Haskell – Guia del laboratori 1

Entorn de treball

Escriurem i editarem el codi de Haskell amb un **editor de text** qualsevol, assegurant-nos que l'**extensió** dels fitxers sigui **.hs**

Per compilar, si tenim Windows podem descarregar i instal·lar el WinHugs, es tracta d'un compilador que inclou una interfície gràfica que facilita el treball.

Altrament, com és el cas de la imatge dels laboratoris, caldrà instal·lar el compilador Hugs i donar les ordres a través de la terminal del sistema.

http://www.haskell.org/hugs/

Als ordinadors de l'aula es recomana ubicar els fitxers de codi .hs a la carpeta Milax, doncs la terminal el buscarà allí per defecte i ens estalviarem d'escriure rutes.

Una vegada oberta la terminal, les ordres habituals són les següents:

Propòsit	Comanda
Executar el compilador	hugs
Carregar un fitxer (load)	:l nomdelfitxer.hs
Recarregar un fitxer posteriorment a una modificació (reload)	:r

Operacions

Aritmètiques

La suma (+), la resta (-) i la multiplicació (*) són les habituals. A continuació es detallen la resta d'operacions més utilitzades i les consideracions que cal tenir presents:

Operació	Funcionament	Exemple
nombre **/** nombre	Divisió. Sempre retorna double	10/2 → 5.0
nombre **`div`** nombre (escrit amb accents oberts, no pas amb apòstrofs)	Divideix i retorna la part entera de la divisió	5 `div` 2 → 2
nombre **`mod`** nombre (escrit amb accents oberts, no pas amb apòstrofs)	Retorna la resta de la divisió	5 `mod` 2 → 1
floor nombre	Retalla el nombre, retornant només la seva part entera	floor 5.26 → 5
abs nombre	Retorna el valor absolut del nombre	abs -5 → 5

Lògiques

Operació	Funcionament	Exemple
condició **&&** condició	And. Retorna cert només si totes les condicions són certes	True && False → False
condició **||** condició	Or. Retorna cert si una de les condicions és certa	True || False → True
not condició	Retorna el contrari de la condició avaluada	not True → False
condició o valor **==** condició o valor	Retorna true si coincideixen els valors que compara	True == False → False 6 == 6 → True
condició o valor **/=** condició o valor	Retorna true si no coincideixen els valors que compara	True /= False → True 6 /= 6 → False

Altres operacions

Operació	Funcionament	Exemple
succ element	Retorna el valor següent	succ 5→ 6 succ 'a' → 'b'
max elem1 elem2	Retorna el màxim dels elements	max 1 2 → 2
min elem1 elem2	Retorna el mínim dels elements	min 1 2→ 1

Notes:

- Per escriure nombres decimals cal fer servir el punt, no pas la coma
- Per escriure caràcters, ho farem entre cometes simples: ‘
- Per escriure strings, ho farem entre cometes dobles: “

Funcions

Per definir una funció a un fitxer .hs cal escriure el nom de la funció seguit dels paràmetres d'entrada que rep (únicament separats per espais) i un signe d'igual (=). Darrere del signe d'igual s'escriu l'operació que es durà a terme amb les variables d'entrada.

Per exemple:

Funció	Exemple de crida
quadratDe x = x*x	quadratDe 5 → 25
suma x y = x + y	suma 1000 2000 → 3000
esMultipleDe3 x = if (x `mod` 3 == 0) then True else False (molt important respectar les majúscules del True i el False)	esMultipleDe3 12 → True

Capçaleres de funció

Prèviament a la definició de la funció es pot incloure una capçalera que determina el tipus de les seves entrades i sortides.

La capçalera sempre comença amb el nom de la funció seguit de ::

Posteriorment als :: es detalla el tipus de les entrades en l'ordre exacte en que es rebran i separades per -> en cas que hi hagi més d'una.

Per últim es detalla el tipus de la sortida, precedit per ->

Funció	Funcionament
quadratDe :: Int -> Int quadratDe x = x*x	Funció que retorna el quadrat de l'enter que es passi com a paràmetre d'entrada
suma :: Int -> Int -> Int suma x y = x + y	Funció que retorna la suma dels dos paràmetres enters d'entrada
esMultipleDe3 :: Int -> Bool esMultipleDe3 x = if (x `mod` 3 == 0) then True else False	Funció que retorna el booleà True si el paràmetre enter d'entrada és múltiple de 3

Quan el tipus d'un paràmetre no és rellevant perquè la funció pot treballar amb diversos tipus, s'utilitzen lletres a la definició del a capçalera.

Llistes

Les llistes sempre han d'emmagatzemar **dades del mateix tipus**.

Podrien inicialitzar-se variables dins un fitxer .hs amb la següent sintaxi:

llistaCompra = ["pa", "llet", "ous"]

parells=[2,4..20]

llistaNombres=[1..20]

llistaDeLlistes = [[2,2],[4,4],[6,6],[8,8]]

llistaLletres=['a'..'z']

Operadors de llistes

Operació	Funcionament	Exemple
llista **++** llista	Retorna una llista resultant de concatenar les dues llistes indicades (han d'emmagatzemar elements del mateix tipus)	llistaNombres ++ parells
element **:** llista	Afegeix un element al principi d'una llista (no serveix per a dos llistes)	"aigua" : llistaCompra
llista **!!** índex	Retorna l'element que ocupa la posició de la llista indicada per l'índex (primera posició = índex 0)	llistaCompra !! 1 → "llet"
head llista	Retorna el primer element de la llista	head llistaCompra → "pa"
tail llista	Retorna tots els elements excepte el primer	tail llistaCompra →["llet", "ous"]
last llista	Retorna l'últim element de la llista	last llistaCompra → "ous"
init llista	Retorna tots els elements de la llista excepte l'últim	init llistaCompra →["pa", "llet"]
length llista	Retorna el nombre d'elements que conté la llista	lenght llistaCompra → 3
null llista	Retorna True si la llista està buida	Null [] → True null llistaCompra → False
reverse llista	Retorna una llista en ordre invers a la indicada	reverse llistaCompra
take numero llista	Retorna els "numero" primers elements de la llista	take 2 llistaCompra → ["pa", "llet"]
drop numero llista	Retorna tots els elements excepte els "numero" primers de la llista	drop 2 llistaCompra → ["ous"]
maximum llista	Retorna l'element amb valor màxim de la llista (si són strings retorna el que comenci per la lletra més propera a la Z)	maximum parells → 20
minimum llista	Retorna l'element amb valor mínim de la llista (si són strings retorna el que comenci per la lletra més propera a la A)	minimum parells → 2
sum llista	Retorna la suma dels elements (han de ser numèrics)	sum [2, 2, 3] → 7
product llista	Retorna el producte dels elements (han de ser numèrics)	product [2, 2, 3] → 12
elem element llista	Retorna true si la llista conté l'element esmentat	elem "pa" llistaCompra → True

Exemples de funcions que reben llistes com a paràmetres d'entrada:

Podem especificar una llista que es passa com a paràmetre d'entrada dividida com a **primer:resta** de manera que és molt més senzill treballar amb les parts de la llista dins el cos de la funció.

Per exemple:

Funció	Detall
long :: [a] -> Int long [] = 0 long (front:rest) = 1 + long rest	Funció que retorna la quantitat d'elements que té una llista
getnth :: Int -> [a] -> a getnth 1 (x:xs) = x getnth n (x:xs) = getnth (n-1) xs	Funció que retorna l'element que ocupa una posició concreta de la llista

Tuples

S'utilitzen per poder agrupar diversos valors, podent ser de diferents tipus. Semblant als objectes de Java (cada valor equivaldria a un atribut).

Podrien inicialitzar-se variables dins un fitxer .hs amb la següent sintaxi:

Definició	Aclariment
mascota = ("Xulina", 4, 1200)	(nom, numero de potes, pes en grams)
cotxe = ("Chevrolet", "Kalos", "Gris", 1200, 65)	(marca, model, color, cilindrada, cavalls)
llibre = ("Don Quijote de la Mancha","Miguel de Cervantes", 1605)	(títol, autor, any)
biblioteca = [llibre, ("Romeo y Julieta", "Shakespeare", 1597), ("Oliver Twist", "Dickens", 1837)]	Llista de llibres

Per afegir nou llibre a llista, podríem fer:

("Tom Sawyer", "Mark Twain", 1876) : biblioteca

Tipus de recursivitat

Exemple 1

La següent funció pinta tants punts com indiqui el paràmetre d'entrada:

```
printdots x = if (x==0) then "" else '.' : printdots(x-1)
```

Tanmateix, és més clar expressar-la d'una de les següent maneres:

Funcions	Detalls
printdots :: Int -> [Char] printdots 0 = [] printdots n = '.' : printdots (n-1)	Recursivitat de pila
printdotsaux :: ([Char],Int) -> [Char] printdotsaux (x,0) = x printdotsaux (x,y) = printdotsaux ('.':x, y-1) printdots' :: Int -> [Char] printdots' n = printdotsaux ([], n)	Recursivitat acumulada. printdotsaux és la classe que fa la feina i printdots' la classe que cridaria l'usuari, estalviant-li de passar la llista buida com a element neutre

Les pautes a seguir per implementar correctament cada un dels tipus de recursivitat són les següents:

Recursivitat de pila

- El cas de sortida retorna l'element neutre de l'operació
- El cas recursiu sempre té dos parts diferenciades. La primera consisteix en realitzar la feina per al cas conegut i la segona és la crida a la mateixa funció però passant-li com entrada el problema simplificat, ambdues parts estan relacionades d'un mode o altre segons el problema (suma, concatenació, etc)

Recursivitat acumulada

- Té un paràmetre extra consistent en la solució parcial del problema
- El cas de sortida consisteix en retornar aquest paràmetre extra
- El cas recursiu consisteix en cridar la mateixa funció, canviants els paràmetres. El paràmetre de la solució s'actualitza amb la feina que pertoqui i la resta de paràmetres es simplifiquen.
- La crida a la funció necessita passar com a paràmetre l'element neutre de l'operació. Pot fer-se un segon mètode més senzill que cridi al primer passant-li l'element neutre i estalviï a l'usuari d'utilitzar aquest paràmetre.

Exemple 2

Un altre exemple, aquesta vegada utilitzant llistes com a paràmetre d'entrada. La funció suma tots els elements de la llista:

Funcions	Detalls
sumlist :: [Int] -> Int sumlist [] = 0 sumlist (front:rest) = front + sumlist rest	Amb recursivitat de pila
xsum :: [Int] -> Int -> Int xsum [] total = total xsum (front:rest) total = xsum rest (front+total) sumlist' :: [Int]->Int sumlist' llista = xsum llista 0	Amb recursivitat acumulada. xsum és la classe que fa la feina i sumlist la classe que cridaria l'usuari, estalviant-li de passar el zero com a element neutre de la suma