Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza
1474

Examen de Teoría

14 de septiembre de 2021

Fundamentos de Informática
Grado en Tecnologías Industriales
Duración: 3 horas

Ejercicio 1 **[3.0 puntos]**

Un usuario, muy preocupado por su factura de la electricidad, decide implementar un programa que le sirva para comparar los precios PVPC con los de las diversas comercializadoras (un usuario puede tener la tarifa PVPC, o acogerse a las tarifas de alguna de las comercializadoras). Vamos a ayudarle...

En la tarifa PVPC (Precio Voluntario para el Pequeño Consumidor), el precio del kWh (kilovatio hora) es distinto cada hora. Por otro lado, las comercializadoras ofrecen tarifas organizadas por bandas (una banda tiene un precio fijo determinado entre dos horas del día), pero unas dividen el día en una sola banda (precio único), otras en 3, otras en 5...

Define los siguientes tipos de datos:

- Un tipo `tpConsumo`, que almacena el consumo de electricidad del usuario a lo largo de un día, para cada hora del día, en kWh.
- Un tipo `tpTarifaPVPC`, que contiene, para cada hora del día, el precio del kWh en euros.
- Un tipo `tpBanda`, que contiene la siguiente información sobre una banda:
  - Horas de inicio y final de la banda (números enteros).
  - Precio del kWh en esa banda.
- Un tipo `tpTarifaComercial`, que contiene los precios para el conjunto de bandas que puede ofrecer cada una de las distintas comercializadoras. Puede haber un máximo de 5 bandas, pero pueden ser menos (como mínimo habrá 1 banda).

A partir de esos tipos de datos, implementa las siguientes funciones:

- Una función que, a partir del consumo de un día del usuario y la tarifa PVPC, devuelva el coste de electricidad de ese día:

```
function precioPVPC(??? consumo: tpConsumo,
                    ??? tarifa: tpTarifaPVPC): ???
```

- Una función que, a partir del consumo de un día del usuario y la tarifa de una comercializadora, devuelva el coste de electricidad de ese día:

```
function precioComercial(??? consumo: tpConsumo,
                         ??? tarifa:  tpTarifaComercial): ???
```

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

Propuesta de solución:

```
program main;
const
  MAXBANDAS = 5;
type
  tpHora = 0..23;
  tpConsumo = array[tpHora] of real;
  tpTarifaPVPC = array[tpHora] of real;
  tpBanda = record
    hini,hfin : tpHora;
    precio    : real;
  end;
  tpTarifaComercial = record
    bandas  : array[1..MAXBANDAS] of tpBanda;
    nbandas : 1..MAXBANDAS;
  end;

function precioPVPC(const consumo: tpConsumo;
          const tarifa: tpTarifaPVPC): real;
var
  h      : integer; precio : real;
begin
  precio := 0;
  for h:=0 to 23 do
  begin
    precio := precio + consumo[h]*tarifa[h];
  end;
  precioPVPC := precio;
end;

function precioComercial(const consumo: tpConsumo;
          const tarifa: tpTarifaComercial): real;
var
  h,b    : integer; precio : real;
begin
  precio := 0;
  for h:=0 to 23 do
  begin
    for b:=1 to tarifa.nbandas do
    begin
      if  (tarifa.bandas[b].hini <= h) and
        (h < tarifa.bandas[b].hfin) then
        precio := precio + consumo[h]*tarifa.bandas[b].precio;
    end;
  end;
  precioComercial := precio;
end;
```

Ejercicio 2 **[3.5 puntos]**

Una conocida red social almacena los años de nacimiento (y de defunción, si se ha producido y registrado en la misma) de sus miembros en un fichero secuencial de registros, de nombre `usuarios.dat'. Cada registro de dicho fichero contiene el nombre y primer apellido del usuario, su alias, y ambos años. Se tienen por tanto las siguientes estructuras de datos:

```
1  tpUsuario = record
2    nom: String; {nombre}
3    ape: String; {primer apellido}
4    alias: String;
5    nac: integer; {anio de nacimiento}
6    def: integer; {anio de defuncion; es cero si no ha fallecido}
7  end;
8  tpFichUsuarios = file of tpUsuario;
```

Todos los registros almacenados en el fichero cuentan con un año válido de nacimiento, y en ningún caso anterior a 1900. En cuanto al de defunción, si la persona ha fallecido el campo correspondiente (`def`) tendrá el año de defunción, y si la persona no ha fallecido en dicho campo se almacenará un valor de cero. Se muestra un ejemplo de fichero al final de este enunciado (el fichero de ejemplo contiene tres registros, pero se ha de resolver el problema para el caso general, en que el fichero puede contener muchos más).

Se pide desarrollar un programa Pascal que, a partir de la información contenida en un fichero `usuarios.dat', muestre por pantalla el año en el que había más personas vivas, sabiendo que estamos en 2021 (es decir, el año en que había más personas vivas entre 1900 y 2021, ambos incluidos).

Observaciones:

1. Los registros no están ordenados en el fichero, ni por año de nacimiento ni de ningún otro modo.
2. Se pueden definir nuevas estructuras de datos.
3. Se valorará reducir el número de veces que se recorre el fichero al mínimo necesario.
4. Si hay dos o más años con el mismo número máximo de personas vivas, basta con mostrar uno cualquiera de ellos.
5. A efectos del cálculo, si una persona nació en el año *m*, cuenta como viva en dicho año *m*. Si una persona falleció en el año *n*, también cuenta como viva en dicho año *n*.

Ejemplo de fichero `usuarios.dat':

nom = Ana	nom = Juan	nom = Maria
ape = Ruiz	ape = Sanz	ape = Diaz
alias = ana96	alias = thor3	alias = mary666
nac = 1996	nac = 1970	nac = 2000
def = 0	def = 2010	def = 0

Propuesta de solución:

```
program analizaFichUsuarios;

type
  tpUsuario = record
    nom: String[25];
    ape: String[25];
    alias: String[25];
    nac: integer; {anio de nacimiento}
    def: integer; {anio de defuncion; es cero si no ha fallecido}
  end;
  tpFichUsuarios = file of tpUsuario;

  tpVecAnios = array[1900..2021] of integer;

var
  user: tpUsuario;
  fich: tpFichUsuarios;
  v: tpVecAnios;
  i: integer;
  maxValor, maxAnio: integer;
begin

  { Inicializar vector }
  for i:=1900 to 2021 do
    v[i] := 0;

  { Volcar info relevante del fichero al vector }
  assign(fich, 'usuarios.dat');
  reset(fich);
  while not eof(fich) do
  begin
    read(fich, user);
    if (user.def <> 0) then {ha fallecido}
    begin
      for i := user.nac to user.def do
        v[i] := v[i] + 1;
    end
    else
    begin
      for i := user.nac to 2021 do
        v[i] := v[i] + 1;
    end;
  end;
  close(fich);

  { Buscar el máximo del vector, el primer máximo vale }
  maxValor := v[1900];
  maxAnio := 1900;
  for i := 1901 to 2021 do
  begin
    if (v[i] > maxValor) then
    begin
      maxValor := v[i];
      maxAnio := i;
    end;
  end;
  writeln('Anio con mas gente viva: ', maxAnio);

```

```
59 end.
```

Ejercicio 3 **[3.5 puntos]**

El **bingo** es un juego de azar similar a la lotería, en el que en cada jugada se extrae de forma aleatoria un número natural dentro de un rango determinado $1..N$ (por ejemplo, mediante el uso de un *bombo* con N bolas numeradas no repetidas). Cada jugador tiene uno o varios *cartones* con un conjunto de M números naturales en el rango $1..N$, divididos en L líneas de igual número de elementos. Un número sólo puede aparecer **una vez como máximo** en cada cartón.

En cada jugada, tras la extracción de un número del bombo, el jugador comprueba si dicho número pertenece a su cartón; si es así lo *marca*. Una vez se extrae un número, éste se retira del bombo y no puede volver a aparecer. Gana el jugador que primero marca todos los números de su cartón. Asimismo, se premia al jugador que primero consigue marcar todos los números de una de las líneas de su cartón.

Se pide implementar un conjunto de módulos de un programa Pascal, necesarios para desarrollar un simulador del juego del bingo. Puedes utilizar las constantes `N` (número máximo en el bombo), `M` (número de elementos en el cartón), y `L` (número de líneas en el cartón), y puedes asumir que `M` siempre será divisible por `L`.

En concreto, deberás implementar en lenguaje Pascal:

a) Un tipo de datos `tpCarton` que implemente un cartón de bingo.

b) Un procedimiento que inicialice de forma aleatoria un cartón de bingo de tipo `tpCarton`:

```
procedure inicializa(??? carton : tpCarton )
```

Los elementos del cartón **NO** tienen que estar ordenados, pero cada elemento sólo puede aparecer una vez como máximo en el cartón. Puedes usar la función `random(N)`, que devuelve un número entero aleatorio en el rango $[0..N]$.

c) Un procedimiento que, dado un cartón de tipo `tpCarton` y un número natural en el rango $[1..N]$, compruebe si ese número está incluido en el cartón, y en el caso afirmativo lo marque:

```
procedure marcar(??? carton : tpCarton, ??? numero : integer)
```

Recuerda que en el bingo cada número sólo puede aparecer **una única vez**.

d) Una función que compruebe si un cartón de tipo `tpCarton` dado como entrada está completamente marcado (es decir, si el jugador *canta bingo* o no):

```
function cantaBingo(??? carton : tpCarton ): ???
```

Puedes definir estructuras de datos o procedimientos y funciones adicionales si lo consideras necesario. Se valorará positivamente la eficiencia de la solución planteada.

Propuesta de solución: