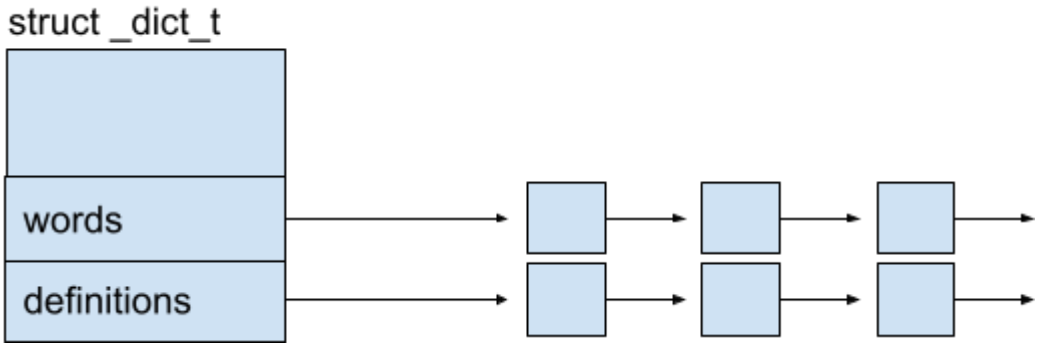
Córdoba 28 de Junio del 2022

Algoritmos y Estructuras de Datos II

Recuperatorio Tema B - TAD Diccionario

Ejercicio 1: Implementación del TAD Diccionario

Implementar el TAD ***dict*** que representa un diccionario de palabras.el TAD almacenará palabras y definiciones (cada palabra tiene exactamente una definición). Este tad debe implementarse mediante dos listas enlazadas comunes de nodos, siguiendo la siguiente estructura:



La estructura principal contiene dos listas enlazadas, una que guarda las palabras, y otra las definiciones. Los índices de ambas listas deben estar *sincronizados*, lo que significa que para la palabra en la posición `k` de la primera lista, la definición se encuentra en el nodo `k` de la segunda lista.

El diccionario debe mantener en todo momento sus nodos **ordenados** según la palabra (como en un diccionario real). Esto debe ser tenido en cuenta al implementar `dict_add`.

Las operaciones del TAD Diccionario se listan a continuación:

Función	Descripción
`dict_t dict_empty(void)`	Crea un diccionario vacío
`dict_t dict_add(dict_t dict, string word, string def)`	Agrega una nueva palabra `word` junto con su definición `def`. En caso que `word` ya esté en el diccionario, se actualiza su definición con `def`.
`value_t dict_search(dict_t dict, string word)`	Devuelve la definición de la palabra `word` contenida en el diccionario `dict`. Si la palabra no se encuentra devuelve `NULL`
`bool dict_exists(dict_t dict, string word)`	Indica si la palabra `word` está en el diccionario `dict`

`unsigned int dict_length(dict_t dict)`	Devuelve la cantidad de palabras que tiene actualmente el diccionario `dict`
`dict_t dict_remove(dict_t dict, string word)`	Elimina la palabra `word` del diccionario. Si la palabra no se encuentra devuelve el diccionario sin cambios.
`dict_t dict_remove_all(dict_t dict)`	Elimina todas las palabras del diccionario `dict`
`void dict_dump(dict_t dict, FILE *file)`	Escribe el contenido del diccionario `dict` en el archivo `file`
`dict_t dict_destroy(dict_t dict)`	Destruye la instancia `dict` liberando toda la memoria utilizada.

AYUDAS:

- El unico archivo que deben completar es `dict.c`
- Se incluye el TAD **string** completo como parte del *kickstart*. Utilizar este TAD para representar las palabras. Esta implementación incluye métodos para comparar palabras, lo cual es útil a la hora de agregar ordenadamente.
- En `dict.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.
- `dict_dump` y `dict_destroy` ya se encuentran implementadas a modo de ejemplo.
- `dict_dump` no debe abrir el archivo ni cerrarlo, solo usar el FILE * para guardar contenido al archivo.
- Se incluye un **`Makefile`** completo.

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.

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 **`dict_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 del Diccionario. Si no es claro cuál es, releer el enunciado.