

# 1 Grupo 20

## 1.1 frecpalhilo

```
1  /*
2  * File:      frecpalhilo.c
3  * Author:    Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4  * Description: file that contains the implementation of the frecpal with
5  *             threads main
6  * Date:      23 / 11 / 19
7  */
8
9  #include <stdlib.h>
10 #include <string.h>
11 #include <stdio.h>
12 #include <pthread.h>
13 #include <unistd.h>
14 #include "utilities.h"
15 #include "hash.h"
16 #include "str_hash.h"
17 #include "str_list.h"
18 #include "hash_list.h"
19 #include "str_ht_list.h"
20 #include "counter_thread.h"
21 #include "error_handler.h"
22
23 #define MAX_WORD_LEN 100
24 #define HASH_SIZE 10007
25
26 pthread_mutex_t mtx;
27 pthread_t end = 0;
28
29 /*
30 * Function : get_txt
31 * -----
32 * Gets a directory name and extracts all files with the extention
33 * txt
34 *
35 * arg : name of the directory
36 */
37 void* get_txt( void *arg ){
38     char *dir_name;
39     char **file_names;
40     int size, i;
41     hash h;
```

```
42     pair *p;
43
44     p = malloc( sizeof(pair) );
45     size = 128;
46     ht_make( &h , HASH_SIZE );
47     dir_name = (char *)arg;
48     file_names = malloc( sizeof(char *) * size );
49
50     *p = traverse_dir( dir_name , file_names , 0 , &size , &h );
51     errorp( p->f , "Error_moving_through_the_given_directory.\n");
52
53     pthread_exit( p );
54 }
55
56
57 /*
58 * Function : count_words
59 * -----
60 * Gets some file names, and counts the words in them
61 *
62 * arg : input type, contains file names, starting index, size of
63 *       array of names and the offset to his files
64 */
65 void* count_words( void *arg ){
66     input *inp;
67     int i, begin , n , mod, aux, e;
68     FILE *fp;
69     str_list l;
70     str_hash H;
71     pair_2 *cnt;
72     char word[MAX_WORD_LEN];
73     int id = 0;
74     char **file_names;
75     char *aux_w;
76     str_node *it, *it2;
77     int ind;
78     ret *retval;
79     str_ht_list_node *np, *np2;
80
81     inp = ( input * )arg;
82     mod = inp->MOD;
83     begin = inp->begin;
84     n = inp->n;
85     file_names = inp->file;
86     e = str_ht_make( &H );
```

```
87 error( e , "Error allocating memory.\n");
88
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 ind = 0;
119 it = l.head;
120
121 while( it != NULL ){
122     cnt[ind].w = it->word;
123     cnt[ind].c = str_ht_find( &H , it->word , 0 );
124     id++;
125     it2 = it;
126     it = it->next;
127     free( it2 );
128 }
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");
181     return -1;
182 }
183
184 /* This thread will look for the txts and return them */
185 e = pthread_create( &txt_thread , NULL , get_txt , argv[2] );
186 error( e , "Error creating txt thread.\n");
187
188
189 e = pthread_join( txt_thread , (void **)&p_aux );
190 error( e , "Error joining txt thread.\n");
191
192 p_aux = (pair *)p_aux;
193 n_txt = p_aux->s;
194 txt_names = p_aux->f;
195 free(p_aux);
196
197 pthread_mutex_init(&mtx, NULL);
198
199 /* If the number of threads given is greater than the number of txt files
200 we will only use 1 thread for file, so the number of threads will become
201 smaller */
202 if ( n_threads > n_txt ) n_threads = n_txt;
203
204 count_threads = malloc( sizeof(pthread_t) * n_threads );
205 errorp( count_threads , "Error allocating memory.\n");
206
207
208 for( i = 0 ; i < n_threads ; i++ ){
209     inp = malloc( sizeof(input) );
210     errorp( inp , "Error allocating memory.\n");
211
212     inp->n = n_txt;
213     inp->MOD = n_threads;
214     inp->begin = i;
215     inp->file = txt_names;
216
217     /* Here we create the counter threads and assing them the corresponding
218     txts */
219     e = pthread_create( &count_threads[i] , NULL , count_words , (void *)inp
220     );
221     error( e , "Error creating count words thread.\n");
```

```
221 }
222
223 /* Here we allocate space for the counter threads output */
224 count_rets = malloc( sizeof(ret*) * n_threads );
225 errorp( count_rets , "Error allocating memory.\n");
226
227
228 i = 0;
229
230 while(i < n_threads){
231     while(!end);
232     thr_id = end;
233     end = 0;
234     e = pthread_join( thr_id ,(void **)&count_rets[i] );
235     error( e , "Error joining count words thread.\n");
236
237     count_rets[i] = (ret*)count_rets[i];
238     i++;
239     pthread_mutex_unlock(&mtx);
240 }
241
242
243 e = str_ht_make( &h );
244 error( e , "Error allocating memory.\n");
245
246 make_str_list( &l );
247
248 /* In this loop we will take all the words given by the counter threads
249 and store them in a hash table where we will update their frequency and in
250 a list so we easily know how many and what words we have */
251 for( i = 0 ; i < n_threads ; i++ ){
252     for( j = 0 ; j < count_rets[i]->size ; j++ ){
253         /* If the word already is in the hash, update its rep count */
254         cont = str_ht_find( &h , count_rets[i]->cnt[j].w , count_rets[i]->cnt[
255             j].c);
256
257         /* Else insert it in the hash and in the list */
258         if ( cont == 0 ){
259             e = str_ht_insert( &h , count_rets[i]->cnt[j].w , count_rets[i]->cnt
260             [j].c);
261             error( e , "Error allocating memory.\n");
262
263             e = str_list_insert( &l , count_rets[i]->cnt[j].w );
264             error( e , "Error allocating memory.\n");
```

```
264     }
265 }
266 }
267
268 free( count_threads );
269 for( i = 0 ; i < n_txt ; i++ ){
270     free( txt_names[i] );
271 }
272 free( txt_names );
273
274 n_words = l.size;
275 words = malloc( sizeof(pair_2)*n_words );
276 errorp( words , "Error allocating memory.\n");
277
278
279 ind = 0;
280 it = l.head;
281 /* Here we pass the words with their rep count from the hash and list,
282 to an array so we can sort it using c qsort, and free the list nodes */
283 while( it != NULL ){
284     words[ ind ].w = it->word;
285     words[ ind ].c = str_ht_find( &h , it->word , 0 );
286     ind++;
287     it2 = it;
288     it = it->next;
289     free(it2);
290 }
291
292 /* Free the hash table space */
293 for( i = 0 ; i < 10007 ; i++ ){
294     np = (h.hash_table[i]).head;
295     while( np != NULL ){
296         np2 = np;
297         np = np->next;
298         free(np2);
299     }
300 }
301 free( h.hash_table );
302
303 /* Sort the words with a custom comparator, so we get the expected order
304 */
305 qsort( words , n_words , sizeof( pair_2 ) , word_freq_comparator );
306
307 for( i = 0 ; i < n_words ; i++ ){
308     printf("%s %d\n", words[i].w , words[i].c );
309 }
```

```
308 }
309
310 return 0;
311 }
```

## 1.2 frecpalproc

```
1  /*
2  * File:         frecpalproc.c
3  * Author:       Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4  * Description:  file that contains the implementation of the frecpal with
5  *              processes main
6  * Date:         23 / 11 / 19
7  */
8
9  #include <stdlib.h>
10 #include <string.h>
11 #include <stdio.h>
12 #include <unistd.h>
13 #include <semaphore.h>
14 #include <sys/wait.h>
15 #include <sys/types.h>
16 #include <fcntl.h>
17 #include <sys/stat.h>
18 #include "utilities.h"
19 #include "hash.h"
20 #include "str_hash.h"
21 #include "str_list.h"
22 #include "hash_list.h"
23 #include "str_ht_list.h"
24 #include "error_handler.h"
25
26 #define MAX_WORD_LEN 100
27 #define HASH_SIZE 10007
28
29
30 int main( int argc , char **argv ){
31
32     int n_proc, n_txt, e, i, j, aux, cnt, status,
33         cont, ind, n_words, fd[2], word_len, fd_fifo;
34
35     char **txt_names;
36     char *** txt_of_proc;
37     char * word;
38     sem_t *semaphore;
39     str_hash h;
```

```
40 str_list l;  
41 str_node *it, *it2;  
42 str_ht_list_node *np, *np2;  
43 pair_2 *words;  
44  
45 if ( argc != 3 ){  
46     printf("Error_in_the_given_input.\n");  
47     return -1;  
48 }  
49  
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  
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 /* father continue */  
81 close(fd[1]);  
82  
83  
84
```

```
85  
86 /* Get txt files names from child process via pipe */  
87  
88 e = read_aux(fd[0], &n_txt, 4);  
89  
90 error(e, NULL);  
91  
92 txt_names = (char **) malloc(sizeof(char *) * n_txt);  
93  
94 errorp(txt_names, NULL);  
95  
96 for( i = 0; i < n_txt; ++i){  
97  
98     e = read_aux(fd[0], &word_len, 4);  
99     error(e, NULL);  
100  
101     txt_names[i] = (char *) malloc(sizeof(char) * (word_len + 1));  
102     errorp(txt_names[i], NULL);  
103  
104     e = read_aux(fd[0], txt_names[i], word_len + 1);  
105     error(e, NULL);  
106  
107 }  
108  
109 close(fd[0]);  
110 /* Wait the child process */  
111 e = wait(&e);  
112  
113 error(e, "Error_in_child_process_get_txt");  
114  
115 /* Create named pipe for reading the work of the counter processes */  
116 unlink("myfifo");  
117 e = mkfifo("myfifo", 0666);  
118 error(e, NULL);  
119  
120 /*fd_fifo = open("myfifo", O_RDONLY);*/  
121  
122 /* Create named semaphore for counter processes coordination while writing  
    in named pipe */  
123 sem_unlink("mySmph");  
124 semaphore = sem_open("mySmph", O_CREAT, 0666, 1);  
125 if (semaphore == SEM_FAILED) {  
126     perror("sem_open(3)_failed");  
127     exit(-1);  
128 }
```

```
129
130     e = sem_close(semaphore);
131     error(e, NULL);
132     /* If the number of processes given is greater than the number of txt
133        files
134        we will only use 1 thread for file, so the number of threads will become
135        smaller */
136     if ( n_proc > n_txt ) n_proc = n_txt;
137
138     /* We store the files names of the txt's that every counter process in
139        their corresponding array */
140
141     txt_of_proc = (char **) malloc(sizeof(char **) *n_proc);
142     errorp(txt_of_proc, NULL);
143
144     for( i = 0 ; i < n_proc ; ++i ){
145
146         txt_of_proc[i] = (char **) malloc(sizeof(char *) * (n_txt / n_proc + (
147             int)((n_txt % n_proc) > 0) + 1));
148         errorp(txt_of_proc[i], NULL);
149
150         txt_of_proc[i][n_txt / n_proc + ((n_txt % n_proc) > 0)] = (char *) NULL;
151     }
152
153     for( i = 0 ; i < n_txt ; ++i ){
154
155         txt_of_proc[i % n_proc][i / n_proc] = (char *) malloc(sizeof(char) * (
156             strlen(txt_names[i]) + 1));
157         errorp(txt_of_proc[i % n_proc][i / n_proc], NULL);
158         strcpy(txt_of_proc[i % n_proc][i / n_proc], txt_names[i]);
159     }
160
161     for( i = 0 ; i < n_proc ; ++i ){
162
163         /* Here we create the counter processes and assing them the
164            corresponding
165            txts */
166
167         e = fork();
168
169         error(e, "Error_creating_count_words_process");
170
171         if( e == 0){
```

```
169
170         e = execv("count_words", txt_of_proc[i]);
171
172         error(e, "Error_in_execution_of_\"count_words\"");
173
174     }
175 }
176
177 fd_fifo = open("myfifo", O_RDONLY);
178
179 e = str_ht_make( &h );
180 error(e, "Error_allocating_memory");
181
182 make_str_list( &l );
183
184 i = 0;
185
186 /* In this loop we will take all the words given by the counter processes
187    and store them in a hash table where we will update their frequency and in
188    a list so we easily know how many and what words we have */
189 while( i < n_proc ){
190
191     e = read_aux(fd_fifo, &aux, 4);
192
193     if(!e) continue;
194
195     if(aux == -1){
196         i++;
197         continue;
198     }
199
200     word = (char *) malloc(sizeof(char) * (aux + 1));
201
202     read_aux(fd_fifo, word, aux + 1);
203
204     read_aux(fd_fifo, &cnt, 4);
205
206     /* If the word already is in the hash, update its rep count */
207     cnt = str_ht_find( &h , word , cnt);
208
209     /* Else insert it in the hash and in the list */
210     if ( cnt == 0 ){
211
212         e = str_ht_insert( &h , word , cnt);
213
```

```

214     error(e, "Error_allocating_memory");
215
216     e = str_list_insert( &l , word );
217     error(e, "Error_allocating_memory");
218
219 }
220 }
221
222 for( i = 0; i < n_proc; ++i ){
223     e = wait(&status);
224     error(e, NULL);
225     if( WIFEXITED(status) ) error(WEXITSTATUS(status), NULL);
226 }
227
228 e = sem_unlink("mySmph");
229 error(e, NULL);
230
231 close(fd_fifo);
232 unlink("myfifo");
233
234 for( i = 0 ; i < n_txt ; ++i ){
235     free( txt_names[i] );
236 }
237 free( txt_names );
238
239 n_words = l.size;
240 words = malloc( sizeof(pair_2)*n_words );
241 errorp(words, "Error_allocating_memory");
242
243 ind = 0;
244 it = l.head;
245 /* Here we pass the words with their rep count from the hash and list,
246 to an array so we can sort it using c qsort, and free the list nodes */
247 while( it != NULL ){
248     words[ ind ].w = it->word;
249     words[ ind ].c = str_ht_find( &h , it->word , 0 );
250     ind++;
251     it2 = it;
252     it = it->next;
253     free(it2);
254 }
255
256 /* Free the hash table space */
257 for( i = 0 ; i < 10007 ; i++ ){
258     np = (h.hash_table[i]).head;

```

```

259     while( np != NULL ){
260         np2 = np;
261         np = np->next;
262         free(np2);
263     }
264 }
265 free( h.hash_table );
266
267 /* Sort the words with a custom comparator, so we get the expected order
268 */
269 qsort( words , n_words , sizeof( pair_2 ) , word_freq_comparator );
270
271 for( i = 0 ; i < n_words ; i++ ){
272     printf("%s□%d\n", words[i].w, words[i].c );
273 }
274
275 return 0;
276 }

```

### 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     it = l.head;
82
83     fd = open("myfifo", O_WRONLY);
84
85     semaphore = sem_open("mySmph", O_RDWR);
86     if (semaphore == SEM_FAILED) {
87         perror("sem_open(3) failed");
88         exit(-1);
89     }
90
91     while( it != NULL ){
92
93         size_word = strlen(it->word);
94         cnt_word = str_ht_find( &H , it->word , 0 );
95
96         /* In section */
97         e = sem_wait(semaphore);
98         error(e, NULL);
99
100         /* Critical Section */
101         write_aux(fd, &size_word, 4);
102         write_aux(fd, it->word, size_word);
103         write_aux(fd, "\0", 1);
104         write_aux(fd, &cnt_word, 4);
105         /* end CS */
106
107         /* out section */
108         e = sem_post(semaphore);
109         error(e, NULL);
110
111         it2 = it;
112         it = it->next;
113         free( it2 );
114     }
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;
137          np = np->next;
138          free(np2);
139      }
140  }
141
142  free( H.hash_table );
143
144  }

```

## 1.4 get txt

```

1
2  #include <stdlib.h>
3  #include <string.h>
4  #include <stdio.h>
5  #include <unistd.h>
6  #include <semaphore.h>
7  #include <sys/types.h>
8  #include <fcntl.h>
9  #include <sys/stat.h>
10 #include "utilities.h"
11 #include "hash.h"
12 #include "str_hash.h"
13 #include "str_list.h"
14 #include "hash_list.h"

```

```

15 #include "str_ht_list.h"
16 #include "error_handler.h"
17
18 #define HASH_SIZE 10007
19
20 /*
21  * Function : get_txt
22  * -----
23  * Gets a directory name and extracts all files with the extention
24  * txt
25  *
26  * argv[1] : name of the directory
27  * argc    : 2
28  */
29 int main( int argc , char **argv ){
30
31     char *dir_name;
32     char **file_names;
33     int size, i, e;
34     hash h;
35     pair *p;
36
37     p = (pair *) malloc( sizeof(pair) );
38     errorp(p,NULL);
39
40
41     size = 128;
42     ht_make( &h , HASH_SIZE );
43     dir_name = argv[1];
44
45     file_names = (char **) malloc( sizeof(char *) * size );
46     errorp(file_names, NULL);
47
48     *p = traverse_dir( dir_name , file_names , 0 , &size , &h );
49     errorp(p->f, "Error_moving_through_the_given_directory");
50
51
52     /* Return names of the found txt files using a pipe in file descriptor 1
53        */
54
55     /* p->s : number of files to return */
56     e = write_aux(1, &(p->s), 4);
57     error(e, NULL);
58

```

```

59  /* (p->f)[i] : txt file name */
60  for( i = 0; i < p->s; ++i){
61
62      size = strlen((p->f)[i]);
63
64      e = write_aux(1, &size, 4);
65      error(e, NULL);
66      /*e = write_aux(2, (p->f)[i], size);*/
67
68      e = write_aux(1, (p->f)[i], size);
69      error(e, NULL);
70      /*e = write_aux(2, "salio", 5);*/
71
72
73      e = write_aux(1, "\0", 1);
74      error(e, NULL);
75  }
76
77  return 0;
78  }

```

## 1.5 utilities

```

1  /*
2  * File:      utilities.c
3  * Author:    Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4  * Description:file that contains the implementation of some useful
5  *             functions used in frecpalhilos
6  * Date:      23 / 11 / 19
7  */
8
9
10 #include <stdio.h>
11 #include <string.h>
12 #include <stdlib.h>
13 #include <sys/types.h>
14 #include <sys/stat.h>
15 #include <unistd.h>
16 #include <fcntl.h>
17 #include <dirent.h>
18 #include "utilities.h"
19 #include "hash.h"
20
21
22 /*
23 * Function : make_path

```

```

24 * -----
25 *   given two arrays of chars, uses string.h basic functions to
26 *   create a new string with the format "path/name"
27 *
28 *   path : pointer to the path name
29 *   name : pointer to the file name
30
31 *   returns an array of chars of the form path/name
32 */
33 char* make_path( char* path , char* name ){
34     char *ret =
35         (char *)malloc( strlen(path) + strlen(name)  + 2 );
36     if ( ret == NULL ){
37         return NULL;
38     }
39     if ( strlen(path) == 0 ){
40         strcpy( ret , name );
41     }
42     else{
43         strcpy( ret , path );
44         ret[strlen(path)] = '/';
45         strcpy(ret + strlen(path)+1, name);
46     }
47     return ret;
48 }
49
50
51 /*
52 * Function : traverse_dir
53 * -----
54 *   Moves through a directory and finds all the txt files, ignoring the
55 *   duplicates inodes, using a hash table to do so
56 *
57 *   dir_name: name of the directory
58 *   txt_names: array to save the names of the txts
59 *   occupied: number of occupied positions in the array
60 *   size: size of the array
61 *   h: hash table of ints
62 *
63 *   returns the address of the array and the amount of names in it
64 */
65 pair traverse_dir( char* dir_name , char** txt_names , int occupied ,
66                   int *size , hash *h){
67     DIR* dirp;
68     struct stat sb;

```

```
69 struct dirent* de;
70 char* name;
71 int e;
72 pair get, ret;
73
74 dirp = opendir( dir_name );
75 if ( dirp == NULL ){
76     ret.f = NULL;
77     ret.s = -1;
78     return ret;
79 }
80 while ( de = readdir(dirp) ){
81     if ( strcmp(de->d_name, ".") == 0 || strcmp(de->d_name, "..") == 0 )
82         continue;
83
84     name = make_path( dir_name , de->d_name );
85     if ( name == NULL ){
86         ret.f = NULL;
87         ret.s = -1;
88         return ret;
89     }
90     e = lstat( name , &sb );
91     if ( e < 0 ){
92         ret.f = NULL;
93         ret.s = -1;
94         return ret;
95     }
96
97     if ( ( sb.st_mode & __S_IFDIR ) == __S_IFDIR ){
98         /* If a directory was found, we traverse it and update values */
99         get = traverse_dir( name, txt_names , occupied , size , h);
100         if ( get.f == NULL ){
101             return get;
102         }
103         txt_names = get.f;
104         occupied = get.s;
105     }
106     else if ( ( sb.st_mode & __S_IFREG ) == __S_IFREG ){
107         if ( strcmp( (de->d_name) + ( strlen( de->d_name ) - 4 ) ,
108             ".txt" ) == 0 ){
109
110             /* If the inode is in the hash table, we ignore it */
111             if ( ht_find( h , sb.st_ino ) ){
112                 continue;
113             }
114         }
115     }
116 }
```

```
114
115     ht_insert( h , sb.st_ino );
116
117     if ( occupied == *size ){
118         /* If the maximum size of the array was reached,
119            we allocate a new array with double size */
120         *size = *size << 1;
121         txt_names = realloc( txt_names , sizeof(char*)*( *size ) );
122         if ( txt_names == NULL ){
123             ret.f = NULL;
124             ret.s = -1;
125             return ret;
126         }
127     }
128     txt_names[occupied++] = name;
129 }
130 }
131
132
133 ret.f = txt_names;
134 ret.s = occupied;
135 return ret;
136 }
137
138
139 /*
140 * Function : word_freq_comparator
141 * -----
142 * Function to compare two pair_2 elements, that contain word and rep count
143 * for that word
144 *
145 * p : pair_2 number 1 to compare
146 * q : pair_2 number 2 to compare
147 *
148 * returns > 0 if q is greater than p, < 0 if p is greater than q and 0 if
149 * they are the same
150 */
151 int word_freq_comparator( const void *p , const void *q ){
152     pair_2 *l , *r;
153     l = ( pair_2 * )p;
154     r = ( pair_2 * )q;
155     if ( l->c > r->c ) return -1;
156     else if ( l->c < r->c ) return 1;
157     return ( strcmp( l->w , r->w ) );
158 }
```

```
159
160
161 /*
162 * Function : int_to_char
163 * -----
164 *  stores the first byte of the int to the first position of the
165 *  array, then right shifts the int bits by 8 and repeats the
166 *  process now with the second position of the array, then for
167 *  the third and finally for the fourth
168 *
169 *  x : int to store
170 *  ret : pointer to the array of chars
171 */
172 void int_to_char(int x, char * ret){
173
174     int i;
175
176     for(i=0; i<4; i++){
177         ret[i] = (char) (x)&255;
178         x >>= 8;
179     }
180
181 }
182
183 /*
184 * Function : str_to_int
185 * -----
186 *  saves the bits from each character in the int, to do so
187 *  it saves the bits from the fourth char, then left shifts 8 bits
188 *  and saves the bits from the third, then left shifts 8 bits and
189 *  repeats for the second and first char
190 *
191 *  c : pointer to the array of chars
192 *
193 *  returns an int that has the bits of the array
194 */
195 int str_to_int( char* c ){
196     int x, i;
197     x = 0;
198     for( i = 3 ; i >= 0 ; i-- ){
199         x = x << 8;
200         x = x | (unsigned char)(c[i]);
201     }
202     return x;
203 }
```

```
204
205 /*
206 * Function : write_aux
207 * -----
208 *  makes calls to syscall read until all len is read or an
209 *  error occurs
210 *
211 *  fd : file descriptor of the file to write
212 *  len : ammount of chars to write from buf
213 *  buf : array of chars to write from
214 *
215 *  returns 0 in case of success or -1 in case of failure
216 */
217 int write_aux(int fd, unsigned char * buf, int len){
218     int e , len2 = 0;
219     while(len2 < len){
220         e = write(fd, buf + len2, len-len2);
221         if ( e <= 0 ) return -1;
222         len2 += e;
223     }
224     return len;
225 }
226
227 /*
228 * Function : read_aux
229 * -----
230 *  makes calls to syscall read untile 1 chars are read and
231 *  stored in buf or an error occurs
232 *
233 *  fd : file descriptor of the file to read from
234 *  len : ammount of chars to read
235 *  buf : array of chars to write
236 *
237 *  returns 0 in case of success or -1 in case of failure
238 */
239 int read_aux( int fd , unsigned char * buf , int len ){
240     int l2, e;
241     l2 = 0;
242     while ( l2 < len ){
243         e = read( fd , buf + l2 , len - l2 );
244         if ( e <= 0 ) return e;
245         l2 = l2 + e;
246     }
247     return len;
248 }
```

```

1  /*
2  * File:      utilities.h
3  * Author:    Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4  * Description: file that contains the signature of some useful
5  *             functions used in frecpalhilos
6  * Date:      23 / 11 / 19
7  */
8
9
10 #include "hash.h"
11
12 #ifndef _UTILITIES_H
13 #define _UTILITIES_H
14
15 typedef struct{
16     char **f;
17     int s;
18 } pair;
19
20 typedef struct {
21     char * w;
22     int c;
23 } pair_2;
24
25 /*
26 * Function : make_path
27 * -----
28 *   given two arrays of chars, creates a path of the form "path/name"
29 *
30 *   path : pointer to the path name
31 *   name : pointer to the file name
32 *
33 *   returns an array of chars of the form path/name
34 */
35 char* make_path( char* path , char* name );
36
37
38 /*
39 * Function : traverse_dir
40 * -----
41 *   Moves through a directory and finds all the txt files, ignoring the
42 *   duplicates inodes
43 *
44 *   dir_name: name of the directory
45 *   txt_names: array to save the names of the txts

```

```

46 *   occupied: number of occupied positions in the array
47 *   size: size of the array
48 *   h: hash table of ints
49 *
50 *   returns the address of the array and the amount of names in it
51 */
52 pair traverse_dir( char* dir_name , char** txt_names , int occupied ,
53     int *size , hash *h );
54
55
56 /*
57 * Function : word_freq_comparator
58 * -----
59 *   Function to compare two pair_2 elements, that contain word and rep count
60 *   for that word
61 *
62 *   p : pair_2 number 1 to compare
63 *   q : pair_2 number 2 to compare
64 *
65 *   returns > 0 if q is greater than p, < 0 if p is greater than q and 0 if
66 *   they are the same
67 */
68 int word_freq_comparator( const void *p , const void *q );
69
70 #endif

```

## 1.6 hash list

```

1  /*
2  * File:      hash_list.c
3  * Author:    Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4  * Description: file that contains the implementation of the functions
5  *             used by the hash list
6  * Date:      23 / 11 / 19
7  */
8
9  #include <stdlib.h>
10 #include "hash_list.h"
11
12
13 /*
14 * Function : hl_insert
15 * -----
16 *   Inserts an int into a list, by inserting at the head of the list
17 *
18 *   l : pointer to a list

```

```
19 * k : integer to insert
20 *
21 * returns 0 on success and -1 on failure
22 */
23 int hl_insert( hash_list *l , int k ){
24     hl_node *aux = malloc( sizeof(hl_node) );
25     if ( aux == NULL ) return -1;
26     aux->key = k;
27     aux->next = l->head;
28     l->head = aux;
29     return 0;
30 }
31
32 /*
33 * Function : hl_make
34 * -----
35 *   Initializes the values of the list
36 *
37 *   l : pointer to a list
38 */
39 void hl_make( hash_list *l ){
40     l->size = 0;
41     l->head = NULL;
42 }
43
44 /*
45 * Function : hl_find
46 * -----
47 *   Looks for an element in the list, by moving through the list
48 *
49 *   l : pointer to a list
50 *   k : integer to find
51 *
52 *   returns 1 on success and 0 on failure
53 */
54 int hl_find( hash_list *l , int k ){
55
56     hl_node *aux = l->head;
57     while ( aux != NULL ){
58         if ( aux->key == k ) return 1;
59         aux = aux->next;
60     }
61     return 0;
62 }
```

```
64 }
65
66 /*
67 * File:      hash_list.h
68 * Author:    Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
69 * Description: file that contains the signatures of the functions and
70 *              structures used by the hash list
71 * Date:      23 / 11 / 19
72 */
73
74 #ifndef _HASH_LIST_H
75 #define _HASH_LIST_H
76
77 typedef struct hl_node {
78     struct hl_node *next;
79     int key;
80 } hl_node;
81
82 typedef struct {
83     int size;
84     hl_node *head;
85 } hash_list;
86
87 /*
88 * Function : hl_insert
89 * -----
90 *   Inserts an int into a list
91 *
92 *   l : pointer to a list
93 *   k : integer to insert
94 *
95 *   returns 0 on success and -1 on failure
96 */
97 int hl_insert( hash_list *l , int key );
98
99 /*
100 * Function : hl_make
101 * -----
102 *   Initializes the values of the list
103 *
104 *   l : pointer to a list
105 */
106 void hl_make( hash_list *l );
```

```
44
45
46 /*
47 * Function : hl_find
48 * -----
49 * Looks for an element in the list
50 *
51 * l : pointer to a list
52 * k : integer to find
53 *
54 * returns 1 on success and 0 on failure
55 */
56 int hl_find( hash_list *l , int key );
57
58 #endif
```

## 1.7 str list

```
1 /*
2 * File:      str_list.c
3 * Author:    Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4 * Description: file that contains the implementation of some functions
5 *              of a string list
6 * Date:      23 / 11 / 19
7 */
8
9 #include <stdio.h>
10 #include <string.h>
11 #include <stdlib.h>
12 #include "str_list.h"
13
14 /*
15 * Function: insert
16 * -----
17 * Inserts the given word in the first position of the given list by
18 * moving its pointers and updates the size of the list
19 *
20 * l: pointer to a list
21 * n: pointer to a word
22 */
23 int str_list_insert( str_list *l , char* w )
24 {
25     str_node *n;
26     n = malloc( sizeof( str_node ) );
27     if ( n == NULL ) return -1;
28     n->word = w;
```

```
29     l->size = l->size + 1;
30     n->next = l->head;
31     l->head = n;
32     return 0;
33 }
34
35
36 /*
37 * Function: make_list
38 * -----
39 * Gets the pointer to the address of a memory block allocated for a list
40 * and initializes its values head to NULL and size to 0
41 *
42 * l: pointer to a list
43 */
44 void make_str_list( str_list *l )
45 {
46     l->size = 0;
47     l->head = NULL;
48 }
49
50
51 /*
52 * File:      str_list.h
53 * Author:    Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
54 * Description: file that contains the signature of some functions
55 *              and structures of a string list
56 * Date:      23 / 11 / 19
57 */
58
59 #ifndef _STR_LIST_H
60 #define _STR_LIST_H
61
62 typedef struct str_node {
63     struct str_node *next;
64     char *word;
65     int reps;
66 } str_node;
67
68 typedef struct {
69     int size;
70     str_node *head;
71 } str_list;
72
73 /*
```

```

25 * Function: str_list_insert
26 * -----
27 * Inserts the given word in the given list
28 *
29 * l: pointer to a list
30 * n: pointer to a word
31 */
32 int str_list_insert( str_list *l , char* w );
33
34 /*
35 * Function: make_list
36 * -----
37 * Gets the pointer to the address of a memory block allocated for a list
38 * and initializes its values
39 *
40 * l: pointer to a list
41 */
42 void make_str_list( str_list *l );
43
44
45
46 #endif

```

## 1.8 str ht list

```

1 /*
2 * File:      str_ht_list.c
3 * Author:    Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4 * Description: file that contains the implementation of the functions
5 *             used by the string hash list
6 * Date:      23 / 11 / 19
7 */
8
9
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14 #include "str_ht_list.h"
15
16 /*
17 * Function: str_ht_list_insert
18 * -----
19 * Inserts the given node in the first position of the given list by
20 * moving its pointers and updates the size of the list
21 *

```

```

22 * l: pointer to a list
23 * n: pointer to a word
24 *
25 * returns 0 on success and -1 on failure
26 */
27 int str_ht_list_insert( str_ht_list *l , char *w , int k )
28 {
29     str_ht_list_node *n;
30
31     n = malloc( sizeof( str_ht_list_node ) );
32     if ( n == NULL ) return -1;
33     n->reps = k;
34     n->word = w;
35     l->size = l->size + 1;
36     n->next = l->head;
37     l->head = n;
38     return 0;
39 }
40
41
42 /*
43 * Function: str_ht_list_make_list
44 * -----
45 * Gets the pointer to the address of a memory block allocated for a list
46 * and initializes its values head to NULL and size to 0
47 *
48 * l: pointer to a list
49 */
50 void str_ht_list_make_list( str_ht_list *l )
51 {
52     l->size = 0;
53     l->head = NULL;
54 }
55
56 /*
57 * Function: str_ht_list_find
58 * -----
59 * Looks for the given word in the list by moving through its nodes,
60 * if it finds it, it adds k to the repetition counter for that
61 * node and returns the old value. Otherwise if the word is not in the list
62 * it returns 0
63 *
64 * l: pointer to a list
65 * c: pointer to an array of char
66 * k: integer to add to the rep count

```



```

67 *
68 * returns: 0 if the words is not in the list or an int that represents
69 *     the number of times that word appeared in the list
70 */
71 int str_ht_list_find( str_ht_list *l , char *c , int k )
72 {
73     str_ht_list_node *np = l->head;
74     while ( np != NULL ){
75         /* If the word is in the list, it updates the repetition count and
76            returns it */
77         if ( strcmp( c , np->word ) == 0 ){
78             np->reps = np->reps + k;
79             return np->reps - k ;
80         }
81
82         np = np->next;
83     }
84     return 0;
85 }

```

```

1  /*
2  * File:          str_ht_list.h
3  * Author:        Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4  * Description:   file that contains the signatures of the functions
5  *               used by the string hash list
6  * Date:          23 / 11 / 19
7  */

```

```

10 #ifndef _LIST_H
11 #define _LIST_H
12
13 typedef struct str_ht_list_node {
14     struct str_ht_list_node *next;
15     char *word;
16     int reps;
17 } str_ht_list_node;

```

```

19 typedef struct {
20     int size;
21     str_ht_list_node *head;
22 } str_ht_list;

```

```

24 /*
25 * Function: str_ht_list_insert

```

```

26 * -----
27 * Inserts the given word in the given list
28 *
29 * l: pointer to a list
30 * n: pointer to a word
31 *
32 * returns 0 on success and -1 on failure
33 */
34 int str_ht_list_insert( str_ht_list *l , char *n , int k );
35
36 /*
37 * Function: str_ht_list_make_list
38 * -----
39 *
40 * Gets the pointer to the address of a memory block allocated for a list
41 * and initializes its values
42 *
43 * l: pointer to a list
44 */
45 void str_ht_list_make_list( str_ht_list *l );
46
47 /*
48 * Function: str_ht_list_find
49 * -----
50 *
51 * Looks for the given word in the given list, if it finds it, it updates
52 * the number of repetitions of that word by k and returns the old number of
53 * repetitions for that word. If it doesnt find it, returns 0.
54 *
55 * l: pointer lo a list
56 * c: pointer to an array of char
57 * k: ammount to add to the rep count of the words
58 *
59 * returns: 0 if the words is not in the list or an int that represents
60 * the number of times that word appeared before updating
61 */
62 int str_ht_list_find( str_ht_list *l , char *c , int k );
63
64 #endif

```

## 1.9 hash

```

1  /*
2  * File:          hash.c
3  * Author:        Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4  * Description:   file that contains the implementation of the functions
5  *               used by the hash

```

```
6  * Date:          23 / 11 / 19
7  */
8
9
10 #include <stdlib.h>
11 #include <stdio.h>
12 #include "hash.h"
13 #include "hash_list.h"
14
15
16 /*
17 * Function : ht_make
18 * -----
19 *   Initializes the values of the hash table, allocates space for the lists
20 *   and initializes the values of the lists
21 *
22 *   h : pointer to a hash table
23 * size : size of the table
24 *
25 * returns 0 on success and -1 on failure
26 */
27 int ht_make( hash *h , int size){
28     int i, e;
29     h->size = size;
30     h->hash_table = malloc( sizeof(hash)*size );
31     if ( h->hash_table == NULL ) return -1;
32     for ( i = 0 ; i < size ; i++ ){
33         hl_make( &(h->hash_table[i]) );
34     }
35     return 0;
36 }
37
38
39 /*
40 * Function : ht_find
41 * -----
42 *   Looks for the given key in the hash by looking in the list indexed by
43 *   the
44 *   value of the hashing function
45 *
46 *   h : pointer to a hash table
47 * k : integer to look in the table
48 *
49 * returns 1 if it was found or 0 if it wasnt found
50 */
51
52
53
54
55 /*
56 * Function : ht_insert
57 * -----
58 *   Inserts the given key in the hash by inserting in the list indexed by
59 *   the
60 *   value of the hashing function
61 *
62 *   h : pointer to a hash table
63 * k : integer to insert in the table
64 *
65 * returns 0 on success or -1 on failure
66 */
67 int ht_insert( hash *h , int k ){
68     int e;
69     e = hl_insert( &( h->hash_table[ hash_function(k,h->size) ] ) , k );
70     return e;
71 }
72
73 /*
74 * Function : hash_function
75 * -----
76 *   returns the value of the hash function of the given integer,
77 *   by using mod with a high prime number to increase effectiveness
78 *
79 *   k : integer to calculate the hash function of
80 * mod : mod to use in the function
81 *
82 * returns the value of the funcion
83 */
84 int hash_function( int k , int mod ){
85     return k%mod;
86 }
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
7  */
8
9
10 #include "hash_list.h"
11
12 #ifndef _HASH_H
13 #define _HASH_H
14
15 typedef struct {
16     hash_list *hash_table;
17     int size;
18 } hash;
19
20
21 /*
22 * Function : ht_make
23 * -----
24 *   Initializes the values of the hash table
25 *
26 *   h : pointer to a hash table
27 *   size : size of the table
28 *
29 * returns 0 on success and -1 on failure
30 */
31 int ht_make( hash *h , int size);
32
33
34 /*
35 * Function : ht_find
36 * -----
37 *   Looks for the given key in the hash
38 *
39 *   h : pointer to a hash table
40 *   k : integer to look in the table
41 *
42 * returns 1 if it was found or 0 if it wasnt found
43 */
44 int ht_find( hash *h , int k );
45
46
47 /*
48 * Function : ht_insert
49 * -----
50 *   Inserts the given key in the hash
51 *
```

```
52 *   h : pointer to a hash table
53 *   k : integer to insert in the table
54 *
55 * returns 0 on success or -1 on failure
56 */
57 int ht_insert( hash * h , int k );
58
59 #endif
```

## 1.10 str hash

```
1  /*
2  * File:          str_hash.c
3  * Author:       Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4  * Description:  file that contains the implementation of the functions
5  *              used by the string hash
6  * Date:         23 / 11 / 19
7  */
8
9
10 #include <stdlib.h>
11 #include <stdio.h>
12 #include <string.h>
13 #include "str_ht_list.h"
14 #include "str_hash.h"
15
16 #define MOD 10007
17 #define PRIME 33
18
19 int prime_pow[101];
20
21
22 /*
23 * Function : set_str_hash
24 * -----
25 *   Initializes the values of an array of prime powers used
26 *   in the hash function
27 */
28 void set_str_hash( ){
29     int i;
30     int cont;
31     cont = 1;
32     for( i = 0 ; i < 101 ; i++ ){
33         prime_pow[i] = cont;
34         cont = (cont*PRIME)%MOD;
35     }
```

```
36 }
37
38
39 /*
40 * Function : str_ht_make
41 * -----
42 *   Initializes the values of the hash table, allocates space for the lists
43 *   and initializes the values of the lists
44 *
45 *   h : pointer to a string hash table
46 *
47 * returns 0 on success and -1 on failure
48 */
49 int str_ht_make( str_hash *h ){
50     int i, e, size;
51     size = MOD;
52     set_str_hash();
53     h->size = size;
54     h->hash_table = malloc( sizeof(str_ht_list)*size );
55     if ( h->hash_table == NULL ) return -1;
56     for ( i = 0 ; i < size ; i++ ){
57         str_ht_list_make_list( &(amp;h->hash_table[i]) );
58     }
59     return 0;
60 }
61
62
63 /*
64 * Function : str_ht_find
65 * -----
66 *   Looks for the given key in the string hash by looking in the list
67 *   indexed
68 * by the value of the hashing function, if found adds reps to the rep value
69 * of that word
70 *
71 *   h : pointer to a string hash table
72 *   w : word to look for
73 *   reps : amount to add to the rep count of the word
74 *
75 * returns the old amount of times that the word appears
76 */
77 int str_ht_find( str_hash *h , char *w , int reps){
78     return str_ht_list_find( &(amp;h->hash_table[ str_hash_function(w) ] ) ,
79                             w , reps );
80 }
81
82
83
84 /*
85 * Function : str_ht_insert
86 * -----
87 *   Inserts the given key in the hash by inserting in the list indexed by
88 *   the
89 * value of the hashing function with the given number of reps
90 *
91 *   h : pointer to a hash table
92 *   k : integer to insert in the table
93 *   reps : number of reps of the word
94 *
95 * returns 0 on success or -1 on failure
96 */
97 int str_ht_insert( str_hash *h , char *w , int reps ){
98     return str_ht_list_insert( &(amp;h->hash_table[ str_hash_function(w) ] ) ,
99                             w , reps );
100 }
101
102
103 /*
104 * Function : str_hash_function
105 * -----
106 *   returns the value of the hash function of the given word,
107 * by using multiplying the i-th letter with a prime raised to the power of
108 * i and adding those values, taking its module by another prime
109 *
110 *   w : word to get the hash function of
111 *
112 * returns the value of the function
113 */
114 int str_hash_function( char *w ){
115     int key, i, len;
116     key = 0;
117     len = strlen( w );
118     for( i = 0 ; i < len ; i++ ){
119         key = ( key + ( w[i] - 'a' + 1 ) * prime_pow[i] ) % MOD;
120     }
121     if ( key < 0 ) key = -key;
122     return key;
123 }
124
125
126
127 /*
128 * File:          str_hash.h
129 * Author:        Jesus Wahrman 15-11540 , Neil Villamizar 15-11523

```

```

4  * Description: file that contains the signature of the functions
5  *              used by the string hash
6  * Date:       23 / 11 / 19
7  */
8
9
10 #include "str_ht_list.h"
11
12 #ifndef _STR_HASH_H
13 #define _STR_HASH_H
14
15 typedef struct {
16     str_ht_list *hash_table;
17     int size;
18 } str_hash;
19
20
21 /*
22  * Function : str_ht_make
23  * -----
24  *   Initializes the values of the hash table
25  *
26  *   h : pointer to a string hash table
27  *
28  * returns 0 on success and -1 on failure
29  */
30 int str_ht_make( str_hash *h );
31
32
33 /*
34  * Function : str_ht_find
35  * -----
36  *   Looks for the given key in the string hash
37  *
38  *   h : pointer to a string hash table
39  *   w : word to look for
40  *   reps : amount to add to the rep count of the word
41  *
42  * returns the old amount of times that the word appears
43  */
44 int str_ht_find( str_hash *h , char *w , int k );
45
46
47 /*
48  * Function : str_ht_insert

```

```

49  * -----
50  *   Inserts the given key in the hash
51  *
52  *   h : pointer to a hash table
53  *   k : integer to insert in the table
54  *   reps : number of reps of the word
55  *
56  * returns 0 on success or -1 on failure
57  */
58 int str_ht_insert( str_hash *h , char *w , int k );
59
60 #endif

```

## 1.11 error handler

```

1  /*
2  * File:       error_handler.c
3  * Author:     Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4  * Description:file that contains the implementation of some useful functions
5  *              to manage errors
6  * Date:       23 / 11 / 19
7  */
8  #include <errno.h>
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 /*
13  * Function : error
14  * -----
15  *   given an integer, print error information if the integer is negative,
16  *   the information printed can be given or by default.
17  *
18  * e: error value
19  * str: error information
20  */
21 void error(int e, char * str){
22     if(e<0){
23         if( str == NULL ) perror("Error");
24         else perror(str);
25         exit(-1);
26     }
27 }
28
29 /*
30  * Function : errorp
31  * -----

```

```

32 *   given a pointer, print error information if the pointer is NULL,
33 *   the information printed can be given or by default.
34 *
35 * e: error value
36 *   str: error information
37 */
38 void errorp(void * e, char * str){
39     if( e == NULL ){
40         if( str == NULL ) perror("Error");
41         else perror(str);
42         exit(-1);
43     }
44 }

1  /*
2  * File:      error_handler.h
3  * Author:    Jesus Wahrman 15-11540 , Neil Villamizar 15-11523
4  * Description:file that contains the signature of some useful functions
5  *           to manage errors
6  * Date:      23 / 11 / 19
7  */
8
9 #ifndef _ERROR_HANDLER_
10 #define _ERROR_HANDLER_
11
12 /*
13 * Function : error
14 * -----
15 *   given an integer, print error information if the integer is negative,
16 *   the information printed can be given or by default.
17 *
18 * e: error value
19 *   str: error information
20 */
21 void error(int e, char * str);
22
23
24 /*
25 * Function : errorp
26 * -----
27 *   given a pointer, print error information if the pointer is NULL,
28 *   the information printed can be given or by default.
29 *
30 * e: error value
31 *   str: error information

```

```

32 */
33 void errorp(void * e, char * str);
34
35 #endif

```

## 1.12 counter thread

```

1  /*
2  * File:      counter_thread.h
3  * 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

```