Algoritmos y Estructuras de Datos II 2020

Laboratorio 3: Tipos Abstractos de Datos

Importante: Para resolver los ejercicios de este laboratorio correctamente te recomendamos leer previamente el material del teórico sobre TAD's.

Ejercicio 1: TAD Contador

Dentro de la carpeta ej1 vas a encontrar los siguientes archivos:

- **counter.h** : Contiene la especificación del TAD Contador.
- **counter.c** : Contiene la implementación del TAD Contador.
- ***main.c*** : Contiene al programa principal que lee uno a uno los caracteres de un archivo chequeando si los paréntesis están balanceados.

Parte a) Implementación de TAD Contador

Para esta parte es necesario que abras los archivos counter.h y counter.c e implementes cada uno de los constructores y operaciones cumpliendo la especificación dada en counter.h.

A continuación se listan algunos aspectos a tener en cuenta antes de resolver el ejercicio:

Encapsulamiento
Lo primero que debemos observar es la forma en la que logramos mantener separadas la especificación del TAD de su implementación. Cuando definimos un TAD es deseable garantizar **ENCAPSULAMIENTO**, es decir, que solamente se pueda acceder y/o modificar a través de las operaciones provistas. **En C, la forma de conseguir encapsular es utilizando punteros.**

Manejo de memoria
Para definir constructores, destructores y operaciones de copia será necesario hacer manejo de memoria. Para el manejo de memoria en C recomendamos que investigues el uso de **malloc** y **free**. Podés ejecutar `$ man malloc` en tu terminal para entender cómo deberías usar malloc, que a diferencia del teórico recibe como argumento la cantidad de memoria que se necesita. En nuestro caso, necesitaremos espacio para almacenar un valor de tipo **struct _counter**. Para obtener el tamaño de un tipo en C recomendamos utilizar la función **sizeof**.

Procedimientos
Los procedimientos (como se ven en el teórico) no existen tal cual en C. Para ello en este ejercicio definimos funciones con tipo de retorno **void,** es decir, funciones que no devuelven ningún valor al llamarlas.

Parte b) Precondiciones en C

En esta parte debemos definir en counter.c todas las precondiciones especificadas en counter.h.
Las precondiciones como se ven en el teórico son implementadas en C usando **assert** al comienzo de la implementación del constructor/operación correspondiente en counter.c. Recomendamos investigar el uso de assert ejecutando `$ man assert` .

Parte c) Uso de TAD Contador para el chequeo de paréntesis balanceados

Para esta parte es necesario que abras el archivo main.c . Allí deberás entender qué es lo que hace la función **matching_parentheses** y llamar al constructor y destructor del contador donde consideres necesario.
¡**Es muy importante** llamar al destructor del TAD una vez este no sea más necesario para poder liberar el espacio de memoria que tiene asignado!

Una vez implementados los incisos a), b) y c), compilá ejecutando:

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c counter.c
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -o counter *.o main.c
```

y ya podés ejecutar el programa, corriendo:

```
$ ./counter input/<file>.in
```

Siendo **<file>** alguno de los nombres de archivo dentro de la carpeta **input.** Asegurate que para aquellos archivos con paréntesis balanceados ejecutar el programa imprima en pantalla *"Parentheses match."* y para aquellos con paréntesis no balanceados imprima *"Parentheses mismatch."*

Ejercicio 2: TAD Lista

Dentro de la carpeta ej2 vas a encontrar los siguientes archivos:

- ***main.c*** : Contiene al programa principal que lee los números de un archivo para ser cargados en nuestra lista y obtener el promedio.
- ***array_helpers.h :*** Contiene descripciones de funciones auxiliares para manipular arreglos.
- ***array_helpers.c :*** Contiene implementaciones de dichas funciones.

Parte a) Especificación de TAD Lista

Para este ejercicio debes crear un archivo **list.h,** especificando allí **todos** los constructores y operaciones vistos sobre el TAD Lista en el teórico.

Te recomendamos que definas el nombre del TAD como `list`, ya que en el archivo main.c lo encontrarás mencionado así.

Existe una diferencia con respecto al teórico con respecto a nuestra TAD Lista en C.
Para simplificar la implementación, nuestras listas serán solamente de tipo **int,** es decir, no soportaremos polimorfismo. Si bien el tipo será fijo (**int**), una buena idea es definir un tipo en list.h, usando **typedef.** Un ejemplo de esto sería definir `typedef int type_elem;` y utilizar **type_elem** en vez de **int** en todos los constructores/operaciones.

Una diferencia con respecto al ejercicio 1 es la **especificación de procedimientos.** En este caso, aquellos procedimientos que modifiquen la lista deben devolver la lista resultante (esto es una limitación de C). Pensar por qué es necesario.

No te olvides de:

- Garantizar encapsulamiento de tu TAD.
- Especificar una función de destrucción y copia.
- Especificar las precondiciones.

Parte b) Implementación de TAD Lista

Para este ejercicio debes crear un archivo **list.c,** e implementar cada uno de los constructores y operaciones declaradas en el archivo list.h . La implementación debe ser como se presenta en el teórico, es decir, utilizando punteros (listas enlazadas).

Parte C) Uso de TAD Lista

Para este ejercicio debes abrir el archivo **main.c,** e implementar las funciones **array_to_list y average.** Para ello, es **MUY IMPORTANTE** que primero reemplaces todas las ocurrencias de **your_list_type** por el nombre que hayas elegido para tu TAD Lista.

Para la implementación de **average** te sugerimos que revises la definición del teórico.

Una vez implementados los incisos a), b) y c), compilá ejecutando:

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c list.c array_helpers.c
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -o average *.o main.c
```

y ya podés ejecutar el programa, corriendo:

```
$ ./average input/<file>.in
```

Siendo **<file>** alguno de los nombres de archivo dentro de la carpeta **input.** Asegurate que el valor de los promedios que se imprimen en pantalla sean correctos y animate a definir tus propios casos de input.