Proyecto FINAL

Objetivos.

En este proyecto de laboratorio se pretende que el alumno implemente un programa en C++ haciendo uso de tipos estructurados (cadenas, arrays, estructuras y ficheros). Además, debe ser capaz de aplicar la programación modular.

A tener en cuenta:

1. La entrega se realizará, como muy tarde, el viernes día 17 de enero (hasta las 12 pm), a través del repositorio en diversión.uv.es.
2. Se entrega por parejas (con la persona con la que se han realizado las prácticas) o individualmente.
3. El proyecto se expondrá oralmente por parejas ante el profesor de laboratorio. El profesor citará a los alumnos con suficiente antelación. El profesor puede solicitar a los estudiantes que (por parejas o de forma individual) amplíen o modifiquen en ese momento el código.
4. Es recomendable realizar una tutoría antes del final del cuatrimestre.
5. Se deben seguir estrictamente los requisitos y especificaciones que se indican en el enunciado.
6. Se debe adjuntar un pequeño documento con una breve explicación de:
   - Las definiciones de tipos utilizados. Justificación de su elección.
   - La implementación de los subprogramas: descripción y justificación de los parámetros de entrada y de salida, tipo de paso de parámetros y tarea que realiza cada función.
   - El diagrama de flujo del programa principal.
   - Cualquier detalle sobre la implementación que consideréis relevante.
7. Se entrega: el fichero .cpp, los ficheros de datos con los que habéis realizado las pruebas y la memoria (pdf) explicativa. La entrega del código y los ficheros de datos se realizará por medio de un repositorio de control de versiones Git en el servidor https://diversion.uv.es. La memoria explicativa y videos adicionales se entregará a través de Aula Virtual. Ver Apartado Procedimiento de Entrega.
8. El profesor os indicara el repositorio que tenéis asignado.

Procedimiento de Entrega:

Durante el desarrollo del proyecto se deben seguir las recomendaciones y forma de trabajo propuestas durante la sesión de control de versiones de la asignatura ISU. Se tendrá en cuenta en la calificación final del proyecto.

Debéis realizar un *commit* cada vez que añadáis una nueva estructura de datos, una función, un fichero o cuando completéis una funcionalidad o en general cualquier cambio que se considere relevante. Los mensajes de los *commit* deben explicar qué se ha completado, evitando mensajes genéricos. Siempre que completéis una tarea del enunciado debe quedar claramente reflejado en el mensaje de *commit*.

Descripción general.

Se quiere implementar una versión sencilla del popular **juego del buscaminas** (en inglés: Minesweeper) en un tablero reducido de tamaño fijo de 8x8 celdas. El número máximo de minas es de 8. Puedes echar unas partidas on-line en `http://buscaminas.eu/`.

Además, se desea almacenar información de los jugadores. El fichero `jugadoresInfo.txt` contiene la información de los jugadores que han jugado hasta el momento y han ganado la partida. Si se ha ganado la partida, se quiere añadir los datos del nuevo jugador: nombre, fecha de nacimiento (día, mes y año) y el número de jugadas realizadas. **A lo sumo** se podrá almacenar 100 jugadores.

Las posiciones (fila y columna) de las minas se pueden inicializar de dos maneras: 1) se pueden detallar en un fichero de configuración previamente rellenado. El fichero contiene una mina por línea, ejemplo: fichero `config.txt`; 2) se pueden generar aleatoriamente al inicio de la partida. El número de minas es también aleatorio y con un máximo de 8 minas.

Los ficheros se pueden descargar de Aula Virtual. Estos ficheros son un ejemplo y **se pueden y deben** modificar para ampliarlos.

Considera las siguientes constantes y definiciones de tipos de datos:

```
const int MAX_MINAS = 8;
```

```
const int FIL = 8;
const int COL = 8;

struct Jugador
{
   . . . // completar
};

const int MAX_JUGADORES = 100;
typedef Jugador Vector[MAX_JUGADORES];

struct Estado
{
  unsigned int nMinas; // número de minas vecinas, en las 8 celdas de alrededor
  bool mina; //existe una mina en esa celda
  bool destapada; /la celda está descubierta ya o no aún
  bool bandera; //la celda está marcada con una bandera o no
};

typedef Estado Tablero[FIL][COL];
```

Se pide:

1) Define los tipos de datos (arrays y estructuras) necesarios para almacenar la información del juego.

2) Implementa una función, **Menu**, para que el usuario elija entre las siguientes opciones. Se debe garantizar que la opción elegida es válida.

```
"a. Lee las posiciones de las minas desde fichero."
"b. Genera aleatoriamente las posiciones de las minas."
"c. Descubre celda."
"d. Marca celda como una mina. Añade una bandera."
"e. Desmarca celda como una mina."

char Menu();
```

3) Implementa una función, **InicializaDesdeFichero**, para inicializar el tablero desde fichero, ejemplo fichero `config.txt`. La función debe inicializar el tablero con las posiciones de las minas indicadas en el fichero. El resto de campos del estado se deben inicializar convenientemente.

```
void InitializaDesdeFichero( Tablero, ifstream &);
```

Es conveniente implementar una función, **NumeroMinasVecinas**, que dada una celda, determine cuántas minas existen en sus 8 celdas vecinas. **Ten en cuenta no salirte del tablero**. La celda podría estar en el borde o en una esquina.

```
unsigned int NumeroMinasVecinas( const Tablero, unsigned int, unsigned int);
```

4) Implementa una función, **InicializaAleatoriamente**, para inicializar el tablero de forma aleatoria. La función debe inicializar la matriz con las posiciones de las minas y el número de minas de forma aleatoria (máximo 8 minas). El resto de campos del estado se deben inicializar convenientemente.

```
void InicializaAleatoriamente( Tablero );
```

5) Implementa una función, **MuestraTablero**, para mostrar por pantalla el estado actual del tablero. Si existe una bandera en la celda se muestra el símbolo "^". En caso contrario, si la celda no ha sido aún destapada se muestra el símbolo punto "." ; y si ha sido destapada y contiene una mina se mostrará el símbolo "#". Si ha sido destapada, no existe mina y no tiene ninguna mina alrededor en sus 8 celdas vecinas se muestra un espacio en blanco. Si ha sido destapada, no existe mina pero sí que existen minas a su alrededor se mostrará la cantidad de minas en sus 8 celdas vecinas.

```
void MuestraTablero( const Tablero );
```

Un ejemplo de la impresión del tablero:

```
-------------------------------------
| . | . | . | . | . | . | ^ | 2 |   |
-------------------------------------
| . | . | . | . | . | . | ^ | 2 | 1 |
-------------------------------------
| . | . | . | . | . | . | . | . | . |
-------------------------------------
| . | . | . | . | . | . | . | . | . |
-------------------------------------
| . | . | . | . | . | . | . | . | . |
-------------------------------------
| . | . | . | . | . | . | 2 | 1 | 1 |
-------------------------------------
| . | . | . | . | . | . | 1 |   |   |
-------------------------------------
| . | . | . | . | . | . | 1 |   |   |
-------------------------------------
| . | . | . | . | . | . | 1 | 1 | 1 |
-------------------------------------
| . | . | . | . | . | . | . | . | . |
-------------------------------------
```

6) Implementa una función, **LeeCelda**, para leer desde teclado la fila y columna de una celda. La función debe garantizar que los datos introducidos son válidos.

```
void LeeCelda(unsigned int &, unsigned int & );
```

7) Implementa una **función recursiva**, **AbreCelda**, que destape una celda conocida su fila y columna. Si el número de minas en las 8 celdas vecinas es cero, la función debe abrir de forma recursiva también sus celdas vecinas (si no han sido aún destapadas). **Ten en cuenta no salirte del tablero (podría ser una celda del borde o de una esquina)**.

Dada una celda con fila i y columna j, sus 8 vecinas vienen determinadas por los índices:

```
(i-1,j-1)     (i-1,j)     (i-1,j+1)
 (i,j-1)       (i,j)       (i,j+1)
(i+1,j-1)     (i+1,j)     (i+1,j+1)
```

```
void AbreCelda(Tablero, unsigned int, unsigned int);
```

8) Implementa una función, **FinJuego**, que determine cuando el juego ha terminado. El juego termina bien porque destapamos una celda con una mina o porque se han procesado todas las celdas del tablero.

```
bool FinJuego(const Tablero );
```

Es conveniente implementar dos funciones auxiliares:

```
bool MinaAbierta(const Tablero );
```

```
bool TodasCeldasProcesadas(const Tablero );
```

9) Implementa una función, **LeeJugadoresFichero**, que lea desde fichero los datos de los jugadores y los guarde en un vector. Debe rellenar la cantidad de jugadores actuales en el vector.

```
void LeeJugadoresFichero( Vector, unsigned int &, ifstream & );
```

10) Implementa una función, **LeeInfoJugador**, que lea la información desde teclado de un jugador. El número de intentos se le pasara como datos de entrada, ya que el programa los calcula automáticamente. El resto de datos personales del jugador se leen desde teclado.

```
Jugador LeeInfoJugador(unsigned int); // numero de intentos
```

11) Implementa una función, **InsertaJugadorVector**, que inserte un jugador en el vector. La función debe incrementar el número actual de elementos. La función tiene como parámetros: los datos del nuevo jugador, el vector y su tamaño. La función inserta el nuevo jugador en el vector. La función devuelve `true` si ha sido posible la inserción o `false` en caso de que no quede memoria.

```
bool InsertaJugadorVector( Jugador, Vector, unsigned int & );
```

12) Implementa una función, **EscribeJugadoresFichero**, para guardar el contenido del vector de jugadores en un fichero.

```
void EscribeJugadoresFichero( const Vector, unsigned int, ofstream & );
```

`Se adjunta el fichero pfinal_alumnos.cpp con el código a completar.`

MUY IMPORTANTE:

- Todos los programas deben seguir la Guía de Estilo de C++ e incluir los comandos de Doxygen para generar la documentación (**incluidas las funciones y definiciones de tipos realizadas por el programador**).
- Subir a Aula Virtual todos los archivos .cpp (de uno en uno) sin comprimir (enunciado y adicionales). Sólo los sube un miembro de la pareja.
- Fecha de entrega: durante la sesión de vuestro grupo de lab.
- No está permitido el uso de herramientas automáticas de generación de código, tipo ChatGPT o similares. Su uso se considerará como copia.
- Sólo se pueden usar los tipos de datos y sentencias de control vistas en clase