# Laboratori de PL: Haskell. Sessió 1

## 1 L'entorn de treball

En les sessions de laboratori treballarem amb el Glasgow Haskell Compiler. A les màquines de l'aula el trobareu a

```
/home/soft/ghc
```

Podeu afegir-ho al arxiu d'autoexecució ".tcshrc" posant

```
set path=($path /home/soft/ghc)
```

Inicialment treballarem amb l'intèrpret

```
/home/soft/ghc/ghci
```

Un cop entrat, proveu a carregar qualssevol dels programes de les sessions de teoria que trobareu a http://www.lsi.upc.edu/~albert/teaching.html

Recordeu que per carregar heu de fer, per exemple,

```
:l factorial.hs
```

També podeu posar `:load`

Podeu preguntar pel tipus inferit utilitzant

```
:t <nom_funció>
```

També podeu posar `:type`

Per sortir podeu escriure

```
:q
```

També podeu posar `:quit`

## 2 Programes amb llistes

Utilitzant com a referència els programes vistos a classe resoleu els següents exercicis:

1. Feu una funció que calculi la mitjana d'una llista d'enters.

2. Feu una funció que calculi la mediana d'una llista d'enters.

3. Feu una funció que donada una llista ens la torna un palíndrom. Per exemple si ens donen [2,4,6] hem de tornar [2,4,6,6,4,2].

4. Feu una funció que aplana una llista de llistes produint llista d'elements. Per exemple, si rep [[2,6],[8,1,4],[],[1]] torna [2,6,8,1,4,1].

5. Feu una funció que donada una llista ens diu si és un palíndrom.

6. Programeu l'algorisme d'ordenació mergesort. Per això programeu inicialment la mescla ordenada. Posteriorment feu la funció d'ordenació utilitzant les funcions predefinides `length`, `take` i `drop`. Aquestes dues últimes donat un enter no negatiu $n$ i una llista $l$ ens donen respectivament el $n$ primers elements de $l$ (o els que hi hagi,si n'hi ha menys) i la resta d'elements d'$l$ sense els $n$ primers.

7. Els nombres de Hamming formen una successió estrictament creixent de nombres que satisfan les següents condicions:

   - El número 1 és a la successió.
   - Si x és a la successió, llavors $2x$, $3x$ i $5x$ també hi són.
   - Cap altre nombre apareix a la successió.

   Utilitzant la mescla i llistes infinites, feu una funció que obtingui els n primers membres de la successió.

# 3 Creació de tipus de dades

En Haskell és molt fàcil crear noves estructures de dades. Considereu per exemple els arbres. El següent codi defineix els arbres binaris.

```
{-- Arbres binaris amb arbres buits --}
data Arbre a = Node a (Arbre a) (Arbre a)
             | Abuit
```

El nom dels tipus i els constructors han de començar per majúscula. La `|` s'usa per expressar les diferents opcions. És a dir, en la nostra definició, un arbre binari, o està buit o està format per un node i dos arbres més.

Noteu que podem usar aquests casos per definir les nostres funcions amb "pattern matching". Noteu també, que estem definit arbres binaris genèrics.

Mireu els arxius `Arbre.hs` i `opsArbre.hs`. Encara que no cal Arbre està definit com un mòdul que és importat per `opsArbre.hs`

Tal com està fet funciona amb l'intèrpret (pel compilador, no funciona directament, però ja veurem com es fa en la propera sessió). En aquest cas fem

```
:l Arbre.h
:l opsArbre.h
```

i després podem cridar a prova.

Utilitzant com a referència aquests programes resoleu els següents exercicis:

1. Feu una funció que calculi l'altura d'un arbre.
2. Feu una funció que ens digui si dos arbres són iguals.
3. Feu funcions que ens donin el recorregut en preordre, inordre, postordre i en amplada d'un arbre.
4. Considereu que l'arbre és un heap d'enters creixent, és a dir, que tots els descendents d'un node són més grans o iguals que ell. Feu operacions d'afegir i eliminar l'arrel. Utilitzeu-los per fer l'ordenació d'una llista.
5. Per aconseguir que el heap és mantingui equilibrat considereu que els nodes són parells que contenen l'enter (que és la informació) i un enter non-negatiu que indica quants elements hi ha al subarbre que encapçala. Adapteu les operacions realitzades de manera que el heap es mantingui equilibrat.
6. Extengueu la definició d'arbre a arbres generals, és a dir, que poden tenir qualsevol quantitat de fills. Afegiu operacions per preguntar quants fill té un arbre no buit i per demanar l'$i$-èssim fill. Modifiqueu alguna de les funcions anteriors per adaptar-la al nou tipus d'arbre.

# 4 Compilació. Entrada/Sortida

Considerem ara que treballarem compilant i muntant el codi, és a dir, generant un executable que cridarem per la línia de comandes.

En aquest cas és necessari llegir l'entrada de dades o bé de la línia de comandes o de l'entrada estàndard (també es pot fer d'un fitxer de text) i escriurem els resultats per la sortida estàndard.

En aquesta sessió només donarem alguns exemples de com fer-ho sense entrar en els detalls dels mecanismes del llenguatge (les *mònades*) que permeten fer-ho.

Mireu els exemples `factmain.hs` i `insert.hs`. Noteu que els programes han de tenir un main.

En aquest cas la forma de compilació és molt similar a la que s'usa amb el compilador `g++` de C++. Així podem fer

```
ghc factmain.hs
```

Cosa que a diferència del g++ ens genera un executable anomenat `factmain`. També podeu fer

```
ghc -o execfact factmain.hs
```

Cosa que gnera un executable anomenat `execfact`

Adapteu alguns del problemes vistos per llistes, per a que es puguin compilar.