Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Examen de Teoría

20 de enero de 2021

Fundamentos de Informática
Grado en Tecnologías Industriales
Duración: 3 horas

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 **[3.0 puntos]**

En el proceso de diseño de vacunas mediante mARN (ARN mensajero) se necesita definir una cadena de ARN que una vez en el organismo provoque la generación de una proteína determinada que impida que el virus ataque a las células. Para probar qué cadena de ARN genera la proteína deseada, se utiliza un programa que simula el proceso biológico.

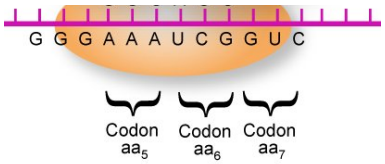
El código presente en el ARN es una **cadena** de moléculas (*nucleótidos*), que sólo pueden ser de cuatro tipos. Cada tipo se representa por un carácter (`'A'`, `'C'`, `'G'`, `'U'`), y la secuencia de ARN se puede representar mediante una **cadena** formada sólo mediante esos cuatro caracteres.

Una proteína es una **secuencia de moléculas llamadas aminoácidos** (que se identifican con un código que es un número entero). Cuando se genera una proteína, cada grupo de 3 nucleótidos consecutivos del ARN (llamado *codon*, una sub–cadena de 3 carateres) se traduce en un trozo de la proteína (aminoácido), que se van enlazando en secuencia. El código de traducción se puede especificar mediante una tabla para los 32 aminoácidos humanos:

```
tpTablaCodigos = array[1..32] of string;
```

en la que en cada posición i aparece el codon correspondiente al aminoácido cuyo código es i.

1	'ACG'
2	'CCU'
3	'GUC'
...	...



Se pide implementar en lenguaje Pascal:

- Un tipo de datos para representar una proteína, teniendo en cuenta que es una **secuencia** de códigos de aminoácidos (enteros), de tamaño variable pero con un máximo de 10000.

  ```
  tpProteina = ....
  ```

- Una función que a partir de un codon y la tabla de traducción nos devuelva el código del aminoácido correspondiente.

  ```
  function codigo(???? codon: string; ???? tabla: tpTablaCodigos): integer;
  ```

- Un procedimiento que a partir de una cadena de ARN y la tabla de traduccón genere la proteina correspondiente:

  ```
  procedure sintetiza(???? arn: string; ???? tabla: tpTablaCodigos;
                      ???? prot: tpProteina);
  ```

NOTA: Supondremos que la longitud de la cadena de ARN siempre es múltiplo de 3.

NOTA: Puedes utilizar el tipo de datos `string` si lo consideras necesario, así como sus operadores y funciones correspondientes: =, <>, `length(...)`, `setlength(...)`.

Ejercicio 2 **[3.5 puntos]**

Durante la primera etapa del proceso de vacunación en los centros de salud pertenecientes a un sector sanitario de Zaragoza, los centros van registrando el número de vacunas administradas cada día de vacunación. Lo hacen en un **fichero secuencial de registros**, de nombre `'vacunacionPrimeraEtapa.dat'`. Cada registro de dicho fichero contiene: el código numérico del centro (un número de 1 a 10, porque hay 10 centros en el sector), la fecha (en formato AAAAMMDD), y el número de vacunas administradas por el centro en esa fecha. Las estructuras de datos correspondientes son las siguientes:

```
tpVacUnDia = record
    centro: integer;
    fecha:  longint;
    numVac: integer;
end;
tpFichVacunas = file of tpVacUnDia;
```

Se pide desarrollar un programa Pascal que:

a) A partir de la información contenida en el fichero `'vacunacionPrimeraEtapa.dat'`, muestre por pantalla el número total de vacunas administradas en la primera etapa en ese sector sanitario ($nTot$), así como la media de vacunaciones por centro (esto es, el total de vacunas administradas entre el número de centros).

b) De cara a la segunda etapa de vacunación se dispone de $nDisp$ dosis. El reparto de estas dosis entre centros se va a hacer de forma proporcional a las vacunas ya administradas por cada centro. Si el centro i ha administrado un total de vac_i dosis en la primera etapa, en la segunda etapa le corresponderán $vac_i/nTot * nDisp$ dosis. Así, el programa pedirá al usuario el número de dosis disponibles para la segunda etapa ($nDisp$), y mostrará por pantalla cuántas dosis le corresponden a cada centro. Si se obtiene un número no entero de dosis a repartir para un centro, se mostrará el entero inmediatamente inferior.

Observaciones:

1. Los registros no están ordenados en el fichero, ni por fecha ni por centro ni de ningún otro modo, ya que cada centro tiene sus propios protocolos sobre cuándo incluir los datos de vacunación en el fichero.
2. Se pueden definir nuevas estructuras de datos si se considera apropiado.
3. El número total de vacunas administradas, así como el número de dosis disponibles para la segunda etapa, pueden almacenarse ("caben") en un dato de tipo entero.
4. Se valorará reducir el número de veces que se recorre el fichero al mínimo necesario.
5. Se muestra a continuación un ejemplo de fichero y un ejemplo de ejecución correspondiente a dicho fichero; en el ejemplo de ejecución, figura en negrita la información introducida por el usuario.

Ejemplo de fichero `'vacunacionPrimeraEtapa.dat'`:

centro = 1 fecha = 20210119 numVac = 200	centro = 2 fecha = 20210118 numVac = 250	centro = 1 fecha = 20210115 numVac = 300	centro = 5 fecha = 20210119 numVac = 500

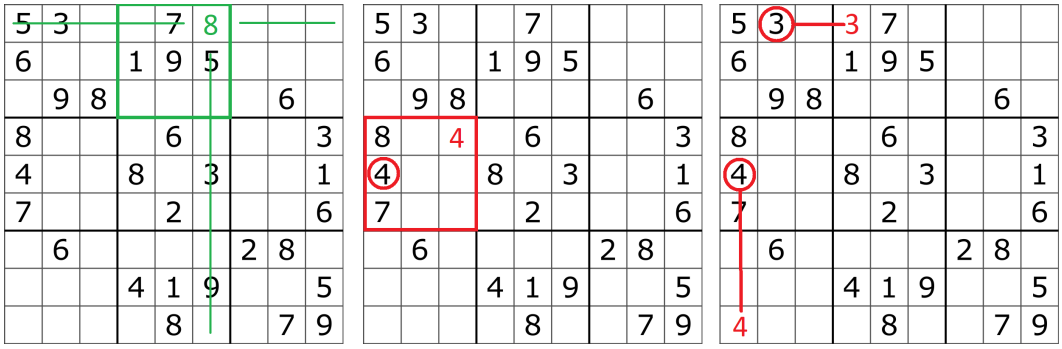
Ejemplo de ejecución:

```
Administradas 1250 vacunas en el sector.
Media por centro = 125.00 vacunas/centro.
Introduzca número vacunas disponibles para segunda etapa:
14000
Vacunas a distribuir por centro:
1 - 5600
2 - 2800
3 - 0
4 - 0
5 - 5600
6 - 0
7 - 0
8 - 0
9 - 0
10 - 0
```

Ejercicio 3 **[3.5 puntos]**

Sudoku es un juego matemático basado en lógica, inventado a finales de los 70. El objetivo es rellenar una cuadrícula de 9×9 celdas (81 casillas) dividida en subcuadrículas de 3×3 (también llamadas "regiones") con las cifras del 1 al 9, partiendo de algunos números ya dispuestos en algunas de las celdas. Para que el **sudoku sea válido**, cada número tiene que aparecer **una** sola vez en cada fila, columna y región.

La figura inmediatamente inferior muestra tres ejemplos de sudokus. El sudoku de la izquierda sería un **sudoku válido**, ya que no hay números repetidos en ninguna fila, columna o región. Un ejemplo es el número 8 en la primera fila, sexta columna, en las que no hay ningún otro número 8, ni tampoco en la región de arriba-centro. Por contra, los ejemplos del centro y la derecha muestran dos ejemplos de **sudokus no-válidos**: En el central, en la región del medio-izquierda hay dos número 4. En el ejemplo de la derecha, en la primera fila hay dos números 3, y en la primera columna dos número 4.



Se pide implementar en Pascal lo siguiente:

- Definir un tipo de datos que permita representar un Sudoku:

```
tpSudoku = ...
```

- Implementar una función que compruebe si un Sudoku es válido o no:

```
function SudokuValido(???? sk: tpSudoku): ????
```

Se pueden definir tantas estructuras de datos y funciones/procedimientos adicionales como se estime necesario. Se valorará el uso adecuado de subprogramas.