1 Codigo

1.1 frecpalhilo

```c
/*
*  File:        frecpalhilo.c
*  Author:      Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
*  Description: file that contains the implementation of the frecpal with
*               threads main
*  Date:        23 / 11 / 19
*/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include "utilities.h"
#include "hash.h"
#include "str_hash.h"
#include "str_list.h"
#include "hash_list.h"
#include "str_ht_list.h"
#include "counter_thread.h"
#include "error_handler.h"

#define MAX_WORD_LEN 100
#define HASH_SIZE 10007

pthread_mutex_t mtx;
pthread_t end = 0;

/*
*  Function : get_txt
*  ----------------------
*   Gets a directory name and extracts all files with the extention
*   txt
*
*   arg : name of the directory
*/
void* get_txt( void *arg ){
  char *dir_name;
  char **file_names;
  int size, i;
  hash h;
  pair *p;

  p = malloc( sizeof(pair) );
  size = 128;
  ht_make( &h , HASH_SIZE );
  dir_name = (char *)arg;
  file_names = malloc( sizeof(char *) * size );

  *p = traverse_dir( dir_name , file_names , 0 , &size , &h );
  errorp( p->f , "Error moving through the given directory.\n");

  pthread_exit( p );
}


/*
*  Function : count_words
*  ----------------------
*   Gets some file names, and counts the words in them
*
*   arg : input type, contains file names, starting index, size of
*     array of names and the offset to his files
*/
void* count_words( void *arg ){
  input *inp;
  int i, begin , n , mod, aux, e;
  FILE *fp;
  str_list l;
  str_hash H;
  pair_2 * cnt;
  char word[MAX_WORD_LEN];
  int id = 0;
  char **file_names;
  char *aux_w;
  str_node *it, *it2;
  int ind;
  ret *retval;
  str_ht_list_node *np, *np2;

  inp = ( input * )arg;
  mod = inp->MOD;
  begin = inp->begin;
  n = inp->n;
  file_names = inp->file;
  e = str_ht_make( &H );
  error( e , "Error allocating memory.\n");

```

```
89    free( inp );
90
91    make_str_list( &l );
92
93    for( i = begin; i < n ; i += mod ){
94
95      fp = fopen( file_names[i] , "r" );
96      errorp( fp , "Error␣opening␣a␣file.\n");
97
98      while( fscanf( fp , "%s" , word ) != EOF ){
99
100       aux = str_ht_find( &H , word , 1 );
101       if ( aux == 0 ){
102         aux_w = malloc( strlen(word) + 1);
103         strcpy( aux_w , word );
104         e = str_ht_insert( &H , aux_w , 1 );
105         error( e , "Error␣allocating␣memory.\n");
106
107         e = str_list_insert( &l , aux_w );
108         error( e , "Error␣allocating␣memory.\n");
109       }
110     }
111
112     fclose( fp );
113   }
114
115   cnt = malloc( sizeof(pair_2)*(l.size) );
116   errorp( cnt , "Error␣allocating␣memory.\n");
117
118
119   ind = 0;
120   it = l.head;
121
122   while( it != NULL ){
123     cnt[id].w = it->word;
124     cnt[id].c = str_ht_find( &H , it->word , 0 );
125     id++;
126     it2 = it;
127     it = it->next;
128     free( it2 );
129   }
130
131   retval = malloc( sizeof(ret) );
132   errorp( retval , "Error␣allocating␣memory.\n");
133
134
135   retval->cnt = cnt;
136   retval->size = l.size;
137
138
139   for( i = 0 ; i < 10007 ; i++ ){
140     np = (H.hash_table[i]).head;
141     while( np != NULL ){
142       np2 = np;
143       np = np->next;
144       free(np2);
145     }
146   }
147
148   free( H.hash_table );
149
150   pthread_mutex_lock(&mtx);
151
152   end = pthread_self();
153
154   pthread_exit( retval );
155 }
156
157
158 int main( int argc , char **argv ){
159
160   int n_threads, n_txt, e, i, j, cont, ind, n_words;
161   pthread_t *count_threads, txt_thread, thr_id;
162   char **txt_names;
163   pair *p_aux;
164   str_hash h;
165   input *inp;
166   str_list l;
167   ret **count_rets;
168   str_node *it, *it2;
169   str_ht_list_node *np, *np2;
170   pair_2 *words;
171
172   if ( argc != 3 ){
173     printf("Error␣in␣the␣given␣input.\n");
174     return -1;
175   }
176
177   n_threads = atoi( argv[1] );
178
179   if ( n_threads == 0 ){
180     printf("Unvalid␣number␣of␣threads.\n");
```

```
  return -1;
}

/* This thread will look for the txts and return them */
e = pthread_create( &txt_thread , NULL , get_txt , argv[2] );
error( e , "Error creating txt_thread.\n");


e = pthread_join( txt_thread , (void **)&p_aux );
error( e , "Error joining txt_thread.\n");

p_aux = (pair *)p_aux;
n_txt = p_aux->s;
txt_names = p_aux->f;
free(p_aux);

pthread_mutex_init(&mtx, NULL);

/* If the number of threads given is greater than the number of txt files
we will only use 1 thread for file, so the number of threads will become
smaller */
if ( n_threads > n_txt ) n_threads = n_txt;

count_threads = malloc( sizeof(pthread_t) * n_threads );
errorp( count_threads , "Error allocating memory.\n");


for( i = 0 ; i < n_threads ; i++ ){
  inp = malloc( sizeof(input) );
  errorp( inp , "Error allocating memory.\n");

  inp->n = n_txt;
  inp->MOD = n_threads;
  inp->begin = i;
  inp->file = txt_names;

  /* Here we create the counter threads and assing them the corresponding
  txts */
  e = pthread_create( &count_threads[i] , NULL , count_words , (void *)inp );
  error( e , "Error creating count_words thread.\n");
}

/* Here we allocate space for the counter threads output */
count_rets = malloc( sizeof(ret*) * n_threads );
errorp( count_rets , "Error allocating memory.\n");


i = 0;

while(i < n_threads){
  while(!end);
  thr_id = end;
  end = 0;
  e = pthread_join( thr_id ,(void **)&count_rets[i] );
  error( e , "Error joining count_words thread.\n");

  count_rets[i] = (ret*)count_rets[i];
  i++;
  pthread_mutex_unlock(&mtx);
}


e = str_ht_make( &h );
error( e , "Error allocating memory.\n");

make_str_list( &l );

/* In this loop we will take all the words given by the counter threads
and store them in a hash table where we will update their frecuency and in
a list so we easily know how many and what words we have */
for( i = 0 ; i < n_threads ; i++ ){
  for( j = 0 ; j < count_rets[i]->size ; j++ ){
    /* If the word already is in the hash, update its rep count */
    cont = str_ht_find( &h , count_rets[i]->cnt[j].w , count_rets[i]->cnt[j].
        c);

    /* Else insert it in the hash and in the list */
    if ( cont == 0 ){
      e = str_ht_insert( &h , count_rets[i]->cnt[j].w , count_rets[i]->cnt[j
          ].c);
      error( e , "Error allocating memory.\n");


      e = str_list_insert( &l , count_rets[i]->cnt[j].w  );
      error( e , "Error allocating memory.\n");
    }
  }
}

free( count_threads );
for( i = 0 ; i < n_txt ; i++ ){
  free( txt_names[i] );
```

```
  }
  free( txt_names );

  n_words = l.size;
  words = malloc( sizeof(pair_2)*n_words );
  errorp( words , "Error allocating memory.\n");


  ind = 0;
  it = l.head;
  /* Here we pass the words with their rep count from the hash and list,
  to an array so we can sort it using c qsort, and free the list nodes */
  while( it != NULL ){
    words[ ind ].w = it->word;
    words[ ind ].c = str_ht_find( &h , it->word , 0 );
    ind++;
    it2 = it;
    it = it->next;
    free(it2);
  }

  /* Free the hash table space */
  for( i = 0 ; i < 10007 ; i++ ){
    np = (h.hash_table[i]).head;
    while( np != NULL ){
      np2 = np;
      np = np->next;
      free(np2);
    }
  }
  free( h.hash_table );

  /* Sort the words with a custom comparator, so we get the expected order */
  qsort( words , n_words , sizeof( pair_2 ) , word_frec_comparator );

  for( i = 0 ; i < n_words ; i++ ){
    printf("%s %d\n", words[i].w , words[i].c );
  }

  return 0;
}
```

1.2 frecpalproc

```
/*
*  File:        frecpalproc.c
*  Author:      Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
*  Description: file that contains the implementation of the frecpal with
*               processes main
*  Date:        23 / 11 / 19
*/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <semaphore.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include "utilities.h"
#include "hash.h"
#include "str_hash.h"
#include "str_list.h"
#include "hash_list.h"
#include "str_ht_list.h"
#include "error_handler.h"

#define MAX_WORD_LEN 100
#define HASH_SIZE 10007


int main( int argc , char **argv ){

  int n_proc, n_txt, e, i, j, aux, cnt, status,
      cont, ind, n_words, fd[2], word_len, fd_fifo;

  char **txt_names;
  char *** txt_of_proc;
  char * word;
  sem_t *semaphore;
  str_hash h;
  str_list l;
  str_node *it, *it2;
  str_ht_list_node *np, *np2;
  pair_2 *words;

  if ( argc != 3 ){
    printf("Error in the given input.\n");
    return -1;
  }

```

```
50  n_proc = atoi( argv[1] );
51
52  if ( n_proc == 0 ){
53    printf("Unvalid␣number␣of␣processes.\n");
54    return -1;
55  }
56
57  /* Create a non named pipe for reading the work of get_txt process */
58
59  e = pipe(fd);
60
61  error(e, "Error␣creating␣a␣non␣nominal␣pipe");
62
63
64  /* This process will look for the txts and return them */
65  e = fork();
66
67  error(e, "Error␣creating␣get_txt␣process");
68
69  if(e == 0){
70    /* child */
71    close(fd[0]);
72    /*dup2(1, fd[1]);*/
73    dup2(fd[1], 1);
74    close(fd[1]);
75
76    e = execl("get_txt", "get_txt", argv[2], NULL);
77
78    error(e, "Error␣in␣execution␣of␣\"get_txt\"");
79
80  }
81  /* father continue */
82  close(fd[1]);
83
84  /* Wait the child process */
85  e = wait(&e);
86
87  error(e, "Error␣in␣child␣process␣get_txt");
88
89  /* Get txt files names from child process via pipe */
90
91  e = read_aux(fd[0], &n_txt, 4);
92
93  error(e, NULL);
94
95  txt_names = (char **) malloc(sizeof(char *) * n_txt);
96
97  errorp(txt_names, NULL);
98
99  for( i = 0; i < n_txt; ++i){
100
101   e = read_aux(fd[0], &word_len, 4);
102   error(e, NULL);
103
104   txt_names[i] = (char *) malloc(sizeof(char) * (word_len + 1));
105   errorp(txt_names[i], NULL);
106
107   e = read_aux(fd[0], txt_names[i], word_len + 1);
108   error(e, NULL);
109
110 }
111
112 close(fd[0]);
113
114 /* Create named pipe for reading the work of the counter processes */
115 unlink("myfifo");
116 e = mkfifo("myfifo", 0666);
117 error(e, NULL);
118
119 /*fd_fifo = open("myfifo", O_RDONLY);*/
120
121 /* Create named semaphore for counter processes coordination while writing in
        named pipe */
122 sem_unlink("mySmph");
123 semaphore = sem_open("mySmph", O_CREAT, 0666, 1);
124 if (semaphore == SEM_FAILED) {
125       perror("sem_open(3)␣failed");
126       exit(-1);
127   }
128
129   e = sem_close(semaphore);
130 error(e, NULL);
131 /* If the number of processes given is greater than the number of txt files
132 we will only use 1 thread for file, so the number of threads will become
133 smaller */
134 if ( n_proc > n_txt ) n_proc = n_txt;
135
136 /* We store the files names of the txt's that every counter process in their
       corresponding array */
137
138 txt_of_proc = (char ***) malloc(sizeof(char **) * n_proc);
139 errorp(txt_of_proc, NULL);
```

```

  for( i = 0 ; i < n_proc ; ++i ){

    txt_of_proc[i] = (char **) malloc(sizeof(char *) * (n_txt / n_proc + ((
        n_txt % n_proc) > i + 1) + 1) );
    errorp(txt_of_proc[i], NULL);

    txt_of_proc[i][n_txt / n_proc + ((n_txt % n_proc) > i + 1)] = NULL;

  }

  for( i = 0 ; i < n_txt ; ++i ){

    txt_of_proc[i % n_proc][i / n_proc] = (char *) malloc(sizeof(char) * (
        strlen(txt_names[i]) + 1) );
    errorp(txt_of_proc[i % n_proc][i / n_proc], NULL);
    strcpy(txt_of_proc[i % n_proc][i / n_proc], txt_names[i]);

  }

  for( i = 0 ; i < n_proc ; ++i ){

    /* Here we create the counter processes and assing them the corresponding
    txts */

    e = fork();

    error(e, "Error creating count_words process");

    if( e == 0){
      /* child */
      e = execv("count_words", txt_of_proc[i]);

      error(e, "Error in execution of \"count_words\"");

    }

  }

  fd_fifo = open("myfifo", O_RDONLY);

  e = str_ht_make( &h );
  error(e, "Error allocating memory");

  make_str_list( &l );

  i = 0;

  /* In this loop we will take all the words given by the counter processes
  and store them in a hash table where we will update their frecuency and in
  a list so we easily know how many and what words we have */
  while( i < n_proc ){

    e = read_aux(fd_fifo, &aux, 4);

    if(!e) continue;

    if(aux == -1){
      i++;
      continue;
    }

    word = (char *) malloc(sizeof(char) * (aux + 1));

    read_aux(fd_fifo, word, aux + 1);

    read_aux(fd_fifo, &cnt, 4);

    /* If the word already is in the hash, update its rep count */
    cont = str_ht_find( &h , word , cnt);

    /* Else insert it in the hash and in the list */
    if ( cont == 0 ){

      e = str_ht_insert( &h , word , cnt);
      error(e, "Error allocating memory");

      e = str_list_insert( &l , word  );
      error(e, "Error allocating memory");

    }
  }

  for( i = 0; i < n_proc; ++i ){
    e = wait(&status);
    error(e, NULL);
    if( WIFEXITED(status) ) error(WEXITSTATUS(status), NULL);
  }

  e = sem_unlink("mySmph");
  error(e, NULL);

```

```
230   close(fd_fifo);
231   unlink("myfifo");
232
233   for( i = 0 ; i < n_txt ; ++i ){
234     free( txt_names[i] );
235   }
236   free( txt_names );
237
238   n_words = l.size;
239   words = malloc( sizeof(pair_2)*n_words );
240   errorp(words, "Error allocating memory");
241
242   ind = 0;
243   it = l.head;
244   /* Here we pass the words with their rep count from the hash and list,
245   to an array so we can sort it using c qsort, and free the list nodes */
246   while( it != NULL ){
247     words[ ind ].w = it->word;
248     words[ ind ].c = str_ht_find( &h , it->word , 0 );
249     ind++;
250     it2 = it;
251     it = it->next;
252     free(it2);
253   }
254
255   /* Free the hash table space */
256   for( i = 0 ; i < 10007 ; i++ ){
257     np = (h.hash_table[i]).head;
258     while( np != NULL ){
259       np2 = np;
260       np = np->next;
261       free(np2);
262     }
263   }
264   free( h.hash_table );
265
266   /* Sort the words with a custom comparator, so we get the expected order */
267   qsort( words , n_words , sizeof( pair_2 ) , word_frec_comparator );
268
269   for( i = 0 ; i < n_words ; i++ ){
270     printf("%s %d\n", words[i].w, words[i].c );
271   }
272
273   return 0;
274 }
```

1.3 count words

```
1
2  /*
3  *  Function : main of count_words
4  *  ----------------------
5  *   Gets some file names, and counts the words in them
6  *
7  *   argc : number of txt files
8  * argv : txt files names
9  */
10
11 #include <stdlib.h>
12 #include <string.h>
13 #include <stdio.h>
14 #include <unistd.h>
15 #include <semaphore.h>
16 #include <sys/types.h>
17 #include <fcntl.h>
18 #include <sys/stat.h>
19 #include "utilities.h"
20 #include "hash.h"
21 #include "str_hash.h"
22 #include "str_list.h"
23 #include "hash_list.h"
24 #include "str_ht_list.h"
25 #include "error_handler.h"
26
27 #define MAX_WORD_LEN 100
28
29 int main( int argc, char ** argv ){
30
31   int i, n, aux, e, fd, size_word, cnt_word;
32   FILE *fp;
33   str_list l;
34   str_hash H;
35   char word[MAX_WORD_LEN];
36   int id = 0;
37   char **file_names;
38   char *aux_w;
39   str_node *it, *it2;
40   str_ht_list_node *np, *np2;
41   sem_t *semaphore;
42
43   n = argc;
44   file_names = argv;
```

```
45
46  e = str_ht_make( &H );
47  error(e, "Error␣allocating␣memory");
48
49  make_str_list( &l );
50
51  for( i = 0; i < n ; ++i ){
52
53    fp = fopen( file_names[i] , "r" );
54    if ( fp == NULL ){
55      printf("Error␣opening␣file␣%s.\n", file_names[i]);
56      continue;
57    }
58
59    while( fscanf( fp , "%s" , word ) != EOF ){
60
61      aux = str_ht_find( &H , word , 1 );
62
63      if ( aux == 0 ){
64
65        aux_w = malloc( strlen(word) + 1);
66        errorp(aux_w, NULL);
67        strcpy( aux_w , word );
68        e = str_ht_insert( &H , aux_w , 1 );
69        error(e, "Error␣allocating␣memory");
70
71        e = str_list_insert( &l , aux_w );
72        error(e, "Error␣allocating␣memory");
73
74      }
75
76    }
77
78    fclose( fp );
79  }
80
81
82  it = l.head;
83
84  fd = open("myfifo", O_WRONLY);
85
86  semaphore = sem_open("mySmph", O_RDWR);
87  if (semaphore == SEM_FAILED) {
88        perror("sem_open(3)␣failed");
89        exit(-1);
90    }
91
92  while( it != NULL ){
93
94    size_word = strlen(it->word);
95    cnt_word = str_ht_find( &H , it->word , 0 );
96
97    /* In section */
98    e = sem_wait(semaphore);
99    error(e, NULL);
100
101   /* Critical Section */
102   write_aux(fd, &size_word, 4);
103   write_aux(fd, it->word, size_word);
104   write_aux(fd, "\0", 1);
105   write_aux(fd, &cnt_word, 4);
106   /* end CS */
107
108   /* out section */
109   e = sem_post(semaphore);
110   error(e, NULL);
111
112   it2 = it;
113   it = it->next;
114   free( it2 );
115 }
116
117 /* In section */
118 e = sem_wait(semaphore);
119 error(e, NULL);
120 /* CS */
121 e = -1;
122 write_aux(fd, &e, 4);
123 /* end CS */
124 /* out section */
125 e = sem_post(semaphore);
126 error(e, NULL);
127
128 close(fd);
129 e = sem_close(semaphore);
130 error(e, NULL);
131
132 /* free the hash table */
133 for( i = 0 ; i < 10007 ; ++i ){
134   np = (H.hash_table[i]).head;
135   while( np != NULL ){
136     np2 = np;
```

```
      np = np->next;
      free(np2);
    }
  }

  free( H.hash_table );

}
```

1.4 get txt

```

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <semaphore.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include "utilities.h"
#include "hash.h"
#include "str_hash.h"
#include "str_list.h"
#include "hash_list.h"
#include "str_ht_list.h"
#include "error_handler.h"

#define HASH_SIZE 10007

/*
*  Function : get_txt
*  ----------------------
*   Gets a directory name and extracts all files with the extention
*   txt
*
*   argv[1] : name of the directory
* argc    : 2
*/
int main( int argc , char **argv ){

  char *dir_name;
  char **file_names;
  int size, i, e;
  hash h;
  pair *p;

  p = (pair *) malloc( sizeof(pair) );
  errorp(p,NULL);


  size = 128;
  ht_make( &h , HASH_SIZE );
  dir_name = argv[1];

  file_names = (char **) malloc( sizeof(char *) * size );
  errorp(file_names, NULL);

  *p = traverse_dir( dir_name , file_names , 0 , &size , &h );
  errorp(p->f, "Error moving through the given directory");


  /* Return names of the found txt files using a pipe in file descriptor 1 */

  /* p->s : number of files to return  */
  e = write_aux(1, &(p->s), 4);
  error(e, NULL);


  /* (p->f)[i] : txt file name */
  for( i = 0; i < p->s; ++i){

    size = strlen((p->f)[i]);

    e = write_aux(1, &size, 4);
    error(e, NULL);

    e = write_aux(1, (p->f)[i], size);
    error(e, NULL);

    e = write_aux(1, "\0 ", 1);
    error(e, NULL);
  }

  return 0;
}
```

1.5 utilities

```
/*
* File:       utilities.c
* Author:     Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
* Description:file that contains the implementation of some useful
*             functions used in frecpalhilos
```

```
* Date:       23 / 11 / 19
*/


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <dirent.h>
#include "utilities.h"
#include "hash.h"


/*
* Function : make_path
* ------------------------------------------------------------
*   given two arrays of chars, uses string.h basic functions to
*   create a new string with the format "path/name"
*
*   path : pointer to the path name
*   name : pointer to the file name

*   returns an array of chars of the form path/name
*/
char* make_path( char* path , char* name ){
  char *ret =
    (char *)malloc( strlen(path) + strlen(name)  + 2 );
    if ( ret == NULL ){
        return NULL;
    }
  if ( strlen(path) == 0 ){
    strcpy( ret , name );
  }
  else{
    strcpy( ret , path );
    ret[strlen(path)] = '/';
    strcpy(ret + strlen(path)+1, name);
  }
  return ret;
}


/*
* Function : traverse_dir
* ------------------------------------------------------------
*   Moves through a directory and finds all the txt files, ignoring the
*   duplicates inodes, using a hash table to do so
*
*   dir_name: name of the directory
*   txt_names: array to save the names of the txts
*   occupied: number of occupied positions in the array
*   size: size of the array
*   h: hash table of ints
*
*   returns the address of the array and the amount of names in it
*/
pair traverse_dir( char* dir_name , char** txt_names , int occupied ,
                   int *size , hash *h){
  DIR* dirp;
  struct stat sb;
    struct dirent* de;
    char* name;
    int e;
    pair get, ret;

    dirp = opendir( dir_name );
    if ( dirp == NULL ){
        ret.f = NULL;
        ret.s = -1;
        return ret;
    }
    while ( de = readdir(dirp) ){
      if ( strcmp(de->d_name,".") == 0 || strcmp(de->d_name,"..") == 0 )
        continue;

      name = make_path( dir_name , de->d_name );
        if ( name == NULL ){
            ret.f = NULL;
            ret.s = -1;
            return ret;
        }
      e = lstat( name , &sb );
      if ( e < 0 ){
            ret.f = NULL;
            ret.s = -1;
            return ret;
        }

      if ( ( sb.st_mode & __S_IFDIR ) == __S_IFDIR ){
```

```
        /* If a directory was found, we traverse it and update values */
        get = traverse_dir( name, txt_names , occupied , size , h);
        if ( get.f == NULL ){
          return get;
        }
        txt_names = get.f;
        occupied = get.s;
      }
      else if ( ( sb.st_mode & __S_IFREG ) == __S_IFREG ){
        if ( strcmp( (de->d_name) + ( strlen( de->d_name ) - 4 ) ,
          ".txt" ) == 0 ){

            /* If the inode is in the hash table, we ignore it */
            if ( ht_find( h , sb.st_ino ) ){
                continue;
            }

            ht_insert( h , sb.st_ino );

        if ( occupied == *size ){
          /* If the maximun size of the array was reached,
             we allocate a new array with double size */
          *size = *size << 1;
          txt_names = realloc( txt_names , sizeof(char*)*(*size) );
          if ( txt_names == NULL ){
                ret.f = NULL;
                ret.s = -1;
                return ret;
             }
          }
          txt_names[occupied++] = name;
        }
      }
    }

    ret.f = txt_names;
    ret.s = occupied;
    return ret;
}


/*
* Function : word_frec_comparator
* ------------------------------------------------------------
*   Function to compare two pair_2 elements, that contain word and rep count
*   for that word
*
*   p : pair_2 number 1 to compare
*   q : pair_2 number 2 to compare
*
*   returns > 0 if q is greater than p, < 0 if p is greater than q and 0 if
*   they are the same
*/
int word_frec_comparator( const void *p , const void *q ){
  pair_2 *l , *r;
  l = ( pair_2 * )p;
  r = ( pair_2 * )q;
  if ( l->c > r->c ) return -1;
  else if ( l->c < r->c ) return 1;
  return ( strcmp( l->w , r->w ) );
}


/*
* Function : int_to_char
* ------------------------------------------------------------
*   stores the first byte of the int to the first position of the
*   array, then right shifts the int bits by 8 and repeats the
*   process now with the second position of the array, then for
*   the third and finally for the fourth
*
*   x : int to store
*   ret : pointer to the array of chars
*/
void int_to_char(int x, char * ret){

  int i;

  for(i=0; i<4; i++){
    ret[i] = (char) (x)&255;
    x >>= 8;
  }

}

/*
* Function : str_to_int
* ------------------------------------------------------------
*   saves the bits from each character in the int, to do so
*   it saves the bits from the fourth char, then left shifts 8 bits
*   and saves the bits from the third, then left shifts 8 bits and
*   repeats for the second and first char
```

```
*
*   c : pointer to the array of chars
*
*   returns an int that has the bits of the array
*/
int str_to_int( char* c  ){
  int x, i;
  x = 0;
  for( i = 3 ; i >= 0 ; i-- ){
    x = x << 8;
    x = x | (unsigned char)(c[i]);
  }
  return x;
}

/*
* Function : write_aux
* ------------------------------------------------------------
*   makes calls to syscall read until all len is read or an
*   error occurs
*
*   fd : file descriptor of the file to write
*   len : ammount of chars to write from buf
*   buf : array of chars to write from
*
*   returns 0 in case of success or -1 in case of failure
*/
int write_aux(int fd, unsigned char * buf,  int len){
  int e , len2 = 0;
  while(len2 < len){
    e = write(fd, buf + len2, len-len2);
    if ( e <= 0 ) return -1;
    len2 += e;
  }
  return len;
}

/*
* Function : read_aux
* ------------------------------------------------------------
*   makes calls to syscall read untile l chars are read and
*   stored in buf or an error occurs
*
*   fd : file descriptor of the file to read from
*   len : ammount of chars to read
*   buf : array of chars to write
*
*   returns 0 in case of success or -1 in case of failure
*/
int read_aux( int fd , unsigned char * buf , int len ){
  int l2, e;
  l2 = 0;
  while ( l2 < len ){
    e = read( fd , buf + l2 , len - l2 );
    if ( e <= 0 ) return e;
    l2 = l2 + e;
  }
  return len;
}
```

```
/*
* File:     utilities.h
* Author:     Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
* Description:  file that contains the signature of some useful
*         functions used in frecpalhilos
* Date:     23 / 11 / 19
*/


#include "hash.h"

#ifndef _UTILITIES_H
#define _UTILITIES_H

typedef struct{
  char **f;
  int s;
} pair;

typedef struct {
  char * w;
  int c;
} pair_2;

/*
* Function : make_path
* ------------------------------------------------------------
*   given two arrays of chars, creates a path of the form "path/name"
*
*   path : pointer to the path name
*   name : pointer to the file name

```

```
*   returns an array of chars of the form path/name
*/
char* make_path( char* path , char* name );


/*
* Function : traverse_dir
* ------------------------------------------------------
*   Moves through a directory and finds all the txt files, ignoring the
*   duplicates inodes
*
*   dir_name: name of the directory
*   txt_names: array to save the names of the txts
*   occupied: number of occupied positions in the array
*   size: size of the array
*   h: hash table of ints
*
*   returns the address of the array and the amount of names in it
*/
pair traverse_dir( char* dir_name , char** txt_names , int occupied ,
  int *size , hash *h );


/*
* Function : word_frec_comparator
* ------------------------------------------------------
*   Function to compare two pair_2 elements, that contain word and rep count
*   for that word
*
*   p : pair_2 number 1 to compare
*   q : pair_2 number 2 to compare
*
*   returns > 0 if q is greater than p, < 0 if p is greater than q and 0 if
*   they are the same
*/
int word_frec_comparator( const void *p , const void *q );

#endif
```

1.6 hash list

```
/*
*  File:        hash_list.c
*  Author:      Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
*  Description: file that contains the implementation of the functions
*               used by the hash list
*  Date:        23 / 11 / 19
*/

#include <stdlib.h>
#include "hash_list.h"


/*
*  Function : hl_insert
*  ----------------------
*   Inserts an int into a list, by inserting at the head of the list
*
*   l : pointer to a list
* k : integer to insert
*
* returns 0 on success and -1 on failure
*/
int hl_insert( hash_list *l , int k ){
  hl_node *aux = malloc( sizeof(hl_node) );
  if ( aux == NULL ) return -1;
  aux->key = k;
  aux->next = l->head;
  l->head = aux;
  return 0;
}


/*
*  Function : hl_make
*  ----------------------
*   Initializes the values of the list
*
*   l : pointer to a list
*/
void hl_make( hash_list *l ){
  l->size = 0;
  l->head = NULL;
}


/*
*  Function : hl_find
*  ----------------------
*   Looks for an element in the list, by moving through the list
*
*   l : pointer to a list
* k : integer to find
```

```
*
* returns 1 on success and 0 on failure
*/
int hl_find( hash_list *l , int k ){

  hl_node *aux = l->head;
  while ( aux != NULL ){
    if ( aux->key == k ) return 1;
    aux = aux->next;
  }
  return 0;
}
```

```
/*
*  File:        hash_list.h
*  Author:      Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
*  Description: file that contains the signatures of the functions and
*               structures used by the hash list
*  Date:        23 / 11 / 19
*/

#ifndef _HASH_LIST_H
#define _HASH_LIST_H

typedef struct hl_node {
  struct hl_node *next;
  int key;
} hl_node;

typedef struct {
  int size;
  hl_node *head;
} hash_list;


/*
*  Function : hl_insert
*  ---------------------
*   Inserts an int into a list
*
*   l : pointer to a list
* k : integer to insert
*
* returns 0 on success and -1 on failure
*/
int hl_insert( hash_list *l , int key );


/*
*  Function : hl_make
*  ---------------------
*   Initializes the values of the list
*
*   l : pointer to a list
*/
void hl_make( hash_list *l );


/*
*  Function : hl_find
*  ---------------------
*   Looks for an element in the list
*
*   l : pointer to a list
* k : integer to find
*
* returns 1 on success and 0 on failure
*/
int hl_find( hash_list *l , int key );

#endif
```

1.7 str list

```
/*
* File:     str_list.c
* Author:     Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
* Description:  file that contains the implementation of some functions
*         of a string list
* Date:     23 / 11 / 19
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "str_list.h"

/*
* Function: insert
* --------------------
*   Inserts the given word in the first position of the given list by
* moving its pointers and updates the size of the list
*
```

```
*   l: pointer to a list
* n: pointer to a word
*/
int str_list_insert( str_list *l , char* w )
{
  str_node *n;
  n = malloc( sizeof( str_node ) );
  if ( n == NULL ) return -1;
  n->word = w;
  l->size = l->size + 1;
  n->next = l->head;
  l->head = n;
  return 0;
}


/*
* Function: make_list
* -------------------------
*   Gets the pointer to the address of a memory block allocated for a list
* and initializes its values head to NULL and size to 0
*
*   l: pointer to a list
*/
void make_str_list( str_list *l )
{
  l->size = 0;
  l->head = NULL;
}
```

```
/*
* File:     str_list.h
* Author:     Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
* Description:  file that contains the signature of some functions
*         and structures of a string list
* Date:     23 / 11 / 19
*/


#ifndef _STR_LIST_H
#define _STR_LIST_H

typedef struct str_node {
  struct str_node *next;
  char *word;
  int reps;
} str_node;

typedef struct {
  int size;
  str_node *head;
} str_list;

/*
* Function: str_list_insert
* -------------------------
* Inserts the given wotd in the given list
*
*   l: pointer to a list
* n: pointer to a word
*/
int str_list_insert( str_list *l , char* w );


/*
* Function: make_list
* -------------------------
*   Gets the pointer to the address of a memory block allocated for a list
* and initializes its values
*
*   l: pointer to a list
*/
void make_str_list( str_list *l );


#endif
```

1.8 str ht list

```
/*
*  File:        str_ht_list.c
*  Author:      Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
*  Description: file that contains the implementation of the functions
*               used by the string hash list
*  Date:        23 / 11 / 19
*/



#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "str_ht_list.h"
```

```

/*
* Function: str_ht_list_insert
* --------------------
*   Inserts the given node in the first position of the given list by
* moving its pointers and updates the size of the list
*
*   l: pointer to a list
* n: pointer to a word
*
* returns 0 on success and -1 on failure
*/
int str_ht_list_insert( str_ht_list *l , char *w , int k )
{
  str_ht_list_node *n;

  n = malloc( sizeof( str_ht_list_node ) );
  if ( n == NULL ) return -1;
  n->reps = k;
  n->word = w;
  l->size = l->size + 1;
  n->next = l->head;
  l->head = n;
  return 0;
}


/*
* Function: str_ht_list_make_list
* --------------------------
*   Gets the pointer to the address of a memory block allocated for a list
* and initializes its values head to NULL and size to 0
*
*   l: pointer to a list
*/
void str_ht_list_make_list( str_ht_list *l )
{
  l->size = 0;
  l->head = NULL;
}

/*
* Function: str_ht_list_find
* --------------------------
* Looks for the given word in the list by moving through its nodes,
* if it founds it, it adds k to the repetition counter for that
* node and returns the old value. Otherwise if the word is not in the list
* it returns 0
*
*   l: pointer lo a list
* c: pointer to an array of char
* k: integer to add to the rep count
*
* returns: 0 if the words is not in the list or an int that represents
*      the number of times that word appeared in the list
*/
int str_ht_list_find( str_ht_list *l , char *c , int k )
{
  str_ht_list_node *np = l->head;
  while ( np != NULL ){
    /* If the word is in the list, it updates the repetition count and
       returns it */
    if ( strcmp( c , np->word ) == 0 ){
      np->reps = np->reps + k;
      return np->reps - k ;
    }

    np = np->next;
  }
  return 0;
}
```

```

/*
*  File:        str_ht_list.h
*  Author:      Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
*  Description: file that contains the signatures of the functions
*               used by the string hash list
*  Date:        23 / 11 / 19
*/


#ifndef _LIST_H
#define _LIST_H

typedef struct str_ht_list_node {
  struct str_ht_list_node *next;
  char *word;
  int reps;
} str_ht_list_node;

typedef struct {
  int size;
```

```
  str_ht_list_node *head;
} str_ht_list;

/*
* Function: str_ht_list_insert
* --------------------------
* Inserts the given word in the given list
*
*   l: pointer to a list
* n: pointer to a word
*
* returns 0 on success and -1 on failure
*/
int str_ht_list_insert( str_ht_list *l , char *n , int k );


/*
* Function: str_ht_list_make_list
* --------------------------
*   Gets the pointer to the address of a memory block allocated for a list
* and initializes its values
*
*   l: pointer to a list
*/
void str_ht_list_make_list( str_ht_list *l );

/*
* Function: str_ht_list_find
* --------------------------
*   Looks for the given word in the given list, if it finds it, it updates
* the number of repetitions of that word by k and returns the old number of
* repetitions for that word. If it doesnt find it, returns 0.
*
*   l: pointer lo a list
* c: pointer to an array of char
* k: ammount to add to the rep count of the words
*
* returns: 0 if the words is not in the list or an int that represents
*      the number of times that word appeared before updating
*/
int str_ht_list_find( str_ht_list *l , char *c , int k );

#endif
```

1.9 hash

```
/*
*  File:        hash.c
*  Author:      Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
*  Description: file that contains the implementation of the functions
*               used by the hash
*  Date:        23 / 11 / 19
*/


#include <stdlib.h>
#include <stdio.h>
#include "hash.h"
#include "hash_list.h"


/*
*  Function : ht_make
*  ----------------------
*   Initializes the values of the hash table, allocates space for the lists
*   and initializes the values of the lists
*
*   h : pointer to a hash table
* size : size of the table
*
* returns 0 on success and -1 on failure
*/
int ht_make( hash *h , int size){
  int i, e;
  h->size = size;
  h->hash_table = malloc( sizeof(hash)*size );
  if ( h->hash_table == NULL ) return -1;
  for ( i = 0 ; i < size ; i++ ){
    hl_make( &(h->hash_table[i]) );
  }
  return 0;
}


/*
*  Function : ht_find
*  ----------------------
*   Looks for the given key in the hash by looking in the list indexed by the
* value of the hashing function
*
*   h : pointer to a hash table
* k : integer to look in the table
*
```

```
* returns 1 if it was found or 0 if it wasnt found
*/
int ht_find( hash *h , int k ){
  return hl_find( &( h->hash_table[ hash_function(k,h->size) ] ) , k );
}


/*
*  Function : ht_insert
*  ---------------------
*   Inserts the given key in the hash by inserting in the list indexed by the
* value of the hashing function
*
*   h : pointer to a hash table
* k : integer to insert in the table
*
* returns 0 on success or -1 on failure
*/
int ht_insert( hash *h , int k ){
  int e;
  e = hl_insert( &( h->hash_table[ hash_function(k,h->size) ] ) , k );
  return e;
}


/*
*  Function : hash_function
*  ---------------------
*   returns the value of the hash function of the given integer,
* by using mod with a high prime number to increase effectiveness
*
*   k : integer to calculate the hash function of
* mod : mod to use in the function
*
* returns the value of the funcion
*/
int hash_function( int k , int mod ){
  return k%mod;
}
```

```
/*
*  File:        hash.h
*  Author:      Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
*  Description: file that contains the signatures of the functions and
*               structures used by the hash
*  Date:        23 / 11 / 19
*/


#include "hash_list.h"

#ifndef _HASH_H
#define _HASH_H

typedef struct {
  hash_list *hash_table;
  int size;
} hash;


/*
*  Function : ht_make
*  ---------------------
*   Initializes the values of the hash table
*
*   h : pointer to a hash table
* size : size of the table
*
* returns 0 on success and -1 on failure
*/
int ht_make( hash *h , int size);


/*
*  Function : ht_find
*  ---------------------
*   Looks for the given key in the hash
*
*   h : pointer to a hash table
* k : integer to look in the table
*
* returns 1 if it was found or 0 if it wasnt found
*/
int ht_find( hash *h , int k );


/*
*  Function : ht_insert
*  ---------------------
*   Inserts the given key in the hash
*
*   h : pointer to a hash table
```

```
* k : integer to insert in the table
*
* returns 0 on success or -1 on failure
*/
int ht_insert( hash * h , int k );

#endif
```

1.10 str hash

```
/*
*  File:        str_hash.c
*  Author:      Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
*  Description: file that contains the implementation of the functions
*               used by the string hash
*  Date:        23 / 11 / 19
*/


#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "str_ht_list.h"
#include "str_hash.h"

#define MOD 10007
#define PRIME 33

int prime_pow[101];


/*
*  Function : set_str_hash
*  ----------------------
*   Initializes the values of an array of prime powers used
* in the hash function
*/
void set_str_hash( ){
  int i;
  int cont;
  cont = 1;
  for( i = 0 ; i < 101 ; i++ ){
    prime_pow[i] = cont;
    cont = (cont*PRIME)%MOD;
  }
}


/*
*  Function : str_ht_make
*  ----------------------
*   Initializes the values of the hash table, allocates space for the lists
*   and initializes the values of the lists
*
*   h : pointer to a string hash table
*
* returns 0 on success and -1 on failure
*/
int str_ht_make( str_hash *h ){
  int i, e, size;
  size = MOD;
  set_str_hash();
  h->size = size;
  h->hash_table = malloc( sizeof(str_ht_list)*size );
  if ( h->hash_table == NULL ) return -1;
  for ( i = 0 ; i < size ; i++ ){
    str_ht_list_make_list( &(h->hash_table[i]) );
  }
  return 0;
}


/*
*  Function : str_ht_find
*  ----------------------
*   Looks for the given key in the string hash by looking in the list indexed
* by the value of the hashing function, if found adds reps to the rep value
* of that word
*
*   h : pointer to a string hash table
* w : word to look for
* reps : amount to add to the rep count of the word
*
* returns the old amount of times that the word appears
*/
int str_ht_find( str_hash *h , char *w , int reps){
  return str_ht_list_find( &( h->hash_table[ str_hash_function(w) ] ) ,
                    w , reps );
}

/*
*  Function : str_ht_insert
*  ----------------------
```

```
*   Inserts the given key in the hash by inserting in the list indexed by the
* value of the hashing function with the given number of reps
*
*   h : pointer to a hash table
* k : integer to insert in the table
* reps : number of reps of the word
*
* returns 0 on success or -1 on failure
*/
int str_ht_insert( str_hash *h , char *w , int reps ){
  return str_ht_list_insert( &( h->hash_table[ str_hash_function(w) ] ) ,
                 w , reps );
}


/*
*  Function : str_hash_function
*  ----------------------
*   returns the value of the hash function of the given word,
* by using multiplying the i-th letter with a prime raised to the power of
* i and adding those values, taking its module by another prime
*
*   w : word to get the hash function of
*
* returns the value of the funcion
*/
int str_hash_function( char *w ){
  int key, i, len;
  key = 0;
  len = strlen( w );
  for( i = 0 ; i < len ; i++ ){
    key = ( key + ( w[i] - 'a' + 1 )* prime_pow[i] ) % MOD;
  }
  if ( key < 0 ) key = -key;
  return key;
}
```

```
/*
*  File:        str_hash.h
*  Author:      Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
*  Description: file that contains the signature of the functions
*               used by the string hash
*  Date:        23 / 11 / 19
*/


```

```
#include "str_ht_list.h"

#ifndef _STR_HASH_H
#define _STR_HASH_H

typedef struct {
  str_ht_list *hash_table;
  int size;
} str_hash;


/*
*  Function : str_ht_make
*  ----------------------
*   Initializes the values of the hash table
*
*   h : pointer to a string hash table
*
* returns 0 on success and -1 on failure
*/
int str_ht_make( str_hash *h );


/*
*  Function : str_ht_find
*  ----------------------
*   Looks for the given key in the string hash
*
*   h : pointer to a string hash table
* w : word to look for
* reps : amount to add to the rep count of the word
*
* returns the old amount of times that the word appears
*/
int str_ht_find( str_hash *h , char *w , int k );


/*
*  Function : str_ht_insert
*  ----------------------
*   Inserts the given key in the hash
*
*   h : pointer to a hash table
* k : integer to insert in the table
* reps : number of reps of the word
*
```

```
* returns 0 on success or -1 on failure
*/
int str_ht_insert( str_hash * h , char *w , int k );

#endif
```

1.11 error handler

```
/*
* File:       error_handler.c
* Author:     Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
* Description:file that contains the implementation of some useful functions
*       to manage errors
* Date:       23 / 11 / 19
*/
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>

/*
* Function : error
* ------------------------------------------------------------
*   given an integer, print error information if the integer is negative,
* the information printed can be given or by default.
*
* e: error value
*   str: error information
*/
void error(int e, char * str){
  if(e<0){
    if( str == NULL ) perror("Error");
    else perror(str);
    exit(-1);
  }
}

/*
* Function : errorp
* ------------------------------------------------------------
*   given a pointer, print error information if the pointer is NULL,
* the information printed can be given or by default.
*
* e: error value
*   str: error information
*/
void errorp(void * e, char * str){
  if( e == NULL ){
    if( str == NULL ) perror("Error");
    else perror(str);
    exit(-1);
  }
}
```

```
/*
* File:       error_handler.h
* Author:     Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
* Description:file that contains the signature of some useful functions
*       to manage errors
* Date:       23 / 11 / 19
*/

#ifndef _ERROR_HANDLER_
#define _ERROR_HANDLER_

/*
* Function : error
* ------------------------------------------------------------
*   given an integer, print error information if the integer is negative,
* the information printed can be given or by default.
*
* e: error value
*   str: error information
*/
void error(int e, char * str);


/*
* Function : errorp
* ------------------------------------------------------------
*   given a pointer, print error information if the pointer is NULL,
* the information printed can be given or by default.
*
* e: error value
*   str: error information
*/
void errorp(void * e, char * str);

#endif
```

1.12 counter thread

```
/*
* File:     counter_thread.h
* Author:     Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
```

```
4  * Description:  file that contains the signature of some structures
5  *         used for a thread function that counts words
6  * Date:     23 / 11 / 19
7  */
8
9
10 #include "utilities.h"
11
12 #ifndef _COUNTER_THREAD_
13 #define _COUNTER_THREAD_
14
15 typedef struct {
16   int n, MOD, begin;
17   char ** file;
18 } input;
19
20 typedef struct {
21   pair_2 * cnt;
22   int size;
23 } ret;
24
25 #endif
```