# Sistemes Operatius II

# Pràctica 2

Juan Marin Vega
Iñaki Úbeda Martínez

# L’aplicació

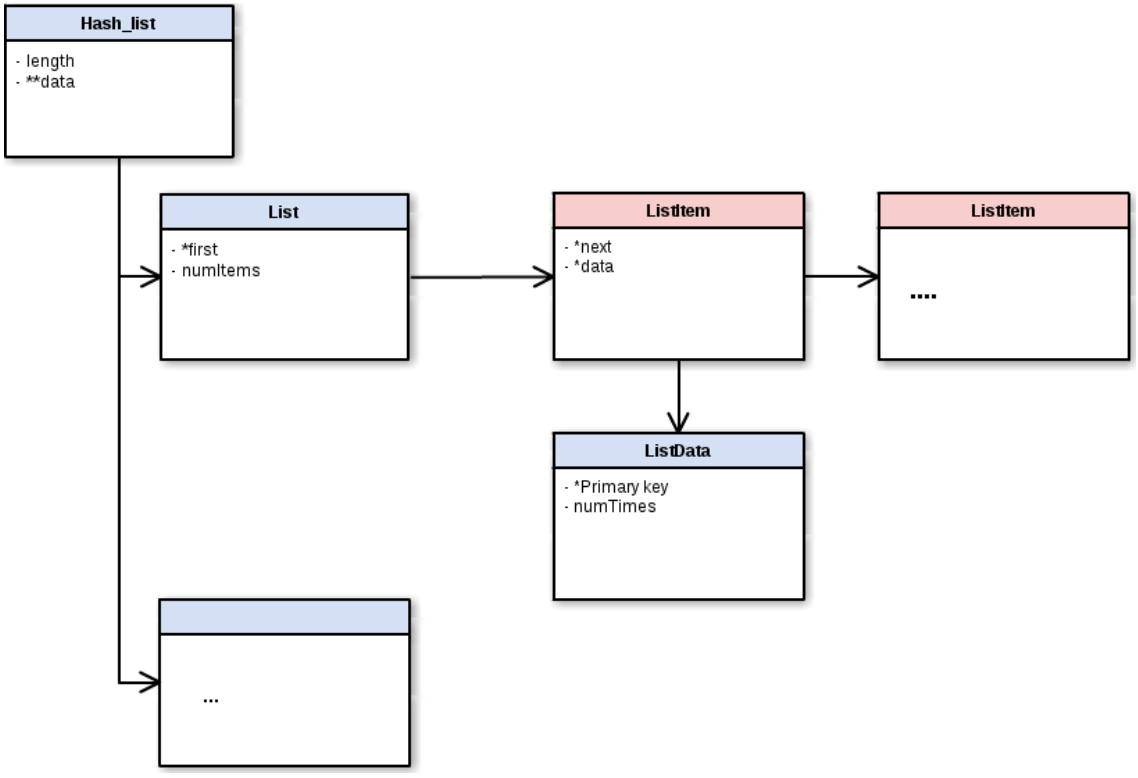
El funcionament de l’aplicació es el següent, quan s’executa el programa es passa un arxiu de configuració amb les rutes del fitxers on haurem de filtrar les paraules segons els criteris establerts a la pràctica anterior.

Per a la indexació de les paraules s’utilitzen dos tipus d'estructures de dades, la primera a nivell local consisteix a una taula hash que a cada posició (de 0 a 99) emmagatzema una llista enllaçada amb les paraules y el número de coincidencies al fitxer.
Un cop es finalitza el filtratge d’un fitxer les dades de la lista hash es passen a un arbre de dades del tipus black tree, aquest arbre té com a primary key la paraula, guarda també el número de coincidencies als fitxers i per últim la informació de les coincidencies a cada fitxer en un vector.

# Estructures utilitzades

Tal i com es comenta al punt anterior s’utilitzen dos tipus d’estructura de dades per tal d’emmagatzemar les paraules filtrades.

### Estructura local



**Estructura global**
Per a la estructura global hem utilitzat la que es dona inicialment afegint uns petits canvis per adaptar-la a les nostres necessitats.
Hem afegit un camp mes per tenir mes accessible el total d'aparicions d'una paraules al llistat de fitxers, s'ha modificat el tipus de la primary key per complir les especificacions de la practica i per ultim el tipus del vector d'informació dels diferents arxius.

Per ultim per guardar la informació de la paraula mes llarga utilitzem un struct amb la longitud de la paraula, el numero de fitxer al que surt (el primer) i un punter a la paraula.

# Proves
La execució de l'algorisme complet

```
time ./practica2 llista.cfg > log
```

dona com a sortida:

```
real  2m15.468s
user  2m9.730s
sys   0m0.639s
```

i al fitxer log podem veure que s'han comptabilitzat fins a 474578 paraules diferents

**Proves amb la taula hash**

Amb una taula hash de 100 posicions i 20 arxius executant:

```
time ./practica2 llista20.cfg > log20
```

obtenim:

```
real  0m3.471s
user  0m3.059s
sys   0m0.331s
```

Amb una taula hash de 500 posicions i 20 arxius executant la mateixa instrucció obtenim:

```
real  0m1.265s
user  0m1.037s
sys   0m0.212s
```

evidentment, com mes gran es la taula hash, menys col·lisions hi hauran i les llistes enllaçades seran mes curtes.

Utilitzant una taula hash de 50 posicions i els mateixos 20 arxius

```
real  0m7.382s
user  0m7.094s
sys   0m0.248s
```

El programa es considerablement mes lent, degut al nombre de col·lisions que hi hauran, provocant que les llistes enllaçades seran mes llargues.

Al executar tot els fitxers per a indexar les paraules s'obté com a paraula més llarga

```
nationalgymnasiummuseumsanatoriumandsuspensoriumsordinaryprivatdoce
nt
```

de longitud 69, ubicada al fitxer:

```
etext03/ulyss12.txt
```

## Consideracions sobre els vectors

Considerem que un vector es una bona opció per quant hi han moltes paraules que surten en molts arxius. Si no es aixi, la llista enllaçada es mes apropiada. El problema es que aixo no es pot saber a priori, tans sols es pot saber si s'han d'analitzar molts o pocs arxius.
A mes a mes, la llista enllaçada suposa utilitzar com a minim:
8 bytes per a un punter que ens permeti mourens entre nodes, 4 bytes per a un enter que indiqui la quantitat de vegades que apareix, 4 bytes per a un enter que indiqui quin arxiu es.

Això suposa un 3 vegades mes memòria, com a mínim, per a representar la quantitat de vegades que surt una paraula respecte d'un arxiu.
També es podrien utilitzar altres metodes per emmagatzemar tota la informació relativa a una paraula, ja que no es necessari tenir tota la informació de totes les paraules a memòria de forma simultanea.

## Execució

Per tal d'executar el programa s'ha de fer make a la carpeta on es troben tots els fitxer .c i .h. Un cop s'ha fet el make s'ha de copiar el fitxer generat a la carpeta base_dades/ i executar dins d'aquesta carpeta l'aplicació amb el format especificat a l'enunciat.