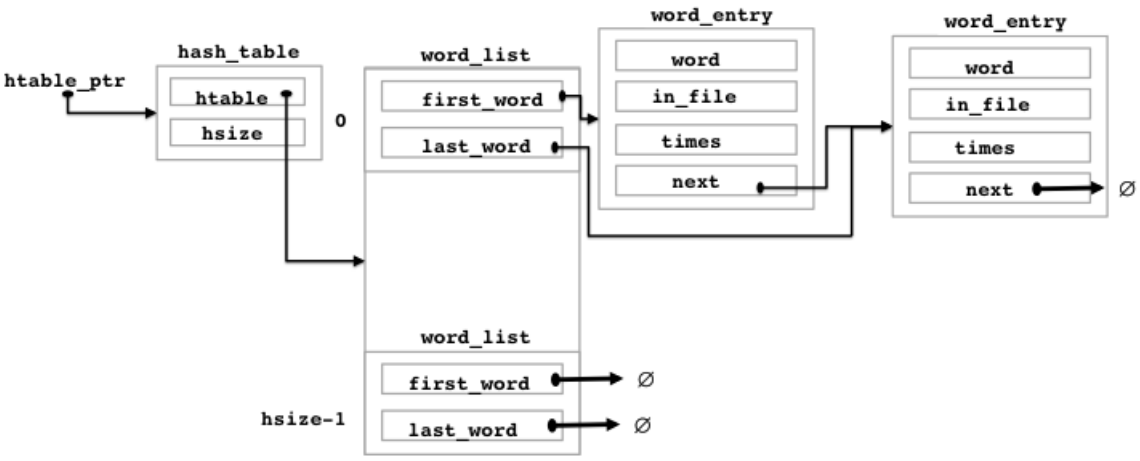
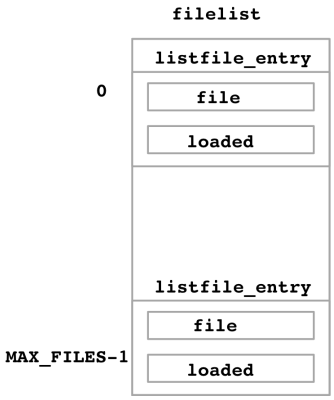
TP7-9: Word search machine

1 Getting started

The goal of the project is to implement a word search engine. The word search machine loads the words of file given by the user to a dictionary. The user can enter a word (e.g. forest) into the application through the application's command line and the application tells him in what files that word exist and how many times. The codes for this lab are in the directory **/share/l3info/CUnix/tp7-9**. The dictionary that stores the words is implemented using a hash table with separate chaining, as the one seen in the course :



An array **filelist** of structures of **listfile_entry** is used to store the names of the files already loaded to the dictionary **file** and their status **loaded**.



2 Implementation

The source files and the functionalities are described below :

- **main.c** : includes the following functions :
  - **int main ()** to manage the creation and deletion of the required data structures and display the menu of choices to the user
  - **int hashcode(char word[], int size)** to return the hash value of a word, as we have seen during the course.
- **file.c** includes the functions dedicated to the manipulation of a file
  - : **listfile_entry * create_filelist(int maxfiles)** : Creation and initialization of the table to keep information for the loaded files
  - **int add_file(char filename[], listfile_entry * filelist, hash_table * htable_ptr):** It verifies that the file to be loaded (given by the user) has not already been loaded. In this case, it reads the file word by word (numbers are excluded). For each word, it verifies if it is the first time that is found in the file. In this case, it adds the word to the linked list at the hash table position given by the hash key by creating a new **word_entry**. In case the word has already been found before in the file, it already exists in the linked list (same word, same file). Then, the number of times is increased by one. No difference should be made between capital and small letters, e.g. all words are stored in small letters. When all the words of the file have been loaded to the dictionary, the status of the file in the array **filelist** is set to 1, which means loaded.
  - **int remove_file(char filename[], listfile_entry * filelist, hash_table * htable_ptr):** It verifies that the file to be removed (given by the user) is loaded. In this case, it searches the dictionary to find all the words that belong to this file and removes them. In case the file is not loaded to the dictionary , it informs the user.
  - **void print_list(listfile_entry * filelist):** Prints the array **filelist** with the names of the loaded files and their status.
  - **void free_filelist(listfile_entry * filelist):** Deallocates the memory for the table that keeps the information about the loaded files
- **hash.c** includes the functions dedicated to the manipulation of the hash table :
  - **hash_table * create_table():** It creates and initializes the hash table.
  - **int search(char word[], listfile_entry *filelist, hash_table *htable_ptr):** It searches the word given by the user in the dictionary and it prints the number of times found in each file. In case the word does not exist, it informs the user.
  - **void print_table(hash_table *htable_ptr, listfile_entry * filelist):** It prints the non empty positions of the dictionary.
  - **void free_table(hash_table * htable_ptr):** Deallocates the memory used for the hash table

3 Exercise

1. Implement the functions of the word search machine in the files **main.c**, **file.c** and **hash.c**
2. Complete the **Makefile** in order to compile correctly your compilation files.
3. Check your program for correct operation and implementation (memory leak, uninitialized values etc) using Valgrind.
4. You are allowed (and advised) to **add additional sub-functions** to your implementation.

OPTIONAL: In order to facilitate the search, it should be possible for the user to enter logical expressions (at least by the operator AND), i.e. word1 AND word2 that will return the files containing both words. More operators (eg OR, NOT) and bracket support are desirable !