Proyecto 2.b
Listas Enlazadas

Algoritmos y Estructuras de Datos II - Laboratorio

Docentes: Natalia Bidart, Matías Bordese, Diego Dubois,
Leonardo Rodríguez, David Arch, Maximiliano Bustos.

Objetivo

El objetivo de este proyecto es extender el código de la primer parte del proyecto 2, para proveer una implementación en C de los siguientes TADs:

- Listas enlazadas[1].
- Tuplas[2].

Instrucciones

En la figura 1 se muestra un diagrama análogo al presentado en el proyecto anterior, con la diferencia de que ahora los TADs a implementar son las listas enlazadas y las tuplas (ver los módulos resaltados en **rojo**).

Las tareas concretas son las siguientes:

- Crear archivos `linked_list.c` y `tuple.c` en donde se provee implementaciones de los tipos de datos lista enlazada y tupla, respectivamente, cumpliendo al pie de la letra las especificaciones dadas en las cabeceras (ya provistas por la cátedra en el proyecto anterior) `linked_list.h` y `tuple.h`.
- Las implementaciones de los tipos mencionados arriba debe ser oculta. Es decir, deben usar la técnica de punteros a estructuras cuando implementen los TADs requeridos.

[1] Ver en wikipedia
[2] Ver en wikipedia

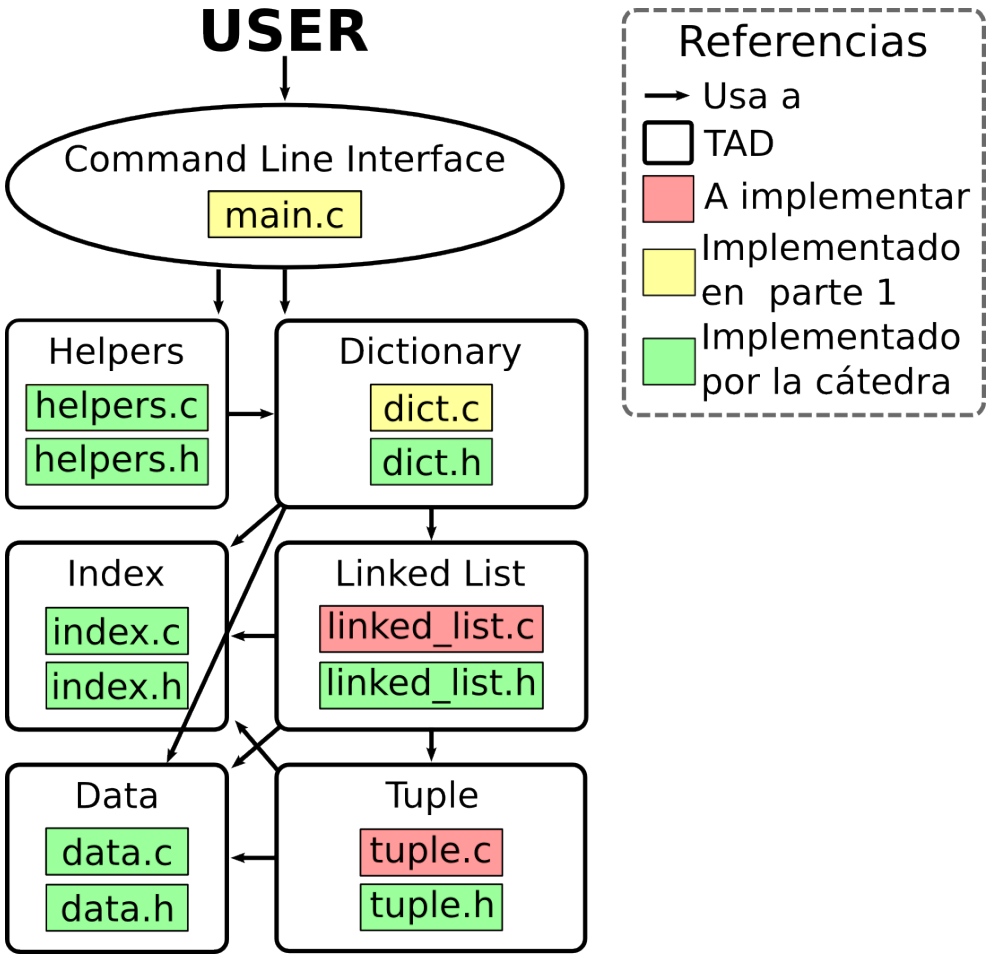

Figura 1: Diagrama de TADs

- El resto de los archivos no deben sufrir cambios (excepto que necesiten arreglar algún bug). Es decir, todos los demás TADs deben quedar igual que en el proyecto anterior (inclusive la interfaz con el usuario).
- Testear esta nueva implementación, confirmando que siempre estén usando código objeto enteramente producido por ustedes. Para ello, antes de compilar, asegurarse de que todos los archivos `.o` fueron borrados del directorio, corriendo este comando (dentro del directorio del proyecto):

  ```
  $ rm *.o
  ```

  o equivalentemente, usando el comando `make` (dado en el teórico del lab):

  ```
  $ make clean
  ```

- El programa resultante debe estar libre de "memory leaks". Usar `valgrind` para confirmar tal situación.

Sobre el TAD `tuple`

El TAD `tuple` a implementar en C es equivalente a la siguiente definición en pseudo-código:

```
(index_t, data_t)
```

Es decir, que la estructura `struct _tuple_t` deberá ser elegida de manera tal que se pueda almacenar dos miembros: un `index_t` y un `data_t`.

Sobre el TAD `linked_list`

El TAD `linked_list` a implementar en C es equivalente a la siguiente definición en pseudo-código:

```
[(index_t, data_t)]
```

que dada la definición de la subsección anterior, se puede escribir de manera equivalente:

```
[tuple_t]
```

Es decir, que la estructura `struct _linked_list_t` deberá ser elegida de manera tal que se pueda almacenar nodos que contengan un `tuple_t` adentro.

Puntos $\star$

Recomendación usual: no empezar con los puntos estrella hasta que no se tengan absolutamente resueltos los puntos obligatorios, incluyendo la corroboración de ausencia de *memory leaks*.

Ejercicio $\star$ 1 *¿Se puede implementar algún método de ordenación con el TAD lista enlazada? Si es así, utilizarlo para que en todo momento la lista permanezca ordenada.*

¿Este cambio produce una mejora en la complejidad de la búsqueda, inserción y/o borrado de palabras en el diccionario?

Ejercicio $\star$ 2 *Detallar en un breve informe (pero* **no** *implementar) qué cambios harían falta para implementar una segunda versión de la lista enlazada tal que almacene elementos genéricos. Es decir, que el tipo de los datos que guarda cada nodo debería ser genérico.*

Incluir cómo afectaría a la interfaz del módulo `linked_list` *y en qué módulos repercutiría este cambio.*

Recordar

- Entrega y evaluación: Martes 8 de Mayo (ojo que el martes 1 de Mayo es feriado!).
- Comentar e identar el código apropiadamente, siguiendo el estilo de código ya provisto por la cátedra.
- Todo el código tiene que usar la librería estándar de C, y no se puede usar extensiones GNU de la misma.
- El programa resultante **no** debe tener *memory leaks* **ni** accesos (read o write) inválidos a la memoria.