Trabajo Práctico: Algoritmos y Estructuras de Datos.

- **Juego**: Escoba de 15

- **Comisión**: X1071 (curso de verano 2024 turno noche).

- **Fecha de entrega**: 28/02/2024.

- **Integrantes**: Gianluca Bolocco, Leandro Fusetti.

Guía de juego:

1. Una vez ejecutado el juego, deberemos indicar la cantidad de jugadores a participar (2 o 3):

```
[-] Ingrese la cantidad de jugadores (2 o 3):
```

2. Una vez ingresado, el juego se da por iniciado, reparte las cartas en la mesa y nos

```
==============[1]==============
============ MESA ============
-Id- ==== CARTA ====
[ 2] ----10 de Copa
[10] ---- 8 de Copa
[27] ---- 4 de Basto
[34] ---- 2 de Copa
========= Jugador 1 ==========
-Id- ==== CARTA ====
[ 0] ----10 de Oro
[15] ---- 7 de Basto
[26] ---- 4 de Copa
[-] Seleccione una carta de su mano (por id):
```

El texto "============ [1] ============" indica el número de la ronda

muestra las cartas que posee el jugador 1, cada carta posee un ID por el cual podremos hacer uso de ella; por ejemplo, si quisiera seleccionar el 7 de Basto, deberé escribir el número de ID 15.

```
[-] Seleccione una carta de su mano (por id): 15
[-] Carta jugada: 7 de Basto
[-] Usted sumo 15!
[-] El 7 de Basto se agrego a las cartas acumuladas del jugador 1
[-] El 8 de Copa se agrego a las cartas acumuladas del jugador 1
```

3. A partir de aquí, de forma automática, se evaluará si la misma puede formar una pareja o grupo de cartas que sumen 15, y además, en el caso de que quede la mesa vacía, se suma una escoba a las estadísticas de puntuación del jugador.

4. Ahora es el turno del jugador 2, como se ha mencionado anteriormente, debe elegir la carta a jugar que pueda sumar 15 con las de la mesa. En este caso no posee ninguna, por lo que la elegida, se descarta de la mano del jugador y se agrega a la mesa.

```
==============[1]==============
============ MESA ============
-Id- ==== CARTA ====
[ 2] ----10 de Copa
[27] ---- 4 de Basto
[34] ---- 2 de Copa
========= Jugador 2 ==========
-Id- ==== CARTA ====
[18] ---- 6 de Copa
[29] ---- 3 de Espada
[36] ---- 1 de Oro
[-] Seleccione una carta de su mano (por id): 18
[-] Carta jugada: 6 de Copa
[-] No hay manera de sumar 15, su carta se descarto.
```

5. Los participantes seguirán jugando nuevas rondas, las cartas se repartirán cada vez que ambos se queden sin cartas y la partida se dará por terminada cuando se acaben las cartas del mazo.
6. Se le dan al último jugador que sumó 15 las cartas que quedaron en la mesa mediante la funcion limpiarMesa().
7. Una vez finalizado la ronda, se desplegará una lista indicando los puntos, el total de cartas acumuladas, el total de Oros, de sietes, quien posee el siete de oro y el acumulado:

```
====== JUGADOR 1 ======
[-] puntos: 3
[-] Cantidad de cartas: 19
[-] Cantidad de Oros: 6
[-] Cantidad de 7: 4
[-] Escobas: 0
[-] Posee el siete de oro.
[-] Acumulado:
-Id- ==== CARTA ====
[ 1] ----10 de Espada
[ 4] ---- 9 de Oro
[ 8] ---- 8 de Oro
[10] ---- 8 de Copa
[12] ---- 7 de Oro
[13] ---- 7 de Espada
[14] ---- 7 de Copa
[15] ---- 7 de Basto
[16] ---- 6 de Oro
[19] ---- 6 de Basto
[21] ---- 5 de Espada
[23] ---- 5 de Basto
[24] ---- 4 de Oro
[27] ---- 4 de Basto
[28] ---- 3 de Oro
[30] ---- 3 de Copa
[31] ---- 3 de Basto
[33] ---- 2 de Espada
[37] ---- 1 de Espada
```

```
====== JUGADOR 2 ======
[-] puntos: 1
[-] Cantidad de cartas: 19
[-] Cantidad de Oros: 3
[-] Cantidad de 7: 0
[-] Escobas: 1
[-] Acumulado:
-Id- ==== CARTA ====
[ 0] ----10 de Oro
[ 2] ----10 de Copa
[ 3] ----10 de Basto
[ 5] ---- 9 de Espada
[ 6] ---- 9 de Copa
[ 7] ---- 9 de Basto
[11] ---- 8 de Basto
[17] ---- 6 de Espada
[18] ---- 6 de Copa
[20] ---- 5 de Oro
[22] ---- 5 de Copa
[25] ---- 4 de Espada
[26] ---- 4 de Copa
[29] ---- 3 de Espada
[34] ---- 2 de Copa
[35] ---- 2 de Basto
[36] ---- 1 de Oro
[38] ---- 1 de Copa
[39] ---- 1 de Basto
```

Funcionamiento del código:

Estructuras utilizadas:

struct Carta{

int num; **(Valor numérico de la carta)**

str10 palo; **(Palo de la carta)**

char ubi; **(Carácter de ubicación)**

}

- Sobre el campo de “ubi”, las ubicaciones posibles para cada carta son:
  - D: deck.
  - T: Table.
  - H: Mano jugador 1.
  - h: Mano jugador 2.
  - X: Mano jugador 3.
  - A: Acumulado jugador 1.
  - a: Acumulado jugador 2.
  - x: Acumulado jugador 3.

struct Jugador{

short puntos=0; **(Puntos)**

int id; **(Que jugador es)**

char mano; **(H,h,X)**

char acum; **(A,a,x)**

short cantCartas=0; **(Cantidad de cartas)**

bool sieteOro; **(Posee siete de oro)**

short cantOro=0; **(Cantidad de oros)**

short cantSiete=0; **(Cantidad de sietes)**

short escobas=0; **(Escobas)**

};

- Los campos de puntuación se resetean al inicio de cada ronda (cuando se terminan las 40 cartas del mazo).
- Se utiliza la estructura de “Nodo” para la pila de sumatoria de 15, en la función jugada, ya que allí precisamos el uso de memoria dinámica.

- Notar que, todo el juego se lleva a cabo alrededor de la modificación de una única estructura que se va iterando, que es el mazo. Es un vector de 40 elementos de tipo carta, vector el cual almacena cada una de las cartas del mazo y su respectiva ubicación.
- Cada carta posee un “id”, que representa la ubicación de la carta en el vector mazo, es decir, que cada vez que utilizamos una carta, lo hacemos por acceso directo.
- Al finalizar cada ronda, la ubicación de todas las cartas del mazo se setean en “D” (deck), para poder iniciar una nueva ronda.

Uso de memoria dinámica:

Decidimos utilizar la estructura pila con el fin de poder administrar las cartas a la hora de hacer la sumatoria de 15.

Los ids de las cartas se van almacenando en la pila, y en caso de sumar 15, se revisan las cartas para sumar la puntuación correspondiente a cada jugador y luego se envían a las cartas acumuladas de cada jugador.

Creímos conveniente utilizar una pila ya que la cantidad de cartas que se pueden llegar a utilizar para sumar 15 varía.

Funciones:

- **void repartir(Carta[],int,char):** Reparte las cartas.
- **void mostrarCartas(Carta[],char):** Muestra las cartas (Mesa o mano).
- **int cartaAleatoria():** Genera un número aleatorio del 0 al 39.
- **short cantCartasT(Carta[]):** Cuenta las cartas que hay en la mesa.
- **void ronda(Jugador&,Carta[]):** función que se ejecuta en cada turno.
- **void jugada(Jugador&, Carta[],short):** Algoritmo que verifica si existe alguna combinación de cartas que sume 15.
- **void calcularPuntaje(Jugador&,Jugador&,Jugador&):** Calcula el puntaje de cada jugador
- **void mostrarPuntaje(Carta[],Jugador):** Muestra el puntaje y cartas acumuladas de cada jugador

Función de combinatoria de cartas:

```
int quince = m[card].num; //acumulador inicializado con la carta seleccionada.
int k=0;

if(cantT != 0){ // Revisa que hayan cartas en la mesa

    while(quince != 15){ // Combinatoria de cartas hasta encontrar una combinacion que sume 15.

        for(int i = k ; i<cantT ; i++){

            if((quince + m[idsTable[i]].num) <= 15){
                quince += m[idsTable[i]].num;
                m[idsTable[i]].ubi = jug.acum;
                push(pila,idsTable[i]); // Una pila con los ids de las cartas que suman 15
            }

            if(quince==15){
                cout<<"[-] Usted sumo 15!"<<endl;
                break;
            }
        }

        k++;

        if(quince != 15){
            quince = m[card].num; // Acumulador
            while(pila != NULL){ // Vaciamos la pila
                m[pop(pila)].ubi = 'T';
            }
        }
        if(k==cantT)break; // Itera tantas veces como cartas hay en la mesa
    }
}
```

Lógica:

- Las cartas vienen ordenadas de manera descendente por valor.
- Verifica si la suma con la siguiente carta es menor o igual a 15 y la suma pasando a la siguiente si se pasa.
- Si no hay manera de sumar 15 con la carta de valor más alto pasa a la siguiente de valor más alto, y así sucesivamente hasta encontrar la primera combinación de cartas que finalmente sume 15.
- La cantidad de veces que intenta combinar cartas es el número de cartas que hay en la mesa.