```
{- Catcgoria A. Functii de baza

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

even, odd :: Integral a => a -> Bool

(+), (*), (-), (/) :: Num a => a -> a -> a

(<), (<=), (>), (>=) :: Ord => a -> a -> Bool

(==), (/=) :: Eq a => a -> a -> Bool

(&&), (||) :: Bool -> Bool -> Bool

not :: Bool -> Bool

max, min :: Ord a => a -> a -> a

isAlpha, isAlphaNum, isLower, isUpper, isDigit ::
Char -> Bool

toLower, toUpper :: Char -> Char

digitToInt :: Char -> Int

ord :: Char -> Int

chr :: Int -> Char

Intervale

[first..], [first,second..], [first..last],
[first,second..last]

-}

{- Categoria B. Functii din biblioteci

sum, product :: (Num a) => [a] -> a

sum [1.0,2.0,3.0] = 6.0

product [1,2,3,4] = 24


and, or :: [Bool] -> Bool

and [True,False,True] = False

or [True,False,True] = True


maximum, minimum :: (Ord a) => [a] -> a

maximum [3,1,4,2]  =  4

minimum [3,1,4,2]  =  1

reverse :: [a] -> [a]

reverse "goodbye" = "eybdoog"

concat :: [[a]] -> [a]

concat ["go","od","bye"]  =  "goodbye"


(++) :: [a] -> [a] -> [a]

"good" ++ "bye" = "goodbye"


(!!) :: [a] -> Int -> a

[9,7,5] !! 1  =  7


length :: [a] -> Int

length [9,7,5]  =  3


head :: [a] -> a

head "goodbye" = 'g'


tail :: [a] -> [a]

tail "goodbye" = "oodbye"


init :: [a] -> [a]

init "goodbye" = "goodby"


last :: [a] -> a

last "goodbye" = 'e'


takeWhile :: (a->Bool) -> [a] -> [a]
```

takeWhile isLower "goodBye" = "good"

take :: Int -> [a] -> [a]

take 4 "goodbye" = "good"

dropWhile :: (a->Bool) -> [a] -> [a]

dropWhile isLower "goodBye" = "Bye"

drop :: Int -> [a] -> [a]

drop 4 "goodbye" = "bye"

elem :: (Eq a) => a -> [a] -> Bool

elem 'd' "goodbye" = True

replicate :: Int -> a -> [a]

replicate 5 '*' = "*****"

zip :: [a] -> [b] -> [(a,b)]

zip [1,2,3,4] [1,4,9] = [(1,1),(2,4),(3,9)

-}

{- Categoria C. Map, Filter, Fold

map :: (a -> b) -> [a] -> [b]

map (+3) [1,2] = [4,5]

filter :: (a -> Bool) -> [a] -> [a]

filter even [1,2,3,4] = [2,4]

foldr :: (a -> b -> b) -> b -> [a] -> b

foldr max 0 [1,2,3,4] = 4

(.) :: (b -> c) -> (a -> b) -> a -> c

($) :: (a -> b) -> a -> b

(*2) . (+3) $ 7 = 20

flip :: (a -> b -> c) -> b -> a -> c

flip (-) 2 3 = 1

-}

`:`	`a -> [a] -> [a]`	Add a single element to the front of a list. `3:[2,3]` ⇝ `[3,2,3]`
`++`	`[a] -> [a] -> [a]`	Join two lists together. `"Ron"++"aldo"` ⇝ `"Ronaldo"`
`!!`	`[a] -> Int -> a`	`xs!!n` returns the `n`th element of `xs`, starting at the beginning and counting from 0. `[14,7,3]!!1` ⇝ `7`
`concat`	`[[a]] -> [a]`	Concatenate a list of lists into a single list. `concat [[2,3],[],[4]]` ⇝ `[2,3,4]`
`length`	`[a] -> Int`	The length of the list. `length "word"` ⇝ `4`

`head,last`	`[a] -> a`	The first/last element of the list.
		`head "word"` ⇝ `'w'`
		`last "word"` ⇝ `'d'`

`tail,init`	`[a] -> [a]`	All but the first/last element of the list.
		`tail "word"` ⇝ `"ord"`
		`init "word"` ⇝ `"wor"`

`replicate`	`Int -> a -> [a]`	Make a list of `n` copies of the item. `replicate 3 'c'` ⇝ `"ccc"`
`take`	`Int -> [a] -> [a]`	Take `n` elements from the front of a list. `take 3 "Peccary"` ⇝ `"Pec"`
`drop`	`Int -> [a] -> [a]`	Drop `n` elements from the front of a list. `drop 3 "Peccary"` ⇝ `"cary"`
`splitAt`	`Int->[a]->([a],[a])`	Split a list at a given position. `splitAt 3 "Peccary"` ⇝ `("Pec","cary")`
`reverse`	`[a] -> [a]`	Reverse the order of the elements. `reverse [2,1,3]` ⇝ `[3,1,2]`
`zip`	`[a]->[b]->[(a,b)]`	Take a pair of lists into a list of pairs. `zip [1,2] [3,4,5]` ⇝ `[(1,3),(2,4)]`
`unzip`	`[(a,b)] -> ([a],[b])`	Take a list of pairs into a pair of lists. `unzip [(1,5),(3,6)]` ⇝ `([1,3],[5,6])`

Figure 6.1 Some polymorphic list operations from `Prelude.hs`