Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Examen de Teoría

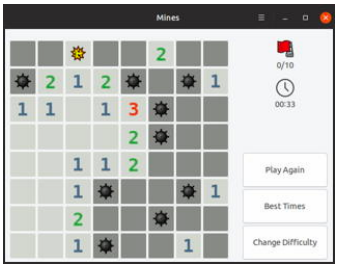
1 de septiembre de 2022

Fundamentos de Informática
Grado en Tecnologías Industriales
Duración: 2h 45m

Entrega:

- Escribe tu nombre completo en **todas** las hojas que entregues como solución del examen.
- Responde a cada ejercicio comenzando **en una hoja diferente**, los ejercicios se entregan por separado.
- La resolución del examen no se puede escribir a lápiz ni con bolígrafo rojo. Lo que esté escrito a lápiz o en rojo será ignorado.

Ejercicio 1 **[3.5 puntos]**



El juego del **Buscaminas** consiste en despejar todas las casillas de un tablero que no oculten una mina. El tablero incialmente se muestra sin ninguna información, y el usuario puede ir seleccionando casillas. Si selecciona una con una mina oculta, pierde. Si selecciona una sin mina, se le muestra en la casilla un numero que indica cuántas casillas de las colindantes contienen una mina. Así, si una casilla tiene el número 3, significa que de las ocho casillas que hay alrededor (si no está en una esquina o borde, en cuyo caso tendría 3 o 5 casillas colindantes en vez de 8) hay 3 con mina y 5 sin mina. Si se descubre una casilla sin número (con un cero) indica que ninguna de las casillas vecinas tiene mina y dichas casillas vecinas se descubren automáticamente.

Se pide, en lenguaje Pascal:

- Definir una estructura de datos **`tpTablero`** que permita almacenar el tablero de la partida:
  - Tiene un tamaño máximo de 100x100, pero se puede jugar a tamaños más pequeños.
  - Para cada casilla es necesario almacenar información que indique: si contiene una mina, si el usuario ha descubierto ya dicha casilla o no, y cuántas casillas colindantes contienen una mina.

A partir de la estructura de datos anterior, implementa los siguientes procedimientos y funciones:

- Un procedimiento que genere el tablero inicial de forma aleatoria, con un número de minas especificado por el usuario:

  ```
  procedure genera_tablero(??? tablero: tpTablero; ??? nminas: integer);
  ```

  El procedimiento debe generar un tablero con toda la información necesaria, incluyendo, para cada casilla, si contiene mina o no, si está descubierta o no, y el numero de casillas colindantes que contienen una mina.

  Puedes usar la función de Pascal estándar (ya existente, no tienes que implementarla):

  ```
  function random(maxval: longint): longint};
  ```

  para generar las posiciones de las minas de forma aleatoria. Esta función toma como parámetro un entero (`longint`) y devuelve un entero aleatorio (`longint`) que está en el rango [0..`maxval`).

- Un procedimiento que modifique el tablero cuando el jugador descubre una casilla:

  ```
  procedure descubre_casilla(??? tablero: tpTablero; ??? fil,col: integer);
  ```

  El procedimiento debe:
  - Asegurarse de que la casilla es válida (está dentro del tablero).
  - Marcar como descubierta la casilla correspondiente.
  - Si no contiene ninguna casilla vecina con mina, marcar como descubiertas también las casillas colindantes.

Puedes definir estructuras de datos o procedimientos y funciones adicionales si lo consideras necesario.

Ejercicio 2 **[3.5 puntos]**

La empresa de transporte público de una ciudad quiere evaluar el uso de sus distintas líneas de autobús por parte de niños y adultos. Para ello, pide a los conductores que apunten, a lo largo de su jornada, para cada persona que sube al autobús, si es un niño o un adulto. La información correspondiente a un día acaba almacenada en un *fichero de texto*, en el que cada línea del fichero contiene información para una ruta de autobús: un entero correspondiente al número de ruta de autobús, un espacio en blanco, y a partir de ahí una secuencia de **`'n'`** y **`'a'`**, en la que hay una `'n'` por cada niño que sube, y una `'a'` por cada adulto que sube. La Figura 1 muestra un ejemplo de un posible fichero de entrada.

Se pide desarrollar un programa Pascal que, dado un *fichero de texto* de entrada **`'buses.txt'`** con la estructura descrita y que contiene la información correspondiente a un día, genere un *fichero de texto* de salida para ese día **`'buses_procesado.txt'`** con la siguiente estructura: cada línea del fichero de salida corresponderá a una ruta de autobús, y tendrá escrito, por este orden, el número de ruta, y luego el porcentaje de niños y porcentaje de adultos que han viajado en esa ruta durante ese día. La Figura 2 muestra un ejemplo del fichero de salida, con el formato que tiene que tener.

Importante: En el fichero de salida, las líneas de autobús deben estar en orden, mientras que en el fichero de entrada no tienen por qué estarlo. Además, el fichero de entrada puede tener varias líneas correspondientes a la misma ruta (distintos conductores que hacen la ruta a lo largo del día).

Observaciones:

- Una ruta puede no haber llevado ningún pasajero. En ese caso, la línea del fichero contendrá únicamente el número de ruta, seguido de un espacio en blanco (por ejemplo, la ruta número 34 en la Figura 1).
- La ciudad cuenta con 26 líneas de autobús, numeradas del 20 al 45.
- El número de veces que cada ruta se hace al día es variable, pero no es relevante, simplemente interesa el total diario (ver la ruta número 23 en el ejemplo).

Figura 1: Ejemplo de fichero de entrada **`buses.txt`**

```
45 annnaaaana
23 an
20 aann
23 nnanna
34
36 aaaa
```

Figura 2: Ejemplo de fichero de salida **`buses_procesado.txt`**

```
20 - n = 50.0%, a = 50.0%
21 -
22 -
23 - n = 62.5%, a = 37.5%
24 -
25 -
...
36 - n = 0.0%, a = 100.0%
...
44 -
45 - n = 40.0%, a = 60.0%
```

Ejercicio 3 **[3.0 puntos]**

La memoria RAM de los ordenadores no es perfecta, y con el paso del tiempo puede estropearse y corromper la información que guarda (lo que recuperas no es lo mismo que lo que has almacenado).

Esos problemas no se pueden solucionar de forma ideal, pero al menos los computadores incorporan un sistema de comprobación que sirve para averiguar si en el dato almacenado se ha corrompido UN bit. Para ello se calcula la **paridad binaria** del dato almacenado, es decir, si al expresarlo en binario contiene un número PAR de unos (paridad 0) o IMPAR (paridad 1). Así, con un único bit extra, podemos saber si el número ha cambiado respecto al original:

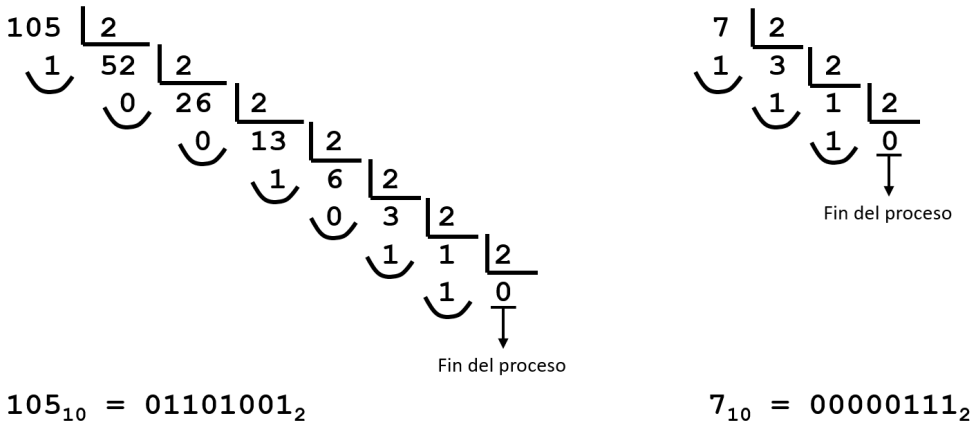
Número (n)	Número en binario (8 bits)	Cantidad de 1s	Paridad
0	00000000	0	0
1	00000001	1	1
5	00000101	2	0
7	00000111	3	1
105	01101001	4	0
211	11010011	5	1

Implementa en Pascal una función:

```
function paridad(n: integer): integer;
```

que calcule la paridad binaria del número que se le pasa como parámetro, es decir, devuelve `0` si al expresarlo en binario el dato contiene un número par de unos, o bien `1` en caso contrario.

Para poder implementar dicha función, hay que saber cómo convertir un número decimal (base 10) a binario (base 2). Para hacer la conversión, se puede: dividir el número decimal entre dos y anotar el resto de dicha división, y hacer esto de forma iterativa hasta que la división entera entre dos da cero. Los valores del resto que hemos ido anotando dan el número en binario, ordenados de forma que el primer resto obtenido es la última cifra del número en binario (y el último resto obtenido es la primera cifra del número en binario). La Figura 1 ilustra el proceso de conversión aquí descrito para dos números de ejemplo.



$105_{10} = 01101001_2$ $7_{10} = 00000111_2$

Figura 1: Conversión de decimal a binario para dos números de ejemplo. En ambos casos, el número en binario se representa con 8 bits.