

```
1 /**
2  * Main.c
3  *
4  * Programa que toma una lista de pares de cadenas de caracteres
5  * <cad1, cad2> de un archivo y reemplaza, en un conjunto de
6  * archivos de texto, todas las ocurrencias de cad1 por cad2.
7  *
8  * Autor: Ka Fung (1810492)
9  * Fecha: 16/06/2020
10 */
11
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <string.h>
15 #include "Lista.h"
16
17 FILE* abrir_archivo(char* archivo);
18 void extraer_palabras(char* archivo, struct Nodo** cabeza);
19 void remplazar_palabras(char* archivo, struct Nodo* cabeza);
20
21 int main(int argc, char **argv) {
22     struct Nodo* lista_cabeza;
23     int i;
24
25     /* Revisar que recibe correctamente los argumentos */
26     if (argc < 3) {
27         printf("Uso: sustituir <palabras.txt> <Archivo1.txt> <Archivo2.txt>... \n");
28         return 1;
29     }
30
31     /* Extraer palabras y almacenarlas en la lista doblemente enlazada */
32     lista_cabeza = NULL;
33     extraer_palabras(argv[1], &lista_cabeza);
34
35     /* Por cada archivo, reemplazar las palabras */
36     for (i = 2; i < argc; i++) {
37         remplazar_palabras(argv[i], lista_cabeza);
38
39         if (i != argc - 1) {
40             printf("\n--\n");
41         }
42     }
43
44     /* Liberar memoria ocupada por la lista */
45     liberar_lista(lista_cabeza);
46
47     return 0;
48 }
49
50 /**
51  * Abre un archivo dado y verifica si se pudo abrir.
52  * Parámetros:
53  *     - archivo: nombre del archivo
54  * Retorno:
55  *     - ptr: puntero al archivo
56  */
57 FILE* abrir_archivo(char* archivo) {
58     FILE* ptr = fopen(archivo, "r");
59     if (!ptr) {
60         printf("Error al abrir el archivo %s\n", archivo);
61         exit(1);
62     }
63     return ptr;
64 }
65
66 /**
67  * Dado un archivo con una lista de palabras a sustituir, extrae
68  * las palabras y las guarda en una lista doblemente enlazada dada.
69  * Parámetros:
70  *     - archivo: nombre del archivo
71  *     - cabeza: puntero a la lista
72  */
73 void extraer_palabras(char* archivo, struct Nodo** cabeza) {
74     FILE* ptr;
75     char temp1[60], temp2[60];
76
77     ptr = abrir_archivo(archivo);
78
79     /* Extrae las palabras y las separa según el char ':' */
80     while(fscanf(ptr, "%[^:]:%[^\n]\n", temp1, temp2) != EOF) {
81         char* word1 = (char *)malloc(sizeof(char) * strlen(temp1) + 1);
82         char* word2 = (char *)malloc(sizeof(char) * strlen(temp2) + 1);
83         if (!word1 || !word2) {
84             printf("Error al reservar memoria para guardar las palabras\n");
85             exit(1);
86         }
87
88         /* Copia las palabras y guarda en la lista doblemente enlazada */
89         strcpy(word1, temp1);
90         strcpy(word2, temp2);
91         insertar_ordenado_lista(cabeza, crear_par(word1, word2));
```

```
92     }
93
94     fclose(ptr);
95 }
96
97 /**
98  * Dado un archivo de texto y una lista de palabras a reemplazar,
99  * reemplaza todas las ocurrencias de las palabras en el archivo.
100  * Parametros:
101  *     - archivo: archivo a reemplazar
102  *     - cabeza: cabeza de la lista doblemente enlazada
103  */
104 void reemplazar_palabras(char* archivo, struct Nodo* cabeza) {
105     FILE* ptr;
106     char ch;
107
108     ptr = abrir_archivo(archivo);
109
110     /* Revisamos por cada coincidencia de char del archivo a reemplazar */
111     ch = fgetc(ptr);
112     while (ch != EOF) {
113
114         /* Revisamos por cada palabra de la lista */
115         struct Nodo* actual = cabeza;
116         while (actual != NULL) {
117             /* Itera mientras coincidan la palabra de la lista y el texto*/
118             int i = 0;
119
120             while (ch == actual->dato->x[i]) {
121                 ch = fgetc(ptr);
122                 i++;
123             }
124
125             /* Si coincide toda la palabra, se imprime */
126             if (i == actual->len) {
127                 printf("%s", actual->dato->y);
128                 break;
129             }
130
131             /* En cambio, se revisa con la siguiente palabra de la lista*/
132             actual = actual->next;
133             fseek(ptr, -i-1, SEEK_CUR);
134             ch = fgetc(ptr);
135         }
136
137         /* Si no coincide ninguna palabra de la lista, se imprime el char */
138         if (!actual) {
139             printf("%c", ch);
140             ch = fgetc(ptr);
141         }
142     }
143
144     fclose(ptr);
145 }
```

```
1 /**
2  * Lista.h
3  * Lista Doblemente Enlazada de Pares de palabras
4  * Autor: Ka Fung (1810492)
5  * Fecha: 16/06/2020
6  */
7
8 #ifndef __LISTA_H__
9     #define __LISTA_H__
10     #include "Par.h"
11
12     /**
13      * Nodo de la lista
14      * Atributos:
15      *     - par: par de palabras
16      *     - len: longitud de la primera palabra del par
17      *     - prev: anterior nodo
18      *     - next: siguiente nodo
19      */
20     struct Nodo {
21         Par* dato;
22         int len;
23         struct Nodo* prev;
24         struct Nodo* next;
25     };
26
27     void insertar_ordenado_lista(struct Nodo** cabeza, Par* palabras);
28     void print_lista(struct Nodo* nodo);
29     void liberar_lista(struct Nodo* nodo);
30 #endif
```

Mejorar documentación

```
1 /**
2  * Lista.c
3  * Lista Doblemente Enlazada de Pares de palabras
4  * Autor: Ka Fung (1810492)
5  * Fecha: 16/06/2020
6  */
7
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11 #include "Lista.h"
12 #include "Par.h"
13
14 /**
15  * Inserta un nuevo nodo en una lista ordenada descendentemente.
16  * Se toma como comparación la longitud de la primera palabra del par.
17  * Parametros:
18  *     - cabeza: cabeza de la lista
19  *     - palabras: palabras a insertar
20  */
21 void insertar_ordenado_lista(struct Nodo** cabeza, Par* palabras) {
22     int len_palabra = strlen(palabras->x);
23     struct Nodo* nuevo = (struct Nodo*)malloc(sizeof(struct Nodo));
24     if (!nuevo) {
25         printf("Error al reservar memoria al insertar una nueva palabra.\n");
26         exit(1);
27     }
28
29     /* Crea nuevo nodo */
30     nuevo->dato = palabras;
31     nuevo->len = len_palabra;
32
33     if (!(*cabeza)) {
34         /* Si la lista esta vacia, simplemente se agrega */
35         *cabeza = nuevo;
36         nuevo->next = NULL;
37         nuevo->prev = NULL;
38     } else if (len_palabra > (*cabeza)->len) {
39         /* Si la palabra es mayor que la primera, se agrega al inicio */
40         nuevo->next = *cabeza;
41         nuevo->prev = NULL;
42         (*cabeza)->prev = nuevo;
43         *cabeza = nuevo;
44     } else {
45         /* En cambio, se busca en donde insertar */
46         struct Nodo* actual = *cabeza;
47         while (actual->next != NULL && len_palabra < actual->next->len) {
48             actual = actual->next;
49         }
50
51         /* Actualiza los punteros */
52         nuevo->next = actual->next;
53         if (actual->next) {
54             actual->next->prev = nuevo;
55         }
56         actual->next = nuevo;
57         nuevo->prev = actual;
58     }
59 }
60
61 /**
62  * Imprime la lista doblemente enlazada
63  * Parámetros:
64  *     - nodo: puntero a la cabeza de la lista
65  */
66 void print_lista(struct Nodo* nodo) {
67     printf("[");
68     while (nodo) {
69         print_par(nodo->dato);
70         if (nodo->next) {
71             printf(", ");
72         }
73         nodo = nodo->next;
74     }
75     printf("]\n");
76 }
77
78 /**
79  * Libera la memoria asignada de la lista doblemente enlazada
80  * Parámetros:
81  *     - nodo: puntero a la cabeza de la lista
82  */
83 void liberar_lista(struct Nodo* nodo) {
84     while (nodo) {
85         struct Nodo* temp = nodo;
86         nodo = nodo->next;
87         liberar_par(temp->dato);
88         free(temp);
89     }
90 }
```

No hagas exit en una función de una librería.

```
1 /**
2  * Par.h
3  * Par de palabras
4  * Autor: Ka Fung (1810492)
5  * Fecha: 16/06/2020
6  */
7
8 #ifndef __PAR_H__
9     #define __PAR_H__
10
11     /**
12      * Par de palabras
13      * Atributos:
14      *     - x: palabra a remplazar
15      *     - y: palabra por remplazar
16      */
17     typedef struct {
18         char *x;
19         char *y;
20     } Par;
21
22     Par* crear_par(char* x, char* y);
23     void print_par(Par* par);
24     void liberar_par(Par* par);
25 #endif
```

```
1 /**
2  * Par.c
3  * Par de palabras
4  * Autor: Ka Fung (1810492)
5  * Fecha: 16/06/2020
6  */
7
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include "Par.h"
11
12 /**
13  * Crea un nuevo par de palabras.
14  * Parametros:
15  *   - x: palabra1
16  *   - y: palabra2
17  * Retorno:
18  *   - par: par de palabras
19  */
20 Par* crear_par(char* x, char* y) {
21     Par* par = (Par*)malloc(sizeof(Par));
22     if(!par) {
23         printf("Error al reservar memoria\n");
24         exit(1);
25     }
26
27     par->x = x;
28     par->y = y;
29     return par;
30 }
31
32 /**
33  * Imprime un par de palabras.
34  * Parametros:
35  *   - par: par de palabras
36  */
37 void print_par(Par* par) {
38     printf("(%s, %s)", par->x, par->y);
39 }
40
41 /**
42  * Libera la memoria asignada de un par de palabras.
43  * Parametros:
44  *   - par: par de palabras
45  */
46 void liberar_par(Par* par) {
47     free(par->x);
48     free(par->y);
49     free(par);
50 }
```