Parcial Llenguatges de Programació

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

12 desembre 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 (1.5 punt): *Merge múltiple sense repetits.* Feu la funció `mergelist :: Ord a => [[a]] -> [a]` que donada una llista de llistes ordenades i sense repetits retorna la llista ordenada i sense repetits que les uneix totes. Per exemple, `mergelist [[10,14,24],[6,9,16,23,30,37],[],[7,14,20,30],[7,13],[9,14]]` és `[6,7,9,10,13,14,16,20,23,24,30,37]`.

Problema 2 (1.5 punt): *Múltiples.* Feu la funció `mults :: [Integer] -> [Integer]` que donada una llista ordenada estrictament creixent (sense repetits) d'enters l genera la llista (infinita) ordenada estrictament creixent (sense repetits) que conté l'1 i tots els nombres que són múltiples només dels nombres de l. Per exemple, `mults [2,3,5,7]` és `[1,2,3,4,5,6,7,8,9,10,12,14,15,16,18,20,21,24,25,27,...`
es valorarà l'eficiència i la simplicitat de la solució. Podeu usar la funció `mergelist`.

Problema 3 (3 punts): *Seqüència de processos.* Cal que feu el següent:

Apartat 3.1: Definiu un data polimòrfic `Procs a`, que permeti representar una seqüència de processos determinats amb els següents constructors:

1. `End`, que no té paràmetres i representa el final de la seqüència.
2. `Skip`, que té un únic paràmetre que és `Procs a`, i representa que el procés es salta l'element de l'entrada i l'envia a la sortida i segueix amb la resta de processos.
3. `Unary`, que té un primer paràmetre que és una funció unària `a -> a` i un segon paràmetre que és `Procs a`, i representa que el procés aplica la funció unària usant l'entrada i segueix amb la resta de processos.
4. `Binary`, que té un primer paràmetre que és una funció binària `a -> a -> a` i un segon paràmetre que és `Procs a`, i representa que el procés aplica la funció binària usant l'entrada i segueix amb la resta de processos.

Com a exemples tenim:

```
let proc1 = Unary (+1) (Binary (*) (Skip (Unary (\x->x*2+1) End)))
let proc2 = Unary (+1) (Skip (Binary (*) End))
```

Apartat 3.2: Definiu la funció `exec :: [a] -> (Procs a) -> [a]` que donada una llista d'entrada i una seqüència de processos, els executa de la següent manera:

1. Si l'entrada és buida, acaba l'execució retornant la llista buida.
2. Si es troba `End`, retorna la llista de l'entrada.
3. Si es troba `Skip`, torna el primer valor de l'entrada com a primer de la sortida i segueix avaluant l'entrada amb la resta de seqüència de processos.
4. Si es troba `Unary`, aplica la funció unària al primer element de l'entrada i el reemplaça pel resultat i segueix executant la resta de processos de la seqüència.

5. Si es troba `Binary`, aplica la funció binària als dos primers elements de l'entrada i els reemplaça pel resultat i segueix executant la resta de processos de la seqüència. Si només hi ha un element a l'entrada fa el mateix però usant dos cops el mateix valor per aplicar la funció binària.

Com a exemples tenim:

```
exec [2,7,3,6] proc1
[21,7,6]

exec [5,1] proc2
[6,1]
```

Problema 4 (4 punts): *Contenidors.* Volem definir la classe `Container` (d'una forma similar a la classe `Evaluable` de la pràctica). Feu els següents apartats:
Apartat 4.1: Definiu la `class Container` amb les següents operacions:

1. `emptyC :: c a -> Bool`, que ens diu si el contenidor és buit.
2. `lengthC :: c a -> Int`, que ens diu el nombre d'elements del contenidor.
3. `firstC :: c a -> a`, que (suposant que no és buit) ens dona el primer element del contenidor.
4. `popC :: c a -> c a`, que (suposant que no és buit) ens dona el contenidor sense el primer elements.

Usant `fistC` i `popC`, hem de poder obtenir tots els elements del contenidor.

Apartat 4.2: Feu que les llistes `[]` siguin `instance` de la class `Container`.

Apartat 4.3: Definiu el data polimòrfic `Tree a`, que representa els arbres generals però amb dos constructors:

1. `Empty` per representar l'arbre buit
2. `Node` que té com a paràmetres un element de tipus `a` i una llista de `Tree a`.

Com a exemple tenim

```
Node 7 [Node 6 [Node 3 [],Empty,Node 6 []],Node 1 [],Node 12 [Node 9 []],Node 5 [Empty,Empty,Node 8 [],Node 9 [],Node 1 []]]
```

Apartat 4.4: Feu que `Tree` sigui instance de la classe `Container`, de manera que el `firstC` i el `popC` ens vagin donant els element seguint el recorregut en preorde de l'arbre.

Apartat 4.5: Feu la funció `iterator :: Container c => c a -> [a]` que ens dona una llista amb tots els elements del contenidor en un determinat ordre. Com a exemple tenim que `iterator t1` és `[7,6,3,6,1,12,9,5,8,9,1]`.