

Examen Final

Algoritmos y Estructuras de Datos II - Taller

El ejercicio consiste en implementar el TAD *color-pairs* que representa una partida correspondiente al juego de cartas “Pares de Colores”.



En el juego de cartas “color numbers”, los jugadores se dividen equitativamente un conjunto de naipes, y luego de mezclarlos, los muestran uno a uno sin verlos previamente. No es necesario utilizar el mazo completo, pero en cada partida, los jugadores siempre comienzan con la misma cantidad de cartas. En cada mano del juego los jugadores, en orden, muestran su cartas tope de mazo, y cada jugador obtiene una diferente cantidad de puntos dependiendo de las cartas mostradas. Esto continúa hasta acabar todas las cartas. El ganador es quien consigue más puntos al finalizar la partida.










































En nuestra implementación del juego simplificamos las reglas, ya que se permitirán **solo dos jugadores**.

Entonces, una partida del juego es una secuencia de cartas, por lo que para implementar el TAD **debe usarse una lista enlazada** de las mismas. Una carta se representa con el TAD *Card* que **está mayormente implementado** y sólo deben completar algunas de sus funciones.

TAD Card

El juego cuenta con un mazo francés de 52 cartas, donde cada una de ellas posee un número del 1 al 13, y un palo (diamante, corazon, trebol, pica). Una partida no necesariamente se juega con el total de cartas, en tanto todos los jugadores tengan al inicio la misma cantidad.

Ejemplo de una baraja inglesa completa, con sus 52 naipes

	As	2	3	4	5	6	7	8	9	10	Jota/Sota/Jack	Reina/Dama	Rey
Tréboles (Clubs)													
Diamantes (Diamonds)													
Corazones (Hearts)													
Picas (Spades)													

El TAD Card tiene la siguiente interfaz:

Función	Descripción
<code>card card_new(unsigned int num, char suit, unsigned int player)</code> -- completar --	Crea una carta con numeración <code>num</code> y palo <code>suit</code> , perteneciente al jugador <code>player</code>
<code>unsigned int card_player(card c)</code>	Retorna el número del jugador que utilizó la carta
<code>unsigned int card_number(card c)</code>	Retorna el número de la carta <code>c</code>
<code>unsigned int card_color(card c)</code>	Retorna el color de la carta <code>c</code>
<code>char card_suit(card c)</code>	Retorna el palo de la carta <code>c</code>
<code>bool card_equal_color(card fst, card snd)</code> -- completar --	Indica si la carta <code>fst</code> es del mismo color que la carta <code>snd</code>
<code>bool card_equal_number(card fst, card snd)</code>	Indica si la carta <code>fst</code> tiene el mismo número que la carta <code>snd</code>
<code>unsigned int card_pair_points(card fst, card snd, unsigned int player)</code> -- completar ---	Retorna el puntaje de jugar la carta <code>fst</code> contra la carta <code>snd</code> , para el jugador <code>player</code>
<code>void card_dump(card c)</code>	Muestra una carta por pantalla
<code>card card_destroy(card c)</code> -- completar ---	Destruye una instancia del TAD Card, liberando toda la memoria utilizada

Número de jugador

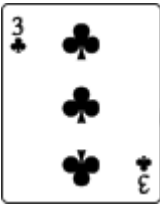
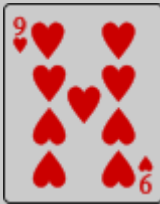





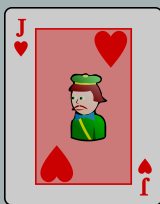
Representa la posición del jugador en la ronda de la partida. El número es un entero entre 1 y N, siendo N la cantidad de jugadores. En nuestra implementación, los números siempre serán 1 o 2.

Cálculo de Puntos de una mano

En cada mano jugada (es decir, un par de cartas, una del jugador 1, y una del jugador 2), cada jugador obtiene la siguiente cantidad de puntos:

- Si los colores de las cartas son iguales, el jugador uno obtiene 1 punto
- Si los números de las cartas son iguales, el jugador 2 obtiene 10 puntos
- Si no se da ninguno de estos casos, ambos obtienen 0 puntos

Ejemplos:

Primera carta	Segunda carta	Puntos Jugador 1	Puntos Jugador 2
		0	0
		1	0
		0	10
		1	10

TAD *color-numbers*

Las funciones a implementar en `war_match.c` son las siguientes:

Función	Descripción
<code>cn_match match_new(void)</code>	Construye una partida de juego
<code>cn_match match_add(cn_match match, card c)</code>	Agregar una carta a la partida

<code>unsigned int match_size(cn_match match)</code>	Devuelve la cantidad de cartas jugadas
<code>unsigned int match_length(cn_match match)</code>	Devuelve la cantidad de manos jugadas
<code>bool is_match_correct(cn_match match)</code>	¿Está la partida correctamente formada?
<code>unsigned int winner_total_points(cn_match match)</code>	Devuelve la suma de puntos del jugador que ganó la partida.
<code>card * match_to_array(cn_match match)</code>	Arreglo dinámico con las cartas de la partida
<code>void match_dump(cn_match match)</code>	Muestra la partida en la pantalla
<code>cn_match match_destroy(cn_match match)</code>	Destruye la partida de juego y todas las cartas

Cartas jugadas - `match_size`

Retorna la **cantidad de cartas** jugadas en la partida, sin importar si la partida es correcta.

Manos jugadas - `match_length`

Retorna la **cantidad de manos** jugadas en la partida. Una “mano” se refiere a una pareja de cartas, cada una jugada por un jugador, que compiten entre ellas. En caso en que la partida no sea correcta, `match_length` **retorna 0**.

Partida correcta - `is_match_correct()`

Se considera que la partida es correcta si:

- El jugador 1 inició la partida.
- La cantidad de cartas es par.
- Cada carta `c` del jugador 1, está seguida de una carta `c'` del jugador 2.
- Nunca dos cartas contiguas pertenecen al mismo jugador.

Para simplificar los siguientes ejemplos, las cartas están representadas como (número, jugador), sin considerar el palo:

<i>match</i>	<i>Retorno</i>	<i>Razón</i>
<code>[(4,1), (5,2)]</code>	true	cumple con todas las propiedades
<code>[(4,1), (5,1)]</code>	false	las cartas pertenecen al mismo jugador
<code>[(4,1)]</code>	false	Falta una carta del jugador 2
<code>[(4,1), (5,2), (6,2)]</code>	false	las últimas 2 cartas pertenecen al mismo jugador

Suma de puntos del ganador - `winner_total_points`

En cada mano, a cada jugador se le asigna el puntaje de la manera explicada en la primera sección. Esta función sólo retorna la suma de puntaje para el jugador que ganó. Si la partida no está formada correctamente **devuelve 0**.

Por ejemplo: (P = pica, D = diamante, C = corazon, T = trebol)

<i>match</i>	<i>ganador</i>	<i>Retorno</i>
[(4,C,1), (5,D,2)]	1	1
[(4,C,1), (4,D,2), (13,T,1), (7,P,2)]	2	10
[(4,C,1), (5,D,2), (13,T,1), (7,P,2), (9,C,1), (10,C,2)]	1	3

Arreglos

La función `match_to_array()` debe devolver un arreglo dinámico con las cartas de la línea de juego en el orden en que fueron agregadas. La cantidad de elementos contenidos en el arreglo se debe corresponder con el valor devuelto por `match_size()`.

Compilación y Test

Se provee un **Makefile** para compilar todo el código y generar un ejecutable. Para ello deben hacer:

```
$ make
```

y luego pueden probar su implementación con los archivos de ejemplo de la carpeta **input**:

```
$ ./test_match -f input/example01.in
```

si todo sale bien debería obtener la siguiente salida:

```
READING input/example01.in
Reading CARDS from file...
Building match: [(4, c, 1), (5, d, 2), (13, t, 1), (7, p, 2), (9, c, 1), (10, c, 2)]
size reported: 6
hands: 3
check correct match: True
winner points: 3
array: [(4, c, 1), (5, d, 2), (13, t, 1), (7, p, 2), (9, c, 1), (10, c, 2)]
DONE input/example01.in.
```

además pueden usar la opción de verificación para comparar los resultados de sus funciones con los valores esperados para el ejemplo. Para ello:

```
$ ./test_match -vf input/example01.in
```

La salida obtenida:

```
READING input/example01.in
Reading CARDS from file...
Building match: [(4, c, 1), (5, d, 2), (13, t, 1), (7, p, 2), (9, c, 1), (10, c, 2)]
size reported: 6 [OK]
hands: 3
check correct match: true [OK]
winner points: 3 [OK]
array: [(4, c, 1), (5, d, 2), (13, t, 1), (7, p, 2), (9, c, 1), (10, c, 2)] [OK]
DONE input/example01.in.
```

```
ALL TESTS OK
```

En caso de error se muestra el valor esperado y además se muestra la cantidad de errores ocurridos. Si

se omite la opción `-f` se lee la línea de juego por la entrada estándar, debiendo ingresar primero la cantidad de elementos y luego las cartas usando la notación `n:p:j` (numero: palo: jugador) separadas por espacios o apretando enter:

```
READING stdin
Reading CARDS from stdin...
4
4:c:1
5:p:2
13:d:1
7:d:2
Building match: [(4, c, 1), (5, p, 2), (13, d, 1), (7, d, 2)]
size reported: 4
hands: 2
check correct match: true
winner points: 1
array: [(4, c, 1), (5, p, 2), (13, d, 1), (7, d, 2)]
DONE stdin.
```

Para realizar test usando todos los ejemplos de la carpeta `input` en modo verificación:

```
$ make test
```

Para además chequear con valgrind:

```
$ make valgrind
```

IMPORTANTE: Pasar los tests no significa aprobar. Tener *memory leaks* resta puntos.