Parcialito Proyecto 2 - Tema B

Resolver los ejercicios listados abajo, en una hora (o menos). El código resultante **no** debe tener memory leaks **ni** accesos (read, write o free) inválidos a la memoria. La entrega es a través de Jaime.

Ejercicio 1

Modificar el TAD `list` tal que éste provea una operación nueva cuya signatura es:

```
list_t list_remove_first(list_t list);
```

Esta operación borra el primer elemento de la lista dada como parámetro. Es muy parecida a `list_remove`, pero en vez de borrar el nodo buscándolo por el valor de un `index`, borra el nodo en la posición 0.

Al igual que con `list_remove`, el nodo que se borra hay que destruirlo completamente para evitar pérdida de memoria.

Ejemplo

Supongamos una variable de tipo `list_t`, de nombre `list`, que referencia una lista con los siguientes elementos:

```
[("perro", "cuadrupedo que ladra"), ("gato", "cuadrupedo que maulla"),
 ("loro", "alumno que habla en clase"), ("perro", "el mejor amigo del hombre")]
```

Las siguientes llamadas a `list_remove_first` modifican la lista como sigue:

```
list = list_remove_first(list)
```

hace que `list` ahora sea:

```
[("gato", "cuadrupedo que maulla"),
 ("loro", "alumno que habla en clase"), ("perro", "el mejor amigo del hombre")]
```

Luego, la llamada:

```
list = list_remove_first(list)
```

hace que `list` sea:

```
[("loro", "alumno que habla en clase"), ("perro", "el mejor amigo del hombre")]
```

Ahora bien, luego de la siguiente llamada:

```
list = list_remove_first(list)
```

`list` ahora contiene:

```
[("perro", "el mejor amigo del hombre")]
```

Por último, si se ejecuta una vez más, se obtiene la lista vacía: `[]`:

```
list = list_remove_first(list)
```

Es totalmente válido hacer más llamadas al método que borra el primer elemento, la lista no debería cambiar (siempre debería ser la lista vacía).

Para la implementación de este método **no** se deben hacer llamadas a otros métodos públicos del TAD lista, pero sí se pueden usar todos los métodos públicos de los otros TADs (es decir, los métodos del index, del data, y del pair).

Ejercicio 2

Crear un archivo `parcialito.c` que provea una función `main` que haga lo siguiente:

1. Crear una lista vacía, y agregarle cada uno de los elementos que se muestran abajo, usando `list_append`:

   - `("perro", "guau")`
   - `("gato", "miau")`
   - `("loro", "prrrr")`
   - `("perro", "super guau")`

   Ayuda: Para crear índices y datos usando cadenas de texto estáticas como las dadas arriba (es decir, strings que no usan memoria dinámica, por ende no hay que liberarlos), pueden hacer lo siguiente:

   ```
   list = list_append(list, index_from_string("perro"),
                      data_from_string("animal que ladra"));
   /* no hay que liberar ni el index ni el data, ni los strings estaticos */
   ```

2. Imprimir la lista en su estado actual usando la siguiente función (copiar y pegar en el código a entregar):

   ```
   void print(list_t list) {
       list_dump(list, stdout);
       printf("\n============\n");
   }
   ```

3. Llamar 5 veces al método nuevo `list_remove_first`, imprimiendo la lista después de cada llamada, invocando al `print` del punto anterior:

Lo de arriba asegura que cuando se ejecute el main, se imprima lo siguiente en pantalla (esto lo va a requerir el test de aceptación en Jaime):

```
perro: guau
gato: miau
loro: prrrr
perro: super guau
============
gato: miau
loro: prrrr
perro: super guau
============
loro: prrrr
perro: super guau
============
perro: super guau
============

============

============
```