TEMA 5

Estructuras de datos no elementales. Vectores y Registros

Parte I. Vectores

1. Introducción.

Hasta ahora hemos trabajado con tipos de datos simples. Los tipos de datos simples tienen como característica común que cada variable representa a un solo dato individual.

Vamos a introducir en este tema los **tipos de datos estructurados**. Estos tipos tienen como característica común que un único identificador puede representar a múltiples datos individuales, pudiendo cada uno de éstos ser referenciado separadamente.

Un tipo de dato estructurado consiste en la agregación de varios datos que a su vez pueden tener una composición simple o estructurada. Se caracterizan por su organización y por las operaciones que se definen sobre ellos. En el nivel más bajo de un tipo estructurado hay tipos simples que se utilizarán en la misma forma que los tipos de datos simples.

Los tipos de datos estructurados que veremos son **vectores** y **registros**.

Es posible plantear problemas muy sencillos cuya resolución resulta complicada con los tipos de datos utilizados hasta ahora.

Ejemplos:

1. Algoritmo que lea una secuencia de enteros (acabando con el 0) y la escriba en orden inverso.

```
23 12 36 72 193 4 0 ---> ---> 4 193 72 36 12 23
```

Para resolverlo, debemos almacenar toda la secuencia de números, ¿cómo hacerlo con los tipos de datos que hemos manejado hasta ahora? Además debemos ir almacenándolos en variables consecutivas en cierto sentido ¿cómo conseguirlo?

2. Contar el número de apariciones de cada letra del alfabeto en un texto dado.

```
Erase una vez. ---> ---> 2A 0B 0C 0D 3E ...
```

Debemos tener para cada letra un contador correspondiente al número de letras aparecidas. ¿Cómo hacemos corresponder a cada letra un contador? ¿Cómo cambiamos de valor el contador correspondiente cada vez?

3. Representar en binario un número entero positivo dado en base decimal

```
434 ----> ----> 110110010
```

Estos problemas ponen de manifiesto la necesidad de estructurar datos.

2. Concepto de Vector.

Un **vector** (o array) es una estructura de datos formada por un conjunto de elementos del mismo tipo que ocupan posiciones consecutivas de memoria. Es un tipo de dato estructurado, es decir, un único identificador se refiere a un conjunto de valores y no a uno único como ocurre con los datos de tipo simple. Veremos más adelante la manera de referirnos a cada uno de sus componentes individuales.

De una manera más formal, podríamos decir que un vector v, es una aplicación de un conjunto finito I de valores escalares, ordenados y consecutivos, denominado conjunto de índices del vector, en otro conjunto D de datos variables de un mismo tipo, conjunto base del vector.

v: **I** → **D**

$I_1 \rightarrow d_1$
$I_2 \rightarrow d_2$
.....
$I_n \rightarrow d_n$

NOTA: Aunque el conjunto de índices I podría ser cualquier subrango de un tipo de datos enumerado, por sencillez vamos a suponer siempre que el conjunto de índices varía siempre de 0 a n-1 (siendo n el tamaño del vector).

3. Declaración de un tipo vector.

La declaración de un tipo de dato vector se realiza con la siguiente sintaxis:

```
tipodef tipoDeDato nombreTipo[tipoDatoÍndice]
```

Donde:

- `nombreTipo` es el nombre que damos al tipo de vector definido
- `tipoDatoÍndice` es el tipo de dato del índice del vector, si está claro bastará con indicar el tamaño (número de datos que puede contener el vector)
- `tipoDeDato` es el tipo de dato de los valores que una variable vector declarada del este tipo (`nombreTipo`) podrá contener

Ejemplos:

tipos

tipodef real tipoNotas[100]

Una vez declarado el tipo, se pueden declarar variables de ese tipo. La declaración de las variables de los tipos anteriores se realiza con la sintaxis habitual (utilizando los tipos de datos que acabamos de crear):

variables

tipoNotas notas1

4. Acceso a las componentes de un vector

Una variable de tipo vector está compuesta por un número fijo de componentes del tipo de base del vector (luego, **todas las componentes de un vector, son del mismo tipo**). Cada componente puede ser referenciada de manera independiente y acceder a su valor, escribiendo el **nombre de la variable de tipo vector, seguido del valor de su índice, encerrado entre corchetes**:

```
nombreVariableVector[expresión]
```

donde la evaluación de `expresión` da como resultado el valor del índice del elemento considerado. El acceso a componentes no definidas de un vector es un error de programación, luego hay que asegurarse en cada momento de que las posiciones que se manejan están dentro del conjunto de índices descrito en la declaración.

Ejemplos:

```
notas1[12]=7.8
notas1[23]=2.4
```

5. Operaciones sobre vectores

- Operaciones permitidas sobre **vectores completos**:
  - **Declaración** de vector (aunque no es una operación de "proceso")
  - Paso como **parámetro.**

    Un vector completo puede ser transferido a una acción o una función como parámetro. Sin embargo, la acción o función debe "conocer" la definición del tipo vector. Un modo adecuado de satisfacer esta condición consiste en definir un tipo vector global. Los parámetros reales y los correspondientes parámetros formales pueden ser declarados entonces miembros de este tipo vector, estableciéndose así la correspondencia precisa.

- El resto de los procesos deben llevarse a cabo trabajando sobre **componentes individuales**.

Ejemplos

Supongamos un entorno de trabajo donde se han declarado el siguiente tipo:

```
tipos
  tipodef real tVector[25]
```

Ejemplo 1. Asignación entre vectores:

```
  …
  variables
   tVector grupo1, grupo2
   //Si los elementos de grupo1 ya han sido
   //introducidos en el ordenador y ahora queremos
   //asignar esos elementos a los de grupo2 podríamos
   //escribir:
   para i=0 hasta 24 hacer
        grupo2[i] = grupo1[i]
   fpara
   …
```

Ejemplo 2. Asignación externa (Lectura):

La operación de asignación externa (lectura de datos) de un vector, debe hacerse asignando un valor a cada componente del vector. Para ello, como en el ejemplo anterior, se debe recorrer el vector mediante una estructura repetitiva:

```
para i=0 hasta n-1 hacer
  leer( v[i] )
fpara
```

O bien, si decidimos escribir una acción de lectura de un vector:

```
acción leeVector(S/ tVector v, E/ entero n)
variable
  entero i
principio
  para i=0 hasta n-1 hacer
    leer( v[i] )
  fpara
fin
```

Ejemplo 3. Salida de datos (Escritura):

Para escribir todas o algunas de las componentes de un vector, debemos recorrer el vector y escribir cada una de las componentes por separado.

```
para i=0 hasta n-1 hacer
    escribir( v[i] )
fpara
```

O bien, si decidimos escribir una acción de escritura de un vector:

```
acción escribeVector(E/ tVector v, E/ entero n)
variable
  entero i
principio
  para i=0 hasta n-1 hacer
    escribir( v[i] )
  fpara
fin
```

6. Recorrido y búsqueda en vectores.

6.1. Recorrido: (tratamiento de cada componente individual)

Una operación frecuente trabajando con vectores, es el recorrido del vector, accediendo a todas o una parte de sus componentes.

El recorrido de los elementos de un vector se hará utilizando una estructura repetitiva en la que hacemos variar el valor del índice entre los límites que nos interese, accediendo así a una componente distinta en cada iteración:

```
para k= i1 hasta in hacer
      Tratar el elemento v[k]
fpara
```

o bien, el recorrido en sentido inverso:

```
para k= in descendiendo hasta i1 hacer
      Tratar el elemento v[k]
fpara
```

6.2. Búsqueda (en vector no ordenado)

Dado un vector no ordenado, se trata de comprobar si un determinado elemento se encuentra en alguna componente del vector o no. Veremos un algoritmo de búsqueda secuencial, en la cual se efectúa la búsqueda recorriendo el vector hasta encontrar el elemento, en caso de que éste sea una de las componentes del vector, o hasta llegar al final del vector, en caso de que el elemento no sea una componente del vector.

```
Especificación:

Interfaz:
 Entrada tVector v, entero n, real elemento
 Salida booleano encontrado

Efecto:
 Condiciones de entrada:
 Efecto producido: devuelve verdad si elemento = v[i],
 para algún i=0...n-1, devuelve falso en caso contrario

función BúsquedaSec(tVector v, entero n,
                    real elemento) devuelve booleano

variable
  entero i
  booleano encontrado

principio

  encontrado=falso
  i=0

  mientras que (i < n) and (not  encontrado ) hacer
    si (elemento == v[i]) entonces
      encontrado = verdad
    si no
      i = i+1
    fsi
  fmq
  devuelve(encontrado)
 fin
```

Resolución de los ejemplos

1) Especificar y diseñar un algoritmo que lea una secuencia de datos de tipo entero, terminada con un 0, y construya otra secuencia con los mismos elementos en orden inverso.

Vamos a recorrer la secuencia guardando los datos en un vector para después recorrer el vector en sentido descendente grabando en la nueva secuencia

```
 Especificación:
 Interfaz:
  Entrada secuencia de enteros
  Salida secuencia de enteros
 Efecto:
  Condiciones de entrada: la entrada contiene un 0
  Efecto producido: Muestra en orden inverso los datos de
  la entrada

algoritmo invertirSecuencia
constantes
 entero Max=1000    //Máxima longitud de la secuencia
tipos
 tipodef entero tvector[Max]
variables
 tvector v
 entero cont, n, i
principio
 cont=0
 leer(n)
 mientras que n ≠ 0 hacer
   v[cont]=n
   cont=cont+1
   leer(n)
 fmq
 para i= cont-1 descendiendo hasta 0 hacer
   escribir( v[i] )
 fpara
fin
```

2) Representar en binario un número entero positivo dado en base decimal

Consideramos definimos los tipos

```
tipodef tBit=0..1
tipodef tBit tDigitos[16]
```

Especificación:

Interfaz:
Entrada entero numero
Salida tDigitos v

Efecto:
Condiciones de entrada: numero es positivo
Efecto producido: v almacena la representación binaria de numero. Además, se visualiza por pantalla dicha representación

```
algoritmo decimalBinario
tipos
   tipodef tBit=0..1
   tipodef tBit tDigitos[16]

variables
  entero numero, cont ,i
  tDigitos v
principio
  cont=0
  leer(numero)
  mientras que numero ≠ 0 hacer
    v[cont]=numero mod 2
    numero=numero div 2
    cont=cont+1
  fmq
  para i=cont-1 descendiento hasta 0 hacer
    escribir(v[i])
  fpara
fin
```

Parte II. Registros

1. Introducción.

Un **registro** es un tipo de dato estructurado, es decir, una agrupación o estructura de datos, cuyos elementos constituyentes **no** necesitan ser del mismo tipo. Cada uno de los datos que forman el registro se denomina **campo**.

2. Declaración de un registro.

Para declarar un registro, declararemos primero un tipo de dato registro que después utilizaremos para declarar variables de ese tipo de dato.

Sintaxis de declaración del tipo registro:

```
tipodef
nombreTipoRegistro = registro
                         tipo1     nombreCampo1
                         tipo2     nombreCampo2
                         ...
                         tipoN     nombreCampoN
                     freg
```

donde `nombreTipoRegistro` es el identificador del tipo de dato registro y cada campo, lleva asociado un identificador `nombreCampo_i` y un tipo de dato `tipo_i` (que es el tipo de dato almacenado en el `campo_i`).

Las declaraciones de los campos suelen escribirse en líneas separadas para una mejor legibilidad, aunque no es imprescindible.

El tipo de un campo individual puede ser un tipo estándar, definido por el programador o estructurado. De aquí que **un campo de un registro pueda ser un vector o incluso otro registro**.

Una vez declarado el tipo registro, la declaración de variables de tipo registro se realiza de la forma habitual:

```
variable
   nombreTipo nombreVariable
```

Ejemplo 1:

Vamos a definir dos registros para almacenar fechas:

```
tipodef
   fecha = registro
             entero mes
             entero día
             entero año
           freg

variables
   fecha     hoy, ayer
```

Ejemplo 2:

Sea el **registro** de un cliente compuesto por los siguientes **campos**:

- nº cliente, de tipo entero.
- nombre del cliente, de tipo cadena.
- saldo, de tipo real.

Si representamos este registro por una variable de tipo registro:

```
   tipodef
        tCuenta = registro
                       entero nClte
                       cadena nombreClte
                       real saldoClte
                  freg
   variables
        tCuenta   cliente
```

Hay que tener en cuenta también que un registro puede ser un elemento individual de otro dato de tipo estructurado, tal como un vector. De este modo podemos definir vectores cuyos elementos son registros, registros cuyos elementos (o alguno de ellos) son vectores ...

Ejemplo:

Algoritmo para almacenar las cuentas de a lo sumo 100 clientes.

Vamos a utilizar un vector denominado clientes cuyos elementos individuales serán registros y cada registro se corresponderá con un cliente:

```
tipodef
          tCuenta = registro
                    entero    nclte
                    cadena    nombreclte
                    real      saldoclte
               freg
tipodef   tCuenta tCliente[100]

variables
     tCliente  clientes
```

3. Operaciones con registros completos.

- Operaciones con registros completos (sin tener que tratar cada una de sus componentes individuales):

  - **Declaración**
  - **Asignación**
  - Paso de registro como **parámetro**.

  Para la asignación y el paso como parámetro se requiere obviamente que ambos registros tengan exactamente la misma estructura (idéntico tipo de dato).

- El resto de operaciones deben hacerse procesando cada campo de manera individual

Ejemplo (asignación):

```
algoritmo ejemplo
tipodef
     tFecha = registro
                    entero    mes
                    entero    día
                    entero    año
              freg
     tCuenta = registro
                    entero nclte
                    cadena tipoclte
                    real saldo
                    tFecha ultimopago
               freg
variables
  tCuenta regA, regB
principio
 ....
 { lectura del registro regB}
      regA = regB
 ...
fin
```

4. Acceso a los elementos individuales de un registro

El procesamiento de elementos individuales de un registro es mucho más frecuente que el procesamiento de registros completos.

Para hacer referencia a un campo de un registro y poder acceder a los elementos individuales del registro se construye un designador de campo. Un designador de campo se forma con: el nombre de la variable de tipo registro, seguido de un punto '.' y del identificador del campo al que se quiere acceder:

```
nombreVariableRegistro.nombreCampo
```

Ejemplos:

```
hoy.mes=6
leer(hoy.año)
regA.nClte=25
regA.saldoClte=regA.Caldoclte + 2345.67
clientes[25].saldoClte=1000
```

Si un campo representa un conjunto de datos estructurados, entonces se puede acceder a los elementos individuales del campo incluyendo los elementos de datos estructurados en el designador de campo. Así, por ejemplo, si un campo representa un vector, un elemento individual del vector puede ser accedido escribiendo:

```
nombreRegistro.nombreCampo [valor del índice]
```

Análogamente, si un campo representa un registro incluido, un elemento individual de ese registro puede ser accedido del siguiente modo:

```
nombreRegistro.nombreCampo.nombreSubcampo
```

donde subcampo se refiere a un campo del registro incluido.

Los elementos individuales del registro pueden utilizarse de la misma manera que las variables ordinarias. Las características particulares (y restricciones) que se aplican a cada elemento, están determinadas por el tipo de dato de estos elementos. De esta forma, si un elemento de registro representa una cantidad de tipo simple, entonces puede aparecer en una sentencia de asignación, dentro de una expresión, dentro de una estructura de control, dentro de una instrucción de E/S, como parámetro dentro de la referencia a una acción o función, etc.

Ejemplo:

Para el ejemplo anterior:

```
regA.saldo = 0
regA.saldo = regular.saldo - pago
regB.nclte = regular.nclte
escribir (regA.nclte, regB.nclte)
promedio = (regB.saldo + regA.saldo) / 2
```

Algunas aplicaciones requieren que una secuencia de registros sea almacenada y procesada en un orden determinado. En tales casos, es conveniente definir un vector cuyos elementos sean registros.

El acceso a un registro determinado se hace:

```
nombreVariableVector [valor del índice]
```

y a un elemento individual como:

```
nombreVariableVector[valor índice].nombreCampo
```

Ejemplo:

Sea el ejemplo anterior de un vector con 100 elementos de tipo registro:

```
tipodef
     tCuenta = registro
                    entero    nclte
                    cadena    nombreClte
                    real      saldoClte
               freg
tipodef   tCuenta tCliente[100]

variables
     tCliente clientes
```

Así, `clientes[23]` se refiere al registro 23 y `clientes[23].saldo` se refiere al saldo del cliente 23.

Se podrían escribir instrucciones como éstas:

```
clientes[23].saldo = 0
escribir (clientes[15].nclte)
```

Ejercicios resueltos

Dado el siguiente entorno de trabajo:

```
tipodef
  tComplejo = registro
                real      parteReal,parteImag
              freg
```

1.

Especificación:

Interfaz:
Entrada real v1, v2
Salida tComplejo x

Efecto:
Condiciones de entrada: v1, v2 son valores cualesquiera.
Efecto producido: Devuelve un número complejo de parte real v1 y parte imaginaria v2.

```
función asignar(real v1, real v2) devuelve tComplejo
variables
  tComplejo  x
principio
  x.parteReal=v1
  x.parteImag=v2
  devuelve(x)
fin
```

2.

Especificación:

Interfaz:
Entrada tComplejo x, y
Salida tComplejo z

Efecto:
Condiciones de entrada:
Efecto producido: Devuelve la suma de los complejos x e y

```
función sumar(tComplejo x, tComplejo y) devuelve tComplejo
variables
  tComplejo  z
principio
  z.parteReal = x.parteReal + y.parteReal
  z.parteImag = x.parteImag + y.parteImag
  devuelve(z)
fin
```

Ejercicios propuestos. Vectores

Suponiendo el siguiente entorno de trabajo en el que se han realizado las siguientes declaraciones:

```
constante
  entero Max
tipodef
      real tvector[Max]
```

1. **función** máximo (tvector v, entero n) **devuelve** real

   {Devuelve el mayor elemento de un vector de tipo *real* de tamaño n}

2. **función** media (tvector v, entero n) **devuelve** real

   {Devuelve la media aritmética de los elementos de un vector de tipo *real* de tamaño n}

3. Construir un algoritmo que lea 20 enteros, los almacene en un vector y diga cuántos de ellos superan la media.

4. Escribe un algoritmo que determine si una frase terminada en un punto es un palíndromo (tiene el mismo sentido leído de izquierda a derecha que de derecha a izquierda). Por ejemplo *dabale arroz a la zorra el abad*.

5. Especificar y diseñar una función que, dado un número entero positivo, diga si es capicúa o no.

6. Programa que calcule los números capicúas de cuatro cifras múltiplos de 45 y los almacene en un vector.

7. Realizar un programa que almacene en un vector todos los números *omirps* comprendidos entre 2 y 1000. Un número natural *n* diremos que es *omirps* si es primo, no es capicúa y el invertido de *n* también es primo.

8. Sea el entorno de trabajo:

constante
Max = 10
tipodef
entero tvector [10]

Sea el número entero *Num* y una variable *v* de tipo *tvector* y de dimensión *n*.. Especificar y diseñar una función que compruebe que las cifras que forman el número *Num* son exactamente las componentes del vector *v* . Las cifras almacenadas en el vector no tienen porqué estar en el orden correcto. Por ejemplo, dado *Num* =1735, *v* =(7,1,3,5) y *n*.=4, la función deberá devolver el valor VERDAD. Suponer por simplicidad que el número *Num* no tiene ninguna cifra repetida.

9. Dado el siguiente entorno de trabajo:

constante
Max = 100
tipodef
entero tvector [Max]

Sea v de tipo tvector con dimensión n. Especificar y diseñar una acción que compruebe si existen componentes consecutivas de v tal que su suma sea cero, en caso afirmativo dar el número de estas componentes. Por ej.: dado v={1,5,-4,-1,7} y n=5, la acción deberá devolver el valor VERDAD, ya que las componentes consecutivas de v, {5,-4,-1}, cumplen la condición, el número de componentes consecutivas es 3.

11. Dos palabras son anagramas cuando una se obtiene a partir de la otra mediante una reordenación de las letras que la forman.

Ejemplos:
- AMOR, MORA, ROMA, RAMO son anagramas entre ellas pero no lo son con AROMA o REMO.
- ROMO y MORO son anagramas entre ellas pero no lo son con ROMA

Especificar y diseñar un subalgoritmo que dados **v** y **w** de tipo **tpalabra** y dimensión **n** y **m** respectivamente determine si **w** es un anagrama de **v**.
Notas:
- Se supone declarado el siguiente tipo de dato:
  tipodef carácter tpalabra[Max]

- MAX es una constante cuyo valor es irrelevante
- n, m <= MAX

12. Sea el siguiente entorno de trabajo:

constantes
MAX = ...
tipodef
entero t_vector[MAX]

Dados dos vectores de tipo t_vector (**VIEJO** de dimensión **nv** y **ALTAS** de dimensión **na**) ordenados de forma creciente, especifica y diseña un subalgoritmo que cree y devuelva un vector (**NUEVO** de dimensión nv+na < MAX) de tipo t_vector ordenado de forma creciente, que incluya los elementos de VIEJO y ALTAS.

Ejemplo.
Dados:
VIEJO: (2, 4, 5, 8, 13, 14, 38)
ALTAS: (6, 7, 10, 12)
Devolvería:
NUEVO: (2, 4, 5, 6, 7, 8, 10, 12, 13, 14, 38)

Nota: Suponer que no hay elementos repetidos

13. Dado el siguiente entorno de trabajo:

constantes

MAX = 100

tipos

tipodef carácter t_lingo[MAX]

tipodef entero t_acierto[MAX]

Sean clave e intento variables de tipo t_lingo y de la misma dimensión, n. Construir un subalgoritmo que compruebe si son iguales las variables anteriores y que devuelva en una variable de tipo t_acierto la siguiente información sobre cada componente de la variable intento:

1. Si la componente i-ésima de intento coincide con la componente i-ésima de clave almacenará un 2.
2. Si la componente i-ésima de intento coincide con alguna componente de clave distinta de la i-ésima almacenará un 1.
3. Si la componente i-ésima de intento no coincide con ninguna de clave almacenará un 0.

Supondremos que todas las componentes de clave e intento son distintas.

14. Supón declarado el siguiente tipo de datos:

tipodef entero tVector[100]

a) Diseña una función con la siguiente cabecera:

> **función** es_suma_parcial(tvector v, entero n, entero k) **devuelve** booleano
>
> que indique si *k* es una suma de componentes consecutivas de *v* comenzando en 0. (Siendo n el número de componentes de v). Es decir, devolverá verdad si para algún *j* (entre 0 y n-1):
>
> k= v[0]+v[2]+...+v[j]

b) Diseñar un subalgoritmo tal que dados *v* y *w* de tipo tvector con número de componentes *nv* y *nw* respectivamente, devuelva verdad si alguna de las componentes de *w* es suma parcial (según lo descrito en el apartado anterior) de componentes de *v*, y devuelva falso en caso contrario. (Puedes utilizar la función del apartado anterior)

15. Supón declarada la siguiente estructura de datos:

tipodef entero tVector[100]

Escribe un subprograma que reciba una estructura de datos de tipo *tVector* así como el número de elementos a tratar de ese vector. Devolverá una estructura de datos de tipo *tVector* con los números primos contenidos en el vector de entrada. Debes crear una función para determinar si un entero positivo es primo.

16. Dada la siguiente definición de tipos:

> **tipodef** entero tVector[100]

Diseña subalgoritmos para resolver los siguientes problemas:

- **a)** Dado un vector, calcular el número de componentes cuyo valor es un número par.
- **b)** Calcular la moda de un vector (valor que más veces aparece repetido y número de veces que aparece).

17. Se han registrado las temperaturas máximas diarias alcanzadas en Logroño durante el año pasado (suponer valores enteros, se han redondeado). Se sabe que esas temperaturas máximas se movieron en un rango entre 1 y 40 grados. Supón definidos los siguientes tipos de datos

tipodef entero tMaximas[366]
tipodef entero tFrecuencias[40]

Diseña un subprograma que construya y devuelva un vector de frecuencias a partir del vector de máximas. El vector de frecuencias contendrá en cada componente " i " el número de días que el año pasado se dio como temperatura máxima " i " grados.

18. Se dispone de una máquina de alta precisión capaz de fabricar tornillos de varios tipos y tamaños.
El departamento de control de calidad quiere controlar el funcionamiento de la máquina, y para ello se ha realizado la siguiente prueba:
Se fabricaron 100 tornillos de un mismo tipo de un determinado diámetro. Las medidas reales de los diámetros de cada tornillo fueron recogidas en una variable de tipo vector tPrueba:

tipodef real tPrueba[500]

Diseña una función que reciba estas medidas, el valor teórico del diámetro y la tolerancia admitida y devuelva verdad si la máquina funciona bien y falso si lo hace mal. Se considera que la máquina funciona mal si en el lote de 100 tornillos hay más de 10 defectuosos (un tornillo se considera defectuoso si su diámetro real y el diámetro teórico se diferencian más de la tolerancia admitida).

Ejercicios propuestos. Registros

1. Definir un tipo de dato (tFecha) para representar una fecha mediante un registro con los campos día, mes y año. Construir una función que dadas dos variables de tipo tFecha indique si son iguales o no.
2. Define un tipo de dato (tAlumno), consistente en un registro con los siguientes campos: Nombre, Apellido1, Apellido2, Nota. Construir acciones de lectura y escritura para este tipo de dato.
3. Define un tipo de dato (tClase), consistente en un vector de tipo tAlumno de tamaño 60. Construye las acciones de lectura y escritura para este tipo de dato utilizando las acciones del ejercicio anterior.
4. Construye funciones que devuelvan la siguiente información sobre la estructura del ejercicio anterior:

   a) Datos del alumno con mayor valor en el campo Nota.

   b) Media de las Notas de todos los alumnos.

5. Los promotores de la *Vuelta Ciclista a España*, quieren crear una base de datos para controlar a los participantes. La información que se desea almacenar para cada corredor es el nombre, dorsal, equipo, tiempo invertido y diferencia de tiempo con el líder de la carrera. Este último dato será calculado a partir de los demás en uno de los apartados del ejercicio. Los datos referentes a tiempos serán considerados enteros.

   a) Proponer las estructuras de datos necesarias para almacenar los datos correspondientes a un corredor (tCorredor) y a todos los corredores (tPelotón). Para ello se dispone de una constante que almacena el número de corredores de la carrera.

   constantes

   NUM_CORREDORES = ... {su valor no es relevante}

   b) Especificar y realizar una acción que devuelva el dorsal y el tiempo invertido del líder de la carrera (el que menos tiempo lleva empleado).

   acción Lider (**E/** tPeloton p, **E/** entero n, **S/** entero tiempo, dorsal)

donde p es el conjunto de corredores y n el tamaño del vector p.

c) Especificar y realizar un subprograma que reciba la información sobre todos los corredores y rellene el valor del campo diferencia para todos ellos.

6. Un grupo financiero quiere informatizar la gestión de sus inversiones. La información que se desea almacenar de cada inversión es el nombre de la empresa de la que posee acciones, el número de acciones que posee, el precio actual de cada acción en el mercado, los precios más altos y más bajos que alcanzaron esas acciones y las fechas (día, mes y año) en que se consiguieron.

a) Proponer las estructuras de datos necesarias para almacenar los datos correspondientes a una inversión (**tInversión**) y a todas las inversiones del grupo financiero (**tGrupo**).

b) Especificar y diseñar un subalgoritmo que devuelva los datos de la inversión cuyas acciones tengan el precio actual en el mercado más alto.

c) Especificar y diseñar un subalgoritmo que dada una empresa, el precio actual en el mercado y la fecha, actualice la base de datos (**tGrupo**).

7. Una empresa distribuidora de muebles desea mecanizar la gestión de su almacén. Para cada mueble se necesita manejar la siguiente información: tipo (puede ser A, B o C), modelo (cadena de caracteres), fecha de fabricación (día, mes y año), dimensiones del mueble (alto, ancho y profundidad), peso, y fecha de entrega (día, mes y año).

a) Declara las estructuras de datos apropiadas para almacenar los datos sobre cada mueble y sobre el stock de muebles que se encuentran en el almacén.

b) Especifica y diseña un subprograma que devuelva los datos del mueble de mayor volumen almacenado.

c) Supón que la empresa distribuidora de muebles desea enviar un camión con muebles de un determinado tipo a una de sus delegaciones. (El camión que se va a utilizar tiene una capacidad máxima de 10000 kilogramos de peso y 90 metros cúbicos de volumen).
Especificar y diseñar un subprograma que dado el stock del almacén y el tipo de mueble, devuelva los datos de los muebles a transportar.

Nota: Supondremos que el criterio seguido para completar el camión será el de ir cargando muebles del tipo solicitado hasta que no quepan más, sin preocuparse si es la carga óptima o no.

8. Supón definida la siguiente estructura de datos:

```
tipodef tCodAsig = (ITI01, ITI02, ITI03, ..., ITI99)

tCodAsig MatricAsig[10]

tMatrícula = registro
                entero dni
                entero nº_asig_de_las_que_se_matricula
                tMAtricAsig asignaturas
             freg

tipodef tMatricula tMatrículaS[NALUMNOS]
```

a) Especificar y diseñar un subalgoritmo que devuelva el número de alumnos matriculados de una única asignatura.

b) Especificar y diseñar un subalgoritmo que compruebe si un determinado código de asignatura existe en el vector de códigos de asignaturas.

c) Especificar y diseñar un subalgoritmo que devuelva el número total de alumnos matriculados en la asignatura ITI01.

9. Se suponen definidos los siguientes tipos de datos:

```
tipos
   tipodef     entero t_apuesta[6]

   tipodef     t_apostante = registro
               entero dni
               cadena nombre
               t_apuesta apuesta
               freg

   tipodef     t_apostante t_apostanteS[100]
```

a) Especifica y diseña una función que, a partir de los datos de un determinado apostante (*t_apostante*) y de la combinación ganadora del sorteo (*t_apuesta*), devuelva el número de aciertos de dicho apostante.

La cabecera será la siguiente:

```
función aciertos(t_apostante a, t_apuesta c_ganadora) devuelve entero
```

b) Especifica y diseña un subalgoritmo que, a partir de los datos de todos los apostantes (*t_apostanteS*) y su número, y de la combinación ganadora (*t_apuesta*), devuelva los datos del apostante (*t_apostante*) con mayor número de aciertos (supón por simplicidad que éste es único y que al menos un apostante ha acertado al menos un número).

(Puedes utilizar la función del apartado anterior)

NOTA:

t_apuesta se utiliza para representar los números de la combinación de un apostante, así como los de la combinación que sale del sorteo de la primitiva.

10. Se suponen definidos los siguientes tipos de datos:

```
tipos

tipodef
   tPais = registro
              cadena      codPaís
              cadena      nombrePaís
              cadena      moneda
           freg
tipodef
   tMoneda = registro
              tPais       país
              real        cambio
                          (cambio respecto al Euro, p.ej., para la peseta 166,386)
              freg
tipodef       tMoneda     tMonedaS[15]
```

a) Especifica y diseña un subalgoritmo que, a partir de los datos de las monedas de la Unión, devuelva el nombre de la moneda de menor valor (mayor cambio).

b) Especifica y diseña un subalgoritmo que calcule el cambio entre monedas de dos países cualesquiera de la Unión.

Recibe:

Los datos de las monedas de los 15,
El nombre de la moneda y la cantidad de ella que se desea cambiar,
El nombre de la moneda a la que se desea cambiar

Devuelve:

Cantidad de la moneda cambiada

Ejemplo:

Cambiar **1000 Pesetas** a **Francos**
Devolvería **39,42**

Ayuda: Utiliza como intermediario el cambio a Euro. Para el ejemplo:
1000 pts / 166,386 = 6,01euros
6,01euro * 6,559 = 39,42 francos

11. Se suponen definidos los siguientes tipos que nos van a permitir gestionar las notas de nuestros alumnos:

```
tipodef
tCalificacion= registro
        entero      identAsig     //para identificar la asig que se califica
        real        calif         //nota en la asignatura
        booleano    presentado    //presentado o no
        freg

tipodef     tCalificacion  tNotaS[10]

tipodef
    tAlumno= registro
        entero      identAlumno   //para identificar al alumno calificado
        entero      numAsig       //número de asignaturas que cursa
        tNotaS      notas         // calificaciones del alumno en todas las //asignaturas que cursa, puede tener no presentados
        freg

tipodef     tAlumno  tClase[100]
```

Se pide diseñar subalgoritmos (acciones o funciones según convenga) para:

a) Leer una calificación (algo de tipo tCalificación) cuyos valores se introducen desde el teclado

b) Leer un alumno (algo de tipo tAlumno) cuyos datos se introducen desde el teclado.

c) “Dado un alumno”, calcular la nota media que ha obtenido en las asignaturas a las que se ha presentado (la media será cero si no se ha presentado a ninguna asignatura).

d) “Dada una clase” proporcionar la mejor nota media y el alumno con mejor nota media.

12. Una compañía de vuelos de bajo coste desea gestionar la información acerca de sus vuelos.

Para ello, se dispone de la siguiente estructura de datos:

```
tipodef
tLugarHora = registro
                cadena      ciudad
                cadena      aeropuerto
                real        hora
             freg

tipodef   booleano  tAsientos[200]
                        //verdad si el asiento está libre, falso si
                        ocupado
tipodef
tVuelo = registro
             cadena         codVuelo
             tLugarHora     salida
             tLugarHora     llegada
             entero         numAsientosAvion //número de asientos
                                             del avión
             tAsientos      asientos
          freg

tipodef   tVuelo  tVueloS[100]
```

a) Especifica y diseña un subprograma que reserve el primer asiento libre en un determinado vuelo (dado el vuelo (registro tVuelo)). Devolverá además un valor lógico indicando si la reserva ha podido realizarse (verdad) o no (falso).

b) Especifica y diseña un subprograma que, dado el vector de vuelos (tVueloS), su tamaño, y el nombre de una determinada ciudad, devuelva el número de vuelos que salen de esa ciudad con algún asiento libre.

c) Especifica y diseña un subprograma que, dado el vector de vuelos (tVueloS), su tamaño, y el código de un determinado vuelo, muestre por pantalla la lista de asientos libres de ese vuelo.

13. Una empresa dedicada a la fabricación de zapatillas desea gestionar la información acerca de sus trabajadores.

Para ello, se dispone de la siguiente estructura de datos:

```
tipodef
entero    tProducciónDiaria[30]    //unidades fabricadas cada día del mes
tipodef
tTrabajador =  re[g]istro
                    entero              dni
                    entero              numDelegación
                    entero              numDíasTrabajados
                    tProduccionDiaria   producción
               freg
tipodef
tTrabajador  tTrabajadorES[50]
```

Especifica y diseña un subprograma que devuelva los datos de los trabajadores de una determinada delegación (dada como parámetro) que superaron algún día un umbral (dado como parámetro) de unidades producidas.

Nota: utiliza los parámetros que consideres necesarios, obviamente de los tipos anteriores o de tipos simples

14. Un centro de salud utiliza las siguientes estructuras de datos para representar la información sobre sus consultas diarias:

```
tipodef     cadena  tDNIpacientes[30]
                //vector que contiene los DNIs de los pacientes
tipodef
tConsulta = registro
                cadena          nombreMédico
                cadena          apellidosMédico
                cadena          especialidad
                entero          numPacientes
                tDNIpacientes   pacientes
            freg
            //Contiene los datos de un médico y sus pacientes
tipodef     tConsulta   tConsultaS[10]
//datos de las consultas que se van a realizar un determinado día
```

a) Especifica y diseña un subprograma que tome los datos que consideres necesarios y devuelva el nombre y apellidos del médico que atiende más pacientes en la consulta.

b) Especifica y diseña un subprograma que tome el DNI de un paciente, las consultas (un dato de tipo tConsultaS) y el número de consultas que va a haber ese día y devuelva las especialidades de los médicos que van a atender a ese paciente.