

Programación 1

Tema 9

Vectores



Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza



Problema

- Para resolver un problema relativo al cambio climático, es necesario manejar la información de las temperaturas medias registradas de forma mensual durante un determinado año en una determinada localidad.
- Queremos calcular la media anual a partir de las medias mensuales.
- ¿Cómo podemos representar esta información?
¿Cómo podemos calcular esta media?

Una (muy mala) solución

```
/*  
 * Pre: Todas las temperaturas son mayores que -273,15 °C.  
 * Post: Devuelve la temperatura media anual a partir de las  
 *       temperaturas medias mensuales.  
 */  
double calcularTemperaturaMedia(  
    double temperaturaEnero, double temperaturaFebrero,  
    double temperaturaMarzo, double temperaturaAbril,  
    double temperaturaMayo, double temperaturaJunio,  
    double temperaturaJulio, double temperaturaAgosto,  
    double temperaturaSeptiembre, double temperaturaOctubre,  
    double temperaturaNoviembre, double temperaturaDiciembre) {  
    return (temperaturaEnero + temperaturaFebrero + temperaturaMarzo  
        + temperaturaAbril + temperaturaMayo + temperaturaJunio  
        + temperaturaJulio + temperaturaAgosto + temperaturaSeptiembre  
        + temperaturaOctubre + temperaturaNoviembre  
        + temperaturaDiciembre) / 12;  
}
```

Vectores o tablas

- ❑ Colección de un número concreto de datos de un mismo tipo
- ❑ Indexados por uno o más índices
- ❑ Operaciones disponibles:
 - Acceso a componentes
- ❑ Operaciones **no** disponibles:
 - ~~Asignación~~
 - ~~Comparación~~

Vectores

□ Sintaxis declaración

- `<declaraciónVector> ::=`
`<tipo> <identificador>`
`“[” <constante-entera> “]”`
`[“=” “{” <listaInicialización> “}”]`
`“;”`
- `<listaInicialización> ::=`
`{<dato> “,”}`

Vectores

□ Sintaxis utilización

- $\langle \text{componente-vector} \rangle ::= \langle \text{identificador} \rangle "[\langle \text{expresión} \rangle]"$
- $\langle \text{expresión} \rangle$ tiene que ser una expresión entera de resultado en el intervalo entre 0 (incluido) y la dimensión del vector (excluida)
- $\langle \text{componente-vector} \rangle$ puede utilizarse como cualquier variable del tipo base del vector.



Vector

```
const unsigned NUM_MESES = 12;
```



Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];
```




Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];
```

t

0	¿?
1	¿?
2	¿?
3	¿?
4	¿?
5	¿?
6	¿?
7	¿?
8	¿?
9	¿?
10	¿?
11	¿?



Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;
```

t	0	¿?
	1	¿?
	2	¿?
	3	¿?
	4	¿?
	5	¿?
	6	¿?
	7	¿?
	8	¿?
	9	¿?
	10	¿?
	11	¿?



Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;
```

t

0	8.9
1	¿?
2	¿?
3	¿?
4	¿?
5	¿?
6	¿?
7	¿?
8	¿?
9	¿?
10	¿?
11	¿?



Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;  
t[1] = t[0] - 1.0;
```

t

0	8.9
1	¿?
2	¿?
3	¿?
4	¿?
5	¿?
6	¿?
7	¿?
8	¿?
9	¿?
10	¿?
11	¿?



Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;  
t[1] = t[0] - 1.0;
```

t

0	8.9
1	7.9
2	¿?
3	¿?
4	¿?
5	¿?
6	¿?
7	¿?
8	¿?
9	¿?
10	¿?
11	¿?

Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;  
t[1] = t[0] - 1.0;  
unsigned m = 2;  
t[m] = 10.7;
```

t

0	8.9
1	7.9
2	¿?
3	¿?
4	¿?
5	¿?
6	¿?
7	¿?
8	¿?
9	¿?
10	¿?
11	¿?

Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;  
t[1] = t[0] - 1.0;  
unsigned m = 2;  
t[m] = 10.7;
```

t

0	8.9
1	7.9
2	10.7
3	¿?
4	¿?
5	¿?
6	¿?
7	¿?
8	¿?
9	¿?
10	¿?
11	¿?

Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;  
t[1] = t[0] - 1.0;  
unsigned m = 2;  
t[m] = 10.7;  
t[m + 1] = 15.2;
```

t

0	8.9
1	7.9
2	10.7
3	¿?
4	¿?
5	¿?
6	¿?
7	¿?
8	¿?
9	¿?
10	¿?
11	¿?

Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;  
t[1] = t[0] - 1.0;  
unsigned m = 2;  
t[m] = 10.7;  
t[m + 1] = 15.2;
```

t

0	8.9
1	7.9
2	10.7
3	15.2
4	¿?
5	¿?
6	¿?
7	¿?
8	¿?
9	¿?
10	¿?
11	¿?

Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;  
t[1] = t[0] - 1.0;  
unsigned m = 2;  
t[m] = 10.7;  
t[m + 1] = 15.2;  
t[4] = t[0] + t[1];
```

t

0	8.9
1	7.9
2	10.7
3	15.2
4	¿?
5	¿?
6	¿?
7	¿?
8	¿?
9	¿?
10	¿?
11	¿?

Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;  
t[1] = t[0] - 1.0;  
unsigned m = 2;  
t[m] = 10.7;  
t[m + 1] = 15.2;  
t[4] = t[0] + t[1];
```

t

0	8.9
1	7.9
2	10.7
3	15.2
4	16.8
5	¿?
6	¿?
7	¿?
8	¿?
9	¿?
10	¿?
11	¿?

Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;  
t[1] = t[0] - 1.0;  
unsigned m = 2;  
t[m] = 10.7;  
t[m + 1] = 15.2;  
t[4] = t[0] + t[1];  
m = 5;  
while (m < 8) {  
    t[m] = 25.0;  
    m++;  
}
```

t

0	8.9
1	7.9
2	10.7
3	15.2
4	16.8
5	¿?
6	¿?
7	¿?
8	¿?
9	¿?
10	¿?
11	¿?

Vector

```
const unsigned NUM_MESES = 12;  
double t[NUM_MESES];  
t[0] = 8.9;  
t[1] = t[0] - 1.0;  
unsigned m = 2;  
t[m] = 10.7;  
t[m + 1] = 15.2;  
t[4] = t[0] + t[1];  
m = 5;  
while (m < 8) {  
    t[m] = 25.0;  
    m++;  
}
```

t	0	8.9
	1	7.9
	2	10.7
	3	15.2
	4	16.8
	5	25.0
	6	25.0
	7	25.0
	8	¿?
	9	¿?
	10	¿?
	11	¿?

Vector

```
const unsigned NUM_MESES = 12;
double t[NUM_MESES];
t[0] = 8.9;
t[1] = t[0] - 1.0;
unsigned m = 2;
t[m] = 10.7;
t[m + 1] = 15.2;
t[4] = t[0] + t[1];
m = 5;
while (m < 8) {
    t[m] = 25.0;
    m++;
}
for (unsigned n = 8; n < NUM_MESES; n++) {
    t[n] = t[n - 1] - 3.0;
}
```

t	0	8.9
	1	7.9
	2	10.7
	3	15.2
	4	16.8
	5	25.0
	6	25.0
	7	25.0
	8	¿?
	9	¿?
	10	¿?
	11	¿?

Vector

```
const unsigned NUM_MESES = 12;
double t[NUM_MESES];
t[0] = 8.9;
t[1] = t[0] - 1.0;
unsigned m = 2;
t[m] = 10.7;
t[m + 1] = 15.2;
t[4] = t[0] + t[1];
m = 5;
while (m < 8) {
    t[m] = 25.0;
    m++;
}
for (unsigned n = 8; n < NUM_MESES; n++) {
    t[n] = t[n - 1] - 3.0;
}
```

t

0	8.9
1	7.9
2	10.7
3	15.2
4	16.8
5	25.0
6	25.0
7	25.0
8	22.0
9	19.0
10	16.0
11	13.0

Vectores

Otra forma de declarar e inicializar

```
double t[] = {8.9, 7.9, 10.7,  
              15.2, 16.8, 25.0, 25.0,  
              25.0, 22.0, 19.0, 16.0,  
              13.0,  
              };
```

t

0	8.9
1	7.9
2	10.7
3	15.2
4	16.8
5	25.0
6	25.0
7	25.0
8	22.0
9	19.0
10	16.0
11	13.0



Vectores

Otra forma de declarar e inicializar

```
double t[NUM_MESES] = {8.9, 7.9};
```

t

0	8.9
1	7.9
2	0.0
3	0.0
4	0.0
5	0.0
6	0.0
7	0.0
8	0.0
9	0.0
10	0.0
11	0.0

Índices fuera de los límites en C++

```
const unsigned DIMENSION = 3;
int v[DIMENSION] = {0, 0, 0};

cout << v[0] << endl;
v[2] = 2;
cout << v[3] << endl;           // ¿?
v[4] = 4;                       // ¿?
cout << v[100] << endl;         // ¿?
v[-4] = -4;                     // ¿?
cout << v[123456789] << endl;   // ¿?
```

Vectores y funciones en C++

- ❑ Solo pueden ser parámetros, no valores devueltos.
- ❑ Como parámetros, son siempre **por referencia**.
 - Como parámetro de salida o entrada y salida
 - ❑ `void f(int v[]);`
 - ❑ Las componentes del vector `v` pueden ser consultadas y modificadas por el procedimiento `f`.
 - Como parámetro solo de entrada
 - ❑ `void g(const int w[]);`
 - ❑ Las componentes del vector `w` solo pueden ser consultadas por el procedimiento `g`, pero no modificadas.

Vectores y funciones en C++

- Sintaxis declaración como parámetro
 - `<parámetro-vector> ::= [“const”]
 <tipo> <identificador>“[]”`



Ejemplo

- Función que, dado un vector de temperaturas mensuales, devuelve su media



Cálculo de la temperatura media anual

t	0	8.9
	1	7.9
	2	10.7
	3	15.2
	4	16.8
	5	25.0
	6	25.0
	7	25.0
	8	22.0
	9	19.0
	10	16.0
	11	13.0

Cálculo de la temperatura media anual

```
/*  
 * Pre: «t» tiene NUM_MESES componentes  
 * Post: Devuelve la temperatura media de las temperaturas  
 *        almacenadas en «t»  
 */  
double temperaturaMediaAnual(const double t[]) {  
    double sumaTemperaturas = 0.0;  
    unsigned i = 0;  
    while (i < NUM_MESES) {  
        sumaTemperaturas += t[i];  
        i++;  
    }  
    return sumaTemperaturas / NUM_MESES;  
}
```

Cálculo de la temperatura media anual (bucle for)

```
/*  
 * Pre: «t» tiene NUM_MESES componentes  
 * Post: Devuelve la temperatura media de las  
 *        temperaturas almacenadas en «t»  
 */  
double temperaturaMediaAnualFor(const double t[]) {  
    double sumaTemperaturas = 0.0;  
    for (unsigned i = 0; i < NUM_MESES; i++) {  
        sumaTemperaturas += t[i];  
    }  
    return sumaTemperaturas / NUM_MESES;  
}
```


Ejemplo de programa completo.

Solo función main

```
int main() {  
    double temperaturas[NUM_MESES];  
    for (unsigned i = 0; i < NUM_MESES; i++) {  
        cout << "Escriba la temperatura del mes "  
              << i + 1 << ": ";  
        cin >> temperaturas[i];  
    }  
    double media = temperaturaMediaAnual(temperaturas);  
    cout << endl;  
    cout << "La temperatura media anual es de "  
          << media << endl;  
    return 0;  
}
```

Ejemplo de programa completo

```
#include <iostream>
using namespace std;
const unsigned NUM_MESES = 12;
double temperaturaMediaAnual(const double t[]) {...}

/* Programa que pide al usuario 12 datos de temperaturas medias mensuales
correspondientes a un año y escribe a continuación en la pantalla la
temperatura media anual correspondiente. */
int main() {
    double temperaturas[NUM_MESES];
    for (unsigned i = 0; i < NUM_MESES; i++) {
        cout << "Escriba la temperatura del mes " << i + 1 << ": ";
        cin >> temperaturas[i];
    }
    double mediaAnual = temperaturaMediaAnual(temperaturas);
    cout << endl;
    cout << "La temperatura media anual es de " << mediaAnual << endl;
    return 0;
}
```

Parámetro de tipo vector
(corchetes sin dimensión)

Declaración de variable
de tipo vector
(dimensión entre corchetes)

Acceso a la componente de
un vector (acceso a un dato
de tipo **double**)

Variable de tipo vector
como argumento
(sin corchetes)

Cálculo de la media

```
/*  
 * Pre: «t» tiene «n» componentes y «n» > 0.  
 * Post: Devuelve el valor medio de los valores  
 * almacenados en las componentes de «t».  
 */  
double media(const double t[], unsigned n) {  
    double suma = 0.0;  
    for (unsigned i = 0; i < n; i++) {  
        suma += t[i];  
    }  
    return suma / n;  
}
```

Cálculo de la media

```
/*  
 * Pre: «t» tiene «n» componentes y «n» > 0.  
 * Post: Devuelve el valor medio de los valores  
 * almacenados en las componentes de «t».  
 */  
double media(const double t[], const unsigned n) {  
    double suma = 0.0;  
    for (unsigned i = 0; i < n; i++) {  
        suma += t[i];  
    }  
    return suma / n;  
}
```

Desviación típica

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

Cálculo de la desviación típica

```
/**  
 * Pre: «t» tiene «n» componentes y n > 1.  
 * Post: Devuelve la desviación típica de los  
 *       valores almacenados en «t»  
 */  
double desviacionTipica(const double t[],  
                        const unsigned n) {  
    double suma = 0.0;  
    for (unsigned i = 0; i < n; i++) {  
        suma += pow(t[i] - media(t, n), 2);  
    }  
    return sqrt(suma / (n - 1));  
}
```

Cálculo de la desviación típica

```
/**  
 * Pre: «t» tiene «n» componentes y  $n > 1$ .  
 * Post: Devuelve la desviación típica de los  
 *       valores almacenadas en «t»  
 */  
double desviacionTipica(const double t[],  
                        const unsigned n) {  
    double mediaAritmetica = media(t, n);  
    double suma = 0.0;  
    for (unsigned i = 0; i < n; i++) {  
        suma += pow(t[i] - mediaAritmetica, 2);  
    }  
    return sqrt(suma / (n - 1));  
}
```



Cálculo del máximo

$n = 8$

$t =$

0	1	2	3	4	5	6	7
8	9.5	1	-3	2	15	7	3

Cálculo del máximo

```
/**  
 * Pre: «t» tiene «n» componentes y  $n > 0$ .  
 * Post: Devuelve el valor máximo almacenado en  
 *       las componentes del vector «t».  
 */  
double maximo(const double t[], const unsigned n) {  
    double maximo = t[0];  
    for (unsigned i = 1; i < n; i++) {  
        if (t[i] > maximo) {  
            maximo = t[i];  
        }  
    }  
    return maximo;  
}
```

Un programa de ejemplo

```
/*  
 * Programa que, a modo de ejemplo, invoca a las tres funciones  
 * anteriores.  
 */  
int main() {  
    const unsigned NUM_DATOS = 7;  
    double vector[NUM_DATOS] = {47.9, 55, 1, 76.3, 92, 250, 79};  
  
    cout << "Media: " << media(vector, NUM_DATOS) << endl;  
    cout << "Desviación típica: "  
        << desviacionTipica(vector, NUM_DATOS) << endl;  
    cout << "Máximo: " << maximo(vector, NUM_DATOS) << endl;  
  
    return 0;  
}
```

Vectores sobredimensionados

```
/*  
 * Programa que, a modo de ejemplo, invoca a las tres funciones  
 * anteriores.  
 */  
int main() {  
    const unsigned NUM_DATOS = 7;  
    double vector[NUM_DATOS] = {47.9, 55, 1, 76.3, 92, 250, 79};  
  
    cout << "Media de los 3 primeros datos: "  
        << media(vector, 3) << endl;  
    cout << "Media de los 4 primeros datos: "  
        << media(vector, 4) << endl;  
    cout << "Media de todos los datos: "  
        << media(vector, NUM_DATOS) << endl;  
    return 0;  
}
```

Letra del DNI

TABLA DE LETRAS DEL DNI					
DNI % 23	letra	DNI % 23	letra	DNI % 23	letra
0	T	8	P	16	Q
1	R	9	D	17	V
2	W	10	X	18	H
3	A	11	B	19	L
4	G	12	N	20	C
5	M	13	J	21	K
6	Y	14	Z	22	E
7	F	15	S		

Letra del DNI (mala solución)

```
/*  
 * Pre:  dni > 0  
 * Post: Devuelve la letra del número  
 *       de identificación fiscal que  
 *       corresponde a un número de  
 *       documento nacional de  
 *       identidad igual a «dni».  
 */  
char letra(const unsigned dni) {  
    unsigned resto = dni % 23;  
    if (resto == 0) {  
        return 'T';  
    } else if (resto == 1) {  
        return 'R';  
    } else if (resto == 2) {  
        return 'W';  
    } else if (resto == 3) {  
        return 'A';  
    }  
    ...  
}
```

```
...  
else if (resto == 15) {  
    return 'S';  
} else if (resto == 16) {  
    return 'Q';  
} else if (resto == 17) {  
    return 'V';  
} else if (resto == 18) {  
    return 'H';  
} else if (resto == 19) {  
    return 'L';  
} else if (resto == 20) {  
    return 'C';  
} else if (resto == 21) {  
    return 'K';  
} else {  
    return 'E';  
}  
}
```

Letra del DNI

```
/*  
 * Pre:  dni > 0  
 * Post: Devuelve la Letra del DNI que corresponde a  
 *       un número de DNI igual a «dni».  
 */  
char letra(const unsigned dni) {  
    const unsigned NUM_LETRAS = 23;  
    const char TABLA_LETRAS_NIF[NUM_LETRAS] = {'T', 'R',  
        'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B',  
        'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K',  
        'E'};  
    return TABLA_LETRAS_NIF[dni % NUM_LETRAS];  
}
```



Inversión de una secuencia

- Programa que solicite en primer lugar al usuario un número positivo n , luego solicite n datos reales y por último los escriba en la pantalla en orden inverso al introducido

Introduzca un número positivo: 8

Introduzca 8 reales: 5 9 -1 0 0 8 25 -25

La secuencia en orden inverso es

-25, 25, 8, 0, 0, -1, 9, 5

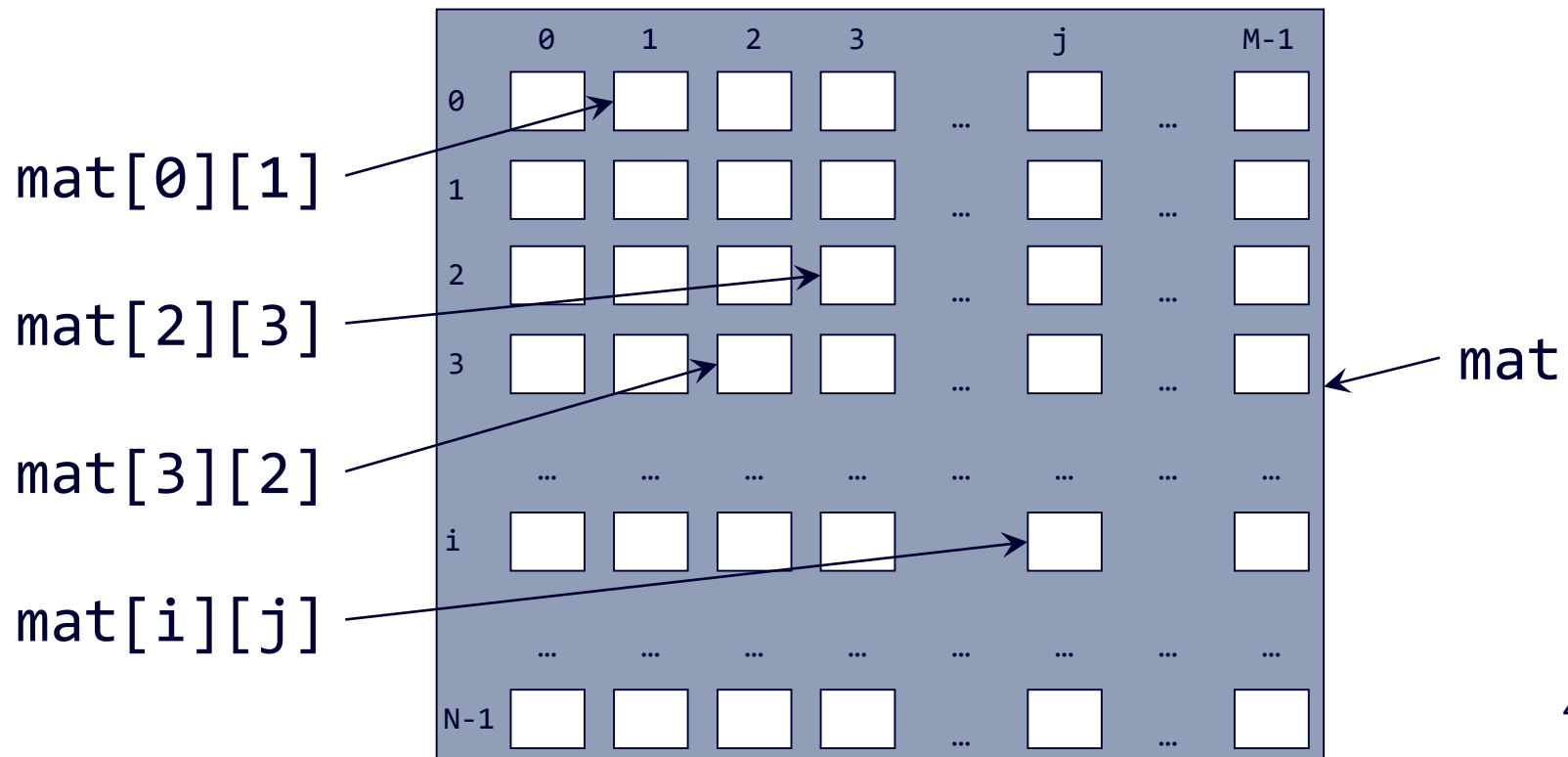
Matrices.

Vectores con varios índices

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1j} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2j} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{ij} & \dots & a_{in} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nj} & \dots & a_{nn} \end{pmatrix}$$

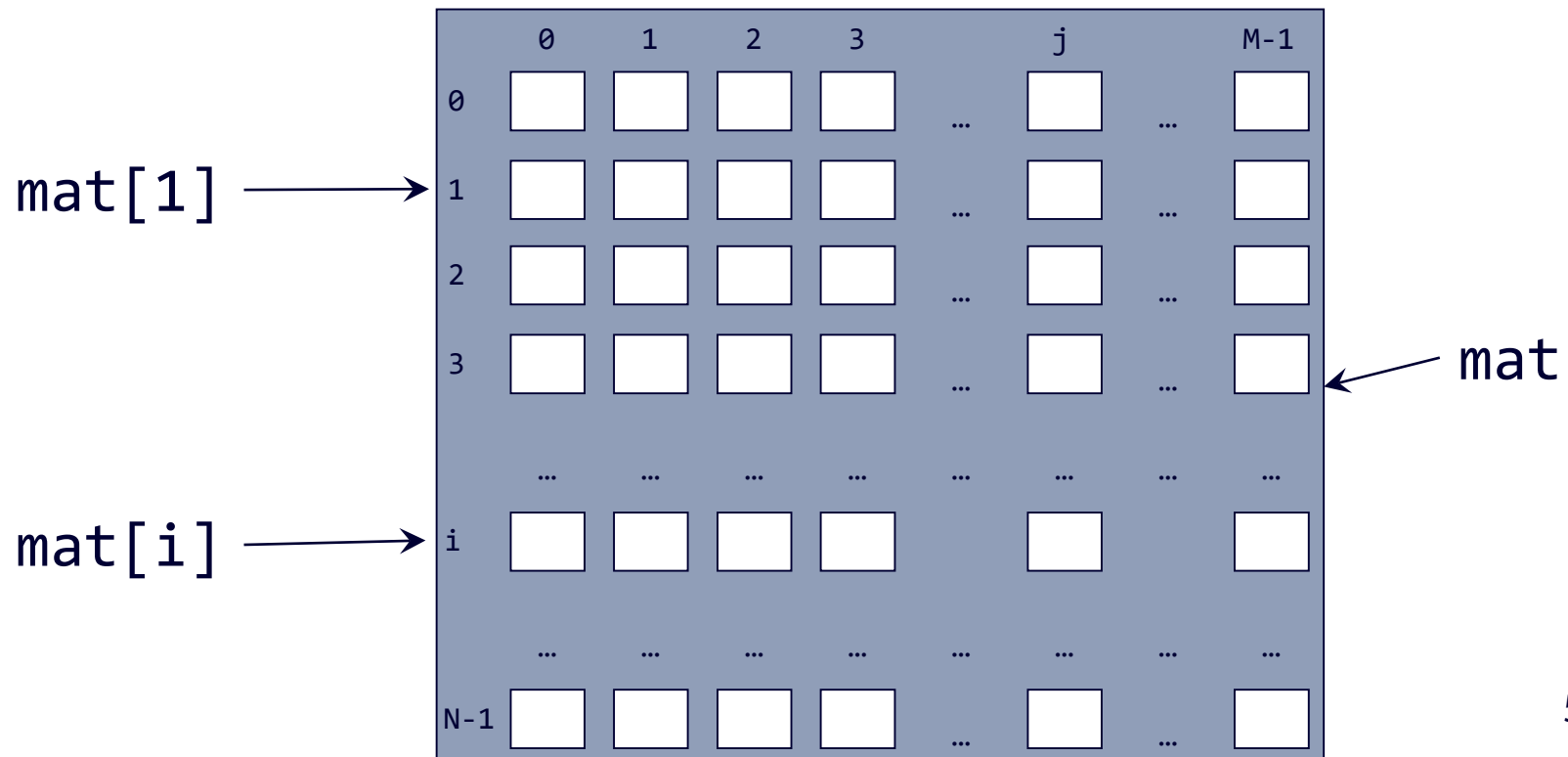
Matrices

```
const unsigned N = 20;  
const unsigned M = 30;  
double mat[N][M];
```

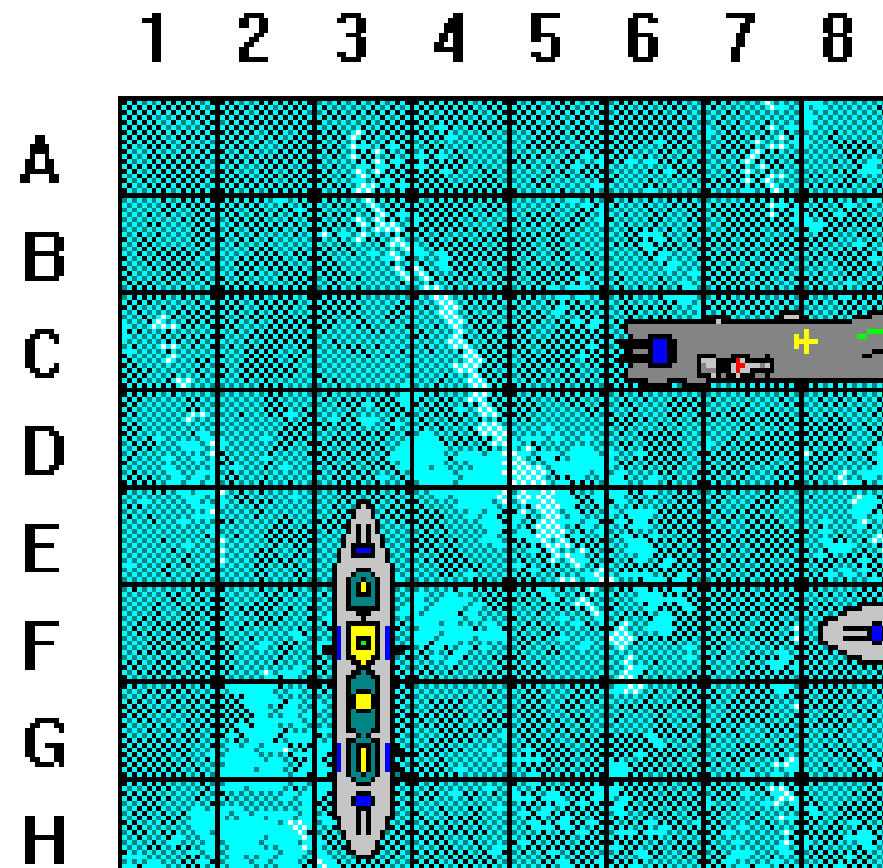


Matrices

```
const unsigned N = 20;  
const unsigned M = 30;  
double mat[N][M];
```



Matrices



Matrices

- ❑ Definición de un tablero para el juego de los barcos
- ❑ Inicialización como «agua»
- ❑ Colocación de un barco en las casillas E3, F3, G3 y H3

Matrices

```
const unsigned NUM_FILAS = 8;  
const unsigned NUM_COLUMNAS = 8;  
const bool AGUA = false;  
const bool BARCO = true;  
  
bool tablero[NUM_FILAS][NUM_COLUMNAS] = {};  
  
// Colocación de un barco en las casillas E3, F3, G3 y H3  
tablero[4][2] = BARCO;  
tablero[5][2] = BARCO;  
tablero[6][2] = BARCO;  
tablero[7][2] = BARCO;
```

Ejercicios con matrices

- ❑ Matriz unidad
- ❑ Suma de matrices
- ❑ Multiplicación de matrices
- ❑ Traspuesta de una matriz
- ❑ Simetría de una matriz

Matriz unidad. Una solución

```
const unsigned DIM = 20;
/*
 * Pre: «matriz» es una matriz cuadrada de DIM x DIM.
 * Post: Inicializa «matriz» como la matriz unidad de
 * tamaño DIM x DIM.
 */
void unidad(double matriz[][DIM]) {
    for (unsigned i = 0; i < DIM; i++) {
        for (unsigned j = 0; j < DIM; j++) {
            if (i == j) {
                matriz[i][j] = 1.0;
            } else {
                matriz[i][j] = 0.0;
            }
        }
    }
}
```

Suma de matrices

```
/*  
 * Pre: «a», «b» y «suma» son matrices  
 * cuadradas de DIM x DIM.  
 * Post: «suma» es la suma matricial  
 * de «a» y «b».  
 */  
void sumar(const double a[][DIM],  
           const double b[][DIM],  
           double suma[][DIM]);
```


Producto de matrices

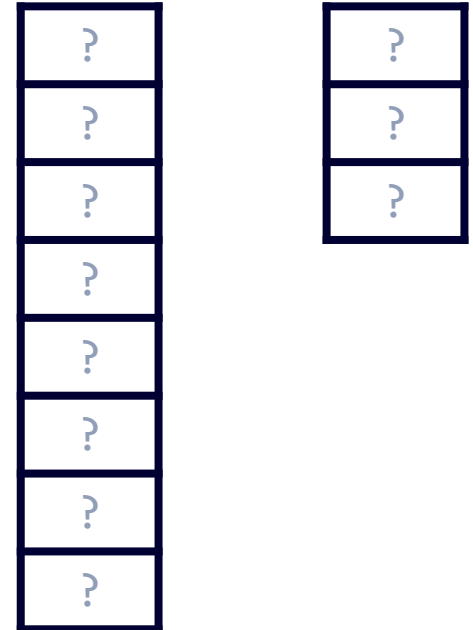
```
/*  
 * Pre: «a», «b» y «producto» son  
 * matrices cuadradas de DIM x DIM.  
 * Post: «producto» es el producto  
 * matricial de «a» y «b».  
 */  
void multiplicar(const double a[][DIM],  
                 const double b[][DIM],  
                 double producto[][DIM]);
```

Simetría de una matriz

```
/*  
 * Pre: «matriz» es una matriz  
 * cuadrada de DIM x DIM.  
 * Post: Devuelve el valor «true»  
 * si y solo si «matriz» es  
 * simétrica.  
 */  
bool esSimetrica(  
    const double matriz[][DIM]);
```


Vectores como parámetros

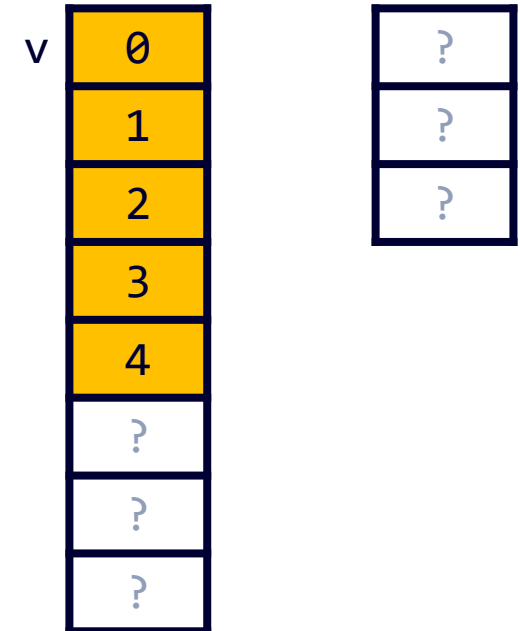
```
void procedimiento1(int u[]) {  
    u[2] = -10;  
}  
  
int main() {  
    int v[5] = {0, 1, 2, 3, 4};  
    procedimiento1(v);  
    return 0;  
}
```



Vectores como parámetros

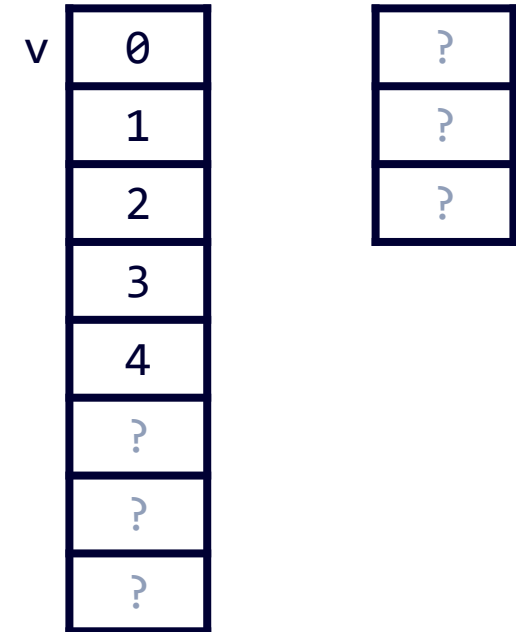
```
void procedimiento1(int u[]) {  
    u[2] = -10;  
}
```

```
int main() {  
     int v[5] = {0, 1, 2, 3, 4};  
    procedimiento1(v);  
    return 0;  
}
```




Vectores como parámetros

```
void procedimiento1(int u[]) {  
    u[2] = -10;  
}  
  
int main() {  
    int v[5] = {0, 1, 2, 3, 4};  
    procedimiento1(v);  
    return 0;  
}
```



El valor de v como argumento es en realidad la dirección de Inicio de v en memoria (la dirección de v[0])

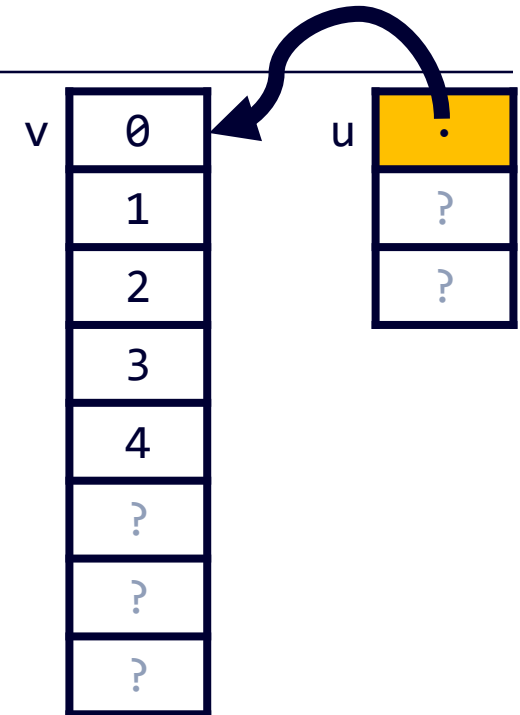
Vectores como parámetros



```
void procedimiento1(int u[]) {  
    u[2] = -10;  
}
```

El parámetro u se asocia con un vector cuya dirección de inicio es la que se ha pasado como argumento

```
int main() {  
    int v[5] = {0, 1, 2, 3, 4};  
    procedimiento1(v);  
    return 0;  
}
```

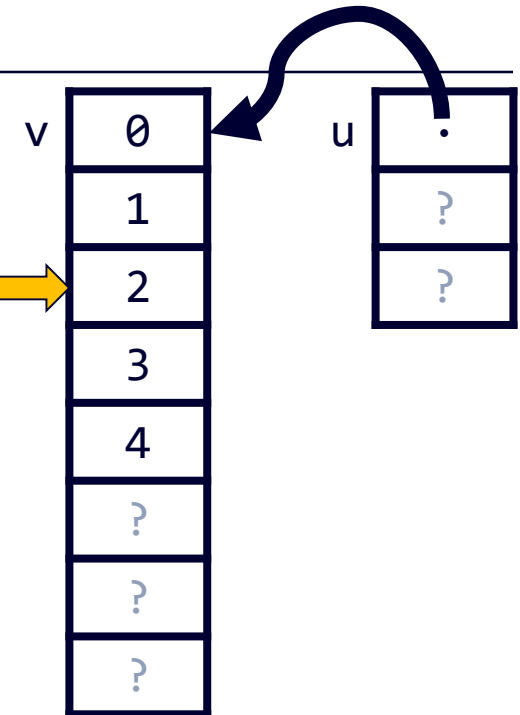


Vectores como parámetros

```
void procedimiento1(int u[]) {  
    → u[2] = -10;  
}
```

Se calcula la dirección de u[2] como la dirección de inicio de u + el tamaño de 2 enteros.

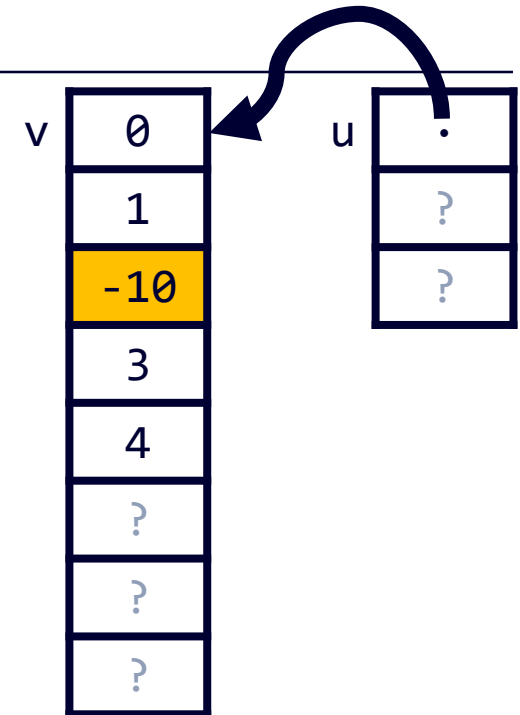
```
int main() {  
    int v[5] = {0, 1, 2, 3, 4};  
    procedimiento1(v);  
    return 0;  
}
```



Vectores como parámetros


```
void procedimiento1(int u[]) {  
    → u[2] = -10;  
}
```

```
int main() {  
    int v[5] = {0, 1, 2, 3, 4};  
    procedimiento1(v);  
    return 0;  
}
```



Vectores como parámetros

```
void procedimiento1(int u[]) {  
    u[2] = -10;  
}  
  
int main() {  
    int v[5] = {0, 1, 2, 3, 4};  
    procedimiento1(v);  
    return 0;  
}
```

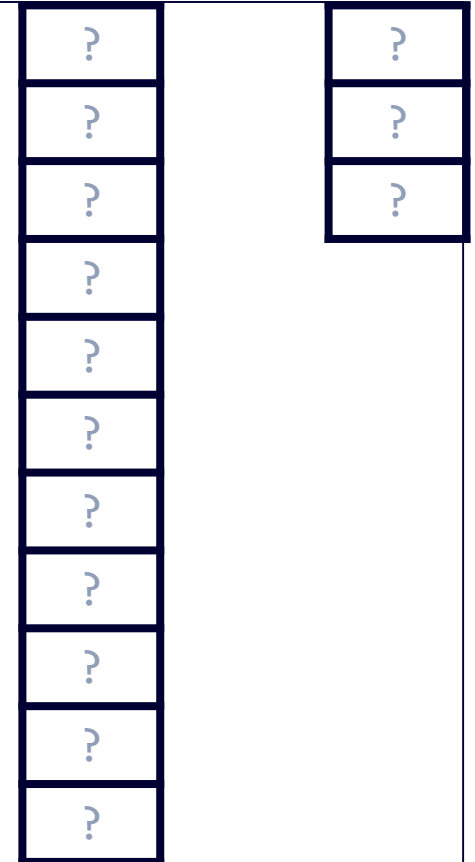


v	0	.
	1	?
	-10	?
	3	
	4	
	?	
	?	
	?	

Matrices como parámetros

```
void procedimiento2(int mat[][3]) {  
    mat[2][1] = -10;  
}
```

```
int main() {  
    int m[3][3] = {{1, 2, 3},  
                   {4, 5, 6},  
                   {7, 8, 9}};  
  
    procedimiento2(m);  
    return 0;  
}
```



Matrices como parámetros

```
void procedimiento2(int mat[][3]) {  
    mat[2][1] = -10;  
}
```

```
int main() {  
    → int m[3][3] = {{1, 2, 3},  
                     {4, 5, 6},  
                     {7, 8, 9}};  
  
    procedimiento2(m);  
    return 0;  
}
```

m	1	?
	2	?
	3	?
	4	
	5	
	6	
	7	
	8	
	9	
	?	
	?	

Matrices como parámetros

```
void procedimiento2(int mat[][3]) {  
    mat[2][1] = -10;  
}
```

```
int main() {  
    int m[3][3] = {{1, 2, 3},  
                   {4, 5, 6},  
                   {7, 8, 9}};  
    procedimiento2(m);  
    return 0;  
}
```

m	1	?
	2	?
	3	?
	4	
	5	
	6	
	7	
	8	
	9	
	?	
	?	

El valor de m como argumento es en realidad la dirección de inicio de m en memoria (la dirección de m[0][0])

Matrices como parámetros

→ **void** procedimiento2(**int** mat[][3]) {

 mat[2][1] = -10;

}

int main() {

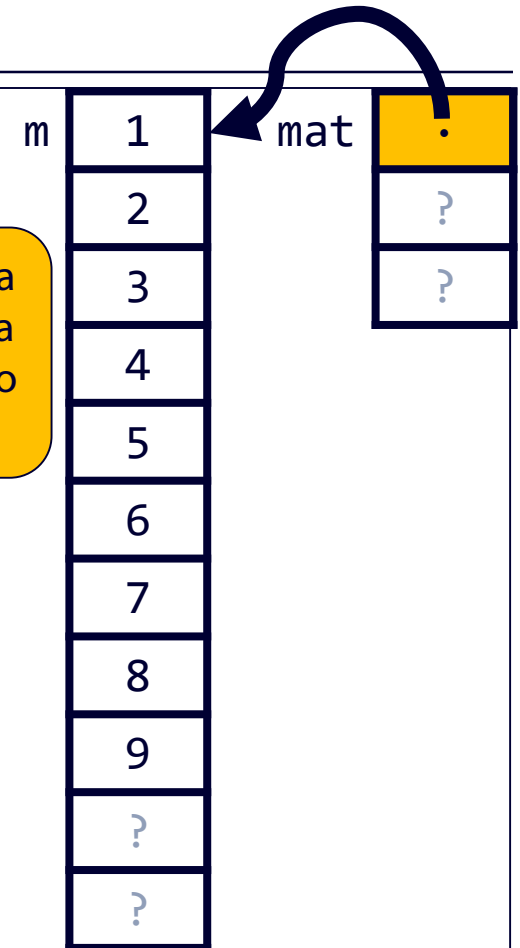
int m[3][3] = {{1, 2, 3},
 {4, 5, 6},
 {7, 8, 9}};

 procedimiento2(m);

return 0;

}

El parámetro mat se asocia con una matriz cuya dirección de inicio es la que se ha pasado como argumento y que tiene que tener 3 columnas.



Matrices como parámetros

```
void procedimiento2(int mat[][3]) {
```



```
    mat[2][1] = -10;
```

```
}
```

```
int main() {
```

```
    int m[3][3] =
```

```
        {4, 5, 6},
```

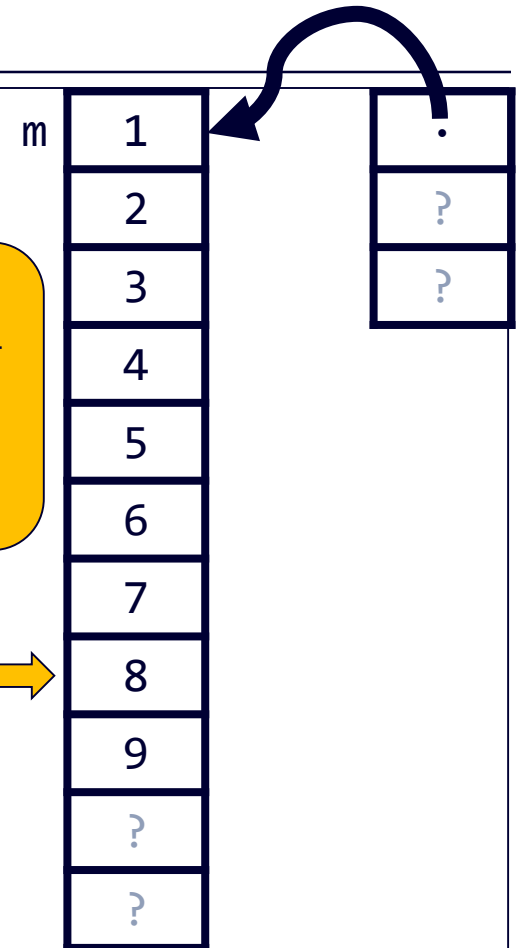
```
        {7, 8, 9}};
```

```
    procedimiento2(m);
```

```
    return 0;
```

```
}
```

Se calcula la dirección de `mat[2][1]` como la dirección de inicio de `mat` + el tamaño de 2 filas completas + el tamaño de 1 entero = dirección de inicio de `mat` + $(2 \times 3 + 1)$ enteros.



Matrices como parámetros

```
void procedimiento2(int mat[][3]) {
```



```
    mat[2][1] = -10;
```

```
}
```

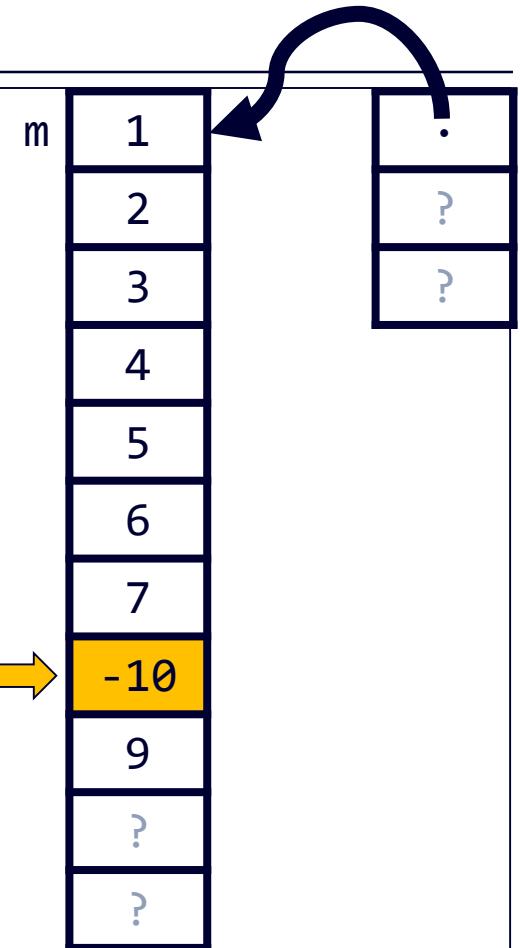
```
int main() {
```

```
    int m[3][3] = {{1, 2, 3},  
                   {4, 5, 6},  
                   {7, 8, 9}};
```

```
    procedimiento2(m);
```

```
    return 0;
```


```
}
```



Matrices como parámetros

```
void procedimiento2(int mat[][3]) {  
    mat[2][1] = -10;  
}
```

```
int main() {  
    int m[3][3] = {{1, 2, 3},  
                   {4, 5, 6},  
                   {7, 8, 9}};  
  
    procedimiento2(m);  
    return 0;  
}
```



m	1	.
	2	?
	3	?
	4	
	5	
	6	
	7	
	-10	
	9	
	?	
	?	

¿Cómo se puede estudiar este tema?

- ❑ Repasando estas transparencias
- ❑ Trabajando con el código de estas transparencias
 - <https://github.com/prog1-eina/tema-09-vectores>
- ❑ Leyendo el material adicional dispuesto en Moodle:
 - Capítulo 9 de los apuntes del profesor Martínez
 - Enlaces a los tutoriales de cplusplus.com y Tutorials Point
- ❑ Trabajando con los problemas de las clases próximas
- ❑ Realizando la práctica 4