# Práctica 2
# *Mastermind*

**Fecha de entrega: 21 de diciembre de 2017**

## 1. Descripción de la práctica

"*Mastermind*" es un conocido juego de ingenio para dos jugadores. En cada partida, uno de ellos elige un código compuesto por una sucesión de colores que el oponente debe descubrir. Para eso, ante cada hipótesis propuesta, el primer jugador debe indicar cuántos de los colores aventurados están colocados en sus posiciones correctas, y cuántos están descolocados.

El número de colores disponibles y la longitud del código dependen de la variante del juego. Nosotros usaremos los colores rojo (`R`), azul (`Z`), verde (`V`), amarillo (`A`), marrón (`M`) y blanco (`B`). Los códigos tendrán una longitud de 4 colores. A modo de ejemplo, la siguiente tabla muestra las respuestas para varias hipótesis si el código oculto fuera BBMM:

| Hipótesis | Colocados | Descolocados |
|---|---|---|
| **RZVA** | 0 | 0 |
| **RRBR** | 0 | 1 |
| **RBRR** | 1 | 0 |
| **MMMM** | 2 | 0 |

En las secciones siguientes se detallan la parte obligatoria de la práctica y la parte opcional.

## 2. Versión 1 (parte obligatoria)

### 2.1. Descripción

En esta versión de la práctica tendréis que programar el juego *Mastermind*. El ordenador tomará el papel de *elegir el código secreto*, y el jugador tendrá que descubrirlo. Al principio se mostrará una descripción del juego junto con un menú en el que el usuario podrá elegir si quiere permitir que el código elegido tenga o no colores repetidos o salir del juego. Después, irá dando hipótesis, se admiten tanto letras mayúsculas como minúsculas, y el ordenador indicará los aciertos de cada una. El número de intentos estará limitado a 15 y cada hipótesis lanzada podrá contener colores repetidos aunque la modalidad de juego elegida sea la que garantiza que no habrá repetidos en el código secreto. Cuando se acierte el código

o se alcance el número máximo de intentos, la partida terminará y se volverá al menú. Si la hipótesis contiene caracteres no válidos o no tiene la longitud adecuada, se le indicará al usuario el error y no contará como un intento.

A continuación puedes ver un ejemplo de ejecución. En cursiva y negrita se muestra lo introducido por el usuario. Verás que, con el fin de poder probar fácilmente la corrección de la práctica, justo al comienzo del juego se muestra un mensaje que contiene el código secreto que el jugador debe tratar de adivinar.

```
      [ Este programa no usa tildes por motivos tecnicos ]

                          Mastermind
                          ==========

Descubre el codigo secreto! En cada partida, pensare un codigo de
colores que tendras que adivinar. En cada intento que hagas te
dare pistas, diciendote cuantos colores de los que has dicho estan
bien colocados, y cuantos no.

Averigua el codigo secreto en el menor numero posible de intentos!


    1. Jugar con un codigo sin colores repetidos
    2. Jugar con un codigo con colores repetidos

    0. Salir

        Elige una opcion: 3
        Opcion incorrecta. Prueba otra vez: -1
        Opcion incorrecta. Prueba otra vez: 1
    [INFO para depuracion] Codigo secreto: BZMR
Introduce el codigo (palabra de 4 letras con alguna de R, Z, V, A, M o B): BRAB
BRAB Colocados: 1; mal colocados: 1
Introduce el codigo (palabra de 4 letras con alguna de R, Z, V, A, M o B): zmzm
ZMZM Colocados: 0; mal colocados: 2
Introduce el codigo (palabra de 4 letras con alguna de R, Z, V, A, M o B): BZZZ
BZZZ Colocados: 2; mal colocados: 0
Introduce el codigo (palabra de 4 letras con alguna de R, Z, V, A, M o B): BzRM
BZRM Colocados: 2; mal colocados: 2
Introduce el codigo (palabra de 4 letras con alguna de R, Z, V, A, M o B): BzMR
BZMR Colocados: 4; mal colocados: 0
Enhorabuena! Lo encontraste!
Te ha costado 5 intento(s).

    1. Jugar con un codigo sin colores repetidos
    2. Jugar con un codigo con colores repetidos

    0. Salir

        Elige una opcion: 0
```

## 2.2. Detalles de implementación

La implementación debe contar con el enumerado `tColor` cuyos posibles valores son `ROJO`, `AZUL`, `VERDE`, `AMARILLO`, `MARRON` y `BLANCO`. Puedes incorporar además valores especiales, como `INCORRECTO`. También es aconsejable que implementes las funciones `color2char` y `char2color` para convertir un `tColor` de y hacia el **`char`** que se le muestra al usuario para representar ese color.

Relacionado con los tipos, debes definir también:

- Una constante `TAM_CODIGO` que estará inicializada a 4 y que indica la longitud del código con la que se juega.
- El tipo `tCodigo` como un vector de tamaño `TAM_CODIGO` que almacena elementos de tipo `tColor`.

Para manejar el tipo `tCodigo` debes contar con, al menos, los siguientes subprogramas:

- **`void`** `codigoAleatorio(tCodigo codigo,` **`bool`** `admiteRepetidos)`: elige aleatoriamente un código y lo devuelve con el parámetro de salida. El segundo parámetro permite indicar si se admiten colores repetidos en él.
- `void compararCodigos(`**`const`** `tCodigo codigo,` **`const`** `tCodigo hipotesis, int& colocados, int& descolocados)`: devuelve el número de colores colocados y descolocados que hay en `hipotesis` con respecto al parámetro `codigo`.

Esta versión debe estar programada de tal forma no requiera modificaciones drásticas en el código la incorporación de nuevos colores o el cambio en el tamaño del código. En concreto, para el tamaño del código debería ser necesario únicamente cambiar la constante `TAM_CODIGO`, mientras que la creación de nuevos colores debería implicar los mínimos cambios posibles, entre los que estarían el cambio del `tColor` y las funciones de conversión de **`char`** a `tColor` y viceversa.

# 3. Versión 2 (parte opcional)

## 3.1. Descripción

La versión 2 de la práctica es una extensión opcional de la primera versión en la que, cuando la máquina detecta que con toda la información proporcionada al jugador éste debería ser capaz de deducir el código secreto, se lo hace saber.

A continuación aparece un ejemplo de ejecución (se omite la explicación inicial). Igual que antes, aparece en negrita y cursiva el texto introducido por el usuario. Además, aparece resaltada la salida nueva de esta versión.

```
    1. Jugar con un codigo sin colores repetidos
    2. Jugar con un codigo con colores repetidos

    0. Salir

        Elige una opcion: 2
    [INFO para depuracion] Codigo secreto: VVAA
Introduce el codigo (palabra de 4 letras con alguna de R, Z, V, A, M o B): RZMB
RZMB Colocados: 0; mal colocados: 0
Introduce el codigo (palabra de 4 letras con alguna de R, Z, V, A, M o B): AAVV
AAVV Colocados: 0; mal colocados: 4
Venga, que con todo lo que te he dicho ya deberias saberlo...
Introduce el codigo (palabra de 4 letras con alguna de R, Z, V, A, M o B): AAAA
AAAA Colocados: 2; mal colocados: 0
Venga, que con todo lo que te he dicho ya deberias saberlo...
Introduce el codigo (palabra de 4 letras con alguna de R, Z, V, A, M o B): VVAA
VVAA Colocados: 4; mal colocados: 0
Enhorabuena! Lo encontraste!
Te ha costado 4 intento(s).

    1. Jugar con un codigo sin colores repetidos
    2. Jugar con un codigo con colores repetidos

    0. Salir

        Elige una opcion: 0
```

## 3.2. Detalles de implementación

Para la versión opcional se necesita una forma de representar el "estado mental" del jugador. Para eso se guarda para cada uno de los códigos posibles si, con las respuestas dadas, ese código podría ser el código secreto o no. Inicialmente todos los códigos son posibles, salvo los que tienen valores repetidos cuando se está jugando sin valores repetidos. Cuando el jugador formula una hipótesis, aquellos códigos en los que no coinciden el número de elementos colocados y descolocados con los de la hipótesis deben calificarse como códigos no posibles.

Declara un tipo `tCodigos` que sea un array de `tCodigo,` el cual nos permite representar todos los códigos posibles, y el tipo `tCodigosPosibles` que sea un array del mismo tamaño que el anterior pero de valores booleanos, que nos permite anotar los códigos que, teniendo en cuenta la información obtenida de hipótesis previas, no podrían ser la solución de la partida actual. Implementa los siguientes subprogramas:

- **`void`** `inicializaIA(`**`bool`** `repetidosPermitidos, tCodigos codigos, tCodigosPosibles posibles)`: que genera el "estado mental" inicial del jugador, es decir, genera todos los códigos posibles y los pone en `codigos` y pone a **`true`** el vector `posibles`, excepto para aquellas entradas con colores repetidos si éstos no se permiten.

- `bool quedaSoloUnoPosible(const tCodigosPosibles posibles)`: que devuelve cierto si, con lo que sabemos, únicamente hay un código secreto válido.
- `void tachaIncompatibles(const tCodigo codigo, int colocados, int descolocados, const tCodigos codigos, tCodigosPosibles posibles)`: dado un "estado mental", tacha todos aquellos códigos que no puedan ser código secreto, teniendo en cuenta que en la hipótesis dada en el parámetro `codigo` hay `colocados` colores en su sitio y `descolocados` colores en un sitio que no es el correcto.

# 4. Entrega de la práctica

La práctica se entregará en el Campus Virtual por medio de la tarea **Entrega de la Práctica 2**, que permitirá subir el archivo `v1.cpp` con el código fuente de la versión 1 si se hace sólo la versión obligatoria o el archivo `v2.cpp` si se realiza también la parte opcional. 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.