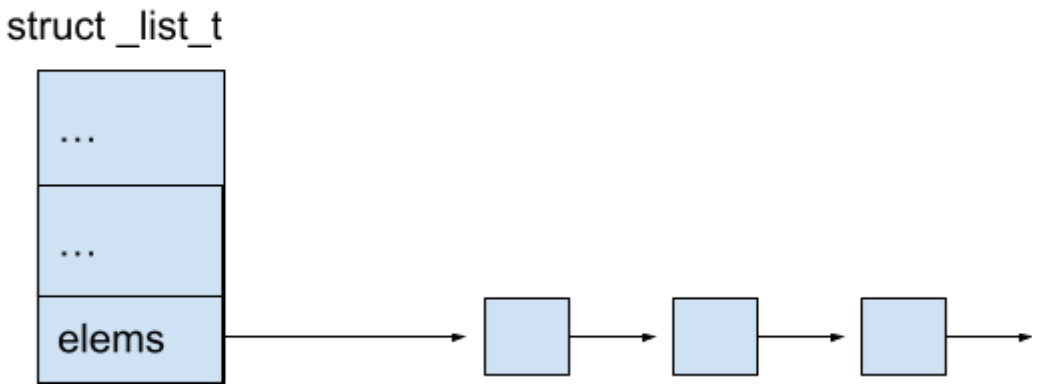
Córdoba, 2 de Junio del 2023

Algoritmos y Estructuras de Datos II

Parcial 2 - TAD Lista de Asociaciones

Ejercicio 1: Implementación del TAD

Implementar el TAD ***list*** que representa una lista de asociaciones (un *diccionario generalizado*). El TAD almacenará claves (key) y datos (value) cumpliendo con la condición de que cada palabra tiene un solo dato asociado. Este TAD debe implementarse mediante una lista enlazada de nodos, siguiendo la siguiente estructura:



La estructura principal contiene la lista enlazada `elems`, en la cual se guarda cada nodo. Cada uno de los nodos de la lista debe contener tanto el *key* como el *value*. En nuestra implementación, las claves serán de tipo `unsigned int`. La lista debe mantener en todo momento sus nodos **ordenados** según la *key*. Esto debe ser tenido en cuenta al implementar `list_add`.

Las operaciones del TAD Diccionario se listan a continuación:

Función	Descripción
`list_t list_empty(void)`	Crea una lista vacía
`list_t list_isEmpty(list_t list)`	Devuelve `true` si la lista está vacía, o `false` si tiene elementos
`list_t list_add(list_t list, int key, elem value)`	Agrega una nueva asociación entre `key` y `value`. En caso que `key` ya esté en lista, se actualiza su `value` asociado.
`elem list_search(list_t list, int key)`	Devuelve el valor asociado a `key`. Esta clave debe existir como precondición.

`bool list_exists(list_t list, int key)`	Indica si el valor `key` está en la lista `list`
`unsigned int list_length(list_t list)`	Devuelve la cantidad de valores que tiene actualmente la lista `list`
`list_t list_remove(list_t list, unsigned int key)`	Elimina la clave `key` de la lista. Si la clave no se encuentra devuelve la lista sin cambios.
`list_t list_remove_all(list_t list)`	Elimina todas los valores de la lista `list`
`void list_to_array(list_t list)`	Devuelve un arreglo en memoria dinámica con cada `value` contenido en `list`
`list_t list_destroy(list_t list)`	Destruye la instancia `list` liberando toda la memoria utilizada.

AYUDAS:

- El unico archivo que deben completar es `list.c`
- En `list.c`, se incluyen las signaturas de varias funciones `static` que creemos pueden ser muy útiles a la hora de completar el TAD. No es necesario que las implementen, pero hacerlo puede facilitar la tarea.
- `list_exists` ya se encuentra implementada a modo de ejemplo.
- Se incluye un **`Makefile`**.

Para verificar que la implementación del TAD funciona correctamente, se provee el programa (**main.c**) mediante el cual pueden ejecutar cada función del TAD, inclusive cargando un diccionario de ejemplo de la carpeta de inputs.

Al llamar al programa, se debe pasar como argumento el nombre del input correspondiente:

```
$ ./listrun input/example-abcde.in
```

Esto lee la lista desde el archivo, carga la lista en un TAD, lo convierte en un array, e imprime el resultado:

```
La lista 'input/example-abcde.in' tiene los siguientes 5 valores:
[ a, b, c, d, e ]
La lista 'input/example-abcde-2.in' tiene los siguientes 5 valores luego
de remover los seleccionados:
[ a, b, c, d, e ]
```

Al llamar al programa, puede pasarse más argumentos, para testear el `list_remove`. Estos argumentos extra deben representar las `keys` que se desea borrar:

```
$ ./listrun input/example-abcde.in 3 10
La lista 'input/example-abcde.in' tiene los siguientes 5 valores:
[ a, b, c, d, e ]
```

```
La lista 'input/example-abcde-2.in' tiene los siguientes 4 valores luego
de remover los seleccionados:
[ a, b, d, e ]
```

El programa resultante no debe dejar *memory leaks ni lecturas/escrituras inválidas*.

Consideraciones:

- Se recomienda usar las herramientas **`valgrind`** y **`gdb`**.
- Si el programa no compila, no se aprueba el parcial.
- Los *memory leaks* bajan puntos.
- Entregar un código muy improlijo resta puntos.
- Si **`list_length()`** no es de orden constante **baja muchísimos puntos.**
- Para promocionar **se debe** hacer una invariante que chequee la propiedad fundamental de la representación del diccionario:
    - El invariante debe chequear al menos:
        - Verificación de caso base.
        - Consistencia entre componentes del **`struct`**.
        - Propiedad fundamental de la lista. Si no es claro cuál es, releer el enunciado.