Cursul 2 - Plan

- Clase(de tipuri) în limbajul Haskell
- Definirea funcţiilor
  - Prin folosirea unor funcţii existente
  - Expresii condiţionale
  - Ecuaţii cu gardă
  - Potrivire şabloane
    - Şabloane tuple
    - Şabloane listă
    - Şabloane întregi
  - Lambda expresii
  - Secţiuni

Clasele de bază

- Tip – colecţie de date
  - Bool, Char, String, Int, Integer, Float
  - Tipul listă
  - Tipul tuplă
  - Tipul funcţie
- Clasă – colecţie de tipuri care suportă operaţii supraîncărcate numite *metode*
  - Eq, Ord, Show, Read, Num, Integral, Fractional

`Eq` – clasa tipurilor cu egalitate

- Tipuri ce au valori care pot fi comparate pentru egalitate şi neegalitate
- Metodele:
  - `(==) :: Eq a => a -> a -> Bool`
  - `(/=) :: Eq a => a -> a -> Bool`
- Tipuri componente:
  - Tipuri de bază: `Bool, Char, String, Int, Integer, Float`
  - Tipuri listă şi tuple cu elemente (componente) tipuri din clasa `Eq`

`Ord` – clasa tipurilor ordonate

- Tipuri ce sunt instanţe ale clasei Eq şi care au valorile total ordonate
- Metodele:
  - `(<) :: Ord a => a -> a -> Bool`
  - `(<=) :: Ord a => a -> a -> Bool`
  - `(>) :: Ord a => a -> a -> Bool`
  - `(>=) :: Ord a => a -> a -> Bool`
  - `min :: Ord a => a -> a -> a`
  - `max :: Ord a => a -> a -> a`
- Tipuri componente:
  - Tipuri de bază: `Bool, Char, String, Int, Integer, Float`
  - Tipuri listă şi tuple cu elemente (componente) tipuri din clasa `Ord`

Exemple

```
Hugs.Base> min [1,2,3,4] [4,3]
[1,2,3,4]
Hugs.Base> min [1,2,3,4] [1,2,3,4,3]
[1,2,3,4]
Hugs.Base> min [1,2,3,4] [1,2,3,1,3]
[1,2,3,1,3]
Hugs.Base> min (1,True) (1,False)
(1,False)
```

`Show` – clasa tipurilor "showable"

- Tipuri ce conţin valori exprimabile prin şiruri de caractere
- Metodele:
  - `show :: Show a => a -> String`
- Tipuri componente:
  - Tipuri de bază: `Bool`, `Char`, `String`, `Int`, `Integer`, `Float`
  - Tipuri listă şi tuple cu elemente (componente) tipuri din clasa `Show`

Exemple

```
Hugs.Base> show 76890
"76890"
Hugs.Base> show [[12,33],[1],[1,2,3]]
"[[12,33],[1],[1,2,3]]“
Hugs.Base> show ("True", True)
"(\"True\",True)"
Hugs.Base> show (True, True)
"(True,True)"
Hugs.Base> show ('a',777)
"('a',777)"
```

`Read` - clasa tipurilor ce pot fi "citite"

- Tipuri ce conţin valori care pot fi convertite din şiruri de caractere
- Metodele:
  - `read :: Read a => String -> a`
- Tipuri componente:
  - Tipuri de bază: `Bool`, `Char`, `String`, `Int`, `Integer`, `Float`
  - Tipuri listă şi tuple cu elemente (componente) tipuri din clasa `Read`

Exemple

```
Hugs.Base> read "1234"
ERROR - Unresolved overloading
*** Type        : Read a => a
*** Expression  : read "1234"

Hugs.Base> read "1234" ::Int
1234
Hugs.Base> read "1234"+ read "12"
1246
Hugs.Base> read "12" < read "1"
False
```

Exemple

```
Hugs.Base> read "ala"::String
"
Program error: Prelude.read: no parse

Hugs.Base> read " "ala" "::String
ERROR - Undefined variable "ala"
Hugs.Base> read " \"ala\" "::String
"ala"
Hugs.Base> read "\"ala\"" < read "\"alo\""

Program error: Prelude.read: no parse

Hugs.Base> "ala" < "alo"
True
```

Num - clasa tipurilor numerice

- Tipuri ce sunt instanţe ale claselor **Eq** sau **Show** şi au valori numerice (**Int, Integer, Float**)
- Metodele:

```
(+) :: Num a => a -> a -> a
(-) :: Num a => a -> a -> a
(*) :: Num a => a -> a -> a
negate :: Num a => a -> a
abs :: Num a => a -> a
signum :: Num a => a -> a
```

Exemple

```
Hugs.Base> negate 989898
-989898
Hugs.Base> abs (-9)
9
Hugs.Base> abs( negate (-2))
2
Hugs.Base> signum( negate (2))
-1
Hugs.Base> signum 666
1
```

`Integral` - clasa tipurilor întregi

- Tipuri ce sunt instanţe ale clasei **`Num`** şi au valori întregi (**`Int, Integer`**)
- Metodele:

```
div :: Integral a => a -> a -> a
mod :: Integral a => a -> a -> a
```

- Metodele pot fi utilizate şi ca operatori infix dacă se scriu **`` `div` ``**, **`` `mod` ``** (atenţie: ` şi nu ’)

Exemple

```
Hugs.Base> 22 `mod` 5
2
Hugs.Base> 22 `mod` 3
1
Hugs.Base> 22 `div` 3
7
Hugs.Base> div 22 3
7
Hugs.Base> mod 22 3
1
```

`Fractional` - clasa tipurilor reale

- Tipuri ce sunt instanţe ale clasei **`Num`** şi au valori neîntregi (**`Float`**)

- Metodele:

```
(/) :: Fractional a => a -> a -> a
recip :: Fractional a => a -> a
```

Exemple

```
Hugs.Base> 7/2
3.5
Hugs.Base> 7./2
ERROR - Undefined variable "./"
Hugs.Base> 7.0/2
3.5
Hugs.Base> 7.3/2
3.65
Hugs.Base> recip 2
0.5
Hugs.Base> recip 1
1.0
Hugs.Base> recip 1.5
0.666666666666667
```

Definirea funcţiilor

- Prin folosirea unor funcţii existente:

```
isDigit :: Char -> Bool
isDigit c = c >= '0' && c <= '9'

even :: Integral a => a -> Bool
even n = n `mod` 2 == 0

rupeLa :: Int -> [a] -> ([a], [a])
rupeLa n xs = (take n xs, drop n xs)
```

Exemple

```
Hugs.Base> :type isDigit
ERROR - Undefined variable "isDigit"
Hugs.Base> :load test.hs
Main> :type isDigit
isDigit :: Char -> Bool
Main> isDigit '5'
True
Main> isDigit 'o'
False
```

Exemple

```
Main> :type rupeLa
ERROR - Undefined variable "rupeLa”
Main> :reload
Main> :type rupeLa
rupeLa :: Int -> [a] -> ([a],[a])
Main> rupeLa 3 [1,2,3,4,5,6]
([1,2,3],[4,5,6])
Main> rupeLa 3 [1,2,3,4,5,6,7]
([1,2,3],[4,5,6,7])
```

Definirea funcţiilor

- Expresii condiţionale:

  `if` *condiţie* **`then`** *res1* **`else`** *res2*

  - *res1* şi *res2* sunt expresii de acelaşi tip
  - Nu există **`if`** fără **`else`** în Haskell

```
semn::Int -> Int
semn n = if n < 0 then -1 else
                 if n == 0 then 0 else 1
```

Exemple

```
Main> :type semn
ERROR - Undefined variable "semn"
Hugs.Base> :reload
Main> :type semn
semn :: Int -> Int
Main> semn (-400)
-1
Main> semn 0
0
```

Definirea funcţiilor

- Expresii case:

```
f1 x =
    case x of
     0 -> 1
     1 -> x + 2
     2 -> x * 3
     _ -> -1
f2 x = case x of
     {0 -> 10; 1 -> 2*x; 2 -> x*x+2; _ -> 0}

Case> f1 1
3
Case> f2 2
6
```

Definirea funcţiilor

- Ecuaţii "guarded":
  - Alternativă la condiţii cu if
  - Secvenţă de expresii logice: gărzi
  - Secvenţă de expresii (rezultate) de acelaşi tip
  - Sintaxa:

nume_func parametri | *garda1* = *exp1*
| *garda2* = *exp2*
...
| *gardan* = *expn*

  - Ultima gardă poate fi **`otherwiswe`** (definită în biblioteca cu valoarea **`True`**)

Exemple

```
semng n      | n < 0      = -1
             | n == 0     = 0
             | otherwise = 1

Main> :type semng
ERROR - Undefined variable "semng"
Main> :reload
Main> :type semng
semng :: Int -> Int
Main> semng (-77)
-1
Main> semng 12
1
Main> semng 0
0
```

Definiţii locale

- O funcţie care are în expresia sa o frază de forma "where ..." spunem că are o definiţie locală; definiţia din fraza where este valabilă doar local
- Exemplu: f(x,y) = (a+1)(a+2), unde a = (x+2)/3

```
f  :: (Float, Float) ->Float
f(x,y) = (a+1)*(a+2) where a = (x+2)/3

f  :: (Float, Float) ->Float
f(x,y) = (a+1)*(b+2)
          where a = (x+y)/3
                b = (x+y)/2
```

Definiţii locale

```
g  :: Integer -> Integer -> Integer
g x y | x <= 10 = x + a
      | x > 10 = x - a
      where a = square(y+1)
```

- Clauza **where** califică ambele ecuaţii gardate

```
Main> g 3 4
28
Main> f (5,6)
35.0
Main> g 11 2
2
```

Definirea funcţiilor

- Tehnica şabloanelor ("pattern matching")
  - Secvenţă de şabloane – rezultate
  - Sintaxa:

nume_func :: tipul_funcţiei

Şablon_1 = exp1

Şablon_2 = exp2

…

Şablon_n = expn

  - Dacă se potriveşte primul şablon, se alege ca rezultat exp1, dacă nu se încearcă al doilea şablon, etc.
  - Şablonul _ ("wildcard") poate fi folosit pentru a exprima potrivirea cu orice valoare

Exemple

```
non :: Bool -> Bool
non False = True
non True = False

conj :: Bool -> Bool -> Bool
conj True True   = True
conj True False  = False
conj False True  = False
conj False False = False

(&) :: Bool -> Bool -> Bool
True & True = True
_ & _ = False
```

Exemple

```
Main> :type non
non :: Bool -> Bool
Main> :type conj
conj :: Bool -> Bool -> Bool
Main> conj (1<2) (2<11)
True
Main>  (1<2) `conj` (2<11)
True
Main> (7-2>5) & (2<9)
False
Main> (7-2>=5) & (2<9)
True
```

Exemple

- O altă modalitate de utilizare a şabloanelor (utilă în evaluarea lazy): dacă primul argument este True (la conjuncţie) rezultatul este valoarea celui de-al doilea

```
(&) :: Bool -> Bool -> Bool
True && b = b
False && _ = False

(||) :: Bool -> Bool -> Bool
False || b = b
True || _ = True
```

Exemple

- Nu se poate folosi numele unui argument de două ori:

```
(&&) :: Bool -> Bool -> Bool
b && b = b
False && _ = False

Test> :load "j:\\2006-07\\PF\\Curs07\\Exemple\\C2\\fun1.hs"
ERROR file:j:\2006-07\PF\Curs07\Exemple\C2\fun1.hs:4 - Repeated variable "b" in
     pattern
```

- Soluţia: folosirea gărzilor:

```
(&) :: Bool -> Bool -> Bool
b & c     | b == c  = b
          | otherwise = False

Fun1> :reload
Fun1> (7-2>=5) & (2<9)
True
Fun1> (7-2>=5) & (21<9)
False
```

Definirea funcţiilor

- Tehnica şabloanelor ("pattern matching"):
  - Şabloane de tip tuple: o tuplă de şabloane este un şablon

```
fst :: (a, b) -> a
fst (x, _) = x
snd :: (a, b) -> b
snd (_, y) = y
```

Definirea funcţiilor

- Tehnica şabloanelor ("pattern matching"):
  - Şabloane de tip listă: o listă de şabloane este un şablon

```
test3a     :: [Char] -> Bool
test3a ['a', _, _] = True
test3a _ = False

Main> test3a [ 'a','c', 'r']
True
Main> test3a [ 'a','c', 'r', 'a']
False
```

Utilizarea operatorului ":" la liste (cons)

- Orice listă este construită prin repetarea operatorului cons(":") care înseamnă adăugarea unui element la o listă
  - Lista [1,2,3,4] înseamnă
    1:(2:(3:(4:[])))

```
test:: [Char] -> Bool
test ('a': _) = True
test _ = False
```

- Şabloanele listă trebuiesc puse în paranteză

Utilizarea operatorului ":" la liste (cons)

```
Hugs.Base> :reload
Main> :type test
test :: [Char] -> Bool
Main> test ['q', 'w', 'e']
False
Main> test ['a', 'q', 'w', 'e']
True
```

Utilizarea operatorului ":" la liste (cons)

```
null :: [a] -> Bool
nul[]    = True
nul(_:_) = False

head :: [a] -> a
head(x:_) = x

tail :: [a] -> [a]
tail (_:xs) = xs
```

Definirea funcţiilor

- Tehnica şabloanelor ("pattern matching"):
  - Şabloane "întregi": expresii de forma n+k unde n este o variabilă întreagă iar k este o constantă întreagă pozitivă

```
pred   :: Int -> Int
pred 0        = 0
pred(n+1)     = n
```

- Şabloanele întregi se potrivesc doar cu expresii pozitive
- Şabloanele întregi trebuiesc puse în paranteză