

# Práctica 1

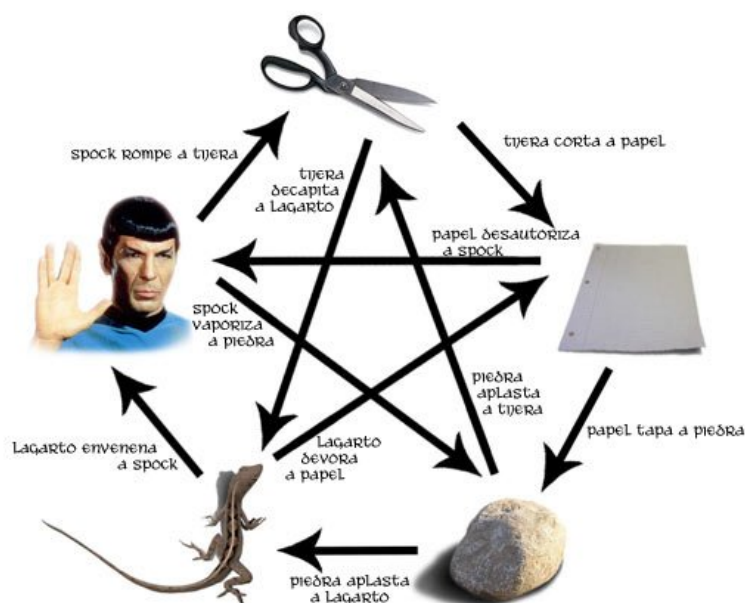
## *Rock-Paper-Scissors-Lizard-Spock*

Fecha de entrega: **19 de noviembre de 2017**

En 1998 Sam Kass comprobó que cuando dos personas que se conocen juegan a *Piedra-Papel-Tijeras* el resultado es, casi el 80% de las veces, un empate. Eso se debe a que tan sólo hay tres elementos. Por eso propuso una variante con dos elementos más, lo que disminuye la posibilidad de empate. Se trata de *Rock-Paper-Scissors-Lizard-Spock*, juego popularizado unos años después por la serie *Big Bang Theory*.

### 1.Descripción de la práctica

En esta práctica tienes que ir desarrollando de manera incremental un programa que permita jugar a *Rock-Paper-Scissors-Lizard-Spock* (en español, *Piedra-Papel-Tijeras-Lagarto-Spock*). En el juego intervienen dos jugadores. Cada jugador hace su elección de entre las alternativas existentes y el ganador se determina atendiendo a las siguientes reglas:



*“Las tijeras ganan al papel (lo cortan), el papel gana a la piedra (la envuelve), la piedra gana al lagarto (lo aplasta), el lagarto gana a Spock (lo envenena), Spock gana a las tijeras (las rompe), las tijeras ganan al lagarto (lo decapitan), el lagarto gana al papel (lo devora), el papel gana a Spock (lo desautoriza), Spock gana a la piedra (la vaporiza) y la piedra gana a las tijeras (las aplasta). Elementos iguales producen un empate.”*

**Versión 1.-** En esta primera versión de la práctica tienes que desarrollar un programa que permita jugar a *Piedra-Papel-Tijeras-Lagarto-Spock*. El jugador (humano) se enfrentará a la máquina.

Una vez iniciado el juego, máquina y jugador harán su elección (primero la máquina -aleatoriamente- y después el jugador) y el programa determinará quién gana cada partida. El programa deberá permitir jugar todas las partidas que sean necesarias hasta que el jugador decida finalizar el juego.

Al finalizar el juego se mostrará el número total de partidas jugadas así como el número de partidas ganadas, empatadas y perdidas por el jugador.

**Versión 2.-** En esta segunda versión de la práctica tienes que ampliar el funcionamiento anterior de manera que sólo se permita jugar a jugadores previamente registrados y que éstos puedan obtener información sobre las reglas del juego.

Las reglas se encuentran almacenadas en un archivo de texto `reglas.txt` que termina con un centinela `XXX` en la última línea.

Cada jugador está registrado usando un apodo (un apodo es una palabra y no puede haber dos apodos iguales). El registro de jugadores se mantiene en un archivo `registro.txt`. El archivo `registro.txt` almacena el apodo de cada jugador registrado en una línea y termina también con el centinela `XXX` en la última línea (por tanto un apodo no podrá ser `XXX`). Por ejemplo, si hay 3 jugadores registrados con los apodos `duende95`, `Zombi` y `oziko85`, éste sería el contenido del archivo:

```
duende95↓  
Zombi↓  
oziko85↓  
XXX
```

En esta nueva versión, lo primero que hará el programa es pedir al jugador que se identifique mediante su apodo. Gracias al registro de apodos que se mantiene en el archivo `registro.txt`, el programa podrá saber si es un jugador que ya está registrado o no. Si no está registrado el jugador o el archivo `registro.txt` no existe, el programa finalizará directamente. Si por el contrario es un jugador registrado, el programa le dará a elegir entre jugar, ver las reglas del juego o salir. Si el jugador elige ver las reglas se mostrará por pantalla el contenido del archivo `reglas.txt`. Por lo demás el programa se comportará de manera similar a la versión 1.

## 2. Detalles de implementación

La práctica debe contar con dos enumerados:

- `tElemento` con un símbolo por cada una de las posibles jugadas (PIEDRA, PAPEL, TIJERAS, LAGARTO, SPOCK) y un último símbolo `INVALIDO` útil para indicar errores.
- `tResultado` para almacenar el resultado de una partida entre un humano y una máquina. Los valores posibles serán `HUMANO`, `COMPUTADORA` y `EMPATE`.

Deberá tener, además, al menos, los siguientes subprogramas:

- `int leeOpcion(string pregunta, string mensajeError, int valMin, int valMax);` que escribe la `pregunta` del parámetro y espera que el usuario introduzca un entero entre `valMin` y `valMax` para devolverlo. Si el entero introducido no está entre los límites escribe `mensajeError` y vuelve a preguntar.
- `string toString(tElemento);` que devuelve la cadena que describe la jugada indicada por el `tElemento` (piedra, papel, tijeras, lagarto, Spock).
- `tElemento leeJugada(string pregunta);` que pide al usuario que escriba la jugada que quiere aplicar (cadenas como las devueltas por el `toString` anterior) y devuelva el `tElemento` correspondiente. Realiza la pregunta una y otra vez hasta que el usuario escribe una opción válida. Para la implementación conviene disponer de una función `tElemento fromString(string)` simétrica a `toString`.
- `bool gana(tElemento jugador1, tElemento jugador2);` devuelve `true` si la jugada del primer jugador vence a la del segundo.
- `bool localizacionJugador (string apodo);` que recibe el apodo proporcionado por la persona que quiere jugar y devuelve un booleano que indica si puede jugar o no (no podrá jugar si el archivo con el registro de jugadores no existe o si, existiendo, no contiene un apodo igual al indicado).
- `bool escribeFichero(string nombreFichero, string marcaFin);` que abre el fichero `nombreFichero` y escribe su contenido línea a línea hasta que se alcanza una línea igual a `marcaFin`.
- `tResultado juegaPartida();` que juega una partida entre el jugador y la máquina, devolviendo el resultado final.
- `void sesionDeJuego();` una sesión de juego completa, en la que se lleva la cuenta de las estadísticas de partidas ganadas, perdidas y empatadas y que se juega hasta que el usuario indique que quiere terminar.

Para generar números aleatorios debes utilizar las funciones `rand()` y `srand(semilla)` de la biblioteca `cstdlib`. Una secuencia de números aleatorios comienza en un primer número entero que se denomina semilla. Para establecer la semilla el programa deberá invocar a la función `srand` con el argumento deseado. Lo que hace que la secuencia se comporte de forma aleatoria es precisamente la semilla. Una semilla habitual es el valor de la hora del sistema que se obtiene con una invocación a `time(NULL)`, de la biblioteca `ctime`, ya que así es siempre distinta para cada ejecución. Así pues, el programa deberá invocar una vez a `srand(time(NULL))`. Una vez establecida la semilla, la función `rand()` genera, de forma pseudoaleatoria, otro entero positivo a partir del anterior. Si quieres que los números aleatorios generados estén en un determinado intervalo, deberás utilizar el operador `%`. Por ejemplo, para obtener un entero aleatorio en el intervalo `[0, limiteSuperior]` hay que usar la expresión `rand() % (limiteSuperior + 1);`.

### 3. Entrega de la práctica

La práctica se entregará en el Campus Virtual por medio de la tarea **Entrega de la Práctica 1**, que permitirá subir el archivo `main.cpp` con el código fuente de la versión 2. Uno de los dos miembros del grupo será el encargado de subirlo, no lo suben los dos.

Recordad poner el nombre de los miembros del grupo en un comentario al principio del archivo de código fuente.

### 4. Comprobaciones parciales de la práctica

El juez alojado en <http://fp.fdi.ucm.es> tiene tres problemas que permiten el envío de versiones parciales de la práctica para comprobar que la implementación de algunos de los subprogramas listados en la sección 2 son correctos.

Los subprogramas se ponen a prueba con un `main` específico para cada una de ellas. Para hacer los envíos al juez, por tanto, debes sustituir el `main` de la práctica por el indicado para ese problema (puedes dejar el otro comentado). Se aconseja *no* esperar a tener la práctica terminada antes de hacer los envíos.

La versión final entregada por el campus *debe superar* estas pruebas para que la entrega se considere válida.

A continuación se detalla cada uno de ellos.

#### 4.1. G-PR1-01 leeOpcion

Pone a prueba el subprograma `leeOpcion` descrito anteriormente. El `main` para probarlo *debe* ser el siguiente (asume incluida la librería `iostream` y añadido el `using namespace std;`):

```
int main() {  
  
    int ncasos;  
    cin >> ncasos;  
  
    for (int i = 0; i < ncasos; ++i) {  
        int a, b;  
        int res;  
        cin >> a >> b;  
        res = leeOpcion("Introduce la opcion:",  
                        "Opcion incorrecta\n", a, b);  
        cout << "La respuesta fue " << res << '\n';  
    }  
  
    return 0;  
}
```

#### 4.2. G-PR1-02 leeJugada

Sirve para comprobar el correcto funcionamiento de `tElemento leeJugada(string)` y `string toString(tElemento)`.

El main con el que hay que probarlo es:

```
int main() {  
  
    int ncasos;  
    cin >> ncasos;  
  
    for (int i = 0; i < ncasos; ++i) {  
        tElemento jugada = leeJugada("");  
        cout << "Jugaste " << toString(jugada) << '\n';  
    }  
  
    return 0;  
}
```

#### 4.3. G-PR1-03 gana

Comprueba que la función `gana` está bien implementada. Pregunta al usuario las dos jugadas y muestra el resultado de la partida.

El main para probarlo es:

```
int main() {  
  
    int ncasos;
```

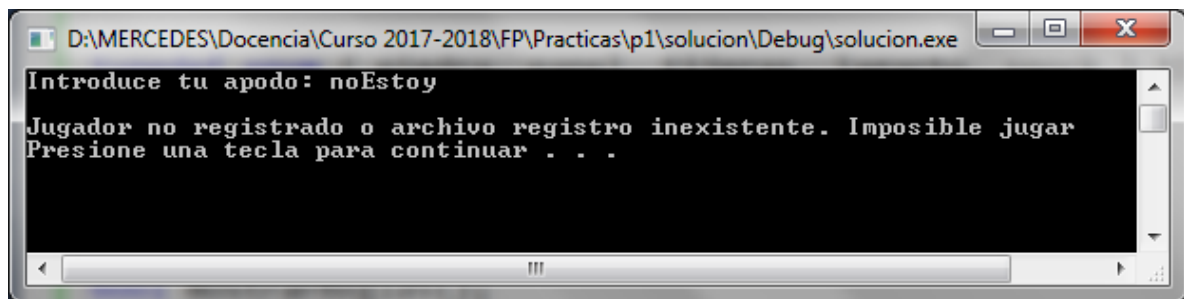
```
cin >> ncasos;

for (int i = 0; i < ncasos; ++i) {
    tElemento jugada1, jugada2;
    jugada1 = leeJugada("");
    jugada2 = leeJugada("");

    if (gana(jugada1, jugada2)) {
        cout << "Gana " << toString(jugada1) << '\n';
    } else if (gana(jugada2, jugada1)) {
        cout << "Gana " << toString(jugada2) << '\n';
    } else {
        cout << "Empate";
    }
}

return 0;
}
```

## Ejemplos de ejecución de la versión 2



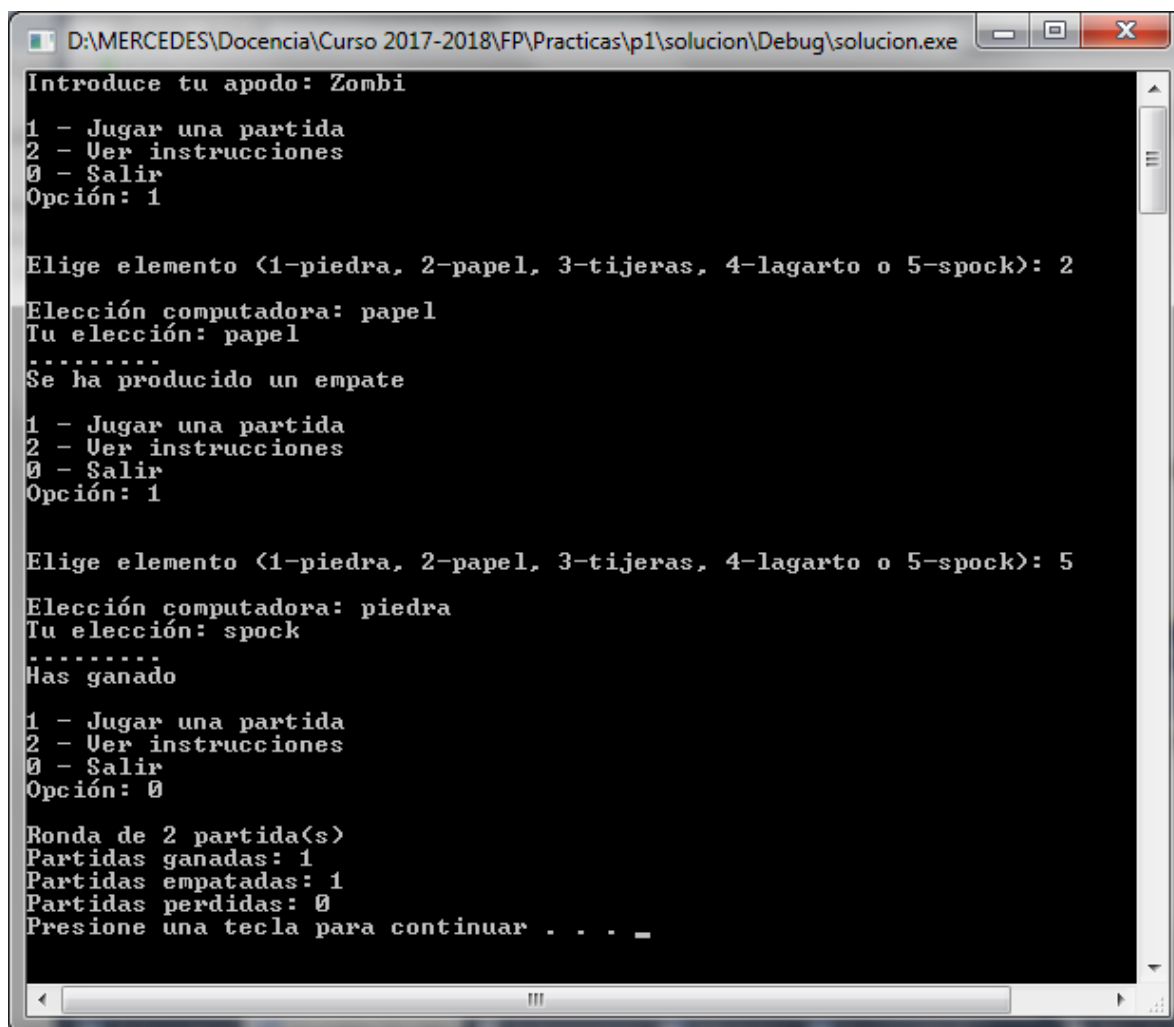
Usuario no registrado

```
D:\MERCEDES\Docencia\Curso 2017-2018\FP\Practicas\p1\solucion\Debug\solucion.exe
Introduce tu apodo: Zombi
1 - Jugar una partida
2 - Ver instrucciones
0 - Salir
Opción: 2
El juego se juega entre dos jugadores: la computadora y el usuario.
En cada jugada, la computadora elige primero su elemento
y luego te pide a ti que lo elijas.
Si ambos elementos son el mismo, entonces se produce un empate.
Si no, quien gane viene establecido por estas reglas:
- Las tijeras cortan el papel
- El papel envuelve a la piedra
- La piedra aplasta al lagarto
- El lagarto envenena a Spock
- Spock rompe las tijeras
- Las tijeras decapitan al lagarto
- El lagarto devora al papel
- El papel desautoriza a Spock
- Spock vaporiza a la piedra
- La piedra aplasta a las tijeras

1 - Jugar una partida
2 - Ver instrucciones
0 - Salir
Opción: 0

Ronda de 0 partida(s)
Partidas ganadas: 0
Partidas empatadas: 0
Partidas perdidas: 0
Presione una tecla para continuar . . . _
```

Usuario registrado (*Zombi*) que visualiza las reglas del juego



The screenshot shows a Windows command prompt window titled "D:\MERCEDES\Docencia\Curso 2017-2018\FP\Practicas\p1\solucion\Debug\solucion.exe". The window contains the following text:

```
Introduce tu apodo: Zombi
1 - Jugar una partida
2 - Ver instrucciones
0 - Salir
Opción: 1

Elige elemento <1-piedra, 2-papel, 3-tijeras, 4-lagarto o 5-spock>: 2
Elección computadora: papel
Tu elección: papel
.....
Se ha producido un empate

1 - Jugar una partida
2 - Ver instrucciones
0 - Salir
Opción: 1

Elige elemento <1-piedra, 2-papel, 3-tijeras, 4-lagarto o 5-spock>: 5
Elección computadora: piedra
Tu elección: spock
.....
Has ganado

1 - Jugar una partida
2 - Ver instrucciones
0 - Salir
Opción: 0

Ronda de 2 partida(s)
Partidas ganadas: 1
Partidas empatadas: 1
Partidas perdidas: 0
Presione una tecla para continuar . . . _
```

Usuario registrado (*Zombi*) que juega dos partidas