PRÁCTICA Nº 6: Pilas.

Fecha de realización: Semana del 6 al 10 de Mayo.

Duración: 2.5 horas presenciales

Objetivos de la práctica

- Desarrollar una aplicación en C++ desde cero.
- Analizar la idoneidad de la utilización del *Tipo Abstracto de Datos* (TAD) **Pila** en la solución del problema propuesto.
- Implementar y utilizar el TAD Pila con representación estática.

Material a entregar

- Solución ejercicios previos (1, 2, 3): **Pr06_Previo.doc**. Se debe subir a Aula Virtual antes del **5 de Mayo a las 23:55**.
- Los ficheros de cabecera e implementación de la clase *Pila* (**`Pila.cpp`, `Pila.h`**).
- Los ficheros de cabecera e implementación de la clase *Hanoi* (**`Hanoi.cpp`, `Hanoi.h`**).
- El programa principal que implementa el desarrollo del juego: **`Pr06.cpp`**.

Introducción

En esta práctica se va a desarrollar un juego de cartas interactivo basado en el problema de las Torres de Hanoi.

En este juego se dispone de una baraja desordenada de 9 cartas, numeradas de 1 a 9, del mismo palo. A partir de esta baraja se crean 3 mazos (o pilas) con 3 cartas cada una.

El objetivo del juego consiste en colocar todas las cartas en orden ascendente en una de las pilas. Para ello, se permiten desplazamientos de cartas de una pila a otra mediante las siguientes reglas:

- Sólo se puede mover una carta en cada jugada, desde una pila origen a una pila destino.
- La carta a mover debe ser la cima de una pila, y al moverla se debe depositar sobre la cima de otra pila.
- La jugada sólo será válida si el número de la carta a mover es **inferior** al número de la carta que corresponde a la cima de la pila donde queremos dejarla.

Puesto que el único movimiento posible es el de mover una carta, el programa debe ir preguntando continuamente al usuario el número de la pila de origen y el de la pila de destino, hasta que el juego haya finalizado.

A partir de la descripción anterior, definimos un nuevo tipo llamado `Hanoi` que representa y permite resolver el juego planteado, y cuya especificación de la interfaz es la siguiente:

```
class Hanoi
{
    public:
        Hanoi();
        void IniciarJuego ();
```

```
        bool MoverCarta (int, int);
        bool JuegoFinalizado ();
        friend ostream & operator << (ostream &, const & Hanoi);
    private:
        static const int MAX_PILAS = 3;
        int numero_jugadas;
        typedef Pila VectorMazos[MAX_PILAS];
        VectorMazos mazos;
};
```

El atributo "mazos" de la clase anterior es un vector de pilas. Por tanto, para que esta clase se pueda construir adecuadamente es preciso implementar previamente el TAD `Pila` visto en clase, el cual permite el almacenamiento secuencial de datos y contiene las operaciones de inserción y eliminación de elementos en un extremo de la secuencia.

Puesto que todas las cartas son del mismo palo y sólo se distinguen por el número, el tipo de datos que almacenará el TAD `Pila` serán números enteros.

Desarrollo de la práctica

Ejercicio 1 (TAD `Pila`): Implementa el TAD `Pila` (ficheros **`Pila.cpp`, `Pila.h`**) con representación estática, de manera similar al visto en clase, que almacene números enteros. Incluye en la definición de la clase el número máximo de elementos que puede almacenar la pila:

```
static const int MAX = 9;
```

Ejercicio 2 (Sobrecarga de operador de salida, clase `Pila`): Sobrecarga el operador << del TAD `Pila` mediante una función `friend` que permita escribir el contenido de la pila:

```
friend ostream & operator << (ostream &, const & Pila);
```

Los elementos de la pila deben imprimirse en formato horizontal, y la cima de la pila debe corresponder al elemento del extremo derecho:

```
1  3  4
```

Ejercicio 3 (Creación de la clase `Hanoi`): Crea la clase `Hanoi` (ficheros **`Hanoi.cpp`, `Hanoi.h`**) e implementa el constructor de la misma. El constructor debe inicializar el juego para poder empezar a jugar en cualquier instante.

Ejercicio 4 (Inicio del juego): Implementa un método público de la clase `Hanoi` que permita iniciar una nueva partida, de manera que se vacíen las pilas de cartas (si no lo estuvieran), y se rellenen con 3 cada una, de manera desordenada.

```
void Hanoi::IniciarJuego ();
```

Ejercicio 5 (Movimiento de una carta): Implementa un método público de la clase `Hanoi` que permita mover una carta desde una pila de origen a una pila destino:

```
bool Hanoi::MoverCarta (int pila_origen, int pila_destino);
```

El método debe devolver un valor booleano indicando si la jugada es válida desde el punto de vista de las reglas del juego. En el caso de que sea válida, incrementaremos el número de jugadas.

Ejercicio 6 (Finalización del juego): Implementa un método público de la clase `Hanoi` que indique mediante un valor booleano si el juego ha finalizado (todas las cartas están en una de las pilas):

```
bool Hanoi::JuegoFinalizado ();
```

Ejercicio 7 (Sobrecarga de operador de salida, clase `Hanoi`): Sobrecarga el operador `<<` de la clase `Hanoi` de manera que imprima por pantalla el estado actual de todas las pilas del juego, así como el número de jugadas realizadas.

El resultado de la operación debe ser similar al siguiente:

```
Numero Jugada: 1
-----------------------------------
Pila #1:  1  3  4
Pila #2:  5  8  2
Pila #3:  9  6  7
-----------------------------------
```

Nota: Utiliza el operador `<<` sobrecargado en el Ejercicio 2 para imprimir el estado de cada una de las pilas.

Ejercicio 8 (Uso del tipo `Hanoi`): Crea un fichero llamado **`Pr_06.cpp`** con la función `main()` que, haciendo uso de la clase `Hanoi`, implemente el desarrollo del juego. Esta función debe implementar la entrada de datos del usuario que corresponde a los movimientos de cartas. Al finalizar el juego, el programa deberá preguntar al usuario si quiere iniciar una nueva partida.

El aspecto final del juego debe ser similar al siguiente:

```
Numero Jugada: 1
-----------------------------------
Pila #1:  1  3  4
Pila #2:  5  8  2
Pila #3:  9  6  7
-----------------------------------
Introduzca la siguiente jugada:
Pila Origen (1-3): 1
Pila Destino (1-3): 2
¡Movimiento incorrecto!
```