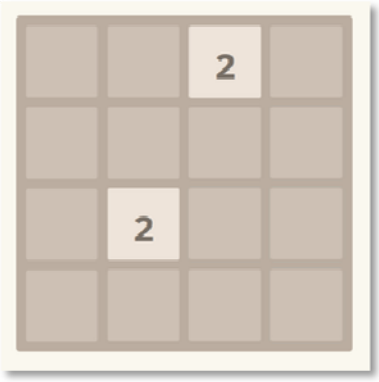
Práctica 3
2048

Fecha de entrega: 29 de marzo

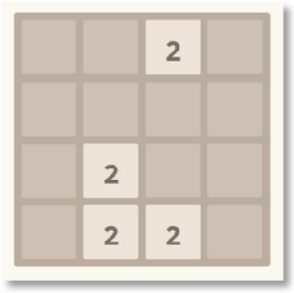
1. Descripción del juego

La práctica consiste en desarrollar un programa en C++ para jugar a **2048**, un juego inventado en marzo de 2014 por Gabriele Cirulli, un joven desarrollador web italiano.

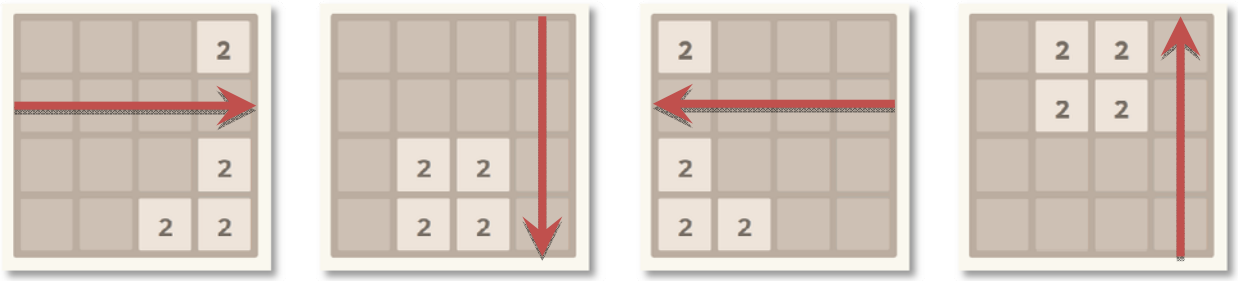


El juego utiliza un tablero de 4 x 4 baldosas que pueden tener fichas de distintos valores, siempre potencias de 2. Inicialmente se ponen dos fichas de valor 2 o 4 en baldosas elegidas aleatoriamente (con mucha más probabilidad de ser un 2 que de ser un 4).

A partir de ese momento el jugador realizará sucesivas jugadas, cada una inclinando el tablero en una dirección (arriba, abajo, derecha o izquierda). Todas las fichas *caerán* en esa dirección, cubriendo los huecos libres para que todas las fichas de cada fila o columna queden juntas hacia el borde correspondiente. Por ejemplo, con este tablero[1]...

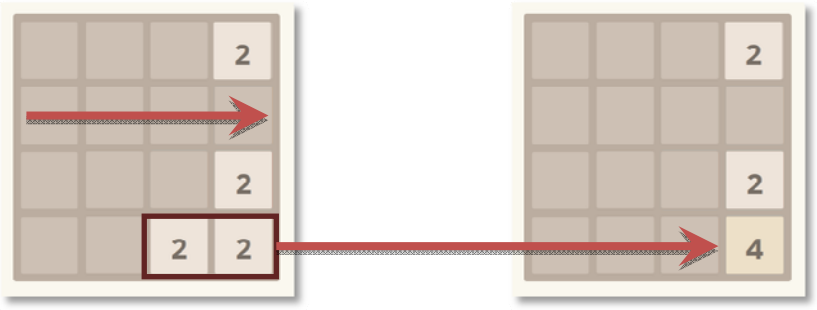


...cada movimiento desplaza las fichas como se muestra a continuación:



[1] Ilustraciones basadas en la que aparece en la entrada de la Wikipedia *2048 (videojuego)*

Una vez desplazadas las fichas, si quedan dos del mismo valor pegadas en alguna fila o columna del sentido del movimiento, entonces se combinan ambas en una sola con la suma de los valores, sustituyendo la que esté más hacia el borde de la dirección elegida y desplazando las demás para cubrir el hueco. La ficha resultante no podrá combinarse con otra nuevamente en la misma jugada. Por ejemplo, inclinando el siguiente tablero hacia la derecha ocurre lo siguiente:

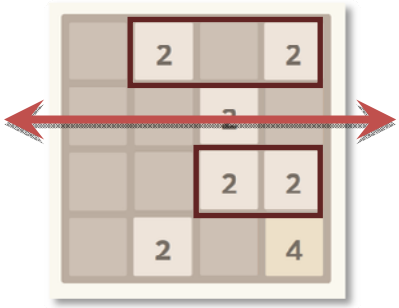


Los valores de las fichas que se consiguen con las combinaciones son obviamente potencias de 2: 4, 8, 16, 32, 64, 128, 256, 512, 1024 y 2048.

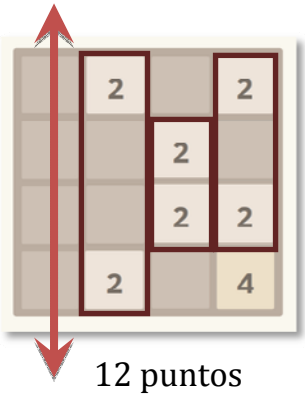
Tras cada jugada, con algún cambio, aparece una nueva ficha, en una baldosa libre, elegida aleatoriamente, de valor 2 o 4 (con mucha más probabilidad de ser un 2).

A medida que van apareciendo nuevas fichas y se realizan combinaciones, el tablero se va poblando de fichas con distintos valores:

El objetivo del juego es conseguir una ficha de valor 2048. En el momento en que se consigue, el juego termina (en la versión original se puede seguir). La puntuación total obtenida es igual a la suma de las puntuaciones de cada jugada. La puntuación de cada jugada es la suma de las combinaciones de fichas que se consigan:



8 puntos

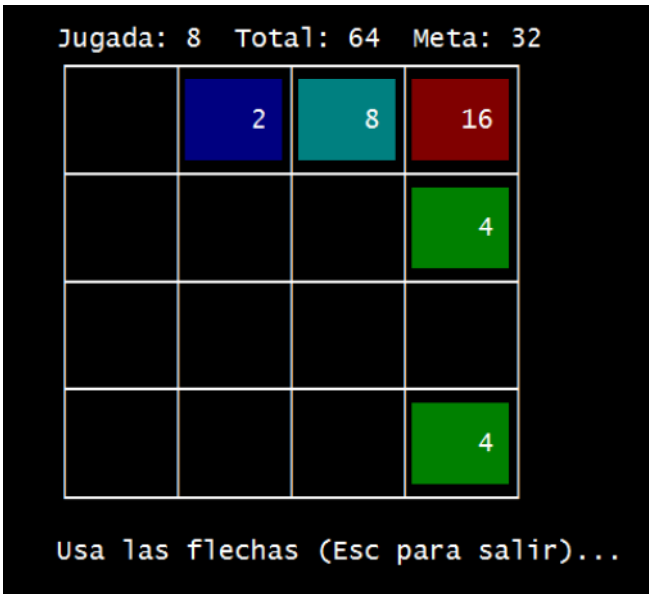


12 puntos

Si, no habiendo conseguido la ficha 2048, se llena el tablero de fichas y no queda ningún movimiento posible, entonces el jugador pierde el juego.

2. El programa

El programa simulará de forma realista, aunque en modo de consola, la dinámica del juego. Utilizará *caracteres gráficos* para dibujar el tablero, así como colores de fondo para las fichas, y leerá teclas especiales, como las teclas de flecha o la tecla Esc.



Principalmente el programa usa un array bidimensional de `DIMxDIM` enteros para mantener el estado del tablero. En la versión clásica `DIM` es 4, pero también hay variaciones del juego con tableros de otras dimensiones como 5x5, 6x6 y 8x8. Declara la constante `DIM` y el tipo `tTablero` (primero lee Estructura el juego del apartado opcional 2.4). Usa el valor 1 (2^0) para las baldosas libres y el valor de la ficha para las ocupadas. El valor 2048 también puede ser otra potencia de 2, define la constante `META` (para pruebas usa 16).

El programa usa también los tipos enumerados `tDireccion` y `tOpcion` con los valores: `Arriba`, `Abajo`, `Derecha` e `Izquierda` y `OP_Arriba`, `OP_Abajo`, `OP_Derecha`, `OP_Izquierda` y `OP_Salir` respectivamente.

2.1. Inicialización y visualización del juego

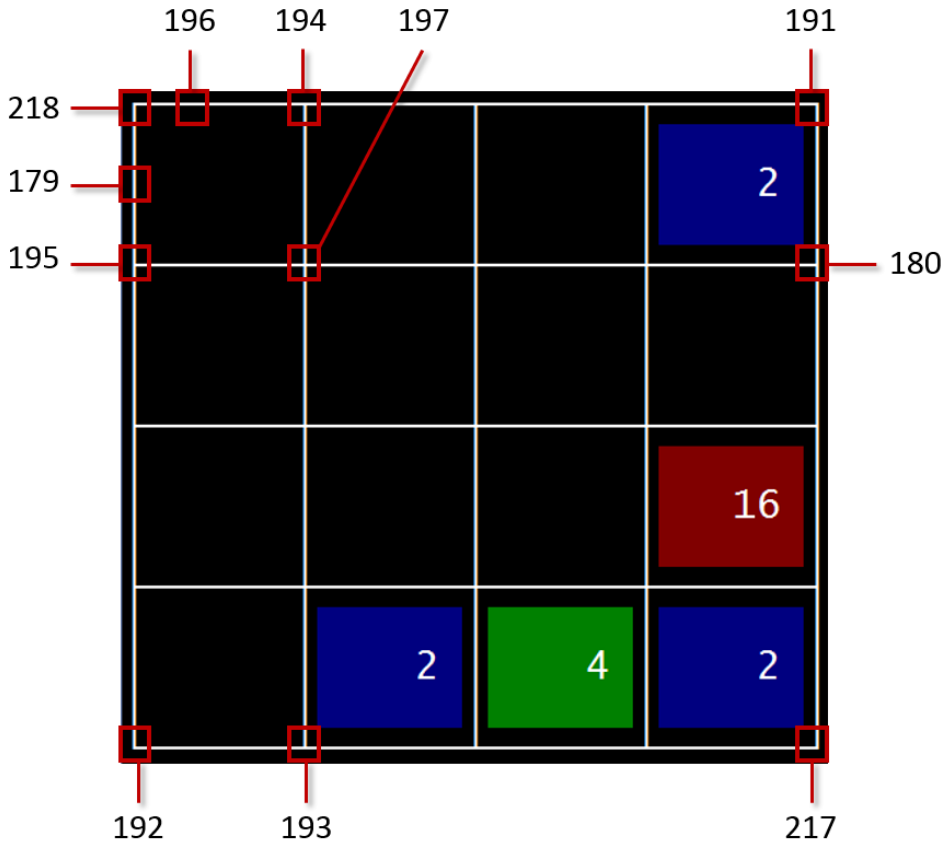
Lo primero que habrá que hacer es inicializar el juego haciendo que todas las baldosas del tablero estén libres (valor 1) menos dos elegidas aleatoriamente que contendrán un 2 o un 4, siendo la probabilidad de ser un 2 del 95% y no pudiendo ser ambas 4.

Una vez que hayamos inicializado el juego lo mostraremos por pantalla tal y como se puede ver en la ilustración anterior. Las celdas libres y los bordes tienen fondo negro. Cada celda usa un recuadro de 3 x 6 con un color de fondo que depende del valor de la celda y un valor que se muestra dentro en color blanco. Sobre el tablero se muestra la última puntuación, la puntuación total y la meta.

Cada vez que vayas a visualizar el estado del juego borra primero el contenido de la ventana, de forma que siempre se muestre el tablero en la misma posición y la sensación sea más visual (durante las pruebas es mejor dejar que se vean los sucesivos estados del juego). Para borrar la consola define y utiliza la función:

```
void clearConsole() { system("cls"); }
```

Tras mostrar las puntuaciones parcial, total y meta en una primera línea, dibuja el tablero en la ventana, teniendo en cuenta que para los bordes de las celdas necesitas caracteres especiales. A continuación te indicamos los códigos que tienes que usar (haciendo un molde a `char`) para obtener cada *carácter gráfico*:



Sin contar los bordes, cada fila comprende tres líneas de pantalla y cada celda seis columnas. Si la celda está vacía, el fondo es negro, pero si hay un valor, los 6 x 3 caracteres se mostrarán con un color de fondo que dependerá del valor de la celda. El valor de la celda se mostrará en la fila de en medio, con un espacio a ambos lados y una anchura de campo de 4.

Al dibujar el tablero, obviamente por filas, ten en cuenta que las esquinas superiores (códigos 218, 194 y 191) e inferiores (códigos 195, 197, 180, 192, 193 y 217) usan caracteres distintos de los bordes interiores (código 196 y 179). Ten en cuenta también que los bordes interiores horizontales (código 196) se repiten seis veces y los bordes interiores verticales (código 179) se repiten tres veces.

Colores de fondo en la consola

Por defecto, el color de primer plano, aquel con el que se muestran los trazos de los caracteres, es blanco, mientras que el color de fondo es negro. Podemos cambiar esos colores, por supuesto, pero utilizando rutinas que son específicas de Windows (hay que incluir la biblioteca `<windows.h>`), por lo que debemos ser conscientes de que el programa no será portable.

En nuestro programa el color de primer plano siempre es blanco, pero queremos utilizar como color de fondo el exponente de la potencia de 2 del valor de la celda, por ejemplo, para las celdas con valor 2 usaremos como color de fondo el 1 (azul), para las celdas con valor 4 el 2 (verde)... Debes cambiar el color de fondo cada vez que tengas que *dibujar* parte de una ficha y volverlo a poner a negro (0) a continuación.

Para cambiar el color de fondo define y utiliza la función:

```
void backgroundTextAtt(int color) {
  SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 15 | (color << 4));
}
```

Basta proporcionar un color (de 0 a 15) y la función `backgroundTextAtt` lo establecerá como color de fondo, con el color blanco (15) para la línea del carácter.

La función `SetConsoleTextAttribute`, de la biblioteca `windows`, permite cambiar tanto el color de fondo como el de la línea del carácter. Disponemos de 16 colores diferentes entre los que elegir, con valores de 0 a 15, tanto para el primer plano como para el fondo. El 0 es el negro y el 15 es el blanco. Los demás son azul, verde, cian, rojo, magenta, amarillo y gris, en dos versiones, oscuro y claro.

Añade al proyecto un módulo (archivos `utilsWin.h` y `utilsWin.cpp`) para todas las funciones cuya implementación requiere `<windows.h>`.

Carga de partidas

También queremos poder cargar partidas a medias. Como podemos cambiar la dimensión del tablero y la meta, los archivos deben contener esta información, así como los puntos alcanzados hasta el momento.

Los archivos comenzaran con una línea que indique la meta y la dimensión y sólo se cargará si estos datos coinciden con los valores de `META` y `DIM` en el programa. Si es así, se leerán `DIMxDIM` enteros colocándolos, por filas, en el tablero. El archivo termina con los puntos alcanzados en la partida hasta ese momento. A la derecha puedes ver un ejemplo de archivo con los datos de una partida.

```
128  4   ← Meta y dimensión
1
1
2
2        ← 1ª fila
4
2
1
8        ← 2ª fila
...
1
1
16
1        ← Última fila
78   ← Puntos
```

Implementación

Para empezar define la función `main` de forma que inicie el generador de números aleatorios y pregunte al usuario si quiere cargar una partida; si es así, pedirá el nombre del archivo y cargará la partida. Si no se quiere cargar una partida, o no se ha podido, iniciará una nueva partida. A continuación visualizará el tablero y preguntará si se quiere guardar el estado de la partida; si es así pedirá el nombre del archivo y la guardará. El programa termina solicitando al usuario que pulse la tecla `Enter` para cerrarse. Define las constantes y los tipos e implementa, al menos, los siguientes subprogramas:

- ✓ **`void inicializa(tTablero tablero)`**: Inicializa el `tablero`, haciendo que todas las celdas estén libres, excepto dos elegidas aleatoriamente. Una de esas celdas será un 2 y la otra 2 o 4, siendo la probabilidad de ser un 2 del 95%.
- ✓ **`void nuevaFicha(tTablero tablero)`**: Crea, en una posición del `tablero` elegida aleatoriamente y que esté libre, una nueva ficha (con una probabilidad del 95% de ser un 2 y una probabilidad del 5% de ser un 4).
- ✓ **`int log2(int num)`**: Devuelve el exponente al que hay que elevar 2 para obtener `num` ($1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, ..., $2048 = 2^{11}$)
- ✓ **`void visualiza(const tTablero tablero, int puntos, int totalPuntos)`**: Muestra el `tablero`, con los `puntos` de la última jugada, el total de puntos y la meta. Te será útil definir constantes para los caracteres especiales y funciones auxiliares: `visualiza(const tTablero tablero, int fila)`, `bordeSuperior()`, `bordeInferior()` y `bordeInter()`.
- ✓ **`bool carga(tTablero tablero, int &puntos)`**: Pide un nombre de archivo e intenta cargar en el `tablero` los datos de archivo, así como los `puntos`. Si no se puede cargar el archivo, inicializa el `tablero` y pone a cero los `puntos`.
- ✓ **`void guarda(const tTablero tablero, int puntos)`**: Pide un nombre de archivo y guarda la meta, la dimensión del `tablero`, los datos del tablero (por filas) y los `puntos` obtenidos hasta el momento.

2.2. Inclinación del tablero

Vamos a añadir la posibilidad de inclinar el tablero y hacer que las fichas *caigan* en esa dirección, cubriendo los huecos libres para que todas las fichas de cada fila o columna queden juntas hacia el borde correspondiente. Para ello tras mostrar el tablero, el programa solicitará al usuario la dirección en la que desea *inclinarlo*. El usuario introducirá dicha dirección con las teclas de flecha. El tablero se inclinará sucesivamente en las direcciones que vaya introduciendo el usuario hasta que este pulse `Esc`, momento en que la partida terminará.

Lectura de teclas especiales

La tecla Esc genera un código ASCII (27), pero las de flecha no tienen códigos ASCII asociados y no se pueden leer con `get`. Cuando se pulsan en realidad se generan dos códigos, uno que indica que se trata de una tecla especial y un segundo que indica de cuál se trata.

La función `ReadConsoleInput` de la biblioteca `windows` nos permite leer directamente del teclado sin tener que esperar a que se pulse la tecla Enter.

Define y utiliza la función:

```
int getKey() {
   int key;
   INPUT_RECORD record;  DWORD num;
   HANDLE hStdIn = GetStdHandle(STD_INPUT_HANDLE);
   do {
      ReadConsoleInput(hStdIn, &record, 1, &num);
   } while(record.EventType !=  KEY_EVENT );
   if (record.Event.KeyEvent.uChar.AsciiChar == 0) { // teclas especiales
      ReadConsoleInput(hStdIn, &record, 1, &num);    // leer otro
      key = record.Event.KeyEvent.wVirtualKeyCode;
  } else key = record.Event.KeyEvent.uChar.AsciiChar;
  return key;
}
```

Las constantes para las teclas de las opciones del menú:

```
const int TE_Salir     = VK_ESCAPE;
const int TE_Arriba    = VK_UP;
const int TE_Abajo     = VK_DOWN;
const int TE_Derecha   = VK_RIGHT;
const int TE_Izquierda = VK_LEFT;
```

Y el tipo `tCoord` formado por dos enteros (`x` e `y`) para las posiciones del tablero.

Modifica la función `main` para que después de mostrar el tablero inicial lo vaya inclinando tantas veces como desee el usuario. Implementa, al menos, los siguientes subprogramas:

- ✓ **`tOpcion menu()`**: Devuelve una de las cinco posibles opciones. `OP_Salir`, si se pulsa la tecla `TE_Salir`, `OP_Arriba` si se pulsa la tecla `TE_Arriba`, etc. Esta función mostrará el mensaje `"Usa las flechas (Esc para salir)..."` y usará la función `getKey` para detectar las teclas de flecha (o Esc). Se seguirán leyendo teclas mientras éstas no se correspondan con una opción válida.

- ✓ **`bool mueveFichas(tTablero tablero, tDireccion direccion)`**: Desplaza las fichas del `tablero` en la `direccion` indicada. Devuelve `true` si se ha producido algún movimiento.
- ✓ **`void obtenerCoordMov(tDireccion dir, int ind, tCoord & ini, tCoord & incr)`**: Determina la celda de inicio `ini` y los incrementos `incr` para recorrer la fila o la columna `ind` de acuerdo con la dirección dada.

2.3. Juego completo

Por fin añadimos todo lo necesario para poder jugar al 2048 tal y como se explicó en el apartado 1. Ahora una vez que el usuario indica la dirección en la que mover el tablero, tenemos que realizar dos tareas: desplazar las fichas en la dirección y realizar las posibles combinaciones de fichas. Tras cada jugada con algún cambio, debe aparecer una nueva ficha, en una baldosa libre, elegida aleatoriamente, de valor 2 o 4 (siendo la probabilidad de ser un 2 del 95%). Ahora el programa debe terminar si:

- El usuario pulsa `Esc`.
- El usuario consigue una baldosa con el valor `META`. En este caso el usuario gana.
- El tablero se llena, no queda ninguna baldosa vacía, y no se puede realizar ninguna combinación. En este caso el usuario pierde.

Modifica la función `main` para poder jugar una partida completa de 2048. Implementa, al menos, los siguientes subprogramas:

- ✓ **`int mayor(const tTablero tablero)`**: Devuelve el valor mayor encontrado en las fichas del `tablero`. Si ese valor es `META`, entonces se ha ganado el juego.
- ✓ **`bool lleno(const tTablero tablero)`**: Devuelve `true` si no quedan baldosas libres en el `tablero`; `false` en otro caso.
- ✓ **`tResultado mueveFichas(tTablero tablero, tDireccion direccion, int & puntos, int & total)`**: Combina las fichas que quedan juntas al mover el `tablero` en la `direccion` indicada. Devuelve los puntos, el total de puntos conseguido con todas las combinaciones realizadas y el resultado (`Pierde`, `Gana`, `Jugando`).

Página de códigos y fuente de los caracteres en la consola

Los códigos indicados para los *caracteres gráficos* se corresponden con la página de códigos 850 (OEM), que no se adapta a la página de códigos del editor de Visual Studio. Los caracteres gráficos no existen en la página de códigos 1252 (ANSI Windows). Para cambiar la página de códigos de `cout`, define y utiliza la función:

```
void cpConsoleOut(int cp) { SetConsoleOutputCP(cp); }
```

Recuerda que la página de códigos ANSI no es compatible con la fuente "mapa de bits". Para cambiar la fuente de `cout`, define y utiliza la función:

```
void fontConsole(bool console){
   CONSOLE_FONT_INFOEX cfi;
   cfi.cbSize = sizeof(CONSOLE_FONT_INFOEX);
   cfi.FontWeight = 400;
   if (console) {
      cfi.nFont = 1;
      cfi.dwFontSize.X = 12; cfi.dwFontSize.Y = 20;
      cfi.FontFamily = 54;
      wcscpy_s(cfi.FaceName, L"Lucida Console"); // consola
   }
   else  {
      cfi.nFont = 8;
      cfi.dwFontSize.X = 12; cfi.dwFontSize.Y = 16;
      cfi.FontFamily = 48;
      wcscpy_s(cfi.FaceName, L"Terminal");    // mapa de bits
   }
   SetCurrentConsoleFontEx(GetStdHandle(STD_OUTPUT_HANDLE), false, &cfi);
 }
```

2.4. Opcional.

Estructura el juego (no es necesario hacer la versión anterior)

Añade al proyecto un módulo (archivos `juego2048.h` y `juego2048.cpp`) para gestionar el juego.

- ✓ Define un tipo estructurado `tJuego2048` que agrupe la meta, la dimensión, el tablero (con los exponentes), los puntos obtenidos, la última dirección, el número de celdas libres y el mayor valor de las celdas.
- ✓ Modifica las cabeceras de las funciones para adaptarlas al nuevo tipo de datos y aprovechar las nuevas variables de estado de la partida para mejorar la eficiencia del programa. Pon las declaraciones de constantes, tipos y prototipos en el archivo `.h`, y la implementación de las funciones en el `.cpp` (utiliza un array de constantes para las potencias de dos de los posibles exponentes).

Permite elegir la meta y la dimensión del tablero cuando se inicia una nueva partida.

Registro de las mejores puntuaciones

Añade al programa la gestión de un archivo (para cada `META` y `DIM`) con las hasta diez mejores puntuaciones de los que han conseguido ganar el juego.

El archivo `records_META_DIM.txt` tiene en cada línea un nombre y una puntuación, y termina con ??? como nombre (centinela). Ejemplo:

```
Luis 21763
Ana 20967
Eva 20682
??? 0
```

Actualiza la función `main()` para que cuando un jugador gane una partida se compruebe si es una de las diez mejores (para la `META` y `DIM` de la partida), si es el caso actualice la lista de mejores puntuaciones, y las muestre al usuario. Al inicio se carga el archivo (si no existe `records_META_DIM.txt` se creará) y al final se guarda.

Para ello debes:

- ✓ Añadir al proyecto un nuevo módulo (archivos `records.h` y `records.cpp`) para el tipo (`tRecord`, `tListaRecords`) y las funciones de la lista de records: `cargar()`, `guardar()`, `mostrar()`, `insertar()` (ordenadamente hasta diez records; si ya había diez, el de menor puntuación desaparece de la lista), y `mostrar()`. Recuerda declarar las constantes, tipos y prototipos en el archivo `.h`, e implementar los prototipos en el `.cpp`.
- ✓ Añadir, en el archivo de la función `main`, el subprograma **`void nuevoRecord(tListRecords & list, int puntos)`**: Comprueba si hay que añadir la nueva puntuación a lista de mejores puntuaciones, y si es así, pide al usuario su nombre, inserta en la lista el nuevo record (nombre y puntos) y la muestra.

3. Requisitos de implementación

No olvides declarar las constantes necesarias para hacer tu código reutilizable y fácil de modificar. No olvides incluir los prototipos de tus funciones.

No utilices variables globales: cada subprograma, además de los parámetros para intercambiar datos, debe declarar las variables locales que necesite usar en el código.

No pueden usarse en la resolución de la práctica elementos de C++ no vistos en clase. El programa no debe utilizar instrucciones de salto no estructuradas como `exit`, break (salvo en las cláusulas de la instrucción switch) y `return` (salvo en la última instrucción de las funciones que devuelven un valor).

Se valorará la estructura y eficiencia del programa.

4. Entrega de la práctica

La práctica se entregará a través del Campus Virtual. Se habilitará una nueva tarea **Entrega de la Práctica 3** que permitirá subir el archivo con el código fuente.

Fin del plazo de entrega: **29 de marzo a las 23:55**