

Prácticas Haskell

Contents

Chapter 1

FormulasProposicionales

```
module FormulasProposicionales (  
    Prop, Formula, Clausula, fncAlista, clausulaLista, formulaAfnc, listaAtupla,  
    esPositivo, esClausula, borrarLista, indice, esClausulaHorn, resolvente,  
    deMorgan, distributiva, resolucion, getFormula, getInt, menu, demoMode  
    ) where
```

Práctica final de Programación Declarativa (PD) de Ingeniería Informática de la UCM del curso 2022-2023

1.1 Types

```
data Prop  
    Atomic predicate
```

```
instance Read Prop  
instance Show Prop  
instance Eq Prop
```

```
data Formula  
    Formula constructor
```

```
instance Read Formula
instance Show Formula
instance Eq Formula
```

```
type Clausula = [Prop]
    Alias for list os propositions as clause
```

1.2 Functions

1.2.1 Parsers

```
fncAlista :: Formula -> [Clausula]
```

Description

Translates a CNF function to a list of clauses

Example

```
>>> fncAlista (And (Or (Atom "A") (Atom "B")) (Not (Atom "B")))
[[Pos "A",Pos "B"],[Neg "B"]]
```

```
clausulaLista :: Formula -> Clausula
```

Description

Translates a Formula wich without (^) to a list of proposition

Example

```
>>> clausulaLista (Not (Atom "q"))
[Neg "q"]
```

```
>>> clausulaLista (Or (Not (Atom "Q")) (Atom "R"))
[Neg "Q",Pos "R"]
```

```
formulaAfnc :: Formula -> Formula
```

Description Applies distributiva and deMorgan to a Formula to change it to its NCF

Example

```
>>> formulaAfnc (And (Not (And (Atom "A") (Atom "B"))) (Atom "C"))
((!"A"^^"C")v(!"B"^^"C"))
```

```
listaAtupla :: [Clausula] -> [Clausula] -> [(Clausula, Clausula)]
```

Description

Aux function for `resolucion` that adds two list of lists to a list of tuples of each list

Examples

```
>>> listaAtupla [[Neg "A"], [Pos "A", Neg "B"]] [[Neg "A"], []]
[[([Neg "A"],[Neg "A"]),([Pos "A",Neg "B"],[])]]
```

1.2.2 Utilities

```
esPositivo :: Prop -> Bool
```

Description

Checks if a proposition is positive or negative

Example

```
>>> esPositivo (Pos "q")
True
```

```
>>> esPositivo (Neg "q")
False
```

```
esClausula :: Formula -> Bool
```

Description

Checks if a formula can be converted to a clause

Example

```
>>> esClausula (Or (Atom "A") (And (Atom "B") (Atom "C")))
False
```

```
>>> esClausula (Or (Not (Atom "A")) (Atom "B"))
True
```

```
borrarLista :: [Clausula] -> Clausula -> [Clausula]
```

Description Aux function for resolution that erases a Clause that is no longer needed

Example

```
>>> borrarLista [[Neg "Q", Neg "S"], [Pos "S", Neg "Q"]] [Pos "S", Neg "Q"]
[[Neg "Q", Neg "S"]]
```

```
indice :: [Clausula] -> Clausula -> Int -> Int
```

Description

Aux function for resolution that gets the index of a given clause

Example

```
>>> indice [[Neg "Q", Neg "S"], [Pos "S", Neg "Q"]] [Pos "S", Neg "Q"] 0
1
```

1.2.3 Operations on Formulas

```
esClausulaHorn :: Clausula -> Bool
```

Description

Checks if a given clause is a Horn's clause (one or zero positive propositions)

Example

```
>>> esClausulaHorn [Neg "A", Pos "B", Pos "C"]
False
```

```
>>> esClausulaHorn [Neg "A"]
True
```

```
>>> esClausulaHorn [Neg "A", Neg "B", Pos "C"]
True
```



```
resolvente :: Clausula -> Clausula -> Clausula
```

Description

Given a Clausula gives the resolvents with another Clausula

Example

```
>>> resolvente (clausulaLista (Or (Atom "P") (Or (Not (Atom "Q")) (Atom "R")))) (clausulaLista (Or
[Neg "Q",Pos "P",Pos "S"]
```

```
deMorgan :: Formula -> Formula
```

Description

Applies deMorgan properties to a Formula

Example

```
>>> deMorgan (Not (And (Atom "A") (Atom "B")))
(!"A"v!"B")
```

```
>>> deMorgan (Not (Or (Atom "A") (Atom "B")))
(!"A"^!"B")
```

```
>>> deMorgan (Not (Or (Atom "A") (Not (Atom "B"))))
(!"A"^"B")
```

```
>>> deMorgan (Not (And ((Not (Or (Atom "A") (Atom "B")))) (Atom "B")))
(("A"v"B")v!"B")
```

```
distributiva :: Formula -> Formula
```

Description

Applies distributed properties to a Formula but only to transform to NCF, not complete distributed.

Example

```
>>> distributiva (And ((Or (Atom "A") (Atom "B"))) (Atom "B"))
(("A"^"B")v("B"^"B"))
```

```
>>> distributiva (Not (Or (And (Atom "A") (Atom "B")) (Atom "C")))
!((("A"∨"C")^(("B"∨"C")))
```

```
resolucion :: [Clausula] -> Clausula -> IO Clausula
```

Description

Makes resolvents for a function until no more resolvents are possible

1.2.4 IO

```
getFormula :: IO Formula
```

Description

Reads a Formula from the standard input

```
getInt :: IO Int
```

Reads an int from the standard input

```
menu :: Formula -> Formula -> IO ()
```

Description

Simple menu to test functions

```
demoMode :: IO ()
```

Description

Demo for the program

Try: f: (And (And (Or (Atom A) (Not (Atom B))) (Or (Not (Atom A)) (Atom B)))) (Atom A) r: (Not (Atom B))

Chapter 2

Interprete

```
module Interprete (  
    Var, Program, showProgram, VarVal, State, mostrarStack,  
    Symbol(Equals, Times, Minus, Plus),  
    Comparator(LEQ, GEQ, Less, Greater, Not, And, Or, Eq), Operation(V, I, O),  
    Instruction(W, A), Comp(Void, CC, CB, CO), isVal, getVal, setVal, resolver,  
    calc, calcular, ejecutarAux, ejecutar, ejecuta, ejecutarAuxIO, ejecutarIO,  
    ejecutaIO, s0, s1, s2, factorial, s3, s4, s5, sumador, testResults,  
    testMode, freeMode, fileMode, getInt, entrypoint, main  
) where
```

Práctica final de Programación Declarativa (PD) de Ingeniería Informática de la UCM del curso 2023-2024

2.1 DATA TYPES

```
type Var = String  
    Alias for Variables  
  
type Program = [Instruction]  
    Alias for a list of instructions  
  
showProgram :: Program -> String
```

Description

Shows a full program

```
type VarVal = (Var, Int)
```

Alias for the program way of handling with variables

```
type State = [VarVal]
```

Alias for the program stack

```
mostrarStack :: State -> IO ()
```

Description

Shows the current state of the stack and waits for an enter key press to continue execution

```
data Symbol
```

Possible arithmetic operations in the program

Constructors

```
= Plus
| Minus
| Times
| Equals
```

```
instance Read Symbol
```

```
instance Show Symbol
```

```
data Comparator
```

Possible logic operations in the program

Constructors

```
= Greater
| Less
| GEQ
| LEQ
| Eq
| Not
| And
| Or
```

```
instance Read Comparator
instance Show Comparator
```

```
data Operation
```

Arithmetic operations in the program. Can use variables or integers.

Constructors

```
=  O      Full operation
      Operation variable to change value
      Symbol   arithmetic operation
      Operation Operation to perform
|  I      Operation is only an integer
      Int      Integer to operate with
|  V      Operation uses a variable
      Var      Variable to get the value from
```

```
instance Read Operation
instance Show Operation
```

```
data Instruction
```

Instructions inside the program, can be assigning new values to variables or while loops

Constructors

```
=  A      Assign operations
      Var  Variable to change value
      Operation Operation to change value
|  W      While loop
      Comp Comparator to end the loop
      Program Set of instructions inside the loop
```

```
instance Read Instruction
instance Show Instruction
```

```
data Comp
```

Boolean comparators for while loops (w)

Constructors

=	CO		Arithmetic comparator
		Operation	Variable to compare
		Comparator	Comparator used
		Operation	Integer to compare to
	CB		Boolean Comparator
		Bool	First part of the comparison
		Comparator	Comparator used
		Bool	Second part of the comparison
	CC		Comparator of comparators
		Comp	Variable to compare
		Comparator	Comparator used
		Comp	Integer to compare to
	Void		Void comparator, used for not clauses

```
instance Read Comp
instance Show Comp
```

2.2 STACK OPERATIONS

`isVal`

```
:: State  current stack of the program
-> Var    Variable to check
-> Bool   Variable is in stack or not
```

Description

Checks if a given variable name is in the stack.

`getVal`

```
:: State  Current stack of the program
-> Var    Variable to check
-> Int    Value of the given variable
```

Description

Gives a the stored value for a value for a variable. If the variable is not in the stack, the function throws an error.

`setVal`

```

::  State    Current stack of the program
->  Var      Variable to update
->  Int      New value
->  State    Updated stack

```

Description

Changes the value of a given variable to a new one.

2.3 INSTRUCTION OPERATIONS

```
resolver :: Comp -> State -> Bool
```

Description

Computes a boolean from the different comparator types

```
calc
```

```

::  Int      First int
->  Symbol   Operation to perform
->  Int      Second int
->  Int      Result

```

Description

Performs an arithmetic operation (`Symbol`) with two Integers

```
calcular
```

```

::  Operation Operation to perform
->  State      Current state of the program
->  Int        Result of the operation

```

Description

Performs an arithmetic operation given the operation and the current stack of the program

2.4 EXECUTION

`ejecutarAux`

```

:: Instruction  Instruction to execute
-> State       Current state of the program
-> State       Next state of the program

```

Description

Auxiliary function for `ejecutar` splitted into the different types of instructions available.

`ejecutar`

```

:: Program  Set of instructions to compute
-> State    Current state of the program
-> State    New state of the program

```

Description

Executes all the instructions in a program in sequence

`ejecuta`

```

:: Program
-> State    Initial state
-> Int      Value of the last initialized variable

```

Description

Executes all the instructions in a given program with a given initial state and returns the last variable declared

2.5 IO EXECUTION

`ejecutarAuxIO`

```

:: Instruction
-> State    Current state of the program
-> IO State Next state of the program

```


Description

Executes an instruction showing the current state of the stack

`ejecutarIO`

```
:: Program   Set of instructions
-> State     Initial stack
-> IO State   Value of last initialized variable
```

Description

Executes an entire program showing the state of the stack before every instruction. See `ejecutarAuxIO`

`ejecutaIO`

```
:: Program   Set of instructions
-> State     Initial stack
-> IO Int     Value of last initialized variable
```

Description

Executes an entire program and returns the value of the R variable showing the state of the stack before every instruction. See `ejecutarIO`.

2.6 EXAMPLES

`s0 :: State`

Initial state for `factorial`

`s1 :: State`

Initial state for `factorial`

`s2 :: State`

Initial state for `factorial`

`factorial :: Program`

Description

Factorial calculation for a number given

```

Y := X
R := 1
while (Y > 0) {
  R := R * Y
  Y := Y - 1
}
```

s3 :: State

Initial state for `sumador`

s4 :: State

Initial state for `sumador`

s5 :: State

initial state for `sumador`

`sumador` :: Program

Description

Summation of from 1 to a given number

```

i := 1
R := 0
while (i < X && R != \textbf{INT\_MAX}){
  R := i + R
  i := i + 1
}
```

2.7 EXECUTION MODES

`testResults`

```

:: Bool      debug mode enabled
-> Program   Instructions to execute
-> State      Initial state for the program
-> Int        Expected result of the program
-> IO ()
```

Description

Executes a program with a given state and shows the result compared to the expected result

`testMode`

```
:: Bool    debug mode enabled  
-> IO ()
```

Description

Execution of varios programs with different initial stacks and showing results compared to expected results

`freeMode`

```
:: Bool    debug mode enabled  
-> IO ()
```

Description

Lets the user write their own program and initial state and shows the result

`fileMode`

```
:: Bool    Debug mode enabled  
-> IO ()
```

Description

Lets the user give a file with a written program an then asks for a initial state. Shows results after execution

2.8 UTILITIES

```
getInt :: IO Int
```

Descripción

Coge un número desde la entrada.

```
entrypoint :: IO ()
```

Description

Main menu for the program, lets the user choose mode and graphical options

```
main :: IO ()
```

Description

Entrypoint

Chapter 3

Labo1

```
module Labo1 (  
    time, yearsSeg, daysSeg, hoursSeg, minsSeg, years, days, hours, mins, segs,  
    sol1a, sol1b, sol2A, sol2B, sol3, calculoDigitos, sumaDigitos, reduccion,  
    bools  
) where
```

3.1 EJERCICIO 1

3.1.1 A)

```
time :: Integer  
    tiempo base del ejercicio 1  
  
yearsSeg :: Integer  
    Segundos totales en un año  
  
daysSeg :: Integer  
    Segundos totales en un día
```

```
hoursSeg :: Integer
```

Segundos totales en una hora

```
minsSeg :: Integer
```

Segundos totales en un minuto

```
years :: Integer
```

Calculo de los años equivalentes a `time`

```
days :: Integer
```

Calculo de las horas equivalentes a `time` quitando `years`

```
hours :: Integer
```

Calculo de las horas equivalentes a `time` quitando `days` y `years`

```
mins :: Integer
```

Calculo de las horas equivalentes a `time` quitando `days`, `years` y `hours`

```
secs :: Integer
```

Calculo de las horas equivalentes a `time` quitando `days`, `years`, `hours` y `mins`

```
sol1a :: (Integer, Integer, Integer, Integer, Integer)
```

Tupla con el tiempo total de `time` en años, días, horas, minutos y segundos

3.1.2 B)

```
sol1b :: Integer -> (Integer, Integer, Integer, Integer, Integer)
```

Descripción

Coge el tiempo dado y lo convierte a (Años, Días, Horas, Minutos, Segundos)

Ejemplos

```
>>> sol1b 1000000  
(0,11,13,46,40)
```

3.2 EJERCICIO 2

3.2.1 A)

```
sol2A :: Integer -> Bool
```

Descripción

Coje el año dado y devuelve True si es bisiesto, False en caso contrario

Ejemplos

```
>>> sol2A 2008  
True
```

```
>>> sol2A 2022  
False
```

3.2.2 B)

```
sol2B :: Integer -> Bool
```

Descripción

Coje el año dado y devuelve True si es bisiesto, False en caso contrario

Ejemplos

```
>>> sol2A 2008
True
```

```
>>> sol2A 2022
False
```

3.3 EJERCICIO 3

```
sol3 :: (Fractional a, Integral a, Foldable t) => t a -> a
```

Descripción

Error de tipado ya que la division `/` no trabaja con `Integer`

Descripción

Calcula la media de un listado de numeros dado

```
>>> sol3 [1,2,3,4]
2.5
```

3.4 EJERCICIO 4

```
calculoDigitos :: Integer -> Integer
```

Descripción

Cálculo del número de dígitos de un numero dado

Ejemplos

```
>>> calculoDigitos 0  
1
```

```
>>> calculoDigitos 1984  
4
```

```
sumaDigitos :: Integral t => t -> t
```

Descripción

Suma todos los digitos de un numero dado

Ejemplos

```
>>> sumaDigitos 0  
0
```

```
>>> sumaDigitos 1984  
22
```

```
reduccion :: Integral t => t -> t
```

Descripción

Suma todos los digitos de un numero dado hasta que el total es menor que 10

Ejemplos

```
>>> reduccion 0  
0
```

```
>>> reduccion 1984  
4
```

3.5 EJERCICIO 5

```
bools :: Bool -> Bool -> Bool
```

Disyunción booleana definida por ajuste de patrones

Chapter 4

Labo2

```
module Labo2 (  
    cuadrados, cuadradoInverso, rsumcos, sumMenores, siguientePrimo, factores,  
    isPrime, iguales, menor, mayorA, ex, filter2, filters, mapx  
    ) where
```

4.1 EJERCICIO 1

4.1.1 A)

```
cuadrados :: (Num b, Enum b) => b -> [b]
```

Descripción

Coje un números y devuelve una lista con los cuadrados desde 0 hasta el número

Ejemplos

```
>>> cuadrados 3  
[0,1,4,9]
```

4.1.2 B)

```
cuadradoInverso :: (Num a, Enum a) => a -> [(a, a)]
```

Descripción

Coje un números y devuelve una lista con los cuadrados desde 0 hasta el número en orden inverso y emparejado con su número inicial

Ejemplos

```
>>> cuadradoInverso 3
[(3,9),(2,4),(1,1),(0,0)]
```

4.1.3 C)

```
rsumcos :: (Floating a, Enum a) => a -> a
```

Descripción

Sumatorio del coseno de 1 hasta el número dado

Ejemplos

```
>>> rsumcos 90
2592.852064773843
```

4.1.4 D)

```
sumMenores :: Integral a => a -> a
```

Descripción

Sumatorio de los números menores que el dado que sean multiplos de 5 o 3

Ejemplos

```
>>> sumMenores 10
33
```

4.1.5 E)

```
siguientePrimo :: Integral a => a -> a
```

Descripción

calcula el siguiente número primo a uno dado.

Usa `isPrime`

Ejemplos

```
>>> siguientePrimo 10
11
```

```
factores :: Integral a => a -> [a]
```

Descripción

Calcula todos los divisores de un número dado

Ejemplos

```
>>> factores 10
[1,2,5,10]
```

```
isPrime :: Integral a => a -> Bool
```

Descripción

Comprueba si un número es primo

Usa `factores`

Ejemplos

```
>>> isPrime 10
False
```

```
>>> isPrime 5
True
```

4.2 EJERCICIO 2

4.2.1 A)

```
iguales :: Eq b => (a -> b) -> (a -> b) -> a -> a -> Bool
```

Descripción

Comprueba si dos funciones son iguales para un rango dado de valores

Ejemplos

```
>>> iguales siguientePrimo sumMenores 0 10
False
```

```
>>> iguales sumMenores sumMenores 0 10
True
```

4.2.2 B)

```
menor :: (Enum a, Num a) => a -> (a -> Bool) -> a
```

Descripción

Devuelve el menor numero mayor que n que compla una función dada

Ejemplos

```
>>> menor 10 isPrime
11
```

4.2.3 C)

```
mayorA :: Enum a => a -> a -> (a -> Bool) -> a
```

Descripción

Mayor número en un intervalo dado que cumple una función dada

Ejemplos

```
>>> mayorA 10 20 isPrime
19
```

4.2.4 D)

```
ex :: Enum a => a -> a -> (a -> Bool) -> Bool
```

Descripción

Comprueba si existe algún número en el intervalo dado que verifique una función

Ejemplos

```
>>> ex 14 16 isPrime
False
```

```
>>> ex 10 15 isPrime
True
```

4.3 EJERCICIO 3

4.3.1 A)

```
filter2 :: [a] -> (a -> Bool) -> (a -> Bool) -> ([a], [a])
```

Descripción

Devuelve dos listas con los elementos de una lista dada que cumplen dos funciones

Ejemplos

```
>>> filter2 [1..10] odd even
([1,3,5,7,9],[2,4,6,8,10])
```

4.3.2 B)

```
filters :: [a] -> [a -> Bool] -> [[a]]
```

Descripción

Devuelve una lista con las listas de numeros que cumplen unas funciones dadas en formato de lista

Ejemplos

```
>>> filters [1..10] [odd, even, isPrime]
[[1,3,5,7,9],[2,4,6,8,10],[2,3,5,7]]
```

4.3.3 C)

```
mapx :: t -> [t -> b] -> [b]
```

Descripción

Devuelve una lista de valores resultado de aplicar a un valor un listado de funciones. Todos los tipos de las funciones de la lista deben ser iguales

Ejemplos

```
>>> mapx 10 [even, odd]
[True,False]
```

```
>>> mapx 10 [siguientePrimo, sumMenores]
[11,33]
```


Chapter 5

Labo2₂₀₂₂

```
module Labo2_2022 (  
  listaPares, listaParesCuadrados, listaPotencias, sumaMenores,  
  primerPrimoMayor, isPrime, iguales, menorA, menor, mayorA, pt, filter2,  
  partition, mapx, filter1, filters  
) where
```

5.1 EJERCICIO 1

5.1.1 A)

```
listaPares :: Integral a => a -> [a]
```

Descripción

Genera una lista de números pares hasta el número dado.

Ejemplo:

```
>>> listaPares 9  
[0,2,4,6,8]
```

5.1.2 B)

```
listaParesCuadrados :: Integral a => a -> [(a, a)]
```

Descripción

Coje un números y devuelve una lista con los cuadrados desde 0 hasta el número en orden inverso y emparejado con su número inicial

Ejemplos

```
>>> listaParesCuadrados 3  
[(3,9),(2,4),(1,1),(0,0)]
```

5.1.3 C)

```
listaPotencias :: Integral a => Int -> [a]
```

Descripción

Genera una lista con las potencias de 3 desde 0 hasta el numero dado

Ejemplo:

```
>>> listaPotencias 3  
[1,3,9]
```

5.1.4 D)

```
sumaMenores :: Int -> Int
```

Descripción

Sumatorio de los números menores que el dado que sean multiplos de 5 o 3

Ejemplos

```
>>> sumaMenores 10
33
```

5.1.5 E)

```
primerPrimoMayor :: Integral a => a -> a
```

Descripción

calcula el siguiente
número primo a uno
dado.

Usa `isPrime`

Ejemplos

```
>>> primerPrimoMayor 10
11
```

```
isPrime :: Integral a => a -> Bool
```

Descripción

Comprueba si un número es primo

Ejemplos

```
>>> isPrime 10  
False
```

```
>>> isPrime 5  
True
```

5.2 EJERCICIO 2

5.2.1 A)

```
iguales :: Eq b => (a -> b) -> (a -> b) -> a -> a -> Bool
```


Descripción

Comprueba si dos funciones son iguales para un rango dado de valores

Ejemplos

```
>>> iguales primerPrimoMayor sumaMenores 0 10  
False
```

```
>>> iguales sumaMenores sumaMenores 0 10  
True
```

5.2.2 B)

```
menorA :: Integral a => a -> a -> (a -> Bool) -> a
```

Descripción

Devuelve el menor numero mayor que n que compla una función dada

Ejemplos

```
>>> menorA 22 128 isPrime  
23
```

```
>>> menorA 14 16 isPrime  
Prelude.head: empty list
```

5.2.3 C)

```
menor :: (Num p, Enum p) => p -> (p -> Bool) -> p
```

Descripción

**Devuelve el menor
numero mayor que n que
compla una función dada**

Ejemplos

```
>>> menor 2077 isPrime  
2081
```

5.2.4 D)

```
mayorA :: Enum p => p -> p -> (p -> Bool) -> p
```

Descripción

Mayor número en un intervalo dado que cumple una función dada

Ejemplos

```
>>> mayorA 10 20 isPrime  
19
```

```
>>> mayorA 14 16 isPrime  
Prelude.last: empty list
```

5.2.5 E)

```
pt :: Enum p => p -> p -> (p -> Bool) -> Bool
```

Descripción

Comprueba si todos los elementos de un intervalo verifican una función

Ejemplos

```
>>> pt 14 16 isPrime  
False
```

```
>>> pt 2 3 isPrime  
True
```

5.3 EJERCICIO 3

5.3.1 A)

```
filter2 :: [a] -> (a -> b) -> (a -> b) -> [[b]]
```

Descripción

Devuelve dos listas con los elementos de una lista dada que cumplen dos funciones

Ejemplos

```
>>> filter2 [1..10] odd even  
([1,3,5,7,9],[2,4,6,8,10])
```

5.3.2 B)

```
partition :: (a -> Bool) -> [a] -> ([a], [a])
```

Descripción

Parte una lista dada entre elementos que verifican una función y otros que no

Ejemplos

```
>>> partition isPrime [0..20]  
([2,3,5,7,11,13,17,19],[0,1,4,6,8,9,10,12,14,15,16,18,20])
```

5.3.3 C)

```
mapx :: t -> [t -> b] -> [b]
```

Descripción

Devuelve una lista de valores resultado de aplicar a un valor un listado de funciones. Todos los tipos de las funciones de la lista deben ser iguales

Ejemplos

```
>>> mapx 10 [even, odd]
[True,False]
```

```
>>> mapx 10 [primerPrimoMayor, sumaMenores]
[11,33]
```

5.3.4 D)

```
filter1 :: [[a]] -> (a -> Bool) -> [[a]]
```

Description

Devuelve una lista con listas de valores que cumplen la propiedad dada

Ejemplo

```
>>> filter1 [[1], [1,2,3,4], [2077,2467, 2277]] isPrime  
[[], [2,3], [2467]]
```

5.3.5 E)

```
filters :: [a] -> [a -> Bool] -> [[a]]
```


Descripción

Devuelve una lista con las listas de numeros que cumplen unas funciones dadas en formato de lista

Ejemplos

```
>>> filters [1..10] [odd, even, isPrime]  
[[1,3,5,7,9],[2,4,6,8,10],[2,3,5,7]]
```


Chapter 6

Labo3

```
module Labo3 (  
    last', reverse', all', min', map', filter', takeWhile', listaNegPos,  
    listaParejas', listaParejas, sufijos, sublistas, permuta, intercalado,  
    sumandos, sumandos'  
) where
```

6.1 EJERCICIO 1

6.1.1 A)

```
last' :: [a] -> a
```

Descripción

Coge el último elemento de una lista dada

Ejemplos

```
>>> last' []  
Lista vacia  
  
>>> last [1,2,3,4]  
4
```

6.1.2 B)

```
reverse' :: [a] -> [a]
```

Descripción

Hace la lista inversa de una dada

Ejemplos

```
>>> reverse' [1,2,3,4]
[4,3,2,1]
```

6.1.3 C)

```
all' :: (a -> Bool) -> [a] -> Bool
```

Descripción

Devuelve true si todos los elementos de la lista cumplen una condición

Ejemplos

```
>>> all' (<10) [1,2,3,4]
True
```

```
>>> all' (>10) [1,20,30,40]
False
```

6.1.4 D)

```
min' :: Ord a => [a] -> a
```

Descripción

Devuelve el menor elemento de una lista dada

Ejemplos

```
>>> min' []  
Lista vacia
```

```
>>> min' [2,5,8,2,0]  
0
```

6.1.5 E)

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

Descripción

Genera una lista de elementos resultantes de aplicar un filtro a otra lista de elementos

Ejemplo

```
>>> map' (*3) [1,2,3,4]  
[3,6,9,12]
```

6.1.6 F)

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

Descripción

Genera una lista de elementos con los elementos de otra lista que hayan pasado un filtro

Ejemplo

```
>>> filter' (>10) [1,20,30,40]  
[20,30,40]
```

6.1.7 G)

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


Descripción

Coge elementos de la lista dada hasta que la condición falla

Ejemplos

```
>>> takeWhile' (<3) [1,2,3,4,5]  
[1,2]
```

```
>>> takeWhile' (>3) [1,2,3,4,5]  
[]
```

6.2 EJERCICIO 2

```
listaNegPos :: [Integer]
```

Descripción

Genera una lista
alternada de elementos
del 1 al 100 de forma [1,
-1]

6.3 EJERCICIO 3

```
listaParejas' :: (Num b, Enum b, Eq b) => b -> [(b, b)]
```

Descripción

Versión acotada de `listaParejas`

Ejemplo

```
>>> listaParejas' 4
[(0,0),(0,1),(1,0),(0,2),(1,1),(2,0),(0,3),(1,2),(2,1),(3,0),(0,4),(1,3),(2,2),(3,1),(
```

```
listaParejas :: Integral a => [(a, a)]
```

Descripción

Genera parejas ordenadas
que cumplen $x + y = z$,
en orden de z

Ver `listaParejas'` para ejemplos

6.4 EJERCICIO 4

6.4.1 A)

```
sufijos :: [a] -> [[a]]
```

Descripción

Genera todos los sufijos
de una lista

Ejemplo

```
>>> sufijos [1,2,3,4]
[[1,2,3,4],[2,3,4],[3,4],[4],[]]
```

6.4.2 B)

```
sublistas :: [a] -> [[a]]
```

Descripción

Genera todas las sublistas de una lista dada

Ejemplo

```
>>> sublistas [1,2,3]
[[1],[2],[3],[1,2],[2,3],[1,2,3]]
```

6.4.3 C)

```
permuta :: [a] -> [[a]]
```

Descripción

Genera todas las permutaciones posibles de una lista dada

Ejemplo

```
>>> permuta [1,2,3]  
[[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]
```

```
intercalado :: a -> [a] -> [[a]]
```

Descripción

Devuelve una lista de listas con el elemento dado en cada posición posible de la lista dada

Ejemplo

```
>>> intercalado 3 [1,2]
[[3,1,2],[1,3,2],[1,2,3]]
```

6.4.4 D)

```
sumandos :: Integral a => a -> [[a]]
```

Descripción

Genera un listado de de listas que al sumar todos sus elementos dan como resultado el numero dado

Ejemplos

```
>>> sumandos 3
[[1,1,1],[1,2],[3]]
```

```
sumandos' :: Integral a => a -> a -> a -> [[a]]
```

Descripción

Genera listados de tamaños desde x hasta n que al ser sumados dan como resultado x , siendo i el numero de listados generados

Ejemplo

```
>>> sumandos' 4 1 1  
[[1,1,1],[1,2],[3]]
```

```
>>> sumandos' 4 2 1  
[[1,1],[2]]
```

```
>>> sumandos' 4 2 2  
[[2]]
```


Chapter 7

Labo3₂022

```
module Labo3_2022 (  
    last', reverse', any', min', map', takeWhile', listaNegPos, listaRepetidos,  
    listaParejas, permuta, intercalado, cuestaPos, actualizarCabecera,  
    meterCuesta, esCuesta  
) where
```

7.1 EJERCICIO 1

7.1.1 A)

```
last' :: [a] -> a
```

Descripción

Coge el último elemento de una lista dada

Ejemplos

```
>>> last' []  
Lista vacia  
  
>>> last [1,2,3,4]  
4
```

7.1.2 B)

```
reverse' :: [a] -> [a]
```

Descripción

Hace la lista inversa de una dada

Ejemplos

```
>>> reverse' [1,2,3,4]
[4,3,2,1]
```

7.1.3 C)

```
any' :: (a -> Bool) -> [a] -> Bool
```

Descripción

Devuelve true si todos los elementos de la lista cumplen una condición

Ejemplos

```
>>> any' (<10) [1,2,3,4]
True
```

```
>>> any' (<10) [10, 20, 30, 5]
True
```

```
>>> any' (>10) [1,2, 3, 4]
False
```

7.1.4 D)

```
min' :: Ord a => [a] -> a
```

Descripción

Devuelve el menor elemento de una lista dada

Ejemplos

```
>>> min' []  
Lista vacia
```

```
>>> min' [2,5,8,2,0]  
0
```

7.1.5 E)

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

Descripción

Genera una lista de elementos resultantes de aplicar un filtro a otra lista de elementos

Ejemplo

```
>>> map' (*3) [1,2,3,4]  
[3,6,9,12]
```

7.1.6 F)

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

Descripción

Coge elementos de la lista dada hasta que la condición falla

Ejemplos

```
>>> takeWhile' (<3) [1,2,3,4,5]  
[1,2]
```

```
>>> takeWhile' (>3) [1,2,3,4,5]  
[]
```

7.2 EJERCICIO 2

```
listaNegPos :: [Integer]
```

Descripción

Genera una lista
alternada de elementos
del 1 al 100 de forma [1,
-2,..]

```
>>> listaNegPos  
[1,-2,3,-4,5,-6,7,-8,9,-10,11,-12,13,-14,15,-16,17,-18,19,-20,21,-22,23,-24,25,-26,27,
```

7.3 EJERCICIO 3

```
listaRepetidos :: [[Int]]
```

Descripción

Genera una lista de listas cuyo contenido sera el numero x repetido x veces

```
>>> take 5 (listaRepetidos)
[[], [1], [2,2], [3,3,3], [4,4,4,4]]
```

7.4 EJERCICIO 4

```
listaParejas :: Integral a => [(a, a)]
```

Descripción

Genera parejas ordenadas que cumplen $x + y = z$, en orden de z

```
>>> take 10 (listaParejas)
[(0,0),(0,1),(1,0),(0,2),(1,1),(2,0),(0,3),(1,2),(2,1),(3,0)]
```

7.5 EJERCICIO 5

```
permuta :: [a] -> [[a]]
```

Descripción

Genera todas las permutaciones posibles de una lista dada

Ejemplo

```
>>> permuta [1,2,3]
[[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]
```

```
intercalado :: a -> [a] -> [[a]]
```

Descripción

Devuelve una lista de listas con el elemento dado en cada posición posible de la lista dada

Ejemplo

```
>>> intercalado 3 [1,2]
[[3,1,2],[1,3,2],[1,2,3]]
```

7.6 EJERCICIO 6

```
cuestaPos :: Ord a => [a] -> [(a, Int)]
```

Descripción

Devuelve una lista con los elementos considerados cuentas y su posicion

Ejemplo

```
>>> cuentaPos [-3,-2,1,0,-1,1]
[(-2,1),(1,2),(1,5)]
```

```
actualizarCabecera :: [(a, Int)] -> a -> [(a, Int)]
```

Descripción

Devuelve una lista con la cabecera actualizada

Ejemplo

```
>>> actualizarCabecera [(3,0)] 2
[(2,1)]
```

```
meterCuesta :: [(a, Int)] -> a -> [(a, Int)]
```

Descripción

Añade una cuesta al listado y actualiza la cabecera

Ejemplo

```
>>> meterCuesta [(3,0)] 2  
[(2,1),(2,1)]
```

```
esCuesta :: Ord a => a -> a -> Bool
```

Descripción

comprueba si un elemento es cuesta

Ejemplo

```
>>> esCuesta 1 0  
False
```

```
>>> esCuesta 2 5  
True
```

Chapter 8

Labo4

```
module Labo4 (  
  Punto(P), pointSum, Direccion(DERECHA, IZQUIERDA, ABAJO, ARRIBA), mueve,  
  destino, trayectoria, Nat(Suc, Cero), (~+), (~*), natToInt, Complejo(C),  
  Medible(medida)  
) where
```

8.1 EJERCICIO 1

```
data Punto
```

Representación de un
punto de forma (x,y)

Constructors

```
= P      Constructor del
        punto
        Int Coordenada X
        Int Coordenada Y
```

```
instance Show Punto
```

Muestra el punto como
 ”(a,b)””

```
pointSum :: (Ord a, Num a) => a -> a -> a
```

Descripción

Suma un numero a una
 parte de una coordenada
 dada. Tiene como rango
 [0 .. 100]

Ejemplos

```
>>> pointSum 100 10
9
```

```
>>> pointSum 0 (-1)
100
```

```
>>> pointSum 100 1
0
```

```
data Direccion
```

Representación de una dirección

Constructors

```
=  ARRIBA
```

```
|  ABAJO
```

```
|  IZQUIERDA
```

```
|  DERECHA
```

```
instance Show Direccion
```

Derivación estandar de Show

```
instance Eq Direccion
```

Derivación estandar de Eq

```
instance Ord Direccion
```

Derivación estandar de Ord

8.1.1 A)

```
mueve :: Punto -> Direccion -> Punto
```

Descripción

Mueve un punto en la dirección dada

Ejemplos

```
>>> mueve (P 0 1) IZQUIERDA  
(100,1)
```



```
>>> mueve (P 0 1) DERECHA  
(1,1)
```

```
>>> mueve (P 0 1) ARRIBA  
(0,2)
```

```
>>> mueve (P 0 1) ABAJO  
(0,0)
```

8.1.2 B)

```
destino :: Punto -> [Direccion] -> Punto
```

Descripción

Mueve un punto en las direcciones dadas

Ejemplos

```
>>> destino (P 0 0) [ARRIBA, ARRIBA, ABAJO, ABAJO, IZQUIERDA, DERECHA, IZQUIERDA, DERECHA]  
(0,0)
```

```
>>> destino (P 0 0) [ARRIBA, ARRIBA, ABAJO, IZQUIERDA, IZQUIERDA, DERECHA]  
(100,1)
```

8.1.3 C)

```
trayectoria :: Punto -> [Direccion] -> [Punto]
```

Descripción

Mueve un punto en las direcciones dadas mostrando los puntos por los que se han pasado

Ejemplos

```
>>> trayectoria (P 0 0) [ARRIBA, ARRIBA, ABAJO, ABAJO, IZQUIERDA, DERECHA, IZQUIERDA,
[(0,0),(0,1),(0,2),(0,1),(0,0),(100,0),(0,0),(100,0),(0,0)]
```

```
>>> trayectoria (P 0 0) [ARRIBA, ARRIBA, ABAJO, IZQUIERDA, IZQUIERDA, DERECHA]
[(0,0),(0,1),(0,2),(0,1),(100,1),(99,1),(100,1)]
```

8.2 EJERCICIO 2

```
data Nat
```

Representación de numeros naturales según la aritmética de Peano

Constructors

= Cero Representación del 0
| Suc Nat Representación de
 cualquier sucesor de
 0

```
instance Show Nat
```

Muestra el punto como
un número normal

```
instance Eq Nat
```

Derivación estandar de
Eq

```
instance Ord Nat
```

Derivación estandar de Ord

8.2.1 A)

```
(~+) :: Nat -> Nat -> Nat
```

Descripción

Suma de dos números siguiendo la aritmética de Peano

Ejemplo

```
>>> (Suc (Cero)) ~+ (Suc (Suc (Cero)))  
3
```

```
(~*) :: Nat -> Nat -> Nat
```

Descripción

Multiplicación de dos números siguiendo la aritmética de Peano

Ejemplo

```
>>> (Suc (Cero)) ~* (Suc (Suc (Cero)))  
2
```

8.2.2 B)

```
natToInt :: Nat -> Int
```

Descripción

Convierte un número representado en aritmética de Peano en un `Int` normal

Ejemplo

```
>>> natToInt (Suc (Suc (Cero)))
2
```

8.2.3 C)

8.3 EJERCICIO 3

```
data Complejo
```

Representación de un numero complejo

Constructors

```
= C Float Float
```

```
instance Num Complejo
```

Implementación especial para (+) (-) y (*)

```
instance Fractional Complejo
```

Implementación especial para `(/)`

```
instance Show Complejo
```

Representa el número
como "a (+-) bi"

```
instance Eq Complejo
```

Derivación estandar de Eq

8.4 EJERCICIO 4

```
class Medible a where
```

Clase para medir otros tipos de datos

Methods

```
medida
```

```
:: a
```

```
-> Int
```

Coge el dato dado y
saca una medida de
él

```
instance Medible Bool
```

Mediremos True como 0 y
false como 1

```
instance Medible a => Medible [a]
```

Suma todas las medidas
de la lista

```
instance (Medible a, Medible b) => Medible (a, b)
```

Suma la medida de los
dos elementos dados

Chapter 9

Labo5

```
module Labo5 (  
    getInt, adivina, Matriz, getFloat, getMatriz, getFilas, getDatosMatriz,  
    dibujaMatriz, mostrarMatrizNueva, formatea, formatear, justify, addSpaces  
    ) where
```

9.1 EJERCICIO 1

```
getInt :: IO Int
```

Descripción

Coge un número desde la entrada.

```
adivina :: Int -> IO ()
```

Descripción

Pide un número por teclado a adivinar. Esto es bastante estúpido de por si solo porque tienes que introducir el número como argumento

9.2 EJERCICIO 2

```
type Matriz = [[Float]]
```

Representación de una matriz como lista de listas

```
getFloat :: IO Float
```

Descripción

Coge un número desde la entrada.

9.2.1 A)

```
getMatriz :: IO Matriz
```

Descripción

Coge una matriz desde la entrada.

Ver `getDatosMatriz` para ver el funcionamiento total

```
getFilas :: Int -> IO [Float]
```

Descripción

Coge una fila de una matriz según el número de columnas dadas.

```
getDatosMatriz :: Int -> Int -> IO Matriz
```

Descripción

Coge todas las filas de una matriz según el número de filas y columnas dadas.

Ver `getFilas` para el funcionamiento completo

9.2.2 B)

```
dibujaMatriz :: Matriz -> IO ()
```

Descripción

Dibuja una matriz dada como argumento.

```
mostrarMatrizNueva :: IO ()
```

Descripción

Pide una matriz por teclado y la muestra una vez construida.

Ver `getMatriz` **y** `dibujaMatriz`.

9.3 EJERCICIO 3

```
formatea :: String -> String -> Int -> IO ()
```

Descripción

Formatea a n columnas cada linea de un fichero dado en otro.

Ver `formatear` para el funcionamiento.

```
formatear :: Int -> String -> String
```

Descripción

Formatea cada linea a un total de n caracteres por fila.

**Ver `justify` para el
funcionamiento completo.**

```
justify :: Int -> String -> String
```

Descripción

**Añade espacios a una
línea si es necesario.**

**Ver `addSpaces` para el
funcionamiento completo.**

```
addSpaces :: Int -> String -> String
```

Descripción

**Añade n espacios a una
línea dada.**

Chapter 10

Lote1

```
module Lote1 (  
    petaFlop, forn1, yearsToSec, frontierInteger, petaFlop', forn1',  
    yearsToSec', frontierInt, petaFlop'', forn1'', yearsToSec'', frontierDouble,  
    time, totalYears, daysToSec, hoursToSec, minsToSec, years, days, hours,  
    mins, segs, solc, sold, f, f', g, h, h', i, i', digitos, sumaDigitos,  
    reduccion, perm, var, comb, factores, isPrime, fibonacci, y, o, no, implica,  
    eq, oEx, (&&&), (|||), (-->), (==), (||=), f8, media, last', init',  
    initLast, concat', auxList, take', drop', splitAt', nub, quicksort, and',  
    or', sum', product', lmedia, reverse', reverse''  
) where
```

10.1 EJERCICIO 1

10.1.1 A)

```
petaFlop :: Integer
```

```
forn1 :: Integer
```

```
yearsToSec :: Integer
```

```
frontierInteger :: Integer
```

```
petaFlop' :: Int
```

Flops in a petaFlop

```
fornl' :: Int
```

Computing power of the best computer of Frontier

```
yearsToSec' :: Int
```

Seconds in a year

```
frontierInt :: Int
```

Power of operation of the
Frontier since the
beginning of time — *!!!*
frontierInt
-7928755680649936896

```
petaFlop'' :: Double
```

Flops in a petaFlop

```
fornl'' :: Double
```

Computing power of the best computer of Frontier

```
yearsToSec'' :: Double
```

Seconds in a year

```
frontierDouble :: Double
```

Power of operation of the
Frontier since the
beginning of time — *!!!*
frontierDouble
5.158595808e35

10.1.2 B)

```
time :: Integer
```

Time for the exercise

```
totalYears :: Integer
```

```
>>> totalYears
317
```

10.1.3 C)

```
daysToSec :: Integer
```

Seconds in a day

```
hoursToSec :: Integer
```

Seconds in an hour

```
minsToSec :: Integer
```

Segundos totales en un minuto

```
years :: Integer
```

Caclulo de los años equivalentes a `time`

```
days :: Integer
```

Calculo de las horas
equivalentes a `time` quitando

`years`

`hours :: Integer`

Calculo de las horas
equivalentes a `time` quitando

`days` *y* `years`

`mins :: Integer`

Calculo de las horas
equivalentes a `time` quitando

`days`, `years` *y* `hours`

`segs :: Integer`

Calculo de las horas
equivalentes a `time` quitando

`days`, `years`, `hours` *y* `mins`

```
solc :: (Integer, Integer, Integer, Integer, Integer)
```

**Tupla con el tiempo total
de time en años, días, horas,
minutos y segundos**

10.1.4 D)

```
sold :: Integer -> (Integer, Integer, Integer, Integer, Integer)
```

Descripción

**Coge el tiempo dado y lo
convierte a (Años, Días,
Horas, Minutos,
Segundos)**

Ejemplos

```
>>> sold 1000000  
(0,11,13,46,40)
```

10.2 EJERCICIO 2

```
f :: Int -> Int -> Int
```

Descripción

**Función f usando
notación infija**

Ejemplos

```
>>> f 7 4  
-14
```

```
f' :: Int -> Int -> Int
```

Descripción

**Función f usando
notación prefija**

Ejemplo

```
>>> f' 7 4
-14
```

```
g :: Int -> Int
```

Descripción

Función g

Ejemplo

```
>>> g (-2)
32
```

```
h :: Int -> Int -> Int -> Int
```

Descripción

Función h usando notación infija

Ejemplo

```
>>> h 1 2 3
0
```

```
h' :: Int -> Int -> Int -> Int
```

Descripción

Función h usando notación posfija

Ejemplo

```
>>> h' 1 2 3
0
```

```
i :: Int -> Int -> Int
```

Descripción

Función i usando notación guardas

Ejemplo

```
>>> i 1 2  
0
```

```
i' :: Int -> Int -> Int
```

Descripción

Función i usando notación if ` then ` else

Ejemplo

```
>>> i 1 2  
0
```

10.3 EJERCICIO 3

10.3.1 A)

```
digitos :: Int -> Int
```

Description

**Gets the number of digits
of a given value**

Example

```
>>> digitos 1
1

>>> digitos 123912891289
12
```

10.3.2 B)

```
sumaDigitos :: Int -> Int
```

Description

**Adds every single digit of
a given number**

Example

```
>>> sumaDigitos 1  
1
```

```
>>> sumaDigitos 18982918192283429138194213189  
82
```

```
reduccion :: Int -> Int
```

Description

Adds digits (`sumaDigitos`) from a number until the result is one single digit. Single digits below 0 are converted to positive values

Example

```
>>> reduccion (-1)  
1
```

```
>>> reduccion 12893213913819238192312819  
2
```

10.3.3 C)

```
perm :: Integer -> Integer
```

Description

Permutations of a given number

Example

```
>>> perm 8  
40320
```

10.3.4 D)

```
var :: Integer -> Integer -> Integer
```

Description

Variations of two given numbers

Example

```
>>> var 2 1  
2
```

```
>>> var 123 5  
25928567280
```

10.3.5 E)

```
comb :: Integer -> Integer -> Integer
```

Description

Combinations of two given numbers

Examples

```
>>> comb 2 1
2
```

```
>>> comb 12425 12423
77184100
```

10.4 EJERCICIO 4

```
factores :: Integral a => a -> [a]
```

Descripción

**Calcula todos los
divisores de un número
dado**

Ejemplos

```
>>> factores 10
[1,2,5,10]
```

```
isPrime :: Integral a => a -> Bool
```


Descripción

Comprueba si un número es primo

Usa `factores`

Ejemplos

```
>>> isPrime 10
False
```

```
>>> isPrime 5
True
```

10.5 EJERCICIO 5

```
fibonacci :: Int -> Integer
```

Descripción

Funcion de fibonacci de aridad 1

Ejemplos

```
>>> fibonacci 2
2

>>> fibonacci 100
573147844013817084101
```

10.6 EJERCICIO 6

```
y :: Bool -> Bool -> Bool
```

Conjunction operator

```
o :: Bool -> Bool -> Bool
```

Disjunction operator

```
no :: Bool -> Bool
```

Not operator

```
implica :: Bool -> Bool -> Bool
```

Implication operator

```
eq :: Bool -> Bool -> Bool
```

Equivalence operator

```
oEx :: Bool -> Bool -> Bool
```

Exclusive disjunction operator

```
(&&&) :: Bool -> Bool -> Bool
```

Conjunction operator

```
(|||) :: Bool -> Bool -> Bool
```

Disjunction operator

```
(-->) :: Bool -> Bool -> Bool
```

Implication operator

```
(==) :: Bool -> Bool -> Bool
```

Equivalence operator

```
(||=) :: Bool -> Bool -> Bool
```

Exclusive disjunction operator

10.7 EJERCICIO 8

```
f8 :: Int -> Int -> Int -> Int
```

Description

La función toma tres argumentos y cumple con las siguientes condiciones:

- (i) Ser estricta en el primer argumento.
- (ii) No ser estricta ni en el segundo ni en el tercer argumento.

(iii) Ser conjuntamente estricta en el segundo y tercer argumento.

Examples

```
>>> f8 1 undefined undefined
1
```

```
>>> f8 0 1 2
3
```

```
>>> f8 0 1 undefined
Prelude.undefined
```

10.8 EJERCICIO 9

```
media :: (Fractional a, Integral a, Foldable t) => t a -> a
```

Description

Arithmetic mean of a list of numbers

Example

```
>>> media [1,2,3,4]
2.5
```

10.9 EJERCICIO 10

```
last' :: [a] -> a
```

Description

Returns the last element
of a given list

Example

```
>>> last' [1,2,3,4]
4
```

```
init' :: [a] -> [a]
```

Description

Returns all element
except the las one of a list

Examples

```
>>> init [1,2,3,4]  
[1,2,3]
```

```
initLast :: [a] -> ([a], a)
```

Description

Returns a tuple with the
`init`' of a list and the `last`' of a
list

Example

```
>>> initLast [1,2,3,4]  
([1,2,3],4)
```

```
concat' :: [[a]] -> [a]
```

Description

Concatenates all given lists into one

Example

```
>>> concat' [[1],[2,3],[4],[5]]  
[1,2,3,4,5]
```

```
auxList :: Int -> Int -> Int -> [a] -> [a]
```

Description

Returns the elements of a list between two given indexes

Examples

```
>>> auxList 0 0 2 [1,2,3,4,5]  
[1,2,3]
```

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

Description

**Takes the n first elements
of a list**

Example

```
>>> take' 2 [1,2,3,4]  
[1,2]
```

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

Description

**Erases the n first
elements of a list**

Example

```
>>> drop' 2 [1,2,3,4]
[3,4]
```

```
splitAt' :: Int -> [a] -> ([a], [a])
```

Description

Splits a list in the given index

Example

```
>>> splitAt' 2 [1,2,3,4]
([1,2], [3,4])
```

```
nub :: Eq a => [a] -> [a]
```

Description

Returns a list without the repeated values of the given list

Example

```
>>> nub [1,2,3,4,1,2,3,4,3,2,6,4,2,0]
[1,2,3,4,6,0]
```

```
quicksort :: Ord a => [a] -> [a]
```

Description

Quicksort implementation

Examples

```
>>> quicksort [4,7,3,6,1,89,52,76,13,0,23]
[0,1,3,4,6,7,13,23,52,76,89]
```

```
and' :: [Bool] -> Bool
```

Description

Conjunction of all elements of a list

Example

```
>>> and' [True,False, True ]  
False
```

```
or' :: [Bool] -> Bool
```

Description

Disjunction of all elements of a list

Example

```
>>> or' [True,False, True ]  
True
```

```
sum' :: [Int] -> Int
```

Description

Sum of al the numbers in the list

Example

```
>>> sum' [1..100]  
5050
```

```
product' :: [Integer] -> Integer
```

Description

Product of al the numbers in the list

Example

```
>>> product' [2..5]  
120
```

```
lmedia :: [[[a]]] -> Float
```

Description

Mean of sizes of all elements in a list of lists

10.10 EJERCICIO 11

10.10.1 A)

```
reverse' :: [a] -> [a]
```

Description

Reverse a given list in $O(n^2)$ time

Example

```
>>> reverse' [1,2,3,4]  
[4,3,2,1]
```

10.10.2 B)

```
reverse'' :: [a] -> [a]
```

Description

Reverse a given list in $O(n)$ time

Example

```
>>> reverse'' [1,2,3,4]  
[4,3,2,1]
```


Chapter 11

Traductor

```
module Traductor (  
  NodeElement, Traduction, Bucket, Hash, leaf, genEmptyTree, insert,  
  insertList, inorder, busqueda, busquedaLista, evens, odds, hash,  
  calcMeanWordCount, calcMeanWordCountExample, rellenadoPorFichero,  
  rellenadoPorFicheroFijo, lecturaPalabras, mostrarTraducciones, demoMode  
) where
```

Práctica final de Programación Declarativa (PD) de Ingeniería Infomática de la UCM

11.1 Types

```
data NodeElement
```

Nodo de la tabla de traduccion usando un hash con colisiones con valores [0 .. 9] para guardar valores

```
instance Show NodeElement
```

```
type Traduction = (String, String)
```

Alias para las tuplas usadas para representar traducciones

```
type Bucket = [Traduction]
```

Elementos de la tabla de traducción asignados a un valor concreto, se guardarán varias traducciones distintas

```
data Hash a
```

Árbol para almacenar todos los nodos, puede ser vacío o un nodo con ramas a ambos lados. Estas ramas también pueden ser vacías

```
instance Show a => Show Hash a
```

11.2 Functions

11.2.1 Generators

```
leaf :: a -> Hash a
```

Description

Genera un árbol que solo tiene raíz, una hoja

Examples

```
>>> leaf (NE 1 [])  
[1] -> []
```

```
genEmptyTree :: [Int] -> Hash NodeElement
```

Description

Genera un árbol binario de n elementos en una lista para meter las palabras dadas

11.2.2 Tree and List manipulators

```
insert :: Hash NodeElement -> Int -> Traduction -> Hash NodeElement
```

Description

Inserta un elemento en el índice dado

Examples

```
>>> insert (Node (leaf (NE 0 [])) (NE 1 []) Nil) 1 ("a","b")
[0] -> []
[1] -> [("a","b")]
```

```
>>> insert Nil 1 ("a","b")
[1] -> [("a","b")]
```

```
insertList :: [Traduction] -> Traduction -> [Traduction]
```

Description

Inserción ordenada de traducciones en una lista.

```
inorder :: Show a => Hash a -> String
```

Description

Muestra el inorden del árbol dado

```
busqueda :: Hash NodeElement -> Int -> String -> String
```

Description

Busca una palabra en el listado dentro del árbol dado.

Usa el índice para encontrar el nodo del listado

Examples

```
>>> busqueda (insert genEmptyTree (hash "hola") ("hola", "hello")) (hash "hola") "hola" "hello"
```

```
>>> busqueda (genEmptyTree) (hash "hola") "hola"
*** Exception: Palabra no encontrada
CallStack (from HasCallStack):
  error, called at hash.hs:190:28 in main:Traductor
```

```
busquedaLista :: [Traduction] -> String -> String
```

Description

Busqueda de una palabra en un listado de traducciones para conseguir la traducción.

Examples

```
>>> busquedaLista [] "hola"
"Palabra no encontrada"

>>> busquedaLista [("Hola", "Hello"), ("A", "B")] "Hola"
"Hello"
```

```
evens :: [a] -> [a]
```

Description

Función auxiliar de `odds` para eliminar los índices pares de una lista

```
odds :: [a] -> [a]
```

Description

Elimina los indices pares de una lista

11.2.3 Utilities

```
hash :: String -> Int
```

Description

Hace un hash de una palabra y se queda con el último dígito para usarlo de índice

Examples

```
>>> hash "Hola"  
8
```

```
>>> hash "Adios"  
0
```

```
calcMeanWordCount :: String -> Float
```


Description

Calcula la media de letras por palabra en una frase.

Usa `foldl` para recorrer cada una de las palabras y añadir sus letras al número total

Más tarde se divide entre el número de palabras para sacar la media de letras sin contar los espacios en blanco

Examples

```
>>> calcMeanWordCount "a b c d"
1
```

```
>>> calcMeanWordCount "ABCD"
4
```

```
calcMeanWordCountExample :: String -> IO ()
```

Description

Coje el texto del fichero dado a la función y calcula la media de letras por palabra.

Examples

```
>>> calcMeanWordCountExample "./pruebas/quijote.txt"  
4.5021253
```

```
>>> calcMeanWordCountExample "./pruebas/call_of_cthulhu.txt"  
4.8737144
```

```
rellenadoPorFichero :: IO (Hash NodeElement)
```

Description

Pide un fichero para generar el árbol de traducciones

```
rellenadoPorFicheroFijo :: IO (Hash NodeElement)
```

Description

Lectura del fichero "datos.txt" para cargar un arbol de traducciones

11.2.4 IO

```
lecturaPalabras :: String -> IO String
```

Description

Lee palabras por el teclado y devuelve un String con todas las palabras introducidas

```
mostrarTraducciones :: IO ()
```

Description

Traduce las palabras dadas por el usuario por teclado.

Genera un árbol vacío, lo llena con un archivo de traducciones y lo usa para traducir una serie de palabras dadas

```
demoMode :: IO ()
```

Description

Modo para probar todas las funciones implementadas en el módulo