Proyecto 2
Tipos Abstractos de Datos

Algoritmos y Estructuras de Datos II - Laboratorio

Docentes: Natalia Bidart, Matías Bordese, Diego Dubois, Leonardo Rodríguez.

1. 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 (Ver en link wikipedia).
- Pares de valores (Ver en link wikipedia).

2. Instrucciones

En la figura 1 se muestra un diagrama análogo al presentado en la primer parte del proyecto 2, 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 de esta segunda parte del proyecto 2 son las siguientes:

- Crear archivos `list.c` y `pair.c` en donde se provee las implementaciones de los tipos de datos lista enlazada y tupla, respectivamente, cumpliendo al pie de la letra las especificaciones dadas en los archivos de cabecera (ya provistas por la cátedra en el proyecto anterior) `list.h` y `pair.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, tal como se hizo en la primer parte con el TAD `dict`.
- El resto de los archivos (todos los `.h` y los `.c` dados por la cátedra o implementados en la parte 1) 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 el nuevo ejecutable, confirmando que siempre estén usando código objeto enteramente producido por ustedes. Para ello, antes de compilar, asegurarse de lo siguiente:
  - que dentro del archivo `Makefile`, la variable `LIBS` no tiene ningún `.o` especificado
  - que todos los archivos `.o` fueron borrados del directorio, corriendo el comando (dentro del directorio del proyecto):
    ```
    $ rm *.o
    ```

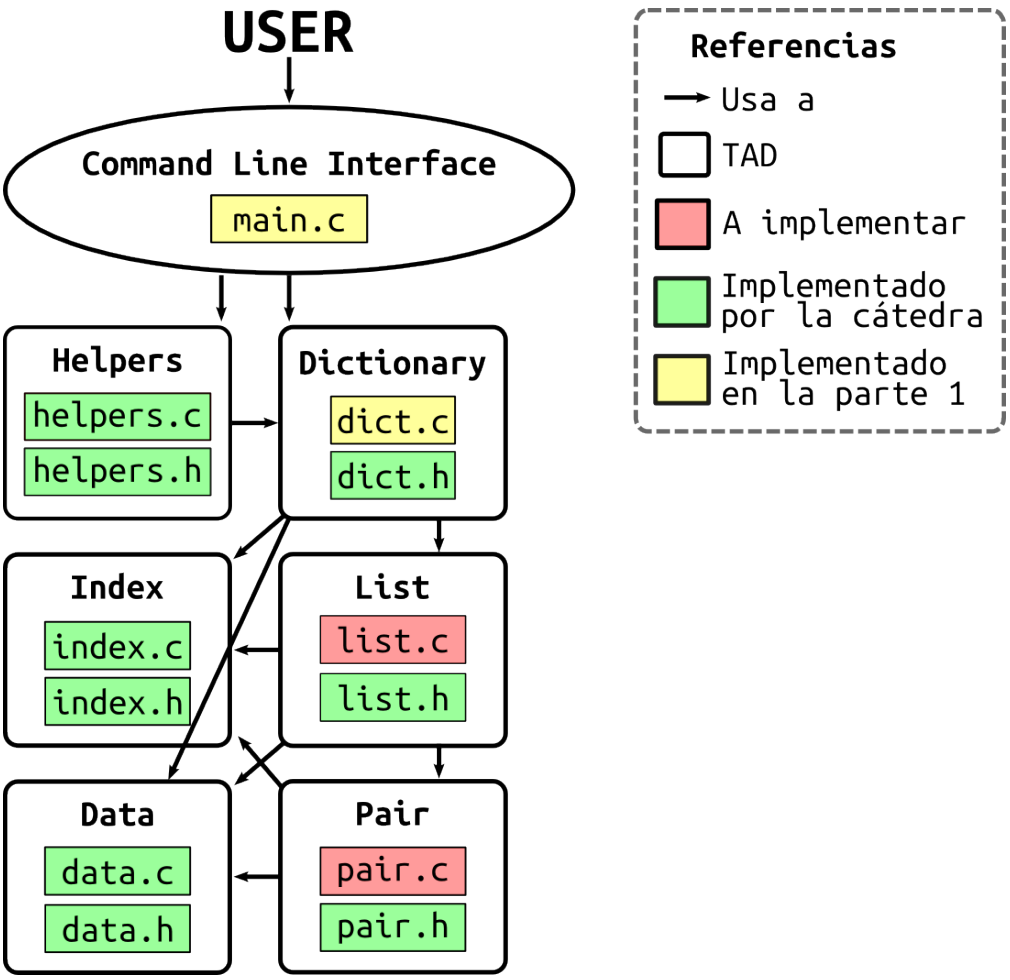

Figura 1: Diagrama de TADs

- El programa resultante debe estar libre de "memory leaks". Usar `valgrind` para confirmar tal situación, como en la primer parte, corriendo el comando:

  ```
  $ valgrind --leak-check=full --show-reachable=yes ./dictionary
  ```

2.1. Sobre el TAD `pair`

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

```
(index_t, data_t)
```

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

2.2. Sobre el TAD `list`

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

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

que dada la definición del tipo nuevo `pair_t` de la subsección anterior, se puede escribir de manera equivalente:

```
[pair_t]
```

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

Para definir la estructura del TAD `list`, seguir el apunte del teórico 05.listasenlazadas.pdf y mapear a C los dos tipos definidos como `node` y `list`. Notar que el tipo `node` es un detalle de implementación de la lista, y no debe ser expuesto bajo ningún concepto en el archivo de cabecera `list.h`.

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* y de accesos inválidos a memoria.

Ejercicio $\star$ 1 *¿Se puede cambiar la implementación del TAD lista enlazada de manera tal que en todo momento la lista permanezca ordenada? Si sí, hacer los cambios necesarios para que esto ocurra. Notar que esto* **no** *significa ordenar la lista completa cada vez que se la modifica.*

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

Ayuda: *Dependiendo de lo que piensen en este punto, pueden que necesiten agregar un método nuevo en el TAD* `list`*, y por ende modificar alguna llamada a este TAD desde* `dict`*.*

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 no sea un tipo concreto.*

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

Recordar

- Entrega y evaluación: Martes 7 de Mayo de 2013.
- Comentar e indentar el código apropiadamente, siguiendo el estilo de código ya provisto por la cátedra en los archivos dados.
- 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.