Fundamentos de Programación I

Tema 5. Estructuras de datos: arrays y registros

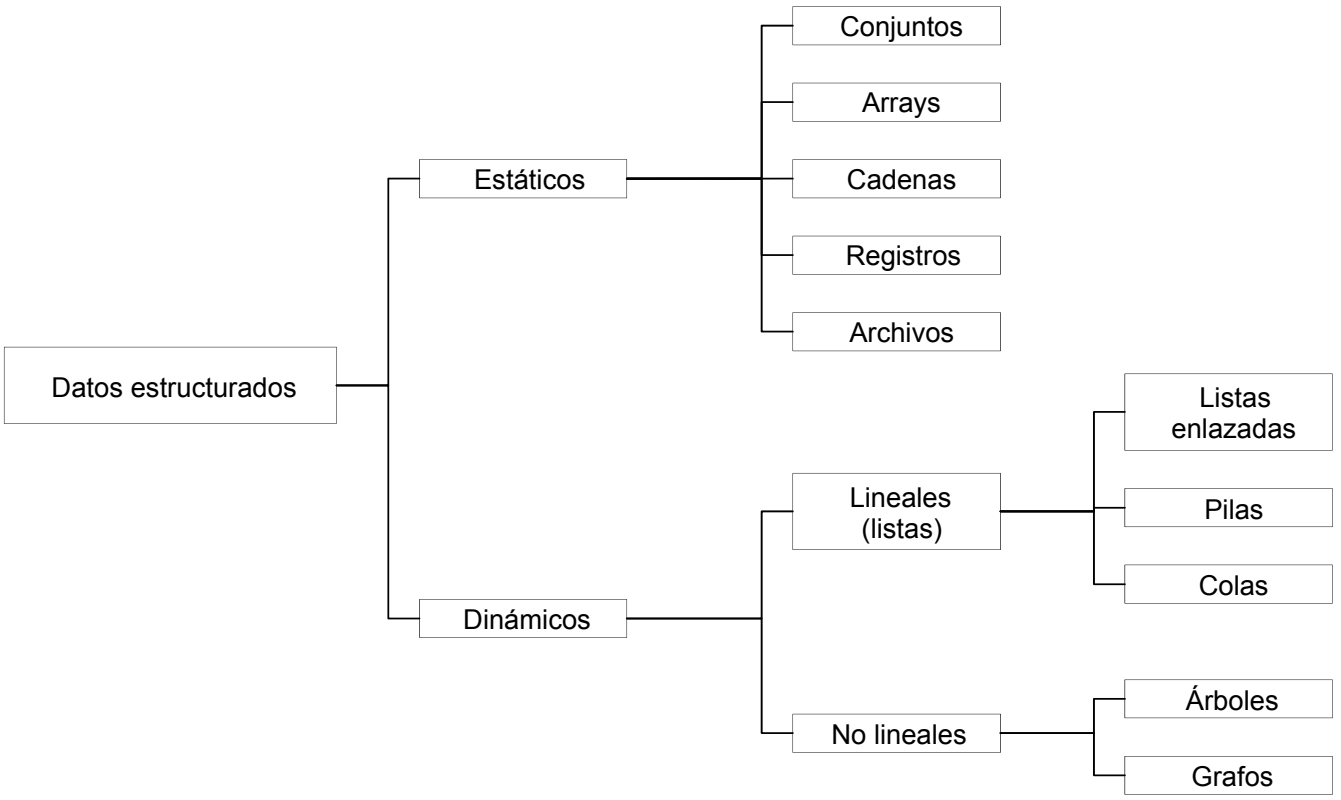
Luís Rodríguez Baena (luis.rodriguez@upsam.net)

Universidad Pontificia de Salamanca (campus Madrid)
Escuela Superior de Ingeniería y Arquitectura

Introducción a las estructuras de datos

- Datos estructurados: colección de datos caracterizada por su organización y las operaciones que se pueden realizar sobre ella.
- Están compuestos por otros tipos de datos más simples.
- Según la forma de asignación de memoria:
  - Estructuras de datos estáticas.
    - ✓ Reservan espacio de almacenamiento en tiempo de compilación.
    - ✓ Su tamaño y posición en memoria no cambian a lo largo de la ejecución del programa.
  - Estructuras de datos dinámicas.
    - ✓ Reservan espacio de almacenamiento en tiempo de ejecución.
    - ✓ Su tamaño y posición en memoria pueden cambiar a lo largo de la ejecución del programa.

Introducción a las estructuras de datos (II)



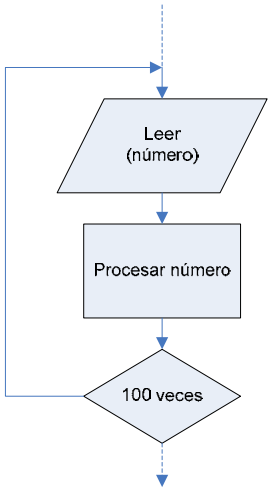
Arrays

- Estructura de datos formada por un conjunto finito y ordenado de elementos homogéneos.
  - Elementos homogéneos: todos los elementos son del mismo tipo: el **tipo base** del array.
  - Serie finita: tiene un tamaño limitado que habrá que definir en la declaración y que no puede cambiar.
  - Ordenado: cada elemento puede ser identificado por la posición que ocupa en la estructura y ser tratado individualmente.
    - ✓ En todo caso, estarán ordenados por su posición dentro de la estructura.
- Se trata de una estructura de acceso directo o aleatorio.
  - Es posible acceder a cada elemento de forma individual.
    - ✓ Los elementos pueden seleccionarse arbitrariamente y son igualmente accesibles.
    - ✓ Para designar a un elemento se utilizará el **selector**.
      - Uno o más índices que indican su posición dentro de la estructura.
  - Dependiendo del número de índices necesario para referenciar un componente habrá:
    - ✓ Arrays unidimensionales (vectores).
    - ✓ Arrays bidimensionales (matrices)
    - ✓ Arrays multidimensionales.

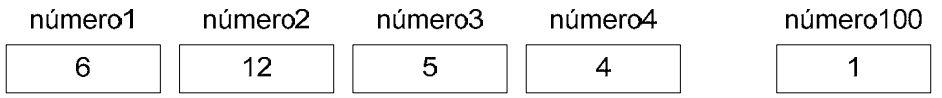
Arrays unidimensionales: vectores

- La referencia a un elemento de hace mediante un único índice.
  - Sería la repetición de una variable del tipo base del array.
- ¿Cómo se pueden tratar de n datos homogéneos?
  - Solución 1: Utilizar una única variable y destruir el contenido anterior en cada asignación.
    - ✓ No se puede recuperar la información.
  - Solución 2: Utilizar n variables.
    - ✓ Impracticable.
  - Solución 3: Utilizar un array de n elementos.
    - ✓ Repite un dato simple n veces.

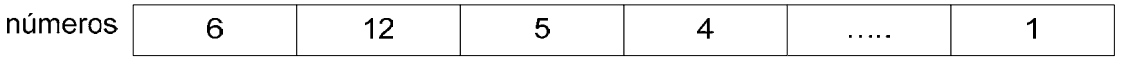
Arrays unidimensionales: vectores (II)



Solución 1



Solución 2



Solución 3

Declaración de variables de tipo vector

- En los tipos de datos estándar, el compilador ya conoce cómo almacenar el dato en memoria y cómo interpretar su contenido.
- En los arrays será necesario además decir cual es su tipo base, su número de elementos y cómo se hará referencia a los componentes individuales.
  - El compilador sabe lo que es un array, pero no sabe ni su tipo, ni su número de elementos ni cómo se referenciará cada elemento.
- Declaración:

```
var
   array[limInf..limSup] de tipoBase : nombreArray
```

Declaración de variables de tipo vector (II)

- *tipoBase*.
  - Indicaría el tipo de dato de cada elemento del array.
  - Puede ser cualquier tipo de dato estándar o definido por el usuario.
- *nombreArray*.
  - Un identificador válido.
- *limInf..limSup*.
  - Doble función:
    - ✓ Es un subrango con todos los valores posible que podrá tomar un índice de array.
      - Los índices debe ser mayores o iguales a *limInf* y mayores o iguales a *limSup*.
    - ✓ Indica también el número de elementos del array: todos los valores posibles entre los dos límites.
  - Normalmente se tratará de valores enteros, pero en algunos lenguajes se puede utilizar cualquier valor de tipo ordinal (como pueden ser los datos de tipo carácter).

Declaración de variables de tipo vector (III)

```
...
var
   //Declara dos vectores con 6 elementos (entre 0 y 5) de tipo entero
   array[0..5] de entero : números, otroVector
   //Declara un vector de 20 elementos de tipo cadena
   //Los índices podrán ir desde 1 hasta 20
   array[1..20] de cadena : nombres
   //Declara un vector de 6 elementos con las ventas
   //realizadas entre los años 2000 y 2005.
   //Los índices a utilizar para referenciar cada
   //tomarán los valores de los años entre 2000 y 2005
   array[2000..2005] de real : ventas
...
```

- Operaciones con el vector.
  - La **única operación** permitida con la **estructura de datos** es la asignación.
  - Es posible asignar un array a otro siempre que sea del mismo tipo.
    - ✓ Mismo número de elementos, mismos límites y mismo tipo de datos.

```
...
 //Asigna a números cada uno de los elementos de otroVector
números ← otroVector
```

Declaración de variables de tipo vector (IV)

- En C la declaración de una variable de tipo array se hace siguiendo este formato:

  tipoDato nombreArray[*númElementos*]

  - Hay que notar que entre corchetes no aparece ni el índice superior ni el índice inferior, sino el número de elementos.
    - ✓ El índice inferior siempre es 0.
    - ✓ El índice superior sería `númElementos-1`.

```
//Declara un array de enteros de 6 elementos.
//Los índices irán entre 0 y 5.
int numeros[6];
//Declara un array 10 elementos de tipo cadena con índices entre 0 y 9
char *nombres[10];
//Declara un array de 6 elementos reales para las ventas entre el año 2005 y 2011
double ventas[6];
```

- En C la asignación no se permite hacer asignación de arrays.
  - La asignación `otroVector = numeros` sería errónea.

Referencia a los elementos de un vector

- Cada elemento de un vector se identifica mediante su índice.
  - Todos los elementos ocupan posiciones contiguas de memoria.
  - Cada índice indica la posición del elemento dentro de la estructura.
    - ✓ Al ser todos los elementos del mismo tipo, es posible acceder al elemento n.
      - Si la variable de tipo array indica el comienzo de la estructura, para acceder al elemento n habrá que saltar n-1 posiciones desde la posición inicial.
  - El acceso a un elemento del vector se realizará mediante el *selector*.
    - ✓ El selector está formado por el nombre del array y su índice (su posición dentro de la estructura) encerrado entre corchetes.
      - `números[2]` hará referencia al segundo elemento del vector `números` (dispositivas 6 y 9), es decir al valor entero 12.

Referencia a los elementos de un vector (II)

- Cada elemento del vector es un dato del tipo base del mismo.
  - `números[2]` será un dato de tipo entero.
    - ✓ Sobre él se podrá hacer cualquiera de las operaciones posibles con un número entero.
  - Cualquiera de las siguientes operaciones se podrán hacer con los elementos de los vectores declarado en la diapositiva 9.

```
...
números[4] ← 3
números[1] ← números[4] + 1
leer(números[3])
nombres[2] ← 'Pepe'
si nombres[2] > 'Juan' entonces
   escribir(nombres[2], ' es mayor que Juan'
fin_si
//Llamada a la función Factorial desarrollada en el Tema 4
//La función Factorial precisa un argumento de tipo entero
//y números[5] es un dato de tipo entero
escribir(Factorial(números[5]))
...
```

Acceso secuencial al vector:
Recorrido

- Un vector es una estructura repetitiva.
  - Los elementos se tratarán mediante una estructura repetitiva.
- El **recorrido** de un vector consiste en recorrer consecutivamente (secuencialmente) todos sus elementos.
  - Se necesitará un contador que vaya tomando todos los valores posibles del índice.
    - ✓ Lo más práctico es utilizar una estructura de tipo desde.
- Para introducir por teclado todos los elementos de `números`:

```
...
desde i ← 0 hasta 5 hacer
   leer(números[i])
fin_desde
...
```

Acceso secuencial al vector:
Recorrido (II)

- Para inicializar todos los elementos del vector `ventas` a 0.

```
...
desde i ← 2000 hasta 2005 hacer
   ventas[i] ← 0
fin_desde
...
```

- Para obtener la suma de todos los elementos del vector `números`.

```
...
suma ← 0
desde i ← 0 hasta 5 hacer
   suma ← suma + números[i]
fin_desde
escribir(suma)
...
```

Acceso secuencial a un vector: Recorrido (III)

- Recorridos en C

```
//Lee todos los elementos de numeros
for(i=0;i<5;i++)
    scanf("%i",&numeros[i]);

//Inicializa todos los elementos del vector ventas a 0
for(i=0;i<6;i++)
    ventas[i] = 0;

//Rellena todos los elementos del vector ventas con números aleatorios
//Es necesario añadir los archivos de cabecera stdlib.h y time.h
srand((unsigned)time(NULL)); //Establece una semilla para generar números aleatorios
for(i=0;i<5;i++)
    numeros[i] = rand();   //rand genera un número aleatorio entre 0 y 32767

//Obtiene la sume de las ventas entre 2005 y 2011
float suma = 0;
for(i=0;i<6;i++)
   suma += ventas[i];
printf("Ventas entre el año 2005 y 2011: %.2f\n",suma);
```

Ejemplo 5.1

Se desea generar un array con la desviación respecto a la media media de cada uno de los elementos de una lista de 100 números reales. El programa deberá mostrar un listado con los elementos de la lista y la desviación media de cada uno de ellos.

- ***Análisis del problema***

  La desviación respecto a la media de un elemento i de una lista se calcula mediante la expresión

$$D_i = \left| x_i - \overline{x} \right|$$

  El algoritmo deberá almacenar los elementos de la lista en un array y calcular la media aritmética de los elementos del mismo. A continuación se rellenará el array con las desviaciones medias aplicando la expresión anterior.

Ejemplo 5.1 (II)

```
algoritmo DesviaciónRespectoMedia
var
   array[1..100] de real : lista, drm
   entero : i
   real : media
inicio
   //Rellenar la lista de elementos por teclado
   desde i ← 1 hasta 100 hacer
      leer(lista[i])
   fin_desde
   //Calcular la media aritmética de la lista
   media ← 0
   desde i ← 1 hasta 100 hacer
      media ← media + lista[i]
   fin_desde
   media ← media / 100
   //Generar el vector con la desviación media
   desde i ← 1 hasta 100 hacer
      drm[i] ← abs(lista[i]- media)
   fin_desde
   //Salida de resultados
   desde i ← 1 hasta 100 hacer
      escribir(lista[i], drm[i])
   fin_desde
fin
```

Declaración de tipos de datos de tipo vector

- Declaración de tipos de datos.
  - El conjunto de tipos de datos ofrecidos por un lenguaje de programación el limitado.
  - Muchos lenguajes permiten definir nuestros propios tipos de datos.
  - La declaración se realiza dentro de la sección de declaraciones de tipos de datos del algoritmo mediante la palabra reservada **`tipos`**.
  - La declaración se realizará entre la cabecera y el cuerpo del programa.

```
algoritmo NombreAlgoritmo
tipos
   definiciónTipoDato = nombreTipoDato
   ...
var
   ...
inicio
   ...
```

Declaración de tipos de datos de tipo vector (II)

- Por ejemplo, se podrían hacer las siguientes declaraciones:

```
const
   numElememento s = 1000
tipos
   entero = miTipoDeDatoEntero
   array[1..100] de real = reales
   array[1..numElementos] de real = vector
```

- Se está declarando un nuevo tipo de dato que podrá contener cualquiera de los tipos de datos que se definen.
  - ✓ Una variable de tipo `miTipoDeDatoEntero`, podrá contener un número entero.
  - ✓ Una variable de tipo `reales`, contendrá un array de 100 posiciones de números reales.
  - ✓ Una variable de tipo `vector`, podrá hacer referencia a un array de números reales con índices comprendidos entre 1 y `numElementos`.

Declaración de tipos de datos de tipo vector (III)

- Una vez hechas esas declaraciones se podrán utilizar variables de esos tipos de datos:

```
var
   vector : lista, otroArrayDeReales
```

- La declaración de tipos de datos es útil cuando se utilizan datos que no están definidos por el lenguaje, como los vectores.
  - Se pueden utilizar como documentación del programa.
  - Se simplifica el mantenimiento y la reutilización del código.
    - ✓ Tal como está definido, `lista`, podrá contener números reales, pero si en un futuro es necesario que almacene enteros, cadenas u otro tipo de dato, sólo será necesario cambiar la definición del tipo.

Declaración de tipos de datos de tipo vector (IV)

- Para definir tipos de datos, C posee la declaración `typedef`.

  `typedef` *`tipoDatoExistente nuevoTipoDato`*

  - Por ejemplo:

    ```
    typedef int MiTipoDeDatosEntero;
    typedef float Reales[100];
    typedef float Vector[NUM_ELEMENTOS]
    ```

  - Declaran respectivamente nuevos tipos de datos para nombrar a un entero, un array de 100 números reales o un array de `NUM_ELEMENTOS` elementos reales.
  - Una vez definidos podemos utilizar esos tipos de datos para declarar variables de ese tipo.

    ```
    int MiTipoDeDatosEntero i;
    Reales numeros;
    Vector lista;
    ```

Uso de arrays en subprogramas

- Los elementos de un array son datos del tipo base del array.
  - Se pueden utilizar los elementos como parámetros actuales, siempre que el parámetro formal requiera un dato de ese tipo.
  - Por ejemplo, si el array `números` contiene números enteros, es posible utilizar un elemento del array siempre que se necesite un argumento de tipo entero.

```
escribir(factorial(números[2]))
```

- Se puede utilizar un array completo como argumento a un subprograma.
  - Cómo parámetro actual sólo habrá que indicar el nombre de la variable de tipo array.
  - Como parámetro formal habrá que indicar que se trata de un array:
    - ✓ Utilizando un tipo de dato previamente definido como array.
    - ✓ Indicando que es un array poniendo corchetes vacíos (`[]`) después del identificador del parámetro.

Uso de arrays en subprogramas (II)

- Una función que devuelve el la suma de elementos de un array de números reales.

```
//Una cabecera alternativa podría ser:
//real función SumaVector(valor real : v[]; valor entero : n)
real función SumaVector(valor vector : v; valor entero : n)
var
   entero : i
   real : suma
inicio
   suma ← 0
   desde i ← 1 hasta n hacer
      suma ← suma + v[i]
   fin_desde
   devolver(suma)
fin_función
...
//La llamada podría ser
var
   vector : lista
...
    escribir(SumaVector(lista,50)
```

Uso de arrays en subprogramas (III)

- Una función que devuelve la posición del mayor de los elementos de un vector de números reales.

```
entero función MáximoVector(valor real: v[]; valor entero : n)
var
   entero : máx, i
inicio
   //máx guarda la posición del máximo
   //Inicialmente se supone que el máximo está en la posición 1
   máx ← 1
   //Se recorre el array para ver si entre el resto de los
   //elementos hay alguno mayor que el que ocupa la posición 1
   desde i ← 2 hasta n hacer
      //Si el elemento i es mayor que el elemento que ocupa
      //la posición máx, i será la nueva posición del máximo
      si v[i] > v[máx] entonces
         máx ← i
      fin_si
   fin_desde
   devolver(máx)
fin_función
```

Uso de arrays en subprogramas (IV)

- Los ejemplos en C.

```
//Procedimiento que lee un array de reales
//En C los arrays se pasan siempre por referencia
//Si se ha hecho la declaración typedef de la diapositiva 21, la cabecera podría ser:
//void leerVector(vector v,int n){
void leerVector(float v[],int n){
   int i;
   for(i=0; i<n;i++){
     printf("Número: ");
     scanf("%f",&v[i]);
   }
}

//Función que devuelve la suma de un array de reales
float sumaVector(float v[], int n){
   float  suma = 0;
   int i;
   for(i=0; i<n;i++)
       suma += v[i];
   return suma;
}
```

Uso de arrays en subprogramas (V)

- Los ejemplos en C.

```
//Función que devuelve la posición del elemento mayor de un vector
int maximoVector(float v[],int n){
    int i, max;
    max = 0;
    for(i=1;i<n;i++){
       if(v[i] > v[max]){
          max = i;
          }
    }
    return max;
}

//Llamadas a las funciones anteriores
...
float lista[n];
leerVector(lista,n);
float suma = sumaVector(lista,n);
printf("Maximo vector: %f\n",lista[maximoVector(lista,n)]);
```

Uso de arrays en subprogramas (VI)

- Una función también puede devolver un dato de tipo array.
  - Una función que inicializa todos los elementos de un vector de enteros a un valor inicial.

```
vector función InicializarVector(valor real : valorInicial; valor entero : n)
var
   entero : i
   vector : v
inicio
   desde i ← 1 hasta n hacer
      v[i] ← valorInicial
   fin_desde
   devolver(v)
fin_función

//La llamada podría ser
var
   vector : lista
...
   lista ← InicializarVector(0, numElementos)
```

Ejemplo 5.2

Implementar una función que devuelva la desviación estándar de una lista de elementos reales. La desviación estándar se calcula mediante la siguiente expresión:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

- ***Análisis del problema***

  Se supone que el programa principal tiene las siguientes declaraciones

```
const
   numElementos = 1000
tipos
   array[1..numElementos] de real = vector
```

Ejemplo 5.2 (II)

También puede resultar interesante implementar una función `MediaVector` que calcule la media de los `n` primeros elementos de un dato de tipo vector.

```
real función MediaVector(valor vector : v; valor entero : n)
var
   entero : i
   real : media
inicio
   media ← 0
   desde i ← 1 hasta n hacer
      media ← media + v[i]
   fin_desde
   devolver(media / n)
fin_función

//Una función alternativa
real función MediaVector(valor vector : v; valor entero : n)
inicio
   devolver(SumaVector(v,n) / n)
fin_función
```

Ejemplo 5.2 (III)

```
real función DesviaciónEstándar(valor vector : v; valor entero : n)
var
   entero : i
   real : suma, media
inicio
   suma ← 0
   media ← MediaVector(v,n)
   desde i ← 1 hasta n hacer
      suma ← suma + (v[i] - media) ** 2
   fin_desde
   devolver(suma / (n-1) ** 0.5)
fin_función
```

Ejemplo 5.2 (IV)

```
//Se suponen hechas las siguientes declaraciones...
#define NUM_ELEMENTOS  1000
typedef float Vector[NUM_ELEMENTOS];  //Define el tipo de datos Vector

float desviacionEstandar(Vector v, int n){
    int i;
    float suma=0, media;

    media = mediaVector(v,n);
    for(i=0; i<n; i++)
           suma = suma + pow(v[i] - media, 2);

    return sqrt(suma);
}
float sumaVector(Vector v, int n){
   float  suma = 0;
   int i;

   for(i=0; i<n;i++)
       suma += v[i];

   return suma;
}
float mediaVector(Vector v, int n){
    return sumaVector(v,n) / n;
}
```

Actualización de un vector

- Supone modificar su contenido sin cambiar la organización interna del mismo.
- Implica las siguientes operaciones:
  - Cambiar un elemento determinado por otro.
  - Añadir un elemento al final de la lista.
  - Insertar un elemento en la posición n de la lista.
  - Eliminar un elemento de la lista.
- Se supone la existencia de un vector con un número máximo de elementos de los cuales sólo están ocupados los n primeros.

```
const
   numElementos = 1000 //Número máximo de elementos del vector
tipos
   array[1..numElementos] de entero = vector
var
   vector : números
   //n es número de elementos ocupados del array.
   //Al comenzar el programa se supone que el vector tiene 0 elementos
   entero : n = 0
```

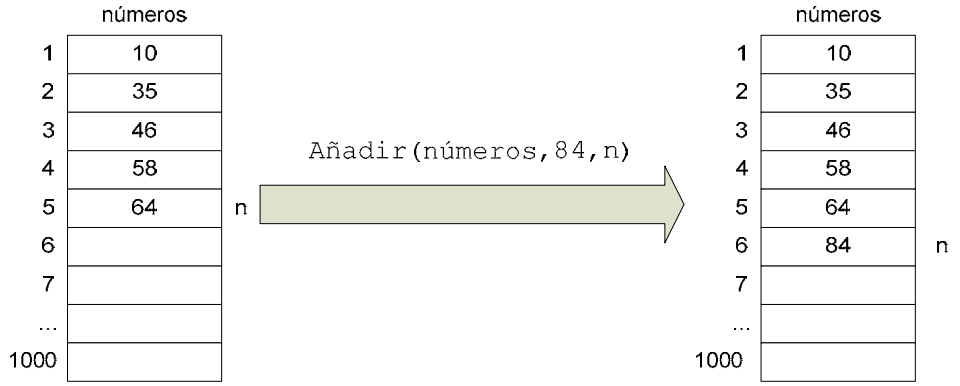
Actualización de un vector: Cambiar

- El procedimiento debería recibir el nuevo elemento `e` (del tipo base del vector), y la posición del elemento a cambiar (`p`) y el número de elementos ocupados (`n`)..
  - El programa debería controlar que la posición es menor que `n`.

```
procedimiento Cambiar(ref vector : v; valor entero: e; ref entero p,n)
inicio
   si p <= n entonces
      v[p] ← e
   si_no
      //No existe el elemento p
   fin_si
fin_procedimiento
```

Actualización de un vector: Añadir

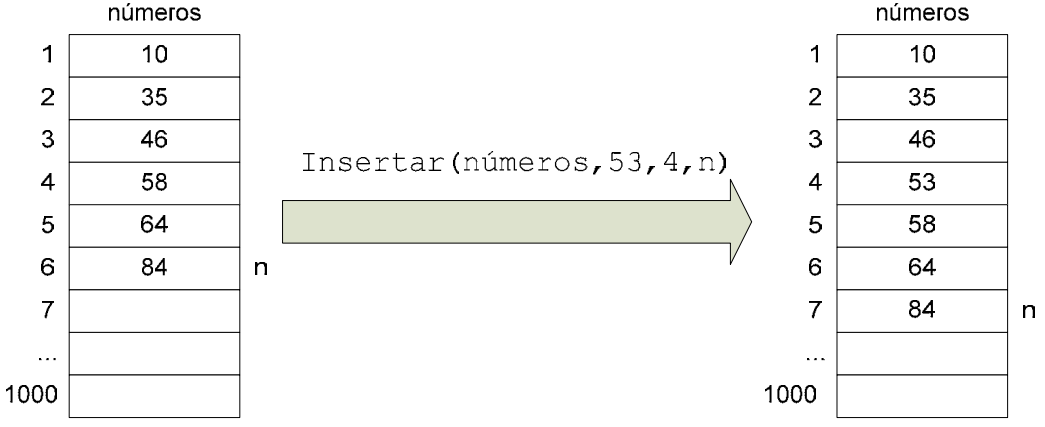
- Habrá que comprobar si el número de elementos ocupados es menor que `n`.
- Será necesario incrementar en número de elementos al finalizar el proceso.
- En el nuevo valor de `n` se coloca el elemento a añadir




```
procedimiento Añadir(ref vector : v; valor entero: dato; ref entero: n)
inicio
   si n < numElementos entonces
      n ← n + 1
      v[n] ← dato
   fin_si
fin_procedimiento
```

Actualización de un vector: Insertar

- Comprobar si hay elementos libres (`n < numElementos`) y que la posición donde se va a insertar (`p`) sea menor o igual que el número de elementos del `vector + 1`.
- Habrá que desplazar todos los elementos situados después de `p` una posición.
- Insertar el nuevo elemento en la posición `p`.
- Incrementar el numero de elementos ocupados en 1.

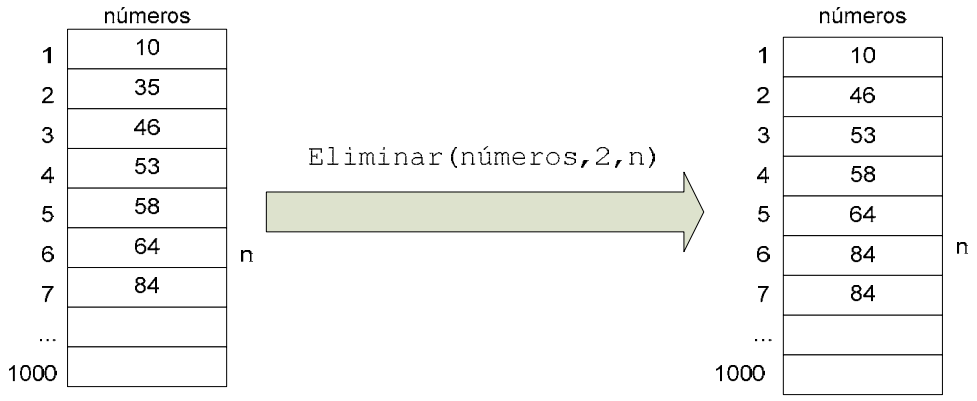


Actualización de un vector: Insertar (II)

```
procedimiento Insertar(ref vector : v; valor entero: dato; valor entero : p;
                       ref entero: n)
var
   entero : i
inicio
   si (n < numElementos) y (p <= n+1) entonces
      desde i ← n hasta p incremento - 1 hacer
         v[i+1] ← v[i]
      fin_desde
      v[p] ← dato
      n ← n + 1
   fin_si
fin_procedimiento
```

Actualización de un vector: Eliminar

- La posición del elemento a eliminar (`p`) no puede sobrepasar el número de elementos ocupados (`n`).
- Desplazar todos los elementos que se encuentren después de `p` una posición.
- Decrementar el número de elementos ocupados.



- El último elemento del vector aparece repetido.
  - Cualquier elemento que se encuentre después de n no existe.

Actualización de un vector: Eliminar (II)

```
procedimiento Eliminar(ref vector : v; valor entero : p;
                       ref entero: n)
var
   entero : i
inicio
   si p <= n entonces
      desde i ← p hasta n-1 hacer
         v[i] ← v[i+1]
      fin_desde
      n ← n - 1
   fin_si
fin_procedimiento
```

Actualización de un vector:
Ejemplo en C

```
void anadir(vector v,tipoDato dato,int *n){
     if(*n<NUM_ELEM){
         v[*n] = dato;
         *n = *n + 1;
     }
     else{
         /*Error array lleno */
     }
}

void cambiar(vector v,tipoDato dato,int pos, int n){
     if(pos < n){
        v[pos] = dato;
     }
     else{
        /* no existe el elemento pos */
     }
}
```

Actualización de un vector:
Ejemplo en C (II)

```
void insertar(vector v,tipoDato dato,int pos, int *n){
     if(*n < NUM_ELEM && pos <= *n){
        int i;
        for(i=*n-1;i>=pos;i--){
            v[i+1] = v[i];
        }
        v[pos] = dato;
        *n = *n + 1;
     }
}
void eliminar(vector v,int pos, int *n){
     if(pos < *n){
        int i;
        for(i=pos;i<*n-1;i++)
           v[i] = v[i+1];
        *n = *n -1;
     }
}
```

Actualización de un vector:
Ejemplo en C (III)

```
//Ejemplo de programa principal
#include <stdio.h>
#define NUM_ELEM 100
typedef int vector[NUM_ELEM];
typedef int tipoDato;

void anadir(vector v,tipoDato dato,int *n);
void cambiar(vector v,tipoDato dato,int pos, int n);
void insertar(vector v,tipoDato dato,int pos, int *n);
void eliminar(vector v,int pos, int *n);

int main(void){
    int i, n=0;
    vector numeros;
    tipoDato dato;
    anadir(numeros,20,&n);
    anadir(numeros,25,&n);
    anadir(numeros,50,&n);
    anadir(numeros,80,&n);
    cambiar(numeros, 30, 1, n);
    insertar(numeros,1,0,&n);
    insertar(numeros,100,5,&n);
    insertar(numeros,25,2,&n);
    eliminar(numeros,6,&n);

    for(i=0;i<n;i++)
        printf("%i\n",numeros[i]);

}
```

Ejercicios con vectores

1. Diseñe un algoritmo que rellene un vector con los primeros 100 números enteros y genere otro con los cuadrados de esos 100 números. Al final del algoritmo se deberá sacar un listado en el que aparezcan ambos datos.
2. Diseñe un algoritmo que rellene un vector de N elementos con números enteros aleatorios entre 1 y 1000 y obtenga la suma de los números pares y de los números impares.
3. Realice las siguientes operaciones:
   - Declare un tipo de dato array para almacenar las notas de n alumnos.
   - Diseñe un subprograma que permita leer las notas de n alumnos.
   - Diseñe un subprograma que permita escribir las notas de n alumnos.
   - Diseñe un subprograma que permita calcular la nota media de n alumnos.
   - Diseñe un programa principal que utilice los subprogramas anteriores.

Ejercicios con vectores (II)

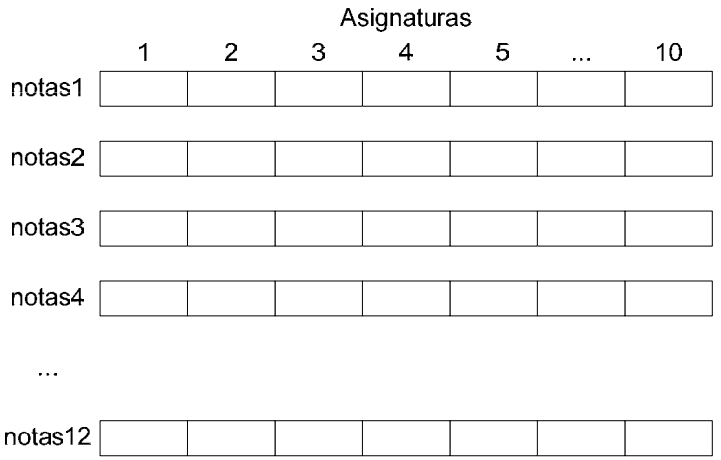
4. Diseñe una función que obtenga el producto de un array de una dimensión por un escalar. El vector, el escalar y el número de elementos del vector se pasarán como argumentos.
   - El producto de un vector a por un escalar k es otro vector y se calcula con:

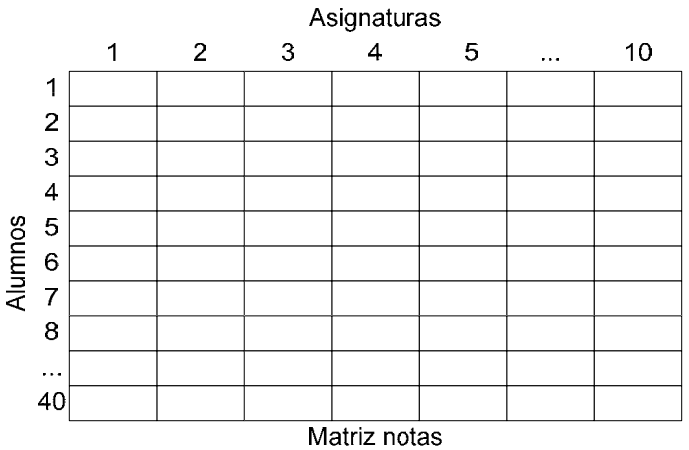
$$k \cdot (a_1 \quad a_2 \quad \ldots \quad a_n) = (k \cdot a_1 \quad k \cdot a_2 \quad \ldots \quad k \cdot a_n)$$

5. Diseñe un subprograma que permita eliminar el elemento más pequeño de un vector.
6. Diseñe una función lógica que permita comparar dos vectores del mismo tipo.

Arrays de dos dimensiones: matrices

- Se utilizan para representar información en forma tabular o la representación de un plano.
- Repetición de un vector un número determinado de veces.
  - Se le puede considerar un vector de vectores.





Arrays de dos dimensiones:
Matrices (II)

- Declaración:

```
var
   array[limInf1..limSup1,limInf2..limSup2] de tipoBase : nombreArray
```

  - Será necesario indicar cómo se hace referencia a cada elemento.
    - ✓ Cada elemento se identifica con dos índices.
    - ✓ Hay que indicar el límite inferior y el límite superior de cada una de las dimensiones.
  - La declaración de la matriz `notas` se hará:

```
var
   array[1..40,1..10] de real : notas
...
```

- En C la declaración se hará:

```
tipoDato nombreArray[numElem1D][numElem2D]
```

  - En realidad sería como un array de arrays: un array en el que cada uno de los elementos es a su vez un array.
  - Hay que indicar el número de elementos de cada una de las dos dimensiones.
    - ✓ Los índices irán entre 0 y el número de elementos – 1.

```
float notas[39][9]
```

Arrays de dos dimensiones:
Matrices (III)

- Operaciones con una matriz.
  - La única operación permitida con la estructura de datos es la asignación.
    - ✓ Es posible asignar un array a otro siempre que sea del mismo tipo.
      - Mismo número de elementos, mismos límites y mismo tipo de datos.
- Acceso a los elementos de una matriz.
  - Cada elemento se identifica por su posición en la primera y en la segunda dimensión.
  - La nota del tercer alumno en la cuarta asignatura será la información contenida en el elemento `notas[3,4]`.
    - ✓ Será un elemento de tipo real y con él se podrán hacer todas las operaciones que se puedan realizar con un dato de tipo real.
  - En C, la referencia al tercer alumno de la cuarta asignatura sería mediante `notas[2][3]`.

Almacenamiento de matrices en memoria

- Los conceptos de fila y columna son conceptos arbitrarios.
  - Se utiliza la palabra fila para indicar la posición de un elemento en la primera dimensión y columna para indicar la posición en la segunda dimensión, pero:
  - Todos los elementos ocupan posiciones consecutivas en memoria.
- A la hora de almacenar los elementos se utilizan dos métodos:
  - Fila mayor: se almacenan de forma correlativa todos los elementos de la primera fila, a continuación los de la segunda, etc.
    - ✓ Para la referencia al elemento i,j de un array de M x N elementos:
      - Posición = N * (i – 1) + j.
  - Columna mayor: se almacenan de forma correlativa todos los elementos de la primera columna, a continuación los de la segunda, etc.
    - ✓ Para la referencia al elemento i,j de un array de M x N elementos:
      - Posición = M * (j – 1) + i.

Almacenamiento de matrices en memoria (II)

Matriz
(array de M * N elementos)

1,1	1,2	1,3	1,4
2,1	2,2	3,3	4,4

Notas
(array de 2 * 4 elementos)

5.0	6.3	4.1	2.5
6.5	8.3	7.0	3.8

El dato 7.0 (elemento 2,3)
estará en la posición

Almacenamiento en memoria por fila mayor

1,1	1,2	1,3	1,4	2,1	2,2
2,3	2,4				

Almacenamiento en memoria por fila mayor

5.0	6.3	4.1	2.5	6.5	8.3
7.0	3.8				

N x (i-1) + j
4 x (2-1) + 3 = 7

Almacenamiento en memoria por columna mayor

1,1	2,1	1,2	2,2	1,3	2,3
1,4	2,4				

Almacenamiento en memoria por columna mayor

5.0	6.5	6.3	8.3	4.1	7.0
2.5	3.8				

M x (j-1) + **i**
2 x (3-1) + 2 = 6

Recorrido de una matriz

- Para recorrer todos los elementos de una matriz será necesario utilizar dos variables que tomen los valores de los índices de la primera y la segunda dimensión.
  - Dos bucles anidados.
- Para leer todos los elementos de la matriz `notas`:

```
desde i ← 1 hasta 40 hacer
   desde j ← 1 hasta 10 hacer
      leer(notas[i,j])
   fin_desde
fin_desde
```

- Un vector sólo admite un tipo de recorrido.
  - En una matriz se pueden recorrer todos los elementos de la primera fila, de la segunda, etc.: **recorrido por filas**.
  - También se pueden recorrer todos los elementos de la primera columna, de la segunda, etc.: **recorrido por columnas**.

Recorrido de una matriz (II)

- Recorrido por filas: hallar la nota media de cada alumno (de cada fila).
  - Es necesario sumar todos los elementos de la primera fila y hallar su media y realizar la misma operación con el resto de filas.
  - El bucle externo será el bucle de la primera dimensión.

```
desde i ← 1 hasta 40 hacer
   //Este fragmento del algoritmo se ejecuta una vez por alumno
   //antes de comenzar a procesar sus notas
   //Si se desea obtener una media por alumno, será necesario
   //inicializar la media tantas veces como alumnos
   media ← 0
   desde j ← 1 hasta 10 hacer
      media ← media + notas[i,j]
   fin_desde
   //Este fragmento del algoritmo se ejecuta una vez por alumno
   //después de procesar sus notas
   //Si se desea obtener una media por alumno, será necesario
   //dividir la media entre el número de alumnos en este punto
   media ← media / 10
   escribir(media)
fin_desde
```

Recorrido de una matriz (III)

- Recorrido por columnas: hallar la nota media de cada asignatura (de cada columna).
  - Es necesario sumar todos los elementos de la primera columnas y hallar su media y realizar la misma operación con el resto de columnas.
  - El bucle externo será el bucle de la segunda dimensión.

```
desde i ← 1 hasta 10 hacer
   //Este fragmento del algoritmo se ejecuta una vez por asignatura
   //antes de comenzar a procesar sus notas
   //Si se desea obtener una media por asignatura, será necesario
   //inicializar la media tantas veces como notas
   media ← 0
   desde j ← 1 hasta 40 hacer
      media ← media + notas[j,i]
   fin_desde
   //Este fragmento del algoritmo se ejecuta una vez por asignatura
   //después de procesar sus notas
   //Si se desea obtener una media por asignatura, será necesario
   //dividir la media entre el número de alumnos en este punto
   media ← media / 40
   escribir(media)
fin_desde
```

Recorrido de una matriz
Implementación en C

```
//Leer todos los elementos
for(i=0;i<m;i++){
    printf("Notas del alumno %i...\n",i+1);
    for(j=0;j<n;j++){
           printf("Asignatura %i:",j+1);
           scanf("%f",&notas[i][j]);
    }
}
//Calcular la media por alumno: recorrido por filas
for(int i = 0; i<40;i++){
    media =0;
    for(int j=0; j<10;j++)
           media += notas[i][j];
    media = media / 10;
    printf("Media alumno %i: %.1f\n",i+1,media);
}
//Calcular la media por asignatura: recorrido por columnas
for(int j = 0; j<10;j++){
    media =0;
    for(int i=0; i<40;i++)
           media += notas[i][j];
    media = media / 40;
    printf("Media asignatura %i: %.1f\n",j+1,media);
}
```

Ejemplo 5.3

Implementar subprograma que devuelva la matriz resultante de multiplicar dos matrices. Sea una matriz A de M x N elementos y una matriz B de O x P elementos para poder realizar el producto es necesario que el número de columnas de la primera (N) sea igual al número de filas de la segunda (O) y la matriz producto resultante será una matriz de M x P elementos.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \\ p_{31} & p_{11} \\ p_{41} & p_{42} \end{pmatrix}$$

y cada elemento i,j de la matriz producto se calculará mediante la expresión:

$$p_{ij} = \sum_{K=1}^{n} a_{ik} \cdot b_{kj}$$

Se consideran las siguientes declaraciones globales

```
const
   numFilas = 100
   numColumnas = 100
tipos
   array[1..numFilas,1..NumColumnas] de entero = tabla
```

Ejemplo 5.3 (II)

```
//Se pasan como argumentos las filas y columnas de la matriz a
//(m y n) y las filas y columnas de la matriz b(o y p).
//Se supone que n es igual a o
procedimiento ProductoMatrices(valor tabla : a,b; ref tabla: prod;
                               valor entero : m,n,o,p)
var
   entero : i,k,j
inicio
   //La matriz producto tiene tantas filas como a
   desde i ← 1 hasta m hacer
      //y tantas columnas como b
      desde j ← 1 hasta p hacer
         //Cada elemento de la matriz producto es un acumulador
         //por lo que habrá que inicializarlo una vez por elemento
         prod[i,j] ← 0
         desde k ← 1 hasta n hacer
            prod[i,j] ← prod[i,j] + a[i,k] * b [k,j]
         fin_desde
      fin_desde
   fin_desde
fin_función
```

Ejemplo 5.3 (III)

```
//Se suponen hechas las siguientes declaraciones:
#define NUM_FILAS 100
#define NUM_COLUMNAS 100

typedef int Tabla[NUM_FILAS][NUM_COLUMNAS];

void productoMatrices(Tabla a, Tabla b, Tabla prod,int m,int n,int o, int p){
    //Se recorre la matriz producto
    //La matriz producto tiene tantas filas como a, es decir m
    for(int i=0;i<m;i++)
          //...y tantas columnas como b, es decir p
          for(int j=0;j<p;j++){
                    //Se calcula el elemento i,j de la matriz producto
                    prod[i][j] = 0;
                    //Se recorre la fila k de a y la columna k de j
              for(int k=0;k<n;k++)
                        prod[i][j] = prod[i][j] + a[i][k]*b[k][j];
          }
}
```

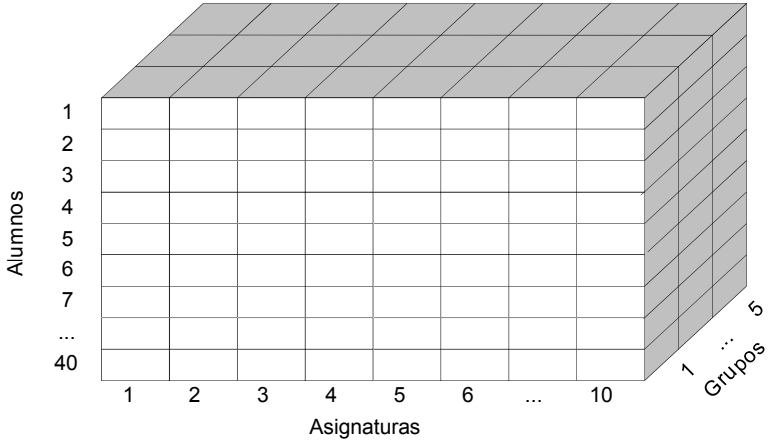
Arrays multidimensionales

- En la declaración aparece un límite inferior y otro superior por cada dimensión.
- Cada elemento se referencia por n índices (uno por cada dimensión).
- Su tratamiento debe un bucle anidado por cada dimensión.
  - Para recorrer una matriz tridimensional son necesarios tres bucles anidados.
- N tipos de recorridos.

Arrays multidimensionales (II)

- Se desea almacenar la información de 10 notas de cada uno de los 40 alumnos de cada uno de los 3 grupos.

```
var
   array[1..40,1..10, 1..3] de real : notas
...
```



- La referencia a una nota precisa de tres índices.
  - notas[10,5,2] hará referencia a la quinta nota del décimo alumno del segundo grupo.

Arrays multidimensionales (III)

- El recorrido precisa de tres bucles anidados.
  - Para calcular la nota media de cada uno de los grupos:

```
desde i ← 1 hasta 3 hacer
   //Este fragmento del algoritmo se ejecuta una vez por grupo
   //antes de comenzar a procesar sus notas
   //Si se desea obtener una media por grupo, será necesario
   //inicializar la media tantas veces como grupos
   media ← 0
   //A partir de aquí habrá que recorrer las notas de todos
   //los alumnos de todas las asignaturas dentro del grupo i
   desde j ← 1 hasta 40 hacer
      desde k ← 1 hasta 10 hacer
         media ← media + notas[j,k,i]
      fin_desde
   fin_desde
   //Este fragmento del algoritmo se ejecuta una vez por grupo
   //después de procesar sus notas
   //Si se desea obtener una media por grupo, será necesario
   //dividir la media entre el número de notas total del grupo
   media ← media / (40 * 10)
   escribir(media)
fin_desde
```

Arrays multidimensionales (IV)

- Ejemplos en C

```
float notas[40][10][3];

//Escribir las notas medias de cada grupo
for(int k=0;k<3;k++){
    float media =0;
    for(int i=0;i<40;i++)
         for(int j=0;j<10;j++)
                  media += notas[i][j][k];
    media = media / 12;
    printf("Nota media del grupo %i: %.2f\n",k+1,media);
}

//Calcular la nota media de cada asignatura
for(int j=0;j<10;j++){
    float media =0;
    for(int i=0;i<40;i++)
         for(int k=0;k<3;k++)
                  media += notas[i][j][k];
    media = media / 120;
    printf("Nota media de la asignatura %i: %.2f\n",j+1,media);
}
```

Ejercicios con matrices

7. Diseñe un procedimiento que devuelva una matriz con la suma de dos matrices con las mismas dimensiones.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{pmatrix}$$

8. Diseñe un algoritmo que permita obtener los puntos de silla de un array de M x N elementos.
   - El punto de silla es el elemento que es al mismo tiempo máximo de fila y mínimo de columna.
9. En una tabla de N x 24 elementos se tienen almacenadas las temperaturas de cada hora de un mes de N días. Diseñe un algoritmo que genere un vector con las temperaturas medias diarias y que devuelva la temperatura media mensual.

Ejercicios con matrices (II)

10. Diseñe un algoritmo que permita obtener un mapa con las estrellas de una zona del espacio. En cada elemento se almacena la intensidad luminosa de un punto. Hay una estrella cuando la intensidad media del punto con todos los colindantes es mayor que la constante K.
11. Diseñe una función que devuelva un valor lógico que indique si una matriz de N x N elementos es un cuadrado mágico. Una matriz es un cuadrado mágico si las suma de cada una de las columnas, de cada una de las filas y de las dos diagonales dan el mismo valor. La tabla que aparece a continuación es un cuadrado mágico.

8	1	6
3	5	7
4	9	2

Ejercicios con matrices (III)

12. En un array de n x m posiciones se almacenan aleatoriamente caracteres X y O. Diseñe las estructuras de datos el programa principal y los módulos necesarios que permitan:
    - Indicar por cada fila y columna la cantidad máxima de caracteres X contiguos situados en cada una de ellas.
    - Indicar en que fila y en qué columna se encuentra la cantidad máxima de caracteres X contiguos.

X	X	O	X	X	2
X	O	O	O	O	1
O	X	O	X	X	3
O	X	X	X	X	4
X	X	X	O	O	3
X	X	O	O	X	2
X	O	X	X	X	3

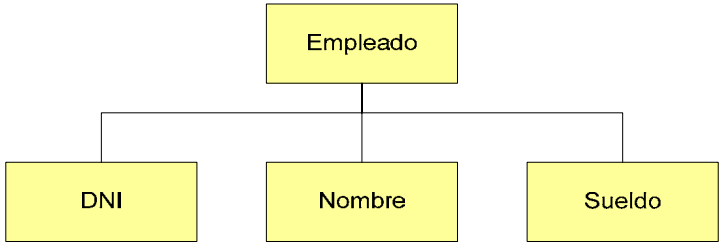
El mayor número de X está en la columna 2

El mayor número de X está en la fila 4

3	4	2	2	2

Registros

- Los registros (o estructuras) son datos estructurados formados por elementos heterogéneos y lógicamente relacionados.
  - Estructura jerárquica con información de distinto tipo referente a un mismo objeto.
    - ✓ A cada componente de un registro se le denomina **campo**.



Declaración de registros

- Se realiza enumerando cada uno de los componentes que lo forman.

```
registro : nombreRegistro
   tipoDeDato : nombreCampo
   tipoDeDato : nombreCampo
   ...
fin_registro
```

  - *nombreRegistro* **es un identificador que se utilizará para hacer referencia a la estructura.**
    - ✓ Se pueden declarar varias variables del mismo tipo.
  - Por cada campo hay que incluir su tipo de dato y su nombre.
    - ✓ *tipoDeDato* será cualquier dato simple o estructurado.
    - ✓ *nombreCampo* será un identificador válido.

```
registro : empleado
   cadena : DNI, nombre
   real : sueldo
fin_registro
```

Declaración de registros (II)

- El tipo de dato de los campos puede ser un tipo de dato estructurado.
  - Registros anidados.

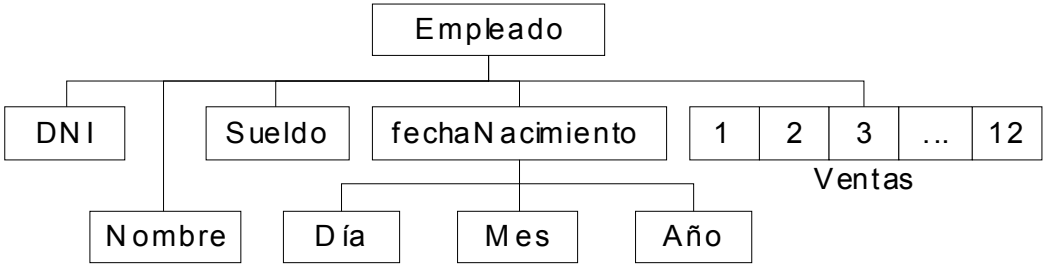
```
registro : empleado
   cadena : DNI, nombre
   real : sueldo
   registro : fechaContrato
      entero : día, mes, año
   fin_registro
fin_registro
```

- Existe una forma alternativa para declarar registros anidados…

```
tipos
   registro = fecha
      entero : día, mes, año
   fin_registro
registro : empleado
   cadena : DNI, nombre
   real : sueldo
   fecha : fechaContrato
fin_registro
```

Declaración de registros (III)

- Los campos de un registro también pueden ser de tipo array.




```
registro : empleado
   cadena : DNI, nombre
   real : sueldo
   fecha : fechaContrato
   vector : ventas
fin_registro
```

```
registro : empleado
   cadena : DNI, nombre
   real : sueldo
   fecha : fechaContrato
   array[1..12] de real : ventas
fin_registro
```

Declaración de registros (IV)

- En C se utiliza la palabra reservada `struct` (los registros son estructuras).

```
struct [nombreRegistro]{
   tipoDato campo;
   ...
} [nombreVariable]
```

  - *nombreRegistro* es un identificador que se utilizará para hacer referencia a la estructura.
    - ✓ Si se declara una variable de tipo estructura se puede omitir el nombre de registro y poner directamente el nombre de la variable.
    - ✓ También se puede omitir el nombre de la variable.
      - En ese caso simplemente se creará un tipo de dato. Posteriormente habrá que declarar una variable de ese tipo.

```
struct{
   char *dni, *nombre;
   float sueldo;
}e;   //Declara la variable e de tipo estructura
struct Empleado{
   char *dni, *nombre;
   float sueldo;
}e;   //Declara la estructura Empleado y una variable de ese tipo
struct empleado{
   char *dni, *nombre;
   float sueldo;
 };   //Declara la estructura Empleado
struct Empleado e; //Declara la variable e de tipo Empleado
```

Declaración de registros (V)

- Registros anidados.

```
struct Empleado {
    char *dni, *nombre;
    float sueldo;
    struct {
       int d,m,a;
    } fechaContrato;
};
```

- Existe una forma alternativa para declarar registros anidados…

```
struct Fecha{
   int d,m,a;
};
struct Empleado {
   char *dni, *nombre;
   float sueldo;
   struct Fecha fechaContrato;
};
```

Declaración de registros (VI)

- En C también los campos pueden ser arrays

```
struct Empleado {
    char *dni, *nombre;
    float sueldo;
    struct Fecha fechaContrato;
    float ventas[12];
};
```

```
#define NUM_ELEMENTOS 12
typedef float Vector[NUM_ELEMENTOS];
...
struct Empleado {
   char *dni, *nombre;
   float sueldo;
   struct Fecha fechaContrato;
   Vector ventas; //Previamente se ha declarado el tipo de dato Vector
};
```

Referencia a los elementos de un registro

- Al declarar una variable de tipo registro, el compilador reserva espacio para cada uno de sus componentes.
  - Al tratarse de datos de distinto tipo, el compilador no puede utilizar el mismo tipo de acceso que utiliza para los elementos de un array.
- La referencia se realiza mediante el selector punto (.).

  nombreDeRegistro.nombreDeCampo
  - Para hacer referencia al nombre de empleado:

    `Empleado.Nombre`
  - Para hacer referencia a un registro anidado, por ejemplo el año del contrato:

    `Empleado.fechaContrato.año`
  - Para hacer referencia a las ventas obtenidas en el mes de mayo:

    `Empleado.ventas[5]`

Operaciones con registros

- Con una estructura de tipo registro sólo se podrá realizar la operación de asignación.
  - Se podrá asignar un registro a otro del mismo tipo.
    - ✓ Dos registros son del mismo tipo si tienen los mismos campos y del mismo tipo de dato.
- Con los elementos individuales de un registro, se podrán realizar las mismas operaciones que se realizan con datos de ese tipo de dato.

```
//Concatena la cadena Pérez al nombre del empleado
empleado.nombre ← empleado.nombre & 'Pérez'
//Lee por teclado el sueldo del empleado
leer(empleado.sueldo)
//escribe la desviación estándar de las ventas del empleado
//utilizando la función del ejemplo 5.2
escribir(DesviaciónEstándar(empleado.ventas,12))
```

- Los registros también se pueden pasar como argumentos a los procedimientos.

Operaciones con registros (II)

```
tipos
   registro = fecha
      entero : día, mes, año
   fin_registro
   registro : empleado
      cadena : DNI, nombre
      real : sueldo
      fecha : fechaContrato
      array[1..12] de real : ventas
      //Al registro anterior se le ha añadido el campo totalVentas
      //que contendrá la suma de las ventas anuales
      real : totalVentas
   fin_registro
...
```

Operaciones con registros (III)

```
procedimiento LeerRegistro(ref empleado : e)
var
   entero : i
inicio
   leer(e.DNI)
   leer(e.nombre)
   leer(e.sueldo)
   leer(e.fechaContrato.día)
   leer(e.fechaContrato.mes)
   leer(e.fechaContrato.año)
   desde i ← 1 hasta 12 hacer
      leer(e.ventas[i])
   fin_desde
fin_procedimiento
//Calcula el total de las ventas anuales del empleado
procedimiento CalcularVentasAnualesEmpleado(ref empleado : e)
var
   entero : i
inicio
   e.totalVentas ← 0 //Se podría haber hecho e.totalVentas=SumaVector(e.ventas,12)
   desde i ← 1 hasta 12 hacer
      e.totalVentas ← e.totalVentas + e.ventas[i]
   fin_desde
fin_procedimiento
```

Operaciones con registros (IV)

- Implementación en C.
  - Tipos de datos utilizados.

```
#define NUM_ELEMENTOS 12

typedef float Vector[NUM_ELEMENTOS];

struct Fecha{
       int d,m,a;
};

struct estructuraEmpleado {
   char *dni;
   char *nombre;
   float sueldo;
   struct Fecha fechaContrato;
   Vector ventas;
   float totalVentas;
};

typedef struct estructuraEmpleado Empleado;
```

- Cuando se pasa una estructura como argumento por referencia en C, se trabaja con punteros a la estructura.
  - Para hacer referencia a los miembros de un puntero a una estructura se utiliza el operador `->`, por ejemplo `e->dni`.

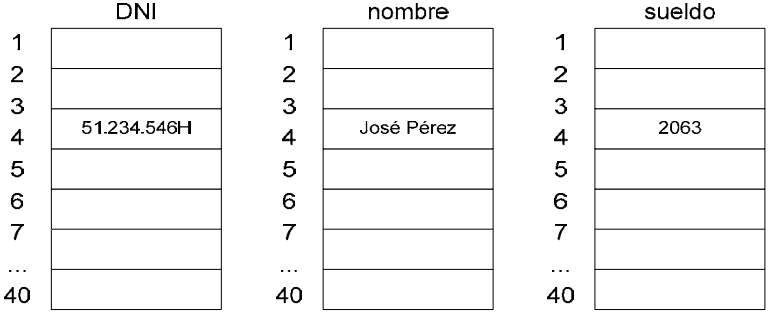
Operaciones con registros (V)

```
//La estructura Empleado se pasa por referencia
void leerEmpleado(Empleado *e){
     printf("DNI: ");
     scanf("%s",e->dni);  //Se hace referencia a los miembros de la estructura con ->
     printf("Nombre: ");
     scanf("%s",e->nombre);
     printf("Sueldo: ");
     scanf("%f",&e->sueldo);
     int i;
     for(i=0;i<12;i++){
        printf("Ventas del mes %i:",i+1);
        scanf("%f",&e->ventas[i]);
     }
     e->totalVentas = sumaVector(e->ventas,12);
}

//La estructura Empelado se pasa por valor
void escribirEmpleado(Empleado e){
     printf("DNI: %s\n",e.dni); //Se utiliza el punto para hacer referencia a los miembros
     printf("Nombre: %s\n",e.nombre);
     printf("Sueldo: %.2f\n",e.sueldo);
     int i;
     for(i=0;i<12;i++){
        printf("Ventas del mes %i: %.2f\n",i+1,e.ventas[i]);
     }
     printf("Total ventas: %f\n",e.totalVentas);
}
```

Arrays de registros

- ¿Cómo se pueden agrupar varios elementos con información heterogénea?
- Arrays paralelos.
  - Dos o más arrays que utilizan el mismo subíndice para referirse a términos homólogos se llaman arrays paralelos.




```
var
   array[1..40] de cadena : DNI
   array[1..40] de cadena : nombre
   array[1..40] de real : sueldo
```

Arrays de registros (II)

- El tipo base de un array también puede ser un registro:
  - Arrays de registros.

```
var
   array[1..40] de empleado : vendedores
```

				fechaContrato			ventas				
	DNI	nombre	sueldo	día	mes	año	1	2	...	12	totalVentas
1											
2											
3	51.234.546H	José Pérez	2063	31	5	1996	...	...	...	...	
4											
5											
6											
7											
...											
40											

Arrays de registros (III)

- A los elementos de un array de registros se accede con la notación ya explicada.

```
//Leer por teclado el DNI del cuarto empleado
leer(vendedores[4].DNI)
//Escribir el año del contrato del cuarto empleado
escribir(vendedores[4].fechaContrato.año)
//Sumar 1000 euros a las ventas de diciembre del cuarto empleado
vendedores[5].ventas[12] ← vendedores[5].ventas[12] + 1000
//Calcular el total de las ventas de los 40 empleados, utilizando
//el procedimiento CalcularVentasAnualesEmpleado ya definido
desde i ← 1 hasta 40 hacer
   CalcularVentasAnualesEmpleado(vendedores[i])
fin_desde
```

Ejemplo 5.4

Un array de N registros almacena los datos de una serie de productos. El array ya está cargado con los datos y por cada producto se guarda su Código, su descripción y su stock. De forma paralela se almacenan en una tabla de Nx7 elementos las cantidades vendidas de cada producto a lo largo de la semana, de manera que en el elemento i,j de la tabla aparecerán las unidades vendidas del producto i en el día j. Por ejemplo, del Bolígrafo Bic Azul (posición 2 del array de registros) existen en el almacén 325 unidades, y el miércoles (fila 2, columna 3 de la tabla) se vendieron 34 unidades.

PRODUCTOS

	Código	Descripción	Stock
1	...	...	...
2	30134	Bolígrafo Bic Azul	325
3	...	...	...
4	...	...	...
...	...	...	...
N	...	...	...

VENTAS

	1	2	3	4	5	6	7
1	...	...	...	...	...	...	...
2	10	12	34	31	54	10	1
3	...	...	...	...	...	...	...
4	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
N	...	...	...	...	...	...	...

Ejemplo 5.4 (II)

- Declarar de todas las estructuras de datos necesarias para realizar las acciones de los puntos siguientes.

```
const
   N = … //Número de productos a manejar
tipos
   registro = Producto
      cadena : código, descripción
      entero : stock
   fin_registro
   array[0..N] de Producto = Productos
   array[1..N,1..7] de entero = Ventas
```

Ejemplo 5.4 (III)

- Diseñar un módulo que permita leer las ventas de cada producto, teniendo en cuenta que la entrada de datos se hará por día de la semana (es decir, primero se leerán las ventas del lunes del producto 1, después las ventas del lunes del producto 2, después las ventas del lunes del producto 3, etc.)

```
procedimiento LeerVentas(ref ventas : tabla; valor entero : n)
var
   entero : i,j
inicio
   desde i ← 1 hasta 7 hacer
   //Leer todas las ventas del día i
      desde j ← 1 hasta n hacer
         leer(tabla[j,i])
      fin_desde
   fin_desde
  fin_procedimiento
fin_procedimiento
```

Ejemplo 5.4 (IV)

- Diseñar un módulo que actualice el stock de cada producto, restando al stock actual las ventas realizadas a lo largo de la semana.

```
procedimiento ActualizarVentas(valor ventas:tabla;ref Productos:p)
var
   entero : i,j
inicio
   desde i ← 1 hasta n hacer
      desde j ← 1 hasta 7 hacer
         p[i].stock ← p[i].stock – tabla[i,j]
      fin_desde
   fin_desde
fin_procedimiento
```

Ejemplo 5.4 (V)

- Estructuras de datos necesarias en C.

```
#define NUM_ELEMENTOS 100


struct estructuraProducto {
   char *codigo;
   char *descripcion;
   float stock;
};

typedef struct estructuraProducto Producto;
typedef Producto Productos[NUM_ELEMENTOS];
typedef float Ventas[NUM_ELEMENTOS][7];
```

Ejemplo 5.4 (VI)

```
void leerVentas(Ventas v,int n){
     int i,j;
     for(j=0;j<7;j++){
        printf("Ventas del dia %i...\n",j+1);
        for(i=0;i<n;i++){
           printf("Ventas del producto %i:",i+1);
           scanf("%f",v[i][j]);
        }
     }
}

void actualizarVentas(Ventas v, Productos p,int n){
     int i,j;
     for(j=0;j<7;j++){
        for(i=0;i<n;i++){
          p[i].stock = p[i].stock - v[i][j];
        }
     }
}
```

Ejercicios

13. Se desea obtener la tabla de puntuación en una liga de futbolín en la que compiten 10 equipos a una sola vuelta. Los resultados se almacenarán en una tabla de dos dimensiones, de forma que cada elemento i,j de la tabla guarda los tantos que ha marcado el equipo i al equipo j en el encuentro. Por ejemplo, un 8 en la posición 2,6 significa que en el encuentro entre el equipo 2 y el equipo 6, el equipo 2 ha metido 8 tantos al equipo 6.

    Los nombres de los equipos y la puntuación obtenida por cada uno de ellos se almacenarán en un array de registros de 10 posiciones (una por equipo) que almacenará el nombre del equipo y la puntuación.

    - Declarar las estructuras de datos necesarias para realizar la aplicación.
    - Diseñe un módulo que permita introducir por teclado el nombre de los equipos y almacenarlo en el array de registros.
    - Diseñe un módulo que permita rellenar la tabla con los tantos obtenidos por los equipos.
    - Diseñe un módulo que permita obtener la puntuación de cada equipo (3 puntos por partido ganado, 1 por partido empatado y 0 por partido perdido).

Ejercicios (II)

14. La V Edición de la Vuelta Ciclista a los Campos Manchegos consta de 5 etapas en las que participan 120 ciclistas. Los resultados de cada etapa se almacenan en un array de 5 filas y 120 columnas en el que cada elemento contiene el tiempo en segundos que ha tardado el corredor en realizar la etapa. Si un corredor no ha finalizado una etapa aparecerá un 0 en el elemento correspondiente. Los corredores se almacenan en un array de 120 posiciones en el que se almacenará el dorsal del corredor y el tiempo total realizado.

Array con los tiempos

Número del participante

Etapas	1	2	3	4	…	120
1						
2						
3						
4						
5						

Array con los ciclistas

	Número	Tiempo
1		
2		
3		
4		
5		
6		
…		
120		

Ejercicios (III)

- Definir las estructuras de datos necesarias para realizar la aplicación.
- Diseñar un módulo que obtenga, a partir de un número de etapa que se pasará como argumento, el dorsal del ganador de la etapa.
- Diseñar una función que devuelva el número del ganador de la prueba y el tiempo total obtenido.

Ejercicios (IV)

15. Una empresa quiere almacenar en una tabla las ventas realizadas por sus vendedores a lo largo de la semana. La empresa cuenta con 5 vendedores y en cada elemento i,j de la tabla se almacenará las ventas realizadas por el vendedor i el día j (j valdrá 1 para el lunes, 2 para el martes, 3 para el miércoles, etc.). Los nombres de los vendedores y el total de ventas semanales se guardarán en un array de registros.

	Nombre vendedor	Total ventas semanales
1	...	
2	...	
3	...	
4	...	
5	...	

2342	3212	4543	1543	3425	5453	1232
3423	5434	4534	3454	4343	6787	4343
3432	4323	6565	7878	9899	6554	3234
5434	3432	4322	3432	3423	5445	3432
5454	4345	4344	3234	5455	5454	5445

Ejercicios (V)

- Realizar las declaraciones de las estructuras de datos necesarias para la aplicación.
- Codifique un módulo que permita introducir los nombres de los vendedores y las ventas realizadas por cada uno de ellos y almacenarlos en las estructuras de datos.
- Codifique un módulo que obtenga el total de ventas mensuales de cada vendedor.
- Codifique un módulo que obtenga en nombre del vendedor que más ventas ha realizado en un día de la semana.