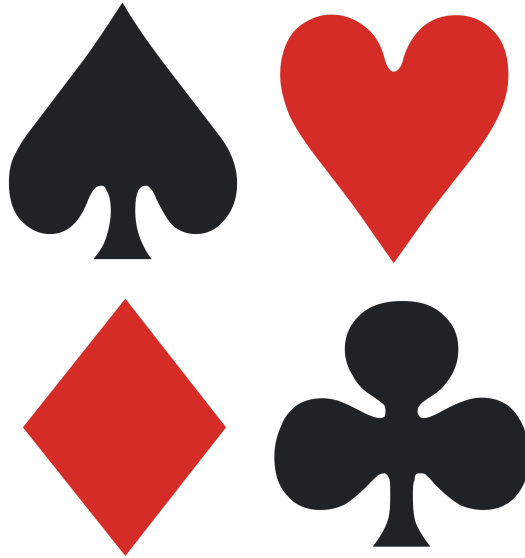


Examen Final

Algoritmos y Estructuras de Datos II - Taller

El ejercicio consiste en implementar el TAD *war-match* que representa una partida correspondiente al juego de cartas “Guerra”.



En el juego de cartas “guerra”, los jugadores se dividen equitativamente un conjunto de naipes, y luego de mezclarlos, los muestran uno a uno sin verlos previamente. En cada mano del juego los jugadores, en orden, muestran su cartas tope de mazo, y el jugador que tiene la carta más alta se lleva como puntos el valor numérico de la menor carta jugada. 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>char card_suit(card c)</code>	Retorna el palo de la carta <code>c</code>
<code>bool card_wins(card fst, card snd)</code>	Indica si la carta <code>fst</code> le gana a la carta <code>snd</code>
<code>unsigned int card_match_points(card fst, card snd)</code> -- completar ---	Retorna el puntaje de jugar la carta <code>fst</code> contra la carta <code>snd</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.

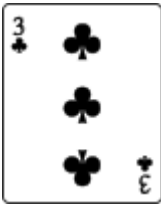





Cálculo de Carta ganadora de una mano

La función `card_wins()` considera que la carta con mayor valor numérico, *sin importar el palo*, es la ganadora. Se considera que dos cartas con el mismo valor numérico están empatadas. En caso de empate, se considera que ninguna carta resultó ganadora.

Cálculo de Puntos

La función `card_match_points()` retorna los puntos obtenidos en una mano, que es el valor numérico de la carta que *perdió*. Se considera que dos cartas con el mismo valor numérico están empatadas, en caso de empate, el puntaje obtenido es 0.

Ejemplos:

Primera carta	Segunda carta	card_match_points()
		3
		10
		0

TAD *War-match*

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

Función	Descripción
<code>war_match match_new(void)</code>	Construye una partida de juego
<code>war_match match_add(war_match match, card c)</code>	Agregar una carta a la partida
<code>unsigned int match_size(war_match match)</code>	Devuelve la cantidad de cartas jugadas
<code>unsigned int match_length(war_match match)</code>	Devuelve la cantidad de manos jugadas
<code>bool is_match_correct(war_match match)</code>	¿Está la partida correctamente formada?
<code>unsigned int winner_total_points(war_match match)</code>	Devuelve la suma de puntos del jugador que ganó la partida.
<code>card * match_to_array(war_match match)</code>	Arreglo dinámico con las cartas de la partida
<code>void match_dump(war_match match)</code>	Muestra la partida en la pantalla
<code>war_match match_destroy(war_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, al jugador con la carta de valor numérico mayor, se le suman los puntos de la carta de valor numérico menor. Esta función sólo retorna la suma de puntaje para el jugador que ganó. Si la partida no está formada correctamente **devuelve 0**.

Por ejemplo:

<i>match</i>	<i>ganador</i>	<i>Retorno</i>
<code>[(4,1), (5,2)]</code>	2	4
<code>[(4,1), (5,2), (13,1), (7,2)]</code>	1	7
<code>[(4,1), (5,2), (13,1), (7,2), (9,1), (10,2)]</code>	2	4 + 9 = 13

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, p ,2), (13, d, 1), (7, d, 2), (9, t, 1), (10, p, 2)]
size reported: 6
hands: 3
check correct match: True
winner points: 13
array: [(4, c, 1), (5, p ,2), (13, d, 1), (7, d, 2), (9, t, 1), (10, p, 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, p ,2), (13, d, 1), (7, d, 2), (9, t, 1), (10, p, 2)]
size reported: 6 [OK]
hands: 3
check correct match: true [OK]
winner points: 13 [OK]
array: [(4, c, 1), (5, p ,2), (13, d, 1), (7, d, 2), (9, t, 1), (10, p, 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 domino-line: [(4, c, 1), (5, p ,2), (13, d, 1), (7, d, 2)]
size reported: 4
hands: 2
check correct match: true
winner points: 7
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.