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



/*
    Jose Matias Gonzalez Valarezo
    15-10627
*/

/*
    Definicion de los pares de caracteres
*/
typedef struct
{
    /* data */

    char cad1[100];
    char cad2[100];

} pair ;

/*
    Definicion de nodos/elementos de la doble lista enlazada
*/

struct nodo
{
    /* data */
    pair data;
    struct nodo* prev;
    struct nodo* next;


};

/*
    Funcion que se encarga crear una nueva instancia
    de un pair, a partir del argumento separador
    y una linea del archivo de palabras.txt
*/
pair createPair(char *linea, char *separador) {
    char *ptr;
    pair newPair;

    ptr = strtok(linea,separador);

    strcpy(newPair.cad1,ptr);
    ptr = strtok(NULL,separador);
    strcpy(newPair.cad2,ptr);


    return newPair;

}


/*
```

```
60      Funcion que crea y agrega un nodo al final de la lista.
61      Si head es NULL entonces la lista esta vacia y agregamos
62      el nuevo nodo en head. En caso contrario debemos recorrer
63      la lista hasta el ultimo nodo y agregar el nuevo nodo luego
64      de este.
65   */
66  void agregarNodo(struct nodo** head, pair p) {
67
68
69      struct  nodo* newNode = malloc(sizeof(struct nodo));
70      struct nodo* temporal;
71      newNode ->data = p;
72      newNode -> next = NULL;
73
74      temporal = *head;
75
76
77      /* Si la lista esta vacia, colocamos el nuevo nodo como cabeza */
78      if (*head == NULL) {
79          newNode -> prev = NULL;
80          *head = newNode;
81          return;
82      }
83
84      /*
85
86          Caso contrario recorremos la lista hasta el ultimo elemento
87          e insertamos al nuevo nodo como el ultimo elemento, colocando el
88          prev del nuevo nodo como el ultimo nodo anterior y next como NULL.
89
90      */
91
92      while (temporal -> next != NULL)
93      {
94          temporal = temporal -> next;
95      }
96
97      temporal -> next = newNode;
98      newNode -> prev = temporal;
99
100 }
101
102
103
104 int main(int argc, char const *argv[])
105 {
106
107
108     /* Declaracion de variables
109         - fp file descriptor
110         - textPalabras, array para alocar strings
111         - copyText, array para alocar strings al leer los archivos
112         a cambiar.
113         - ptr, char para tomar subStrings
114         - separador, string con constante valor ":"
115         - N int con valor fijo, para calcular malloc de arrays
116         - j variable para un ciclo for, para verificar strings.
117     */
118     FILE *fp;
119     char *textPalabras;
```

```
120     char copyText[100];
121     char *ptr;
122     char *separador= ":";
123     int N = 100;
124     /* Variables para manejar el input*/
125     int j = 1;
126 
127 
128 
129     /* Inicializamos la cabeza de la lista */
130     struct nodo *head = NULL;
131 
132 
133     /*
134         Crear pares a partir del archivo palabras.txt
135         la posicion 1 de argv siempre debe contener
136         el primer archivo de texto que contiene los
137         pares cad1:cad2
138     */
139 
140     fp = fopen(argv[1],"r");
141 
142     if (fp == NULL)
143     {
144         printf("Error al abrir el archivo \n");
145         exit(1);
146     }
147 
148       /* Alocando memoria para un array de chars */
149 
150     textPalabras = (char*) malloc( sizeof(char) *N  );
151 
152     /* Verificando error al apartar memorio */
153 
154     if (textPalabras == NULL) {
155         printf("Error obteniendo espacio de memoria\n");
156         exit(1);
157     }
158 
159 
160     /*
161         Leemos el archivo de los pares de palabras y los
162         procesamos con las funciones createPair y agregarNodo
163         de manera que dentro de la lista enlazada quedan
164         almacenados las keys Cad1.
165     */
166     while (fscanf(fp,"%s",textPalabras) != EOF)
167     {
168         pair par;
169 
170         par = createPair(textPalabras,separador);
171 
172         agregarNodo(&head, par);
173 
174     }
175 
176     /* Cerramos el archivo */
177 
178     fclose(fp);
179 
```

```
180
181     /* Abriendo txt */
182
183     for(j = 2; j < argc; j++) {
184         fp = fopen(argv[j], "r");
185
186         /* Verificacion de error al abrir un archivo */
187         if(fp == NULL ) {
188             printf("Error al abrir el archivo\n");
189             exit(1);
190         }
191
192
193         /*
194             Recorremos el archivo que contiene el texto
195             a modificar, en cada lectura verificamos
196             si la palabra que se esta leyendo actulmente
197             se encuentra en la lista enlazada, en caso
198             positivo la reemplazamos por su respectiva
199             Cad2, en caso negativo simplemente imprimimos
200             la palabra.
201         */
202
203         while ( !feof(fp) )
204         {
205
206             /*
207                 - subString string utilizado para almacenar signos de puntuacion.
208                 - dobleCheck,tripleCheck y finalCheck son strings utilizados para
209                 almacenar las palabras y modificarlas en distintas situaciones.
210                 - temporal es un apuntador a un nodo, en este caso al head de la
211                 lista enlazada.
212                 - i,k,v son ints utilizados para los ciclos for de esta seccion.
213                 - verificacion es un int utilizado para colocar el caracter \0 al
214                 final de una palabra al construirla caracter por caracter.
215                 - flag es un int , que como indica su nombre cumple la funcion de
216                 un flag para un condicional.
217             */
218             char subString[2];
219             char dobleCheck[100];
220             char tripleCheck[100];
221             char finalCheck[100];
222             struct nodo* temporal;
223             int i;
224             int k;
225             int verificacion;
226             int flag;
227             int posicion;
228             int v;
229
230             fscanf(fp,"%s ",copyText);
231
232             /*
233                 Verificamos caracter por caracter para verificar si hay
234                 un signo de puntuacion en la palabra, en caso positivo,
235                 se lo quitamos a la palabra y lo guardamos para ser agregado al
    final.
236             */
237
238             for(i = 0; i < strlen(copyText); i++){
```

```
239
240                 ptr = NULL;
241                 if(copyText[i] == '.' || copyText[i] == ',' || copyText[i] == ';' ){
242
243                     subString[0] = copyText[i];
244                     subString[1] = '\0';
245
246
247                     ptr = strtok(copyText,subString);
248
249                 }
250             }
251             /*
252                 Verificamos la palabra caracter por caracter, mientras vamos
253                 generando substrings, si conseguimos que un substrings es
254                 igual a una cadena cad1 la guardamos junto a variables de
255                 contexto para saber cual palabra debemos copiar/modificar antes
256                 de imprimir.
257             */
258              for(k = 0; k < strlen(copyText); k++){
259                 flag = 0;
260                 verificacion = k + 1;
261                 dobleCheck[k] = copyText[k];
262                 dobleCheck[verificacion] = '\0';
263
264                 /*
265                     Verificamos primero el head
266                     si conseguimos la palabra la retornamos
267                     caso contrario buscamos en el siguiente
268                     elemento de la lista
269                 */
270
271                 temporal = head;
272
273                 if (strcmp(temporal->data.cad1,dobleCheck) == 0) {
274                     strcpy(tripleCheck,temporal->data.cad2);
275                     flag = 1;
276                 }
277
278                 /*
279                     Verificacion de la lista de nodos, menos el ultimo
280                 */
281
282                 while(temporal->next != NULL) {
283                     if (strcmp(temporal->data.cad1,dobleCheck) == 0) {
284                         strcpy(tripleCheck,temporal->data.cad2);
285                         flag = 1;
286                     }
287                     temporal = temporal->next;
288                 }
289
290                 /*
291                     Verificacion del ultimo nodo
292                 */
293
294                 if (strcmp(temporal->data.cad1,dobleCheck) == 0) {
295                     strcpy(tripleCheck,temporal->data.cad2);
296                     flag = 1;
297                 }
298
```

```
299
300                 /* Si se consiguio una cadena cad1 entra en el if*/
301                 if(flag == 1) {
302
303                     strcpy(finalCheck,tripleCheck);
304                     strcpy(tripleCheck,"");
305                     posicion = k;
306                 }
307
308             }
309
310             /*
311                 Verificamos primero el head
312                 si conseguimos la palabra la retornamos
313                 caso contrario buscamos en el siguiente
314                 elemento de la lista
315             */
316
317             temporal = head;
318
319             if (strcmp(temporal->data.cad1,copyText) == 0) {
320                 strcpy(copyText,temporal->data.cad2);
321             }
322
323
324             /*
325                 Verificacion de la lista de nodos, menos el ultimo
326             */
327
328             while(temporal->next != NULL) {
329                 if (strcmp(temporal->data.cad1,copyText) == 0) {
330                     strcpy(copyText,temporal->data.cad2);
331                 }
332                 temporal = temporal->next;
333             }
334
335
336             /*
337                 Verificacion del ultimo nodo
338             */
339
340             if (strcmp(temporal->data.cad1,copyText) == 0) {
341                 strcpy(copyText,temporal->data.cad2);
342             }
343
344
345             /*
346                 Si finalCheck es distinto de "" y posicion > 0
347                 entonces vamos a cambiar lo que obtuvimos del texto
348                 por la cadena guardada en finaCheck, caracter por caracter.
349                 Por ultimo reiniciamos posicion y finalCheck
350             */
351             if (posicion != 0 && strcmp(finalCheck,"") != 0 ) {
352
353
354                 for (v = posicion;v >= 0; v-- ) {
355                     copyText[v] = finalCheck[v];
356                 }
357                 posicion = 0;
358                 strcpy(finalCheck,"");
```

```
359
360             }
361
362             /*
363                 Si eliminamos un signo de puntuacion, se lo agregamos
364             */
365             if (ptr != NULL ) {
366
367                 strcat(copyText,subString);
368             }
369
370             printf("%s ",copyText);
371
372         }
373
374         /* Si faltan archivos por procesar ... */
375         if ( j + 1 < argc) {
376              printf("\n--");
377         }
378         printf("\n");
379         fclose(fp);
380     }
381
382
383
384     return 0;
385 }
386
```