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

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.

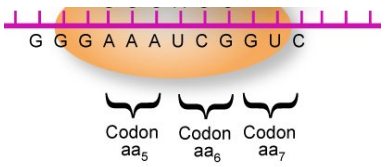
El código presente en el ARN es una **cadena** de moléculas (*nucleótidos*), que sólo pueden ser de cuatro tipos. Cada tipo se representa por un carácter (`'A'`, `'C'`, `'G'`, `'U'`), y la secuencia de ARN se puede representar mediante una **cadena** formada sólo mediante esos cuatro caracteres.

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(...)`.

Propuesta de solución:

```
program proteina;
const
  NUMAA  = 32;
  MAXTAM = 10000;
type
  tpTablaCodigos = array[1..NUMAA] of string;
  tpProteina = record
    aa  : array[1..MAXTAM] of integer;
    naa : integer;
  end;

function codigo(const codon: string; const tabla: tpTablaCodigos): integer;
var
  i,icod : integer;
begin
  icod := 0;
  for i:=1 to NUMAA do
    if tabla[i] = codon then icod:=i;

  codigo := icod;
end;

procedure sintetiza(const arn: string; const tabla: tpTablaCodigos;
                    var   prot: tpProteina);
var
  codon: string;
  icod : integer;
  i,j : integer;
begin
  prot.naa := 0;
  i:=0;
  while i<length(arn) do
  begin
    setlength(codon,3);
    for j:=0 to 2 do
      codon[j] := arn[i+j];
    icod := codigo(codon,tabla);
    if icod>0 then
    begin
      prot.naa := prot.naa+1;
      prot.aa[prot.naa] := icod;
    end;
    i := i+3;
  end;
end;

begin
end.
```

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:

```
1  tpVacUnDia = record
2      centro: integer;
3      fecha:  longint;
4      numVac: integer;
5  end;
6  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

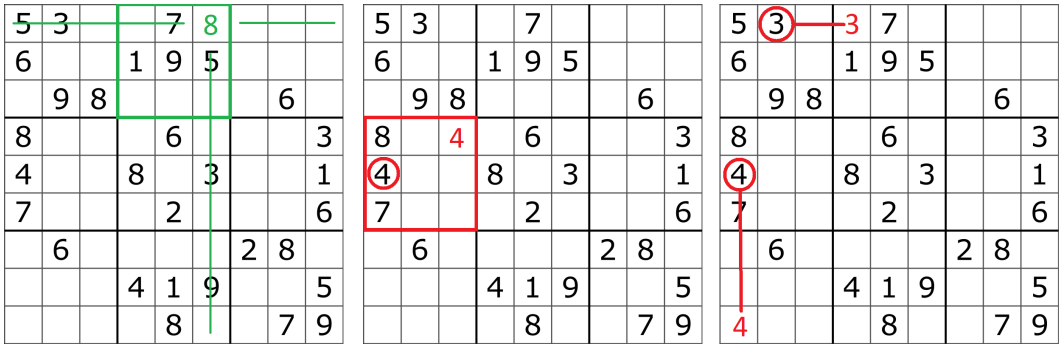
Propuesta de solución:

```
program analizaVacunas;
const
  NCENTROS = 10;
type
  tpVacUnDia = record
    centro: integer;
    fecha: longint;
    numVacunas: integer;
  end;
  tpFichVacunas = file of tpVacUnDia;
  tpCentros = array[1..NCENTROS] of integer;

var
  fich: tpFichVacunas;
  vac: tpVacUnDia;
  vec: tpCentros; { num vacunas admin por cada centro }
  nTot: integer;
  media: real;
  nDisp: integer;
  i: integer;
begin
  assign(fich, 'vacunacionPrimeraEtapa.dat');
  reset(fich);
  { Inicializar vector }
  for i := 1 to NCENTROS do
    vec[i] := 0;
  nTot := 0;

  while not eof(fich) do
  begin
    read(fich, vac);
    nTot := nTot + vac.numVacunas;
    vec[vac.centro] := vec[vac.centro] + vac.numVacunas;
  end;
  close(fich);
  media := nTot / NCENTROS;
  writeln('Se han administrado ', nTot, ' vacunas en este sector sanitario.');
  writeln('La media por centro es de ', media:1:2, ' vacunas/centro.');

  writeln('Introduce num. vacunas disponibles para la segunda etapa: ');
  readln(nDisp);
  writeln('Vacunas a distribuir por centro: ');
  for i := 1 to NCENTROS do
    writeln(i, ' - ', trunc(vec[i]/nTot*nDisp));

end.
```

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.

Propuesta de solución 1 (más ineficiente):

```
program ComprobarSudoku;

type
    tpSudoku = array[1..9, 1..9] of char;

{ Pre:  -
  Post: Comprueba que el caracter 'num' contiene un caracter entre '1' y '9'.
        Si no, se asume vacío. }
function esNumeroValido( num : char ): boolean;
begin
  esNumeroValido := (num >= '1') and (num <= '9');
end;

{ Pre:  Asumimos que sudoku[i,j] es numero válido
  Post: Devuelve si hay error en la columna j }
function errorEnColumna( const sudoku : tpSudoku; i: integer; j : integer): boolean;
var
  ii : integer;
begin
  errorEnColumna := false;

  for ii:=1 to 9 do
    if (ii <> i) and esNumeroValido(sudoku[ii,j]) then
      if sudoku[ii,j] = sudoku[i,j] then
        errorEnColumna := true;
end;

{ Pre:  Asumimos que sudoku[i,j] es numero válido}
{ Post: Devuelve si hay error en la fila i }
function errorEnFila( const sudoku : tpSudoku; i : integer; j : integer): boolean;
var
  jj : integer;
begin
  errorEnFila := false;

  for jj:=1 to 9 do
    if (jj <> j) and esNumeroValido(sudoku[i,jj]) then
      if sudoku[i,jj] = sudoku[i,j] then
        errorEnFila := true;
end;

{ Pre:  Asumimos que sudoku[i,j] es numero válido}
{ Post: Devuelve si hay error en la region de i, j }
function errorEnRegion( const sudoku : tpSudoku; i : integer; j : integer): boolean;
var
  ri0, rj0 : integer;
  ii, jj : integer;
begin
  errorEnRegion := false;

  {Computamos la primera posición de la región en que se encuentra i,j}
  ri0 := ((i-1) div 3)*3 + 1;
  rj0 := ((j-1) div 3)*3 + 1;

  { Iteramos sobre todas las posiciones dentro de la región}
  for ii:=ri0 to ri0+2 do
    for jj:=rj0 to rj0+2 do
      { Comprobamos que ii,jj sea distinto de la posicion que buscamos,
```

```
59        * y que contenga un numero valido (ie no esté vacio). }
60        if (ii <> i) and (jj <> j) and esNumeroValido(sudoku[ii,jj]) then
61          if sudoku[ii,jj] = sudoku[i,j] then
62            errorEnRegion := true;
63  end;
64
65  { Pre:  Asumimos que sudoku[i,j] es numero válido}
66  { Post: Devuelve si hay error en la fila, columna, o región de i, j }
67  function hayError( const sudoku : tpSudoku; i : integer; j : integer): boolean;
68  begin
69    hayError := errorEnFila(sudoku, i, j) or
70          errorEnColumna(sudoku, i, j) or
71          errorEnRegion(sudoku, i, j);
72  end;
73
74  { Pre:  - }
75  { Post: Devuelve si hay un error en el sudoku }
76  function SudokuValido(const sudoku : tpSudoku): boolean;
77  var
78    i,j: integer;
79  begin
80    SudokuValido := true;
81    for i:=1 to 9 do
82      for j:=1 to 9 do
83        if esNumeroValido(sudoku[i,j]) then
84          if hayError( sudoku, i, j ) then
85            SudokuValido := false;
86  end;
87
88  begin
89  end.
```

Propuesta de solución 2 (eficiente):

```
1
2  program ComprobarSudoku;
3  const
4      MAX = 3;
5  type
6      tpSudoku = array[1..9, 1..9] of char;
7    tpArea = record
8      i0,j0 : integer;
9      i1,j1 : integer;
10   end;
11   tpContador = array['1'..'9'] of integer;
12
13 { Pre:
14 * Post: Comprueba que el caracter 'num' contiene un caracter entre '1'
15 *      y '9'. Si no, se asume vacío. }
16 function esNumeroValido( num : char ): boolean;
17 begin
18   if (num < '1') or (num > '9' ) then
19     esNumeroValido := false
20   else
21     esNumeroValido := true;
22 end;
23
24 function areaValida( const a: tpArea): boolean;
25 begin
26   areaValida := (a.i0>0) and (a.j0>0) and (a.i1 < 10) and (a.j1 < 10)
27             and (a.i0 <= a.i1) and (a.j0 <= a.j1)
28             and (((a.i1-a.i0+1)*(a.j1-a.j0+1)) = 9);
29 end;
30
31 { Pre: Asumimos que sudoku[i,j] es numero válido
32 * Post: Devuelve si hay error en la columna j }
33 function errorEnArea( const sudoku : tpSudoku; const a : tpArea): boolean;
34 var
35   ii, jj : integer;
36   ci : char;
37   c : tpContador;
38 begin
39   errorEnArea := false;
40   for ci:='0' to '9' do
41     c[ci] := 0;
42
43   if not areaValida(a) then
44     writeln('ERROR');
45
46   for ii:=a.i0 to a.i1 do
47     for jj := a.j0 to a.j1 do
48     begin
49
50       if esNumeroValido(sudoku[ii,jj]) then
51       begin
52         c[sudoku[ii,jj]] := succ(c[sudoku[ii,jj]]);
53         if c[sudoku[ii,jj]] > 1 then
54           errorEnArea := true;
55       end;
56     end;
57 end;
58
```

```
59  function errorEnFilas( const sudoku : tpSudoku ): boolean;
60  var
61    jj : integer;
62    a : tpArea;
63    error : boolean;
64
65  begin
66    a.i0:=1; a.i1 := 9;
67    errorEnFilas := false;
68
69    for jj:=1 to 9 do
70    begin
71      a.j0:=jj; a.j1:=jj;
72      error := errorEnArea(sudoku, a);
73      if error then
74        errorEnFilas := true;
75    end
76  end;
77
78  function errorEnColumnas( const sudoku : tpSudoku ): boolean;
79  var
80    ii : integer;
81    a : tpArea;
82    error : boolean;
83  begin
84    a.j0:=1; a.j1 := 9;
85    errorEnColumnas := false;
86
87    for ii:=1 to 9 do
88    begin
89      a.i0:=ii; a.i1:=ii;
90      error := errorEnArea(sudoku, a);
91      if error then
92        errorEnColumnas := true;
93
94    end
95  end;
96
97  function errorEnRegiones( const sudoku : tpSudoku ): boolean;
98  var
99    ri, rj : integer;
100   a : tpArea;
101   error : boolean;
102
103 begin
104   errorEnRegiones := false;
105
106   {Computamos la primera posición de la región en que se encuentra i,j}
107
108   for ri := 1 to 3 do
109     for rj := 1 to 3 do
110     begin
111       a.i0 := (ri-1)*3 + 1;
112       a.j0 := (rj-1)*3 + 1;
113       a.i1 := a.i0+2;
114       a.j1 := a.j0+2;
115
116       error := errorEnArea(sudoku, a);
117       if error then
118         errorEnRegiones := true;
```

```
119
120     end;
121 end;
122
123 { Pre: ...
124 * Post: Devuelve si hay un error en el sudoku }
125 function SudokuValido(const sudoku : tpSudoku): boolean;
126 begin
127   if errorEnFilas(sudoku) or
128      errorEnColumnas(sudoku) or
129      errorEnRegiones(sudoku) then
130     esSudokuValido := false
131   else
132     esSudokuValido := true;
133 end;
134
135 end.
```