Córdoba 1 de Junio del 2021

Segundo Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA A

Ejercicio 1

Completar en el archivo **`balanced.c`** el código necesario para que funcione el algoritmo de chequeo de delimitadores balanceados. En particular se debe programar la función:

```
static char * take_delimeters(char *expression, size_t *count)
```

que dada una cadena `expression` debe devolver una nueva cadena (en nueva memoria) cuyo contenido debe ser el resultado de quedarse solo con los delimitadores que se encuentran en `expression`. Además debe dejar en `*count` la cantidad de delimitadores que se encontraron (o en otras palabras, el tamaño de la cadena devuelta). Por ejemplo:

```
expression = "{[(2 * 3) + 4] - (7* 8)} / 2";
only_delimeters = take_delimeters(expression, &n_delimeters);
{only_delimeters --> "{[()](()}" && n_delimeters == 8}
```

El programa resultante no debe dejar *memory leaks*.

Una vez compilado el programa puede probarse ejecutando:

```
$ ./balanced "{[(2 * 3) + 4] - (7* 8)} / 2"
```

Obteniendo como resultado:

```
la expresión está balanceada
```

Otro ejemplo de ejecución:

```
$ ./balanced "(x + y) * (x - y"
```

que genera la siguiente salida

```
la expresión NO está balanceada
```

Córdoba 1 de Junio del 2021

Segundo Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA A

Ejercicio 2

Implementar el TAD *Truco-Card* que representa una carta de las que se utilizan para jugar al juego de naipes denominado *Truco*. No es necesario conocer el juego, solo se deben seguir ciertas pautas que se detallan a continuación.

Imagen ilustrativa de un antiguo juego de PC que simulaba partidas de truco

El TAD tiene la siguiente interfaz

Función	Descripción
`truco_card truco_card_new(int num, char p)`	Crea una carta con número `num` y palo `p`
`bool truco_card_win(truco_card a, truco_card b)`	Indica si la carta `a` le gana a la carta `b`
`bool truco_card_tie(truco_card a, truco_card b)`	Indica si las cartas empatan
`void truco_card_dump(truco_card c)`	Muestra la carta en la pantalla
`truco_card truco_card_destroy(truco_card c)`	Destruye una carta

Deben definir la estructura de representación interna `struct _s_truco_card` así como la invariante de representación que se debe verificar cuando corresponda. Las cartas representan la baraja española que tienen numeración entre 1 y 12 y una figura denominada “palo” que puede ser “espada”, “oro”, “basto” o “copa”. En el juego de naipes “Truco” se juega sin las cartas con numeración 8 y 9, por lo que no son cartas válidas y no debe poder crearse una instancia con esos valores. Para poder determinar el resultado de las funciones *boolenas* `truco_card_win()` y `truco_card_tie()` se debe seguir la siguiente jerarquía de cartas:

Rango	Cartas			
15	1 de espada			
14	1 de basto			
13	7 de espada			
12	7 de oro			
11	3 de copa	3 de basto	3 de espada	3 de oro
10	2 de copa	2 de basto	2 de espada	2 de oro
9	1 de copa		1 de oro	
8	12 de copa	12 de basto	12 de espada	12 de oro
7	11 de copa	11 de basto	11 de espada	11 de oro
6	10 de copa	10 de basto	10 de espada	10 de oro
-	9 de copa	9 de basto	9 de espada	9 de oro
-	8 de copa	8 de basto	8 de espada	8 de oro
3	7 de copa		7 de basto	
2	6 de copa	6 de basto	6 de espada	6 de oro
1	5 de copa	5 de basto	5 de espada	5 de oro
0	4 de copa	4 de basto	4 de espada	4 de oro

La función `truco_card_tie()` devuelve `true` si y sólo si las dos cartas tienen **el mismo** rango. La función `truco_card_win()` devuelve `true` si y sólo si la carta `a` tiene rango mayor (estricto) a la carta `b`.

a) Implementar el TAD *Truco-Card* siguiendo las pautas de encapsulamiento vistas en la materia.

b) Crear el archivo **`main.c`** donde se permita al usuario ingresar dos cartas. Luego debe mostrarse por pantalla primero la carta ganadora y posteriormente la perdedora. Si las cartas empatan debe mostrar un mensaje *"¡La mano está parda!"* y luego mostrar las dos cartas en cualquier orden.

IMPORTANTE: El TAD *Truco-Card* y el programa en **`main.c`** deben estar libres de *memory leaks* y de *invalid reads / writes*

Córdoba 1 de Junio del 2021

Segundo Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA A

Ejercicio 3

En **`balanced.c`** hay una nueva implementación del algoritmo de chequeo de delimitadores balanceados. En esta ocasión se utilizará una cola para guardar los delimitadores de la expresión de entrada. Deben completar el algoritmo utilizando las funciones del TAD Cola listadas a continuación:

Función	Descripción
`queue queue_empty(void)`	Crea una cola vacía
`queue queue_enqueue(queue q, queue_elem e)`	Inserta un elemento a la cola
`bool queue_is_empty(queue q);`	Indica si la cola está vacía
`unsigned int queue_size(queue q)`	Obtiene el tamaño de la cola
`queue_elem queue_first(queue q)`	Obtiene el primer elemento de la cola (el próximo a atender).
`queue queue_dequeue(queue q)`	Quita un elemento de la cola
`queue queue_destroy(queue q)`	Destruye una instancia del TAD Cola

El programa **no debe tener *memory leaks***

Ejercicio 4

En **`queue.c`** se da una nueva implementación incompleta del TAD Cola que deben completar. La implementación utiliza listas enlazadas de una manera particular. Analizar cómo se almacenan los elementos en la cadena de nodos y terminar de implementar las funciones `queue_first()` y `queue_dequeue()`. **No pueden cambiar las implementaciones del resto de las funciones**, solo pueden tocar las funciones marcadas para completar. Para verificar que esta nueva implementación del TAD funciona correctamente, pueden utilizar el programa del ejercicio anterior.