii dacă este vorba de un constructor de tip sau de unul de date. De exemplu în semnătura unei funcții (care este tipul ei) ceea ce apare este constructorul de tip. Iar pe rândurile următoare, unde este descris procesul de calcul al valorilor apar constructorii de date.

Notați și faptul că în cazul tipului Triunghi constructorul de date Tripleta se comportă ca o funcție cu semnătura:

```
Tripleta  :: a -> a -> a -> Triunghi a
```

Motiv pentru care o dată din acest tip se va scrie (Tripleta ... ...) cu trei argumente din același tip.

3.2. Tipuri utilizator și reuniunea

Tipurile reuniune sunt în esență tipuri utilizator dotate cu mai mulți constructori de date. Pot exista mai multe feluri de date, fiecare cu constructorul ei și ele sunt toate incluse în același tip. Constructorii

pot fi însoțiți sau nu de variabile de tip. Vom vedea (la exemplul cu arborii) că tipurile astfel construite pot fi și recursive. Dar începem cu niște tipuri nerecursive:

```
data Vreme = Buna | Rea
data Ziua  = Luni | Marti | Miercuri | Joi |
                  Vineri | Sambata | Duminica
```

Atenție când treceți cu scrierea pe rândul următor. Aveți grijă să nu scrieți mai din stânga decât pe precedentul, în cadrul aceleiași structuri sintactice. Ar avea efectul unei închideri de structură.
De altfel tipul Bool, este definit (ne confirmă Paul Hudak, John Peterson și Joseph H. Fasel în "A Gentle Introduction to Haskell 98") exact așa:

```
data Bool = True | False
```

Pentru acest tip, Bool, Haskell 98 are o funcție "show" în interpretor, deci putem afișa fiecare din aceste valori atunci când o expresie se evaluează la așa ceva:

```
> True
True
```

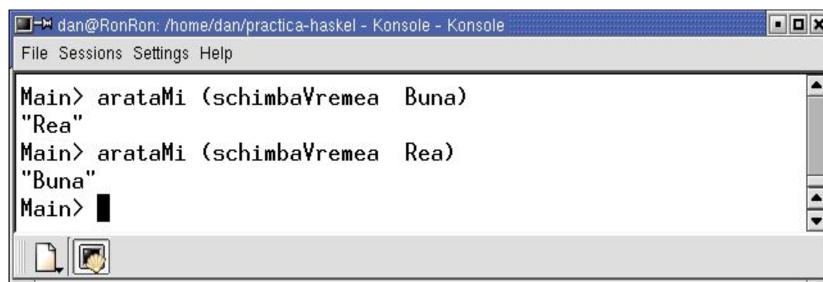
Notă: Avansații în lucrul cu Haskell 98 știu să declare un tip de date nou astfel încât sistemul să creeze automat și funcțiile "show" necesare noului tip. Acest mod de a le declara nu e prezentat în acest subcapitol.

Pentru tipul nostru, Vreme, vom recurge la scrierea unei funcții separate pentru afișare. Micul nostru program ar putea arăta așa:

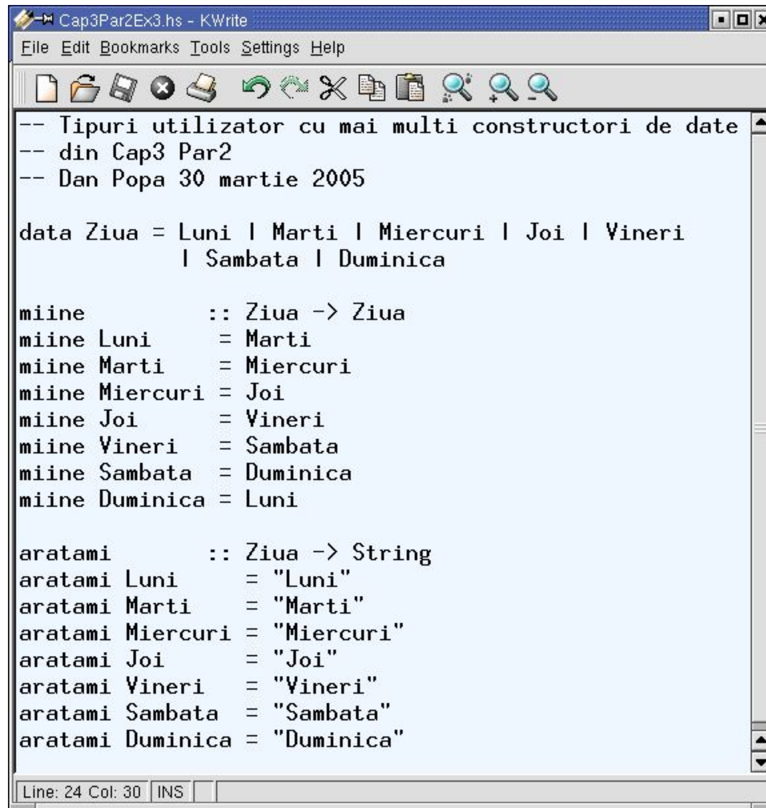
```
data Vreme = Buna | Rea

schimbaVremea      :: Vreme -> Vreme
schimbaVremea Buna = Rea
schimbaVremea Rea  = Buna

arataMi            :: Vreme -> String
arataMi Buna       = "Buna"
arataMi Rea        = "Rea"
```

Cap3Par2Ex2.hs

Și ar rula ca în imaginea de mai jos:



Asemănător arată și exemplu cu zilele săptămânii. Aici sunt ceva mai mulți constructori de date, fără variabile, câte unul pentru fiecare zi a săptămânii. Ei se comportă, intuitiv vorbind, ca niște constante. Puteți să vi-i imaginați ca atare.



Funcția “aratami” (scrisa fara “-”) va converti numele zilei la tipul string în vederea afișării. O altă variantă ar fi să scrieți o asemenea funcție folosind patternuri cu gardă.

Iată și interpretorul Haskell răspunzând la întrebări despre zilele săptămânii: