```
#ifndef __FUNCTIONS_H__
#define __FUNCTIONS_H__

#include "types.h"

// ---------------------------------------------------------------------
// file.c
// ---------------------------------------------------------------------

// create and initialize file table of capacity maxfile
listfile_entry *
create_filelist(int maxfiles);

// add words from file to table
int
add_file(char filename[],
         listfile_entry * filelist,
         hash_table * htable_ptr);

// remove file from file table
int
remove_file(char filename[],
            listfile_entry * filelist,
            hash_table * htable_ptr);

// print file table
void
print_list(listfile_entry * filelist);

// free file table
void
free_filelist(listfile_entry * filelist);

// ---------------------------------------------------------------------
// hash.c
// ---------------------------------------------------------------------

// create hash table
hash_table *
create_table();

// search a word in table and print it
// returns : true if found, false otherwise
int
search_word(char word[],
            listfile_entry * filelist,
            hash_table * htable_ptr);

// print table contents
void
print_table(hash_table * htable_ptr,
            listfile_entry * filelist);

// free hash table
void
free_table(hash_table * htable_ptr);

// ---------------------------------------------------------------------
// main.c
// ---------------------------------------------------------------------

// compute hash value for word
```



```
// returns : N;/ 0 <= N < size
int
hashcode(char word[], int size);

#endif // __FUNCTIONS_H__
```

```
#ifndef __TYPES_H__
#define __TYPES_H__

#define MAX_LENGTH 50        // maximum word length of an entry
#define MAX_FILES 20         // maximum number of files
#define MAX_ENTRIES 1023     // capacity of hash table

// elements of the word list
typedef
struct word_entry
{
  char word[MAX_LENGTH];
  int in_file; // index of file in file table
  int times;   // how many times does the word exist
  struct word_entry * next;
}
  word_entry;

// simple linked list of word entries
typedef
struct
{
  word_entry * first_word;
  word_entry * last_word;
}
  word_list;

// a hash table is an array of word_list + maximum number of elements in the arr
ay
typedef
struct
{
  word_list * htable;
  int hsize; // capacity of array
}
  hash_table;

// names of files loaded in the hash table + loaded status
typedef
struct
{
  char filename[MAX_LENGTH];
  int loaded;                  // true if file loaded
}
  listfile_entry;


#endif // __TYPES_H__
```

```c
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "../include/types.h"
#include "../include/functions.h" // extern functions declarations

// ----------------------------------------------------------------------------
// inner functions declarations
// ----------------------------------------------------------------------------

int filelist_not_full(listfile_entry *);

//-----------------------------------------------------------------------------
// global functions definitions
//-----------------------------------------------------------------------------

/**
   Create and initialize file table of capacity maxfiles

   parameters :
   maxfiles : capacity of file table

   returns : pointer to table or NULL in case of error
 */
listfile_entry *create_filelist(int maxfiles)
{
  listfile_entry *listfile_ptr;
  listfile_ptr = (listfile_entry*)malloc(sizeof(listfile_entry) * maxfiles);

  return listfile_ptr;
}

/**
   add words from file to table
   - checks if the file has already been loaded
   - updates the file table (if file not already loaded)
   - reads every word in the file (idem)
   - updates the hash table (idem)

   parameters :
   filename   : name of file :)
   filelist   : pointer to table of files
   htable_ptr : pointer to hash table

   returns :
    1 if file already present in table
    2 if no space left in filelist
   -1 if file doesn't exist or can't be read
   -2 if allocation error
    0 if everything ok
*/
int add_file(char filename[],listfile_entry * filelist, hash_table * htable_ptr)
{
  FILE *file;
  char buffer[MAX_LENGTH];

  int return_value;
  int i;
```



```c
  for (i = 0; i < MAX_FILES; i++) {
    if(strcmp(filename, filelist[i].filename) == 0 && filelist[i].loaded) {
      return_value = 1;
    }
  }

  if(filelist_not_full(filelist) == 0) {
    return_value = 2;
  }

  if(file == NULL) {
    return_value = -1;
  }
  file = fopen(filename, "r");

  //gÃ©rer la filelist

  while (fscanf(file, "%s", buffer) == 1) {
    int hash = hashcode(buffer, MAX_LENGTH);
    //test si le mot existe ou pas
    strcpy(htable_ptr->htable->first_word->word, buffer);
    htable_ptr->htable->first_word->times = 1;
  }
  if(feof(file)) {
    return_value = 0;
  }
  else {
    return_value = -2;
  }




  return return_value; // all fine
}

/**
   remove file from file table

   parameters :
   filename   : name of file to remove
   filelist   : pointer to table of files
   htable_ptr : pointer to hash table

   returns :
   -1 if file not in table
    0 if file removed
*/
int remove_file(char filename[], listfile_entry * filelist, hash_table * htable_
ptr)
{

  // TO BE COMPLETED

  return 0;
}

/*
  print file table (only loaded files)
```

```
  parameters :
    filelist : pointer to table of files
*/
void print_list(listfile_entry * filelist)
{

  // TO BE COMPLETED

}

/**
   free file table

parameters :
   filelist   : pointer to table of files
*/
void free_filelist(listfile_entry * filelist)
{
  // TO BE COMPLETED
}

// ************************************************************************
// inner functions
// ************************************************************************

/**
  returns:
    1 if space left in the filelist
    0 if filelist is full
**/
int filelist_not_full(listfile_entry * filelist) {
  return 1;
}
```

```c
#include <ctype.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../include/functions.h" // global functions declarations

// ----------------------------------------------------------------------
// inner functions declarations
// ----------------------------------------------------------------------

// TO BE COMPLETED

//-----------------------------------------------------------------------
// global functions definitions
//-----------------------------------------------------------------------

/**
   create and initialize hash table

   returns : pointer to table or NULL if creation error
*/
hash_table *create_table()
{
  int i;
  hash_table *htable_ptr;
  htable_ptr = (hash_table*)malloc(sizeof(hash_table));
  htable_ptr->hsize = MAX_ENTRIES;
  htable_ptr->htable = (word_list*)malloc(sizeof(word_list) * htable_ptr->hsize)
;
  for (i = 0; i < htable_ptr->hsize; i++) {
    htable_ptr->htable[i].first_word = NULL;
    htable_ptr->htable[i].last_word = NULL;
  }
  return htable_ptr;
}

/**
   search a word in table ; print word if found, with number of occurrences
   and file where word is found

   parameters :
   word : the word to look for
   filelist   : pointer to table of files
   htable_ptr : pointer to hash table

   returns : true if found, false otherwise
*/
int search_word(char word[], listfile_entry * filelist, hash_table * htable_ptr)
{

  // TO BE COMPLETED
  //printf("%s\n", );
  return true;
}

/**
   print table contents

   parameters :
   htable_ptr : pointer to hash table
   filelist   : pointer to table of files
```



```c
*/
void print_table(hash_table * htable_ptr, listfile_entry * filelist)
{

   // TO BE COMPLETED

}


/**
   free hash table

   parameters :
   htable_ptr : pointer to hash table
*/
void free_table(hash_table * htable_ptr)
{

  // TO BE COMPLETED

}

// ----------------------------------------------------------------------
// inner functions definitions
// ----------------------------------------------------------------------

// TO BE COMPLETED
```

```c
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../include/types.h"
#include "../include/functions.h"

//----------------------------------------------------------------------------

int main()
{
  // create hash table
  hash_table *hash_tablet = create_table();
  // create filelist array
  listfile_entry *filelist = create_filelist(MAX_FILES);

  // display menu
  while (1) {
    int nbchoices = 0;
    fprintf(stderr, "\nChoisir une action\n");
    fprintf(stderr, "%d. Load a file in dictionary\n", ++nbchoices);
    fprintf(stderr, "%d. Search a word in dictionary\n", ++nbchoices);
    fprintf(stderr, "%d. Remove file from dictionary\n", ++nbchoices);
    fprintf(stderr, "\n");
    fprintf(stderr, "%d. Print dictionary\n", ++nbchoices);
    fprintf(stderr, "%d. Print file list\n", ++nbchoices);
    fprintf(stderr, "\n0. Quit\n");
    int choice;
    while (1) {
      fprintf(stderr, "Your choice ? ");
      scanf("%d", & choice);
      if (choice >= 0 && choice <= nbchoices) { break; }
      fprintf(stderr, "\nError %d is an incorrect choice\n", choice);
    }
    if (choice == 0) { break; }

    fprintf(stderr, "-------------------------------------------------\n");

    // TO BE COMPLETED

    switch (choice) {

      // Load a file in dictionary
    case 1:

      // TO BE COMPLETED

      break;

      // Search a word in dictionary
    case 2:

      // TO BE COMPLETED

      break;

      // Remove file from dictionary
    case 3:

      // TO BE COMPLETED

      break;
```



```c
      // Print dictionary
    case 4:

      // TO BE COMPLETED

      break;

      // Print file list
    case 5:

      // TO BE COMPLETED

      break;
    }
    fprintf(stderr, "-------------------------------------------------\n");

  }

  // the end : free allocated memory

  // TO BE COMPLETED

  return 0;
}

// compute hash value for word
// returns : N ; 0 <= N < size
int hashcode(char word[], int size)
{
  int hash = 0;
  while(*word != '\0') {
    hash += *word++;
  }
  return (hash % size);
}
```