UNIDAD 3
ACTIVIDAD PRÁCTICA
MEMORIA DINÁMICA

U-tad
CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL

Centro adscrito a
Universidad
Camilo José Cela

ÍNDICE

ORGANIZACIÓN DE LA ACTIVIDAD PRÁCTICA

Nombre de la práctica	
Tipo de tarea	Individual
Entregables	Archivo zip con los diferentes ejercicios del alumno (Revisar apartado: "Como realizar mi entrega")

DESCRIPCIÓN DE LA ACTIVIDAD PRÁCTICA

- Ejercicio (10 puntos)

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, int tam, 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:
  - o 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.
  - o 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.
  - o Visualizar los intentos:
    - ♣ Presentará por pantalla toda la información recogida de cada intento, indicando si se acierta la fila o la columna
  - o 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
*******************************
```

RECOMENDACIONES E INDICACIONES PARA LA ENTREGA

- Claridad en el código:

Asegúrate de que tu código sea claro y esté bien estructurado. Sigue las buenas prácticas de programación, como el uso de indentaciones y comentarios, para facilitar su lectura y comprensión.

- Uso de identificadores descriptivos:

En la implementación en C, utiliza nombres de variables que describan claramente su propósito (por ejemplo, exponente en lugar de solo exp).

- Revisa los datos de entrada:

Antes de implementar los algoritmos en C, verifica que tienes claro cuáles son los posibles datos que un usuario puede introducir en tu programa y asegúrate de que no da errores al introducir datos no admitidas como letras cuando se soliciten números.

RÚBRICA DE CORRECCIÓN

La rúbrica para corregir el ejercicio seguirá los criterios listados a continuación, que tienen distintos pesos respecto al total de la nota.

Criterios	Excelente	Satisfactorio	No satisfactorio	Insuficiente
Ejercicio 1: Busca Minas	De 10 a 7.5 puntos: Programa	De 7.49 a 5 puntos: Programa funcional con	De 4.99 a 2.5 punto: Programa con errores significativos	De 2.49 a 0 puntos: Programa

	funcional y correcto, con buen uso de condicionales y bucles y funciones, estructuras, argc/argv, gestión de memoria dinámica lógica clara y sin errores importantes. Y revisión de parámetros de entrada.	algunos errores menores en la lógica o en la implementación, pero produce resultados correctos.	que no produce el resultado correcto, aunque hay un intento de implementación.	incorrecto o muy incompleto, sin funcionalidad significativa.

COMO REALIZAR LA ENTREGA

Para realizar le entrega de esta práctica se pide al alumno entregas un archivo zip con el nombre de NombreAlumno_PrimerApellido_PracticaTema3.zip.

El zip debe contener los siguientes archivos:

- Un archivo llamado ejercicio1.c. Este archivo debe contener el código compilable del ejercicio 1.

Además, se solicita al alumno que todos los archivos vayan iniciados con un comentario multilínea con indicando su nombre y el número del ejercicio.