# Parcial Llenguatges de Programació

Grau en Enginyeria Informàtica
Temps estimat: 1h i 45m

8 Abril 2016

Nomes podeu entrar al compte de la forma `lpXX` que se us ha assignat. Entrar en qualsevol altre compte o que algú altre usi el vostre compte invalidarà el vostre examen i es considerarà còpia.
Per accedir al racó aneu a `https://examens.fib.upc.edu`
Cal que lliureu el codi via racó amb els comentaris que considereu necessaris en un arxiu "examen.hs" executable en l'entorn ghci sense activar cap opció addicional (només fent `ghci examen.hs`) i que solucioni els problemes que es llisten a continuació.

A la vostra solució heu de mantenir tots els noms que indiqui l'enunciat.
Imprimirem la vostra solució amb la comanda
`a2ps -1 -r -f 8 --borders=0 --no-header --header=Examen examen.hs -o examen.ps`
comproveu que el que envieu té una indentació correcte i no es surt dels límits de la pàgina.

Cal que al començar la solució de cada problema afegiu una línia comentada indicant el problema i subapartat que ve a continuació. Per exemple,
`-- Problema 3.1`
Es valorarà l'ús de funcions d'ordre superior predefinides i la simplicitat de les solucions. Ara bé, només es poden usar funcions predefinides de l'entorn Prelude: no podeu fer cap `import`.

A l'arxiu proves.txt trobareu exemples de crides i la seves solucions per a tenir alguns exemples de prova en format text.

**Problema 1 (1 punt)**: *Subconjunts repetits.* Feu la funció `allsets :: a -> [[a]]` que donat un valor $x$, genera la llista infinita que conte totes les llistes en ordre creixent de longitud que contenen zero o més vegades el valor $x$. Per example, `allsets 3` és `[[],[3],[3,3],[3,3,3],[3,3,3,3],[3,3,3,3,3],...`
Es valorarà que **no** useu recursivitat.

**Problema 2 (1.5 punt)**: *Divisors agrupats.* Feu la funció `alldivisors:: Int -> [[Int]]` que donat un enter $x$ genera la llista que conté una llista per cada divisor $d$ diferent de 1 que té $x$, on $d$ apareix tans cops com vegades divideix a $x$. La llista estarà ordenada creixentment segons el valor que contingui cada subllista. Per exemple,
`alldivisors 14283` és
`[[3,3,3],[9],[23,23],[27],[69,69],[207],[529],[621],[1587],[4761],[14283]]`

**Problema 3 (3 punts)**: *Expressions generals.* Feu les següents definicions i funcions.

**Apartat 3.1**: Definiu un data polimòrfic `Expr a`, que permeti representar expressions que poden ser variables amb el constructor `Var` i un nom descrit amb un `String`, constants amb el constructor `Const` i un valor del tipus `a` o bé operadors amb el constructor `Func` , un nom descrit amb un `String` i una llista d'arguments que són novament expressions (`Expr a`). Noteu que són com les expressions numèriques de la pràctica.

Per al cas d'enters tenim que un valor del tipus `Expr Int` és

```
Func "op2" [Func "op1" [Const 1,Var "x1"],Var "ma",Func "op4" [Var "x2",Func "op1" [Const 1,Var "x1"]]]
```

o, per a que es llegeixi millor

```
Func "op2" [Func "op1" [Const 1,Var "x1"],
            Var "ma",
            Func "op4" [Var "x2",
                        Func "op1" [Const 1,Var "x1"]
                       ]
           ]
```

Si per poder veure les expressions per pantalla o per algun dels exercicis següents (d'aquest problema o d'un altre) us cal fer que `Expr` sigui d'alguna de les classes predefinides (com ara `Show` o `Eq`), feu-ho de la manera més simple possible (és a dir, no implementeu coses innecessàries).

**Apartat 3.2**: Definiu la funció `constLeafs :: Expr a -> [a]` que donada una expressió ens retorna la llista de valors constants (és a dir, sota un `Const`) que apareixen (d'esquerra a dreta) a l'expressió. Per exemple, per l'exemple anterior el resultat seria `[1,1]`. A l'arxiu adjunt `proves.txt` trobareu un altre exemple.

**Apartat 3.3**. Feu que el tipus `Expr` sigui `instance` de la classe `Functor` on (fmap) és la funció que fa el map de la funció donada aplicant-la a totes les constants (és a dir, els valors que estan sota un `Const`). Recordeu que la classe Functor es defineix amb:

```
class Functor f where
 fmap :: (a->b) -> f a -> f b
```

La vostra definició ha d'incloure un cas com

```
fmap g (Var x) = Var x
```

Afegiu aquest i la resta de casos.
Com a exemple, si feu `fmap` amb `even` i l'arbre de l'exemple anterior obtindreu

```
Func "op2" [Func "op1" [Const False,Var "x1"],Var "ma",Func "op4" [Var "x2",Func "op1" [Const False,Var "x1"]]]
```

A l'arxiu adjunt `proves.txt` trobareu un altre exemple.

**Problema 4 (2 punts)**: *Fusions compatibles.* Considereu que tenim una llista de parells que representa una llista d'assignacions amb claus i valors. Les claus es representen amb un String i els valor son genèrics. Pel cas de que els valors siguin enters podem tenir la següent llista

```
[("ma",8),("x2",5),("x3",12)]
```

Feu la funció `join :: Eq a => [(String,a)] -> [(String,a)] -> Maybe [(String,a)]` que donades dues llistes ordenades per clau i sense claus repetides torna `Nothing` si les assignacions no són compatibles (és a dir tenen valors diferents per a la mateixa clau) i un `Just` de la fusió ordenada sense repetits de les dues assignacions, en cas contrari. Per exemple, això és el que obtindrem amb les següents crides

```
> join [("x1",3),("x2",5)] [("ma",8),("x2",5),("x3",12)]
Just [("ma",8),("x1",3),("x2",5),("x3",12)]

> join [("x1",3),("x2",5)] [("ma",8),("x2",6),("x3",12)]
Nothing
```

A l'arxiu adjunt `proves.txt` trobareu un altre exemple.

**Problema 5 (2.5 punts)**: *Matching d'expressions.* Una expressió $e_1$ fa *matching* amb una expressió $e_2$ quan existeix una assignació de les variables de $e_1$ a expressions que fa que les dues expressions ($e_1$ amb l'assignació i $e_2$) siguin exactament iguals. Per exemple, `Func "bin" [Var "x1",Var "x1"]` fa matching amb
`Func "bin" [Func "un" [Const 2] , Func "un" [Const 2]]`, ja que a "x1" se li assigna `Func "un" [Const 2]`.

Feu la funció `match:: Eq a => Expr a -> Expr a -> Maybe [(String,(Expr a))]` que retorna `Nothing` si no fan matching i `Just` de l'assignació que les fa iguals, en cas contrari.

Heu d'usar la funció `join` i només us heu de preocupar de les variables de la primera expressió. Tingueu en compte que les variables poden aparèixer més d'un cop (com a l'exemple) i que, per a que faci matching, el valor que se li assigna a cada aparició ha de ser sempre el mateix. A l'arxiu adjunt `proves.txt` trobareu dos exemples.

Podeu suposar que dos operadors (en un constructor `Func`) amb el mateix nom (String) tenen sempre el mateix nombre de fills.