Proyecto 3
Diccionario implementado sobre Cinta de elementos

Algoritmos y Estructuras de Datos II
Laboratorio

19 de abril de 2011

El objetivo de este proyecto es usar las abstracciones de cintas para el manejo de secuencias. También será un objetivo secundario el aprender a interpretar y utilizar código hecho por terceras personas.

Este proyecto consiste en implementar un TAD diccionario sobre listas de asociaciones. A diferencia del proyecto anterior[1] la lista de asociaciones deberá ser implementada sobre un TAD cinta de elementos para manipular listas enlazadas (como se vio en el teórico del taller). Como las cintas no están en el lenguaje, va a haber que implementarlas.

Además se deberá proveer las funcionalidades de lectura y escritura del diccionario en archivos de disco. Para ello se darán ya programadas una cinta de lectura y otra de escritura de tuplas en disco.

El diseño de estas modificaciones será el siguiente:

- Hacer un nuevo TAD *cinta de tuplas en memoria* como se vio en el teórico. Se debe escribir un archivo `cinta.h` con la parte pública del mismo un `cinta.c` con su implementación. Un esquema del `.h` puede ser el siguiente:

```
#include "tuple.h"
#include "bool.h"

typedef struct celda *aList;
typedef struct sCinta *Cinta;

aList
aList_vacia(void);
/*
DESC: Constructor de lista vacia.
*/
```

[1] En el proyecto anterior la lista de asociaciones se implementó con arreglos, los cuales ya están disponibles en C.

```
aList
aList_destroy(aList l);
/*
DESC: Destruye todos los elementos de la lista (vacia o no)
      con su contenido.
*/

Cinta
cinta_create(aList l);
/*
DESC: Constructor de cinta.
PRE: { l = L }
  c = cinta_create(l);
POS: { Pi(c) ++ Pd(c) = L }
*/

void
cinta_arr(Cinta c);
/*
DESC: Arranca la cinta.
PRE: { L = Pi(c) ++ Pd(c) }
   cinta_arr(c);
POS: { Pi(c)=[] /\ Pd(c)=L }
*/

Bool
cinta_fin(Cinta c);
/*
DESC: Pregunta si llego al final.
DEF: cinta_fin(c) == (Pd(c) = [])
*/

void
cinta_av(Cinta c);
/*
DESC: Avanza la cinta.
PRE: { Pi(c) = L1 /\ Pd(c) = e:L2 }
   cinta_av(c);
POS: { Pi(c)=L1:e /\ Pd(c)=L2 }
*/

Tuple
```

```
cinta_elec(Cinta c);
/*
DESC: Elemento corriente de la cinta.
DEF: Pd(c) /= [] ==> cinta_elec(c) = head(Pd(c))
*/

void
cinta_ins(Cinta c, Tuple t);
/*
DESC: inserta una tupla.
PRE: { t= T; Pi(c) = L1 /\ Pd(c) = L2 }
   cinta_ins(c,t);
POS: { t=T /\ Pi(c)=L1 /\ Pd(c)=t:L2 }
*/

void
cinta_del(Cinta c);
/*
DESC: Borra el elemento corriente.
PRE: { Pi(c) = L1 /\ Pd(c) = e:L2 }
   cinta_del(c);
POS: { Pi(c)=L1 /\ Pd(c)=L2 }
*/

aList
cinta_destroy(Cinta c);
/*
DESC: Destructor.
PRE: { Pi(c) = L1 /\ Pd(c) = L2 }
   l = cinta_destroy(c);
POS: { l = Pi(c)++ Pd(c) }
*/
```

Nota: Notar que el tipo `aList` solo tiene el contructor de lista vacía y además, el destructor del tipo `Cinta` devuelve la lista actual contenida en la cinta. Esto hace que el usuario de esta librería solo pueda modificar los elementos en la lista unicamente utilizando las operaciones de la cinta (`cinta_ins` y `cinta_del`).

- Con este TAD implementar todas las funciones de la lista de asociaciones utilizando las mismas signaturas que en el proyecto pasado. O sea que por ahora hay que dejar el mismo archivo `listaAsoc.h` del proyecto pasado y cambiar el `listaAsoc.c` para que implemente la lista de asociaciones con una cinta, en vez de un arreglo.

- Dejando todos los demás TAD's igual que en el proyecto anterior (inclusive la interfase con el usuario) testear esta nueva implementación.

Ejercicio ⋆ 1 *¿Se puede implementar algún método de ordenación con el TAD cinta? Si es así, utilícelo para que en todo momento la lista permanezca ordenada.*
¿Este cambio produce una mejora en la complejidad de la búsqueda e inserción de palabras en el diccionario?

La segunda etapa del proyecto será incluir la funcionalidades de lectura y escritura del diccionario en disco. Para ello se brindará ya programadas las cintas de lectura y escritura de tuplas en disco. Las mismas se encuentran en un archivo comprimido ubicado en la página de la materia junto a este enunciado. Para usarlos se debe tener en cuenta lo siguiente:

- Se encuentran implementados dos TAD's: cinta de lectura de tuplas (par `Key Data`) y cinta de escritura de tuplas en disco. La primera está programada en los archivos `cr.h` y `cr.c`. La segunda está programada en los archivos `cw.h` y `cw.c`[2].

- Estos programas usan algunos de los TAD's programados en el proyecto anterior. Para poder compilar estos programas hay que copiarlos en el mismo directorio donde están programados los TAD's `Key`, `Data`, `Tuple` y `Bool` (pueden estar los otros TAD's también).

- Tener en cuenta que para que estos programas compilen correctamente **se deben haber respetado las interfaces** de estos TAD's dadas en el enunciado del proyecto anterior. Si esto no se hizo la cinta no va andar y va a tener que rehacer parte del proyecto anterior `:(`.

- La especificación de cada función de estos dos nuevos TAD's están en los archivos `cr.h` y `cw.h`. Leerlas detenidamente. También puede ver las implementaciones en los archivos `.c` correspondientes. El archivo `pru.c` también sirve como muestra de uso de los mismos (probar compilar y ejecutar este programa). Cualquier duda consulte a su ayudante.

- El formato de los archivos que guardan las tuplas es:

  ```
  <palabra1> : <definicion1>
  <palabra2> : <definicion2>
             ...
  ```

  Los archivos se pueden modificar con un editor de texto plano.

  Junto con los programas de las cintas hay un diccionario y un programa de prueba como ejemplo del funcionamiento básico de la librería. Se recomienda estudiarlo.

- Con las cintas de lectura y escritura implementar las funciones

[2]También hay un archivo llamado error.h que contiene un formato unificado de salida para los errores.

```
ListaAssoc
la_fromFile(char *nomfile);

void
la_toFile(char *nomfile, ListaAsoc la);
```

en el TAD lista de asociaciones respectivamente. Las mismas leen y escriben en el archivo `nomfile` la lista respectivamente. Ver que para hacer esto va a tener que modificar el `.h` y el `.c` de este TAD.

- Con estas últimas dos funciones implementar las siguientes

```
Dict
dict_fromFile(char *nomfile);

void
dict_toFile(char *nomfile, Dict d);
```

en el TAD diccionario. Las mismas leen y escriben en el archivo `nomfile` el diccionario respectivamente. Ver que para hacer esto va a tener que modificar el `.h` y el `.c` de este TAD.

- Agregar a la interfase con el usuario la posibilidad de leer y escribir diccionarios en disco.

Ejercicio $\star$ 2 *Los TAD's implementados en los archivos* `cr.c` *y* `cw.c` *hacen uso de librerías externas para la visualización de errores y la librería* `assert`*. Investigue estas librerías y úselas en todos los programas.*

Para probar el diccionario se puede usar el que está en la página de la materia.

Además, de forma general, para hacer cada TAD se deberá tener en cuenta:

- Todos los TAD's deberán estar implementados con la técnica de punteros a registros vista en el teórico del laboratorio.
- Cada TAD se deberá escribir en un par de archivos separados, un `.h` ("headers") y un `.c`.
- Para implementar correctamente los TAD's, cada `.h` deberá exportar únicamente las funcionalidades del TAD que define, ocultando todos los aspectos que tienen que ver con su implementación.
- No definir las estructuras que implementan los TAD's en los archivos "headers" ya que allí solo va la parte pública del TAD y no los detalles de implementación.
- Los programas deben estar libres de "memory leaks".

- Recordar que todos los `.h` deben tener su cerradura `ifndef`.
- Los programas deben compilar con los parametros `-Wall -ansi -pedantic` del compilador `gcc` sin ningún error ni alerta.
- Para obtener nota B el programa debe funcionar sin errores y debe cumplir con todos los puntos anteriores.
- Para obtener nota B^+ el programa debe funcionar sin errores, debe cumplir con todos los puntos anteriores y se deben hacer todos los puntos estrella.