main.c

```c
/**
 * Entry-point de programa de sustituir.
 *
 * Programa que toma una lista de pares de strings <str1, str2> y reemplaza,
 * en un conjunto de archivos de texto, todas las ocurrencias de str1 por str2.
 * Imprime el resultado por salida estándar, separando cada archivo del
 * conjunto de archivos con el delimitador '--'.
 *
 * Uso:
 * > sustituir <ruta a archivo de pares> {lista de nombres de archivos}
 *
 * El archivo de pares consiste en texto plano, donde cada línea contiene dos
 * cadenas de caracteres separadas por ':'. Ejemplo:
 *
 * ———————————————————————————
 * Hola:hola                   |
 * sustituir:cambiar           |
 * hola como estas:que tal     |
 * string:cadena de caracteres |
 * ———————————————————————————
 *
 * Autor: Christopher Gómez.
 * Fecha: 04-06-2022.
 */
#include <stdio.h>
#include <stdlib.h>

#include "pair.h"
#include "list.h"
#include "utils.h"

int main(int argc, char **argv) {
    Node *word_list;
    int i;

    if (argc < 3) {
        printf("Uso: sustituir <archivo pares> {lista de nombres de archivos}.\n");
        exit(1);
    }

    /* Extrae la lista de palabras e itera por todos los archivos
    proporcionados en el argv, con la función de reemplazar */
    word_list = extract_words_from_file(argv[1]);
    for (i = 2; i < argc; i++) {
        replace_words(argv[i], word_list);
        if (i + 1≠ argc)
            printf("\n--\n");
    }

    return 0;
}
```

pair.h

```c
#ifndef __PAIR_H__
#define __PAIR_H__

/**
 * Estructura de par de cadenas de caracteres.
 */
typedef struct Pair {
    char *first, *second; /** Primer y segundo elemento del par */
} Pair;

Pair *Pair_new(char *first, char *second);
void Pair_print(Pair *pair);

#endif
```

Mejorar documentación

pair.c

```c
/**
 * Implementación de par.
 *
 * Autor: Christopher Gómez.
 * Fecha: 04-06-2022.
 */
#include <stdlib.h>
#include <stdio.h>

#include "pair.h"

/**
 * Crea una par de cadenas de caracteres.
 *
 * @param first: Primer elemento del par.
 * @param second: Primer elemento del par.
 * @return Un apuntador a Pair si la creación fue exitosa.
 *     NULL en caso contrario.
 */
Pair *Pair_new(char *first, char *second) {
    Pair *p = malloc(sizeof(Pair));
    if (!p)
        return NULL;

    p->first = first;
    p->second = second;

    return p;
}

/**
 * Imprime una representación en string del par de cadenas de caracteres.
 *
 * Para propósitos de depuración.
 *
 * @param pair: Apuntador a estructura Pair.
 */
void Pair_print(Pair *pair) {
    printf("('%s', '%s')", pair->first, pair->second);
}
```

list.h

```h
#ifndef __LIST_H__
#define __LIST_H__

#include <aio.h>

#include "pair.h"

/**
 * Estructura de nodo para lista circular doblemente enlazada.
 */
typedef struct Node {
    struct Node *prev, *next; /** Elementos anterior y siguiente */
    Pair *data; /** Dato del nodo, de tipo Pair */
    int length; /** Longitud de la primera palabra del par */
} Node;

Node *List_new();
u_int8_t List_push(Node **head, Pair *data, int length);
void List_print(Node *head);

#endif
```

Mejorar formato, para facilitar legibilidad

list.c

```c
/**
 * Implementación de lista circular, donde cada nodo contiene un
 * par de cadenas de caracteres y la longitud de la cadena de la
 * primera coordenada.
 *
 * Soporta las operaciones de creación e inserción ordenada.
 *
 * Autor: Christopher Gómez.
 * Fecha: 04-06-2022.
 */
#include <stdlib.h>
#include <stdio.h>

#include "list.h"
#include "pair.h"

/**
 * Crea una lista doblemente enlazada sin elementos.
 *
 * @return Un apuntador a la cabeza de la lista si la creación
 *     fue exitosa.
 *     NULL en caso contrario.
 */
Node *List_new() {
    /* Asigna memoria dinámicamente para crear la lista */
    Node *head = malloc(sizeof(Node));
    if (!head)
        return NULL;

    /* La cabeza de la lista apunta a sí misma */
    head->next = NULL;
    head->prev = NULL;
    head->data = NULL;

    return head;
}

/**
 * Añade un nodo en orden en la lista.
 *
 * @param head: Apuntador a la direccion de la cabeza de la lista.
 * @param data: Dato a almacenar en el nodo, de tipo Pair.
 * @param length: Longitud de la primera palabra del par.
 * @return 1 si la operación fue exitosa.
 *     0 en caso contrario.
 */
u_int8_t List_push(Node **list, Pair *data, int length) {
    Node *head = *list;

    if (!head->data) {
        /* Si la lista es nueva, coloca el dato en el nodo */
        head->data = data;
        head->length = length;
    } else {
        /* De otra forma, crea una nueva entrada para la lista
        dinámicamente */
        Node *new_node = malloc(sizeof(Node));
        if (!new_node)
            return 0;

        /* Coloca el dato en el nodo */
        new_node->data = data;
        new_node->length = length;

        if (length ⩾ head->length) {
            /* Si el campo length es ⩾ que el de la cabeza */
            new_node->next = head;
            new_node->prev = NULL;
            head->prev = new_node;

            /* Ahora el nuevo nodo es la cabeza de la lista */
            *list = new_node;
        } else {
            Node *cur = head;

            /* Busca la posición en la que debe insertarse el nuevo nodo:
            antes del primer nodo que contenga un campo length menor o
            igual, o al final de la lista si no lo hay */
            while (cur->next ≠ NULL && cur->next->length > length)
                cur = cur->next;

            /* Se agrega el nodo entre cur y cur->next */
            new_node->next = cur->next;
            new_node->prev = cur;
            if (cur->next)
                cur->next->prev = new_node;
```

list.c

```c
            cur->next = new_node;
        }
    }

    return 1;
}

/**
 * Imprime una representación en string de todos los pares de la lista.
 *
 * Para propósitos de depuración.
 *
 * @param list: Apuntador a la cabeza de la lista
 */
void List_print(Node *head) {
    Node *cur = head;

    printf("[");
    while (cur != NULL && cur->data != NULL) {
        Pair *p = cur->data;
        Pair_print(p);
        if (cur->next)
            printf(", ");
        cur = cur->next;
    }
    printf("]");
}
```

utils.h

```c
#ifndef __UTILS_H__
#define __UTILS_H__

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

#include "list.h"

FILE *open_file(char *path, char *mode);
void verify_malloc(void *ptr);
Node *extract_words_from_file(char *path);
void replace_words(char *path, Node *l);

#endif
```

utils.c

```c
/**
 * Utiles para el programa sustituir.
 *
 * Autor: Christopher Gómez.
 * Fecha: 13-06-2022.
 */
#define PAIRMATCH "%[^:\n]:%[^\n]\n"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "pair.h"
#include "list.h"
#include "utils.h"

/**
 * Verifica que un puntero no sea nulo. Si es nulo, termina el programa.
 *
 * @param ptr Puntero a verificar.
 */
void verify_pointer(void *ptr) {
    if (!ptr) {
        printf("Hubo en error asignando memoria.\n");
        exit(1);
    }
}

/**
 * Abre un archivo y verifica que no haya habido errores.
 *
 * @param path Ruta del archivo a abrir.
 * @return Puntero al archivo abierto.
 */
FILE *open_file(char *path, char *mode) {
    FILE *fp = fopen(path, mode);
    if (!fp) {
        printf("Hubo en error abriendo el archivo %s.\n", path);
        exit(1);
    }
    return fp;
}

/**
 * Lee un archivo con el formato indicado y extrae pares de cadenas de
 * caracteres en una lista ordenada en orden descendiente del tamaño de
 * la cadena.
 *
 * @param path: Cadena de caracteres con la ruta del archivo a leer.
 * @return Apuntador a la cabeza de una lista enlazada de pares de palabras.
 */
Node *extract_words_from_file(char *path) {
    FILE *fp;
    char p1[50], p2[50];
    Node *l;

    fp = open_file(path, "r");

    l = List_new();
    verify_pointer(l);

    /* Lee hasta encontrar ':' y escribe en p1,
    luego lee hasta encontrar un salto de línea. */
    while (fscanf(fp, PAIRMATCH, p1, p2) != EOF) {
        int p1_length = strlen(p1);
        char *word1 = malloc(p1_length + 1);
        char *word2 = malloc(strlen(p2) + 1);
        Pair *p;

        verify_pointer(word1);
        verify_pointer(word2);

        /* Aquí se guarda el par y se enlaza a la lista. */
        p = Pair_new(strcpy(word1, p1), strcpy(word2, p2));
        verify_pointer(p);

        if (!List_push(&l, p, p1_length)) {
            printf("Hubo en error agregando el par %s:%s a la lista.\n", word1, word2);
            exit(1);
        }
    }
    fclose(fp);

    return l;
}

/**
```

No hagas exit en una librería. El dueño del p

utils.c

```c
 * Lee un archivo y reemplaza en él todas las ocurrencias de las cadenas de
 * caracteres indicadas por otras, contenidas en una lista con pares de cadenas.
 *
 * El análisis del archivo se hace de izquierda a derecha, por lo que se
 * reemplazan primero las ocurrencias más a la izquierda, tomando como prioridad
 * a las cadena en su orden de aparición en la lista.
 *
 * Escribe en salida estándar el resultado.
 *
 * @param path: Cadena de caracteres con la ruta del archivo a leer.
 * @return Apuntador a la cabeza de una lista enlazada de pares de palabras.
 */
void replace_words(char *path, Node *head) {
    FILE *fp;
    char cur_char;

    fp = open_file(path, "r");

    /* Por cada letra del archivo a leer */
    while ((cur_char = fgetc(fp)) ≠ EOF) {
        Node *cur_node = head→data? head: NULL;
        long int cur_pos = ftell(fp) - 1;

        /* Por cada palabra en la lista */
        for (; cur_node; cur_node = cur_node→next) {
            char *cur_word = cur_node→data→first;

            /* Por cada letra en la palabra */
            for (;cur_char == *cur_word; cur_word++)
                cur_char = fgetc(fp);

            /* Si la palabra coincide, la reemplaza */
            if (*cur_word == '\0') {
                printf("%s", cur_node→data→second);
                fseek(fp, -1, SEEK_CUR);
                break;
            }

            /* Si no, pasa a la siguiente de la lista */
            fseek(fp, cur_pos, SEEK_SET);
            cur_char = fgetc(fp);
        }

        /* Si se llegó al final de la lista, se imprime el caracter */
        if (!cur_node) printf("%c", cur_char);

        /* Si se llegó al final del archivo, termina el ciclo */
        if (cur_char == EOF) break;
    }
}
```