**MINISTERUL EDUCAŢIEI, CULTURII ŞI CERCETĂRII AL REPUBLICII MOLDOVA**

**UNIVERSITATEA DE STAT „ALECU RUSSO” DIN BĂLŢI**

**FACULTATEA DE ŞTIINŢE REALE, ECONOMICE ȘI ALE MEDIULUI**

**CATEDRA DE MATEMATICĂ ȘI INFORMATICĂ**

**MONADELE ÎN HASKELL**

**REFERAT LA DISCIPLINA ,, PROGRAMAREA FUNCȚIONALĂ”**

**Autor**:

Studentul grupei IS21Z

Țurcanu Tudor

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Conducător științific:**

Olesea SKUTNIȚKI

magistru, lect. univ.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**BĂLȚI, 2020**

Cuprins:

[INTRODUCERE 3](#_Toc57055533)

[Monadele. Construcția -DO 4](#_Toc57055534)

[Monadele și monadele tranformere. 5](#_Toc57055535)

[CONCLUZII 8](#_Toc57055536)

[Bibliografie 9](#_Toc57055537)

# 

# INTRODUCERE

Haskell este un limbaj de programare cu scop general , static tastat , pur funcțională limbaj de programare cu tipul de inferență și evaluare leneș . Dezvoltat pentru a fi potrivit pentru predare, cercetare și aplicații industriale, Haskell a inițiat o serie de caracteristici avansate ale limbajului de programare, cum ar fi clasele de tip , care permit supraîncărcarea operatorului în condiții de siguranță . Implementarea principală a lui Haskell este Glasgow Haskell Compiler (GHC). Acesta poartă numele logicianului Haskell Curry .

Haskell este un limbaj ce este utilizat în multe domenii, cum ar fi în cel academic și industrial, la fel având o popularitate destul de mare. Haksell este un limbaj ușor de a fi înțeles, modern, și având posibilități de Cross-Platform.

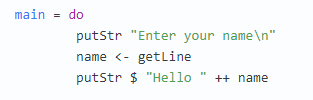
Monadele sunt structuri algebrice (M,>>= ,return) provenite din topologia algebrica sau/si din teoria categoriilor. Au intrat insa in "computer science" pe usa din dos :) a cresteri productivitatii muncii si extensibilitatii aplicatiilor.

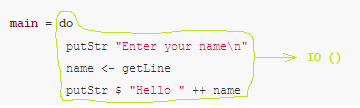
Avantajul teoretic major al folosirii monadelor il constituie faptul ca pot separa DESCRIEREA unei succesiuni de calcule (pe care o exprimati in Haskell in do-notatie) atat de MODUL LOR DE INLANTUIRE (exprimat prin operatorul "bind") cat si de FORMA STRUCTURILOR DE DATE (exprimata prin constructorul de date al monadei si functia return care produce valori monadice).Practicianul programator percepe insa monadele si do-notatia ca un limbaj imperativ inclus in Haskell. Legile monadei garanteaza functionarea imperativa a do-notatiei, conform cu intuitia.

Monada este un tip de date al acțiunilor compuse. Monada este o clasă de constructori de tip ale căror valori reprezintă astfel de acțiuni. Poate că IO este cea mai recunoscută: valoarea IO a este „o rețetă pentru obținerea unei valori din lumea reală”.

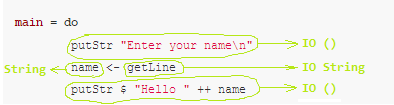
# Monadele. Construcția -DO

Exemplu:

  
Fiecare do-construcție are tipul “**m a**”,unde “**m**” – este monada. În cazul dat monada este **IO**.

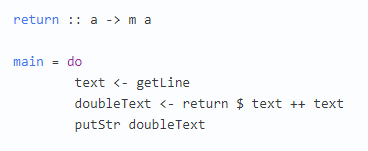


Fiecare linie din do-construcție la fel are tipul „**m a**”. Unde „**a**” în fiecare linie poate fi diferit.

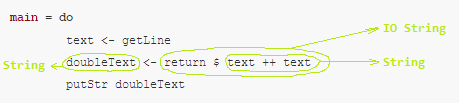


Simbolul “**<-**” , transformă tipul “**IO String**” în tipul „**String**”

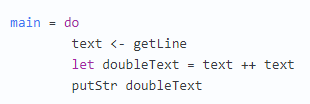
Dacă trebuie să efectuăm câteva calcule în monadă care nu au legătură cu această monadă, atunci putem folosi funcția **return**.



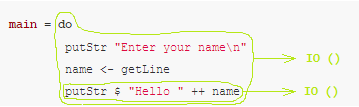
Funcția **return** înfășoară orice tip „a” în tipul monadic „m a”.



În acest exemplu, folosind **return**, expresia de tip "String" este convertită în tipul "IO String", care este apoi extins înapoi la "String". Alternativ, în construcția-do puteți utiliza cuvântul cheie *let*.



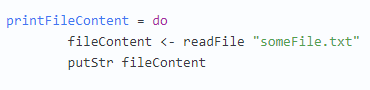
Toată construcția-do ia tipul ultimei linii.



Să presupunem că doriți să citiți conținutul unui fișier. Pentru aceasta aveți funcția *readFile*.



După cum puteți vedea, funcția returnează „IO String”. Dar avem nevoie de conținutul fișierului ca „String”. Aceasta înseamnă că trebuie să ne executăm funcția în interiorul do-construcției.



Aici variabila *fileContent* este de tip “String” și putem lucra cu ea ca un șir obișnuit (de exemplu, îl afișăm). Rețineți că funcția *printFileContent* este de tipul "IO ()"



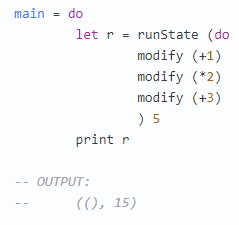
# Monadele și monadele tranformere.

Imaginați-vă că o monadă este un spațiu în cadrul căruia puteți efectua anumite acțiuni specifice acestui spațiu.

De exemplu, în monada „IO”, puteți trimite text pe consolă.



Monada “State” are o stare externă pe care o putem modifica.

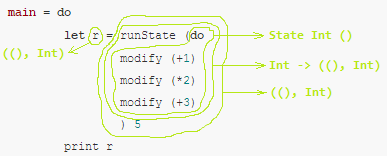


În acest exemplu, se ia numărul 5, se adaugă 1 la el, rezultatul se înmulțește cu 2, apoi se adaugă încă 3. Rezultatul este 15.

Folosind funcția *runState*



Deci “se lansează” monada dată.

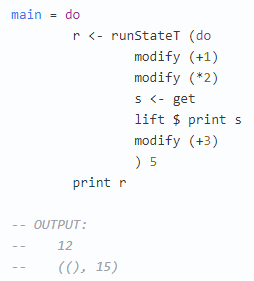


Monada poate fi privită din două părți: din interior și din exterior. Din interior, putem efectua unele acțiuni specifice unei anumite monade. Și afară - putem „rula”, „imprima”, o putem converti într-un tip non-monadic.

Acest lucru permite să includem o do-construcție în alta, ca în exemplul de mai sus. Monada *IO* este singura monadă care nu poate fi privită din exterior. Totul ajunge să fie cuibărit în *IO*. Monada *IO* este fundamentul.

Exemplul de mai sus are anumite limitări. În interiorul monadei *State*, nu putem efectua acțiunile disponibile în *IO*.

Ați ajuns „suspendați în aer”, ați pierdut contactul cu solul,pentru a rezolva această problemă, există tranformatoarele de monade.



Acest program face la fel ca cel anterior. Am înlocuit *State* cu *StateT* și am adăugat două linii:

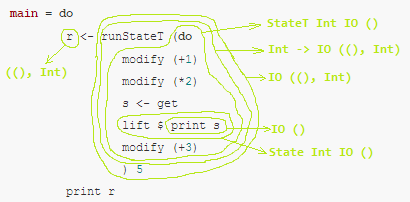


cu care se trimite rezultatul intermediar la consolă.

Aici *runStateT* pornește monada *StateT*, iar funcția de ridicare „ridică” operația disponibilă în *IO* către monada *StateT*.



Examinați cu atenție modul în care tipul este convertit secvențial în acest exemplu.



Operațiunea „print s” este de tip „IO ()”. Cu lift se „ridică” pană la tipul „StateT Int IO ()”. Construcția interioară a do-ului este acum de tipul "StateT Int IO ()". Se „rulează” și se obține tipul „Int -> IO ((), Int)”. Apoi se conectează valoarea „5” și se obține tipul „IO ((), Int)”.

Întrucât s-a obținut tipul „IO”, poate fi folosit în do-construcția externă. Săgeata „<-” elimină tipul monadic și returnează „((), Int)”. Rezultatul „((), 15)” este transmis la consolă.

În interiorul *StateT*, poate fi schimbată starea externă și pot fi efectuate operații de Întrare / ieșire. Adică. monada *StateT* nu se „atârnă în aer” ca *State*, ci rămâne legată cu monada IO externă.

Astfel, un program poate avea o grămadă de monade incluse unul în celălalt. Unele dintre aceste monade vor fi legate între ele, altele nu.

# CONCLUZII

Avantajul practic al folosirii monadelor pentru programare in Haskell il constituie faptul ca ele definesc un fel de sublimbaj al Haskell-ului, cu nuanta imperativa.

E ca un fel de limbaj imperativ inclus in limbajul functional Haskell. Mai mult, programele scrise în el sunt simple deoarece poate ascunde detalii ale lor in felul cum este definit operatorul >>= (pronunta"bind").

Exemplu: folositi o *[monadă a stărilor](https://wiki.haskell.org/Monada_a_starilor" \o "Monada a starilor)* și toate detaliile mai putin semnificative le ascundeti în stare și în definirea lui ">>=".

Un alt avantaj: Haskell fiind un limbaj functional pur, teoria spune ca nu puteti pur si simplu insera "print-uri" intr-o functie care se executa cu speranta ca ea va afisa outputuri utile pentru depanare.

Monadele permit sa depasiti limitarea de mai sus: Puteti scrie functia in do-notatie in maniera monadica si folosi o [monada cu stare si I/O](https://wiki.haskell.org/index.php?title=Monada_cu_stare_si_I/O&action=edit&redlink=1" \o "Monada cu stare si I/O (page does not exist)) (string de I/O) iar la finalul executiei veti gasi in stringul de I/O toate mesajele emise.

# Bibliografie

[https://wiki.haskell.org/Monade](https://vk.com/away.php?utf=1&to=https%3A%2F%2Fwiki.haskell.org%2FMonade)

[https://habr.com/ru/post/315022/](https://vk.com/away.php?utf=1&to=https%3A%2F%2Fhabr.com%2Fru%2Fpost%2F315022%2F)