Programación II. Listas. Ejercicio entregable
(semana 23-27 marzo 2020)

Dada la interfaz para trabajar con ***listas enlazadas simples*** que se proporciona más abajo, escribe el código C de una función main que realice las siguientes acciones:

- Crear una lista e imprimir su tamaño (debería ser 0).
- Insertar en ella, por el principio, los números pares desde 0 hasta 10 (ambos inclusive) en ese orden.
- Imprimir la lista, imprimiendo primero su tamaño y luego sus elementos.
- Crear otra lista e imprimir su tamaño (debería ser 0).
- Insertar en ella, por el principio, los números impares desde 1 hasta 9 (ambos inclusive) en ese orden.
- Imprimir la lista, imprimiendo primero su tamaño y luego sus elementos.
- Crear una tercera lista e imprimir su tamaño (debería ser 0).
- Ir extrayendo **alternativamente** elementos del comienzo de la lista 1 y del comienzo de la lista 2, e ir insertando cada uno de ellos en la tercera lista, también por el inicio. No hay que fijarse en el valor de los números, simplemente deben alternarse: uno de una lista, uno de la otra (y así sucesivamente).
- Imprimir las 3 listas.

En esta función se deben controlar los posibles errores (*) y se debe gestionar la memoria adecuadamente.
A continuación se indica un ejemplo de salida del programa:

```
Creating list 1... Size = 0
Inserting elements into list...
Printing list 1:
Size = 6. Elements: 10 8 6 4 2 0

Creating list 2... Size = 0
Inserting elements into list...
Printing list 2:
Size = 5. Elements: 9 7 5 3 1

Creating list 3... Size = 0
Combining elements from list 1 and list 2 into list 3...
Extracted element from list 1: 10
Extracted element from list 2: 9
Extracted element from list 1: 8
Extracted element from list 2: 7
Extracted element from list 1: 6
Extracted element from list 2: 5
Extracted element from list 1: 4
Extracted element from list 2: 3
Extracted element from list 1: 2
Extracted element from list 2: 1
Extracted element from list 1: 0

Printing list 1:
Size = 0. Elements:

Printing list 2:
Size = 0. Elements:

Printing list 3:
Size = 11. Elements: 0 1 2 3 4 5 6 7 8 9 10

Freeing memory...
```

Otras pruebas que puedes hacer para comprobar que funciona bien:
Insertar en la primera lista los elementos 0, 2, 4 y 6 y en la segunda 1 y 3.
Insertar en la primera lista los elementos 0 y 2 y en la segunda 1, 3, 5 y 7.
Insertar en la primera lista los elementos 0 y 2 y en la segunda ninguno.
No insertar ningún elemento en la primera lista e insertar en la segunda el 1.
No insertar ningún elemento en ninguna lista.

(*) Se puede crear una función auxiliar free_all_memory que reciba todos los punteros a la memoria reservada (o que se reservará) dinámicamente, y que libere todo aquello que no sea NULL.

list.h (disponible también como fichero adjunto en la tarea Moodle, con la documentación completa en inglés)

```
#ifndef LIST_H
#define LIST_H

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

typedef struct _List List;

/* Puntero a función que libera un elemento de cualquier tipo */
typedef void (*P_ele_free)(void*);

/* Puntero a función que, dado un puntero a un elemento, reserva memoria para
otro de ese mismo tipo y copia su valor */
typedef void *(*P_ele_copy)(const void*);

/* Puntero a una función que recibe un manejador de fichero y un puntero a
elemento e imprime el elemento en el fichero*/
typedef int (*P_ele_print)(FILE *, const void*);

/* INTERFAZ PÚBLICA DE LISTAS */

/* Crea y devuelve una nueva lista para trabajar con elementos que se liberan,
se copian y se imprimen con las funciones que recibe esta función como argumento
(f1, f2 y f3 respectivamente */
List *list_new(P_ele_free f1, P_ele_copy f2, P_ele_print f3);

/* Indica si la lista que recibe como argumento está vacía o no */
Boolean list_isEmpty(const List *pl);

/* Inserta un nuevo elemento al principio de la lista apuntada por pl.  e es el
puntero que almacena la dirección del  elemento a insertar (es decir, es el
puntero a ese elemento). Insert hace copia del elemento que recibe */
Status list_pushFront(List *pl, const void *e);

/* Inserta un nuevo elemento al final de la lista apuntada por pl.  e es el
puntero que almacena la dirección del  elemento a insertar (es decir, es el
puntero a ese elemento). Insert hace copia del elemento que recibe */
Status list_pushBack(List *pl, const void *e);

/* Extrae el primer elemento de la lista apuntada por pl y lo devuelve.  Extraer
no hace copia del elemento, simplemente devuelve un puntero al mismo */
void *list_popFront(List *pl);

/* Extrae el último elemento de la lista apuntada por pl y lo devuelve.  Extraer
no hace copia del elemento, simplemente devuelve un puntero al mismo */
void *list_popBack(List *pl);

/* Libera toda la memoria ocupada por la lista apuntada por pl */
void list_free(List *pl);

/* Devuelve el tamaño (n° de elementos) de la lista apuntada por pl */
int list_size(const List *pl);

/* Imprime los elementos de la lista apuntada por pl en el fichero abierto
identificado por el manejador fp */
int list_print(FILE *fp, const List *pl);

#endif     /* LIST_H */
```

Funciones para el manejo de ***números enteros***:

```
/* Libera la memoria ocupada por el entero al que apunta p */
void int_free(void *p);

/* Reserva memoria para un nuevo número entero, copia en esa memoria el valor
contenido en el entero apuntado por p y devuelve un puntero al nuevo número
entero creado */
void *int_copy(const void *p);

/* Imprime el contenido del entero apuntado por p en el fichero abierto
identificado por el manejador f. Devuelve el nº de caracteres impresos */
int int_print(FILE *f, const void *p);
```