Córdoba, 11 de Agosto del 2023

Examen Final

Algoritmos y Estructuras de Datos II - Taller

El ejercicio consiste en implementar el TAD *MatchTimeline* que representa la ficha de un partido de fútbol, en donde se indica en orden cronológico los acontecimientos en el partido: goles, tarjetas y cambios. Para esta implementación, se va a considerar una lista enlazada, donde cada nodo contendrá la información de un evento. A modo de ejemplo, una ficha de un partido a implementar, es algo como lo siguiente:

Argentina	**2**	**2**	**Francia**
Gol - 10	23’		
Gol - 11	36’		
Amarilla - 24	45’		
	54’		Amarilla - 14
	80’		Gol - 10
	81’		Gol - 10
	86’		Amarilla - 26
	90’		Amarilla - 9
Amarilla - 8	90’		

(Para la simplificación del caso, es de la final del mundo pero solo del partido, sin el alargue ni los penales)

Cada nodo en la lista enlazada que representa al *MatchTimeline* debe tener la siguiente información:

- Equipo al cual se le registra el evento
- Tipo del evento
- Tiempo
- Número del jugador

Para esta implementación vamos a tener en cuenta las siguientes simplificaciones:

- El equipo se va a representar como local y visitante, con el siguiente tipo de dato enumerado:

```
typedef enum {Home, Away} team;
```

- Para el tipo de evento, vamos a considerar solo goles, tarjetas amarillas y tarjetas rojas, con el siguiente tipo de dato enumerado:

```
typedef enum {Goal, YellowCard, RedCard} event;
```

- El tiempo va a ser representado por un *unsigned int,* que se corresponde con el minuto en el que sucedió el evento, es decir un número entre 0 y 90 (no se tienen en cuenta tiempos adicionales)
- El número de jugador va a ser representado por un *unsigned int,* y para simplificar, vamos a asumir que cada equipo utiliza a los jugadores con números entre el 1 y el 11

La propiedad fundamental de este TAD es que la lista enlazada de los eventos están ordenados cronológicamente por el **time**.

Las funciones a implementar en **matchTimeline.c** son las siguientes:

Función	Descripción
`matchTimeline matchTimeline_empty();`	Devuelve un nuevo 'matchTimeline' creado vacío
`matchTimeline matchTimeline_score_goal(matchTimeline mt, team team, time t, player_number pn);`	registra un nuevo gol
`matchTimeline matchTimeline_receive_redCard(matchTimeline mt, team team, time t, player_number pn);`	registra una nueva tarjeta roja
`matchTimeline matchTimeline_receive_yellowCard(matchTimeline mt, team team, time t, player_number pn);`	registra una nueva tarjeta amarilla, debe tener en cuenta de que si es la segunda tarjeta debe sacar tarjeta roja también
`bool matchTimeline_is_time_and_score_valid(matchTimeline mt);`	Devuelve True si el matchTimeline es válido. Es válido cuando los registros están ordenados cronológicamente, y los goles de home (away) se corresponden con la cantidad de registros de goles de home (away)
`unsigned int matchTimeline_get_score(matchTimeline mt, team team);`	Devuelve la cantidad de goles del equipo *team* **IMPORTANTE: ESTE METODO DEBE SER CONSTANTE**
`unsigned int matchTimeline_size(matchTimeline mt);`	Devuelve la cantidad de registros que tiene el matchTimeline **IMPORTANTE: ESTE METODO DEBE SER CONSTANTE**
`void matchTimeline_print(matchTimeline mt);`	Imprime en pantalla la ficha del partido, en donde en la primer línea se debe imprimir el resultado **IMPORTANTE: Es un método solo de lectura, no debe modificar la estructura**
`event *matchTimeline_events_array(matchTimeline mt);`	Devuelve un arreglo que representa el tipo de cada evento en el orden en que ocurrieron. Este método debe pedir memoria para match timeline y la memoria pedida debe ser liberada después
`matchTimeline matchTimeline_destroy(matchTimeline mt);`	Destruye el registro del match Timeline y libera todos los recursos pedidos.

Resultado del partido

La función "`matchTimeline_get_score`" devuelve el resultado del partido, pero de a un equipo a la vez, es decir si se le pasa como atributo `Home` devuelve los goles del equipo local, y si se le pasa como atributo `Away`, los goles del equipo visitante. **ESTE MÉTODO DEBE SER CONSTANTE!**

Ficha correcta

En "`matchTimeline_is_time_and_score_valid`" se debe chequear las siguientes condiciones:

- Los eventos deben estar en orden cronológico
- La cantidad de goles que tiene *Home* se debe corresponder a la cantidad de registros con equipo *Home*
- La cantidad de goles que tiene *Away* se debe corresponder a la cantidad de registros con equipo *Away*

A continuación damos algunos ejemplo abreviando:

- Goal -> G
- YellowCard -> Y
- RedCard -> R
- Home -> H
- Away -> A

En cada caso las filas van a tener el formato:

```
Minuto' - Equipo - Tipo de evento - Número de jugador
```

Ficha	*Retorno*	*Razón*
Home 1 - 1 Away 15' - A - G - 9 **35' - H - Y - 2** **32' - H - G - 5**	`false`	Este caso es incorrecto porque el último registro ocurre antes del penultimo
Home 0 - 0 Away	`true`	**correcto**, No tiene ningún registro
Home 1 - 0 Away 25' - A - Y - 11 40' - H - G - 7 50' - A - R - 11	`true`	**correcto**, Los registros están ordenados y la cantidad de goles coincide
Home 1 - 0 Away 5' - H - Y - 5 15' - A - G - 9 35' - H - Y - 2 75' - A - Y - 8	`false`	**incorrecto**, Home tiene 1 gol, pero ningún registro y Away no tiene goles pero si tiene un registro de gol.

Arreglos

La función `matchTimeline_events_array()` debe devolver un arreglo dinámico con los tipos de eventos sucedidos en el partido, en el orden apropiado, es decir, para el partido de ejemplo al inicio exámen (Argentina 2 - 2 Francia), debería ser:

```
[Goal Goal YellowCard YellowCard Goal Goal YellowCard YellowCard
YellowCard]
```

La cantidad de elementos contenidos en el arreglo se debe corresponder con el valor devuelto por `matchTimeline_size`().

Ejercicio 1: matchTimeline.c

Implementar las funciones del TAD MatchTimeline, de acuerdo a las especificaciones provistas.

Ejercicio 2: main.c

En el archivo main.c se encuentran varios tests ya hechos y otros por completar. Tomando como ejemplos los ya realizados, completar los tests que falten con las indicaciones en los comentarios.

Consideraciones:

- Se provee el archivo **`Makefile`** para facilitar la compilación.
- Se recomienda usar las herramientas **`valgrind`** y **`gdb`**.
- Si el programa no compila, no se aprueba el final.
- Los *memory leaks* bajan puntos.
- Entregar un código muy improlijo puede restar puntos.
- Si las funciones que pide que sean de orden constante, no lo son baja muchísimos puntos.
- Para aprobar **se debe** hacer una invariante que cheque propiedades del TAD que no sean triviales.
- Se pueden agregar funciones auxiliares que crean convenientes para modularizar el código
- **Se debe entregar matchTimeline.c y main.c**

Ejercicio extra solo para alumnos LIBRES

Actualmente se está pensando en implementar una tarjeta blanca la cual se entrega como premio de juego limpio a los jugadores. Esta tarjeta tiene la particularidad de que solo es entregada en los últimos 15 minutos de juego y el jugador que la recibe no puede tener ni tarjetas amarillas ni rojas.

Se pide implementar la lógica esta lógica y dos tests más: Uno que pruebe agregar una tarjeta blanca y otro que no permite agregar la tarjeta blanca por exceso de tarjetas rojas o amarillas.

- **Deberán entregar además el matchTimeline.h**