```c
/**
 * Main.c
 *
 * Programa que toma una lista de pares de cadenas de caracteres
 * <cad1, cad2> de un archivo y reemplaza, en un conjunto de
 * archivos de texto, todas las ocurrencias de cad1 por cad2.
 *
 * Autor: Ka Fung (1810492)
 * Fecha: 16/06/2020
 */

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

FILE* abrir_archivo(char* archivo);
void extraer_palabras(char* archivo, struct Nodo** cabeza);
void remplazar_palabras(char* archivo, struct Nodo* cabeza);

int main(int argc, char **argv) {
    struct Nodo* lista_cabeza;
    int i;

    /* Revisar que recibe correctamente los argumentos */
    if (argc < 3) {
        printf("Uso: sustituir <palabras.txt> <Archivo1.txt> <Archivo2.txt>... \n");
        return 1;
    }

    /* Extraer palabras y almacenarlas en la lista doblemente enlazada */
    lista_cabeza = NULL;
    extraer_palabras(argv[1], &lista_cabeza);

    /* Por cada archivo, reemplazar las palabras */
    for (i = 2; i < argc; i++) {
        remplazar_palabras(argv[i], lista_cabeza);

        if (i != argc - 1) {
            printf("\n--\n");
        }
    }

    /* Liberar memoria ocupada por la lista */
    liberar_lista(lista_cabeza);

    return 0;
}

/**
 * Abre un archivo dado y verifica si se pudo abrir.
 * Parámetros:
 *      - archivo: nombre del archivo
 * Retorno:
 *      - ptr: puntero al archivo
 */
FILE* abrir_archivo(char* archivo) {
    FILE* ptr = fopen(archivo, "r");
    if (!ptr) {
        printf("Error al abrir el archivo %s\n", archivo);
        exit(1);
    }
    return ptr;
}

/**
 * Dado un archivo con una lista de palabras a sustituir, extrae
 * las palabras y las guarda en una lista doblemente enlazada dada.
 * Parámetros:
 *      - archivo: nombre del archivo
 *      - cabeza: puntero a la lista
 */
void extraer_palabras(char* archivo, struct Nodo** cabeza) {
    FILE* ptr;
    char temp1[60], temp2[60];

    ptr = abrir_archivo(archivo);

    /* Extrae las palabras y las separa según el char ':' */
    while(fscanf(ptr, "%[^:]:%[^\n]\n", temp1, temp2) != EOF) {
        char* word1 = (char *)malloc(sizeof(char) * strlen(temp1) + 1);
        char* word2 = (char *)malloc(sizeof(char) * strlen(temp2) + 1);
        if (!word1 || !word2) {
            printf("Error al reservar memoria para guardar las palabras\n");
            exit(1);
        }

        /* Copia las palabras y guarda en la lista doblemente enlazada */
        strcpy(word1, temp1);
        strcpy(word2, temp2);
        insertar_ordenado_lista(cabeza, crear_par(word1, word2));
```

```
    }

    fclose(ptr);
}

/**
 * Dado un archivo de texto y una lista de palabras a reemplazar,
 * reemplaza todas las ocurrencias de las palabras en el archivo.
 * Parametros:
 *      - archivo: archivo a reemplazar
 *      - cabeza: cabeza de la lista doblemente enlazada
 */
void remplazar_palabras(char* archivo, struct Nodo* cabeza) {
    FILE* ptr;
    char ch;

    ptr = abrir_archivo(archivo);

    /* Revisamos por cada coincidencia de char del archivo a reemplazar */
    ch = fgetc(ptr);
    while (ch != EOF) {

        /* Revisamos por cada palabra de la lista */
        struct Nodo* actual = cabeza;
        while (actual != NULL) {
            /* Itera mientras coincidan la palabra de la lista y el texto*/
            int i = 0;

            while (ch == actual->dato->x[i]) {
                ch = fgetc(ptr);
                i++;
            }

            /* Si coincide toda la palabra, se imprime */
            if (i == actual->len) {
                printf("%s", actual->dato->y);
                break;
            }

            /* En cambio, se revisa con la siguiente palabra de la lista*/
            actual = actual->next;
            fseek(ptr, -i-1, SEEK_CUR);
            ch = fgetc(ptr);
        }

        /* Si no coincide ninguna palabra de la lista, se imprime el char */
        if (!actual) {
            printf("%c", ch);
            ch = fgetc(ptr);
        }
    }

    fclose(ptr);
}
```

```c
/**
 * Lista.h
 * Lista Doblemente Enlazada de Pares de palabras
 * Autor: Ka Fung (1810492)
 * Fecha: 16/06/2020
 */

#ifndef __LISTA_H__
    #define __LISTA_H__
    #include "Par.h"

    /**
     * Nodo de la lista
     * Atributos:
     *      - par: par de palabras
     *      - len: longitud de la primera palabra del par
     *      - prev: anterior nodo
     *      - next: siguiente nodo
     */
    struct Nodo {
        Par* dato;
        int len;
        struct Nodo* prev;
        struct Nodo* next;
    };

    void insertar_ordenado_lista(struct Nodo** cabeza, Par* palabras);
    void print_lista(struct Nodo* nodo);
    void liberar_lista(struct Nodo* nodo);
#endif
```

```c
/**
 * Lista.c
 * Lista Doblemente Enlazada de Pares de palabras
 * Autor: Ka Fung (1810492)
 * Fecha: 16/06/2020
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Lista.h"
#include "Par.h"

/**
 * Inserta un nuevo nodo en una lista ordenada descendentemente.
 * Se toma como comparación la longitud de la primera palabra del par.
 * Parametros:
 *      - cabeza: cabeza de la lista
 *      - palabras: palabras a insertar
 */
void insertar_ordenado_lista(struct Nodo** cabeza, Par* palabras) {
    int len_palabra = strlen(palabras->x);
    struct Nodo* nuevo = (struct Nodo*)malloc(sizeof(struct Nodo));
    if (!nuevo) {
        printf("Error al reservar memoria al insertar una nueva palabra.\n");
        exit(1);
    }

    /* Crea nuevo nodo */
    nuevo->dato = palabras;
    nuevo->len = len_palabra;

    if (!(*cabeza)) {
        /* Si la lista esta vacia, simplemente se agrega */
        *cabeza = nuevo;
        nuevo->next = NULL;
        nuevo->prev = NULL;
    } else if (len_palabra > (*cabeza)->len) {
        /* Si la palabra es mayor que la primera, se agrega al inicio */
        nuevo->next = *cabeza;
        nuevo->prev = NULL;
        (*cabeza)->prev = nuevo;
        *cabeza = nuevo;
    } else {
        /* En cambio, se busca en donde insertar */
        struct Nodo* actual = *cabeza;
        while (actual->next != NULL && len_palabra < actual->next->len) {
            actual = actual->next;
        }

        /* Actualiza los punteros */
        nuevo->next = actual->next;
        if (actual->next) {
            actual->next->prev = nuevo;
        }
        actual->next = nuevo;
        nuevo->prev = actual;
    }
}

/**
 * Imprime la lista doblemente enlazada
 * Parámetros:
 *      - nodo: puntero a la cabeza de la lista
 */
void print_lista(struct Nodo* nodo) {
    printf("[");
    while (nodo) {
        print_par(nodo->dato);
        if (nodo->next) {
            printf(", ");
        }
        nodo = nodo->next;
    }
    printf("]\n");
}

/**
 * Libera la memoria asignada de la lista doblemente enlazada
 * Parámetros:
 *     - nodo: puntero a la cabeza de la lista
 */
void liberar_lista(struct Nodo* nodo) {
    while (nodo) {
        struct Nodo* temp = nodo;
        nodo = nodo->next;
        liberar_par(temp->dato);
        free(temp);
    }
}
```

```c
/**
 * Par.h
 * Par de palabras
 * Autor: Ka Fung (1810492)
 * Fecha: 16/06/2020
 */

#ifndef __PAR_H__
    #define __PAR_H__

    /**
     * Par de palabras
     * Atributos:
     *     - x: palabra a remplazar
     *     - y: palabra por remplazar
     */
    typedef struct {
        char *x;
        char *y;
    } Par;

    Par* crear_par(char* x, char* y);
    void print_par(Par* par);
    void liberar_par(Par* par);
#endif
```

```c
/**
 * Par.c
 * Par de palabras
 * Autor: Ka Fung (1810492)
 * Fecha: 16/06/2020
 */

#include <stdio.h>
#include <stdlib.h>
#include "Par.h"

/**
 * Crea un nuevo par de palabras.
 * Parametros:
 *     - x: palabra1
 *     - y: palabra2
 * Retorno:
 *    - par: par de palabras
 */
Par* crear_par(char* x, char* y) {
    Par* par = (Par*)malloc(sizeof(Par));
    if(!par) {
        printf("Error al reservar memoria\n");
        exit(1);
    }

    par->x = x;
    par->y = y;
    return par;
}

/**
 * Imprime un par de palabras.
 * Parametros:
 *    - par: par de palabras
 */
void print_par(Par* par) {
    printf("(%s, %s)", par->x, par->y);
}

/**
 * Libera la memoria asignada de un par de palabras.
 * Parametros:
 *   - par: par de palabras
 */
void liberar_par(Par* par) {
    free(par->x);
    free(par->y);
    free(par);
}
```