Ejercicio 1.-Encuentra la mina

Se pretende generar un programa que esconda una mina en una matriz.

- Al inicio, el programa guardará en una variable oculta al usuario las coordenadas (fila/columna) de una bomba. Se generará un tablero formado por casillas en las que el jugador debe buscar esa bomba. Con cada intento del usuario se debe comprobar si las coordenadas introducidas coinciden con las de la bomba. En el momento que se encuentre la bomba acaba el programa. En caso de no encontrarla, se marcan con 'O' las casillas exploradas y con '?' las casillas no exploradas.
- Se debe generar una matriz cuadrada de caracteres de un tamaño dado por el usuario. El tamaño de la matriz se pasa como parámetro de entrada por argc/argv, y se reservará memoria dinámica en función de ese valor.
    - Se pide implementar una función que cree un array dinámico de dos dimensiones de un tamaño dado:
        - char** crearMatriz(int numFilasColumnas)
- Los elementos de la matriz pueden contener el valor:
        - INCOGNITA = '?'
        - AGUA = 'O'
    - Se pide implementar una función que rellene un array dinámico doble con un valor por defecto. Al inicio, se rellenará con el valor '?' los elementos de la matriz doble creada anteriormente usando esta función:
        - void rellenaMatriz(char** matriz, char valor)
- El programa pedirá al usuario coordenadas (fila/columna) dentro de la matriz creada anteriormente. El objetivo es encontrar la bomba escondida, se debe almacenar cada intento del usuario y el resultado del mismo. En caso de no haberse encontrado la bomba, este intento guardará información de ayuda:
        - La bomba está en esa fila
        - La bomba está en esa columna
        - Bomba encontrada
    - Se debe crear una estructura adecuada para almacenar esos intentos, y un array dinámico que crecerá con cada nuevo intento.
        - intento_t pedirIntento();
            - Esta función pide una fila/columna al usuario, lo almacena en una estructura "intento_t" y devuelve al usuario.
        - void testeaIntento(char** matriz, intento_t* nuevoIntento, coordenadas_t coordenadasBomba)
            - Una vez se tienen las coordenadas del intento, se usa esta función para testear si se ha encontrado la bomba. Se pide implementar una estructura "coordenadas_t" que almacene las coordenadas fila/columna de la bomba. Se usarán esas coordenadas en esta función para el test
            - En caso de acierto, se marca el intento como "bomba encontrada" y acaba el programa.

            - En caso de fallo, se marca en el intento si ha acertado la fila o la columna, y se marca en la matriz esas coordenadas como casilla explorada (‘O’)
        - void insertaIntentoEnLista(listaIntentos_t* intentos, intento_t nuevoIntento);
            - Después de haber testeado el intento y no haber encontrado la bomba, se guardará el intento en una estructura “listaIntentos_t” que almacenará el array dinámico de intentos, y el número de intentos que hay en ese momento.
- Para interactuar con el programa, se debe crear un menú que pida datos al usuario. Las opciones de menú son las siguientes:
    - Buscar la bomba:
        - Pedirá por pantalla fila y columna, testeará esas coordenadas y guardará la siguiente información en un array dinámico de intentos:
            - fila, columna, si la bomba se encuentra en esa fila o en esa columna.
        - Toda la información de los distintos intentos se guardará en un array cuya memoria se gestionará de forma dinámica.
    - Visualizar la matriz:
        - Presentará por pantalla la matriz indicando con el caracter ‘?’ que no se ha buscado en esa posición y con ‘O’ que ha sido un intento fallido.
    - Visualizar los intentos:
        - Presentará por pantalla toda la información recogida de cada intento, indicando si se acierta la fila o la columna
    - Salir
        - Presenta la matriz antes de salir, en este caso indicando una ‘X’ donde se encontraba la bomba.
        - Libera toda la memoria dinámica del programa antes de acabar
- Pista: Piensa que primero tendrás que pedir memoria para los punteros de cada fila y luego para cada array de caracteres, cuyo tamaño coincide con el de las columnas. Al ser una matriz cuadrada el tamaño será el mismo tanto para las filas como para las columnas.

  Ejemplo de salida por pantalla:

```
$ ./a.out 5
*******************************
Introduzca una opcion
Para buscar pulse 1
Para visualizar los intentos pulse 2
Para ver la matriz pulse 3
Para salir pulse 0
*******************************
1
Introduce fila y columna:       1 1
*******************************
Introduzca una opcion
Para buscar pulse 1
Para visualizar los intentos pulse 2
Para ver la matriz pulse 3
Para salir pulse 0
*******************************
1
Introduce fila y columna:       2 2
*******************************
```

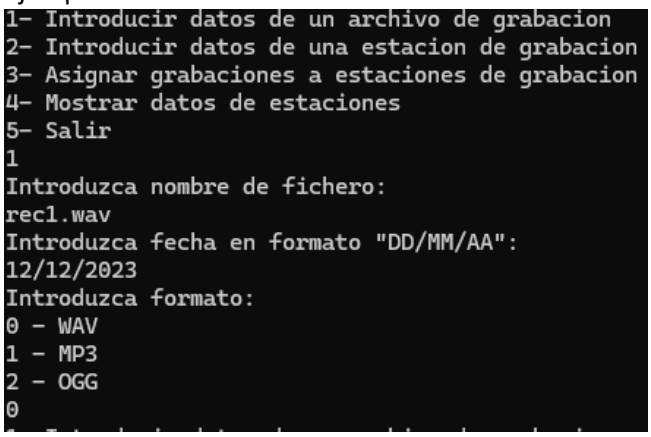
```
Introduzca una opcion
Para buscar pulse 1
Para visualizar los intentos pulse 2
Para ver la matriz pulse 3
Para salir pulse 0
******************************
2

Intento 0
[1][1]...en esa columna


Intento 1
[2][2]... en esa fila

******************************
Introduzca una opcion
Para buscar pulse 1
Para visualizar los intentos pulse 2
Para ver la matriz pulse 3
Para salir pulse 0
******************************
3
? ? ? ? ?
? O ? ? ?
? ? O ? ?
? ? ? ? ?
? ? ? ? ?
******************************
Introduzca una opcion
Para buscar pulse 1
Para visualizar los intentos pulse 2
Para ver la matriz pulse 3
Para salir pulse 0
******************************
1
Introduce fila y columna:       2 1
**********BOOM**********
```

-Ejercicio 2.- Listas de grabaciones

Un proyecto de análisis de eco-acústica necesita almacenar los datos de grabaciones de tal manera que un archivo de audio esté relacionado con sus metadatos de grabación (lugar, zona geográfica, etc..), y una estación de grabación pueda tener una lista de grabaciones asociadas. Es necesario almacenar los datos en dos listas que crecerán de forma dinámica, y deben ocupar el mínimo tamaño de memoria posible. Para ello es necesario crear estructuras que almacenen la siguiente información:

- Archivos de grabación:
    - Nombre del archivo: Conjunto de caracteres alfanuméricos
    - Fecha de grabación: Fecha en formato DD/MM/AAAA
    - Formato usado: Puede ser WAV, OGG, MP3
    - Estación asignada: Conjunto de caracteres alfanuméricos que se corresponden con el nombre de alguna estación. Si no ha sido asignado a ninguna estación, contendrá una cadena vacía.
- Estación de grabación:
    - Nombre de la estación: Conjunto de caracteres alfanuméricos
    - Coordenadas geográficas: Latitud/longitud donde se ubica esta estación
    - Lista de grabaciones: Una lista dinámica que contiene las grabaciones realizadas por esta estación. Para poder mantener el mínimo de memoria, cada grabación introducida debería apuntar a una posición dentro en una lista de grabaciones aparte.

El programa tendrá un menú al inicio que permitirá introducir datos en listas/arrays dinámicos usando las estructuras anteriormente citadas, permitiendo búsquedas en ellas. El menú será el siguiente:

- 1- Introducir datos de un archivo de grabación
    - Al seleccionar esta opción, se pedirán los datos de “nombre de archivo”, “fecha de grabación”, “formato usado”, y se añadirá a la lista de archivos de grabación. No se piden los datos de “estación asignada” (Se puede dejar como cadena vacía o apuntando a NULL).
    - Ejemplo:
    - 

- 2- Introducir datos de una estación de grabación

- Se pedirán los datos “Nombre de estación” y “Coordenadas geográficas” para poder crear una nueva estación, y se añadirá a la lista de estaciones de grabación. No se piden los datos de “lista de grabaciones” (se puede dejar una lista/array apuntando a NULL)
- Ejemplo:

```
1- Introducir datos de un archivo de grabacion
2- Introducir datos de una estacion de grabacion
3- Asignar grabaciones a estaciones de grabacion
4- Mostrar datos de estaciones
5- Salir
2
Introduzca nombre de estacion:
pajareras
Introduzca coordenadas en formato "latitud longitud":
37.0427289 -6.4344467
```

- 3- Asignar grabaciones a estaciones de grabación.
  - Se mostrará la lista de grabaciones que no tengan asignada una estación. Por cada una de ellas, se pedirá el nombre de una estación, y se añadirá una referencia/puntero de la grabación a la lista de grabaciones de esa estación. Por último, se asignará el nombre de la estación a la grabación seleccionada.
  - En caso de introducir datos erróneos de estación, se mostrará un error y continuará con la siguiente grabación
  - Ejemplo:

```
1- Introducir datos de un archivo de grabacion
2- Introducir datos de una estacion de grabacion
3- Asignar grabaciones a estaciones de grabacion
4- Mostrar datos de estaciones
5- Salir
3
Nombre archivo:rec1.wav
Fecha de grabaci¾n: 12/12/2023
Tipo: WAV
Estaci¾n no asignada
Introduzca nombre de estacion:
pajareras
Nombre archivo:rec2.mp3
Fecha de grabaci¾n: 13/12/2023
Tipo: MP3
Estaci¾n no asignada
Introduzca nombre de estacion:
asd
Error: Estacion no encontrada
```

- 4- Mostrar datos de estaciones
  - Por cada estación, se mostrará sus datos y la lista de grabaciones asignadas.
  - Ejemplo:

```
1- Introducir datos de un archivo de grabacion
2- Introducir datos de una estacion de grabacion
3- Asignar grabaciones a estaciones de grabacion
4- Mostrar datos de estaciones
5- Salir
4
========================================
Nombre estacion: pajareras
Latitud: 37.042728
Longitud: -6.434447
--- Grabacion 0 ---
Nombre archivo:rec1.wav
Fecha de grabaci%n: 12/12/2023
Tipo: WAV
```

- 5- Salir
    - Al acabar el programa, se debe liberar toda la memoria dinámica asignada