

## 6

## Recorrido y búsqueda en arrays

Grados en Ingeniería Informática, Ingeniería del Software e Ingeniería de Computadores

Miguel Gómez-Zamalloa Gil

(adaptadas del original de Luis Hernández Yáñez y Pablo Moreno Ger)



Facultad de Informática  
Universidad Complutense de Madrid



# Índice

---

Recorrido de arrays	Pág. 3
Arrays completos	
Arrays no completos con centinela	
Arrays no completos con contador	
Ejemplos	
Generación de números aleatorios	
Búsquedas en arrays	Pág. 17
Arrays completos	
Arrays no completos con centinela	
Arrays no completos con contador	
Ejemplo	
Recorridos y búsquedas en cadenas	Pág. 27
Más ejemplos de manejo de arrays	Pág. 30
Arrays multidimensionales	Pág. 43
Inicialización de arrays multidimensionales	
Recorrido de un array bidimensional	
Recorrido de un array N-dimensional	
Búsqueda en un array multidimensional	



# Fundamentos de la programación

---

## Recorrido de arrays



# Recorrido de arrays

## *Esquema de recorrido*

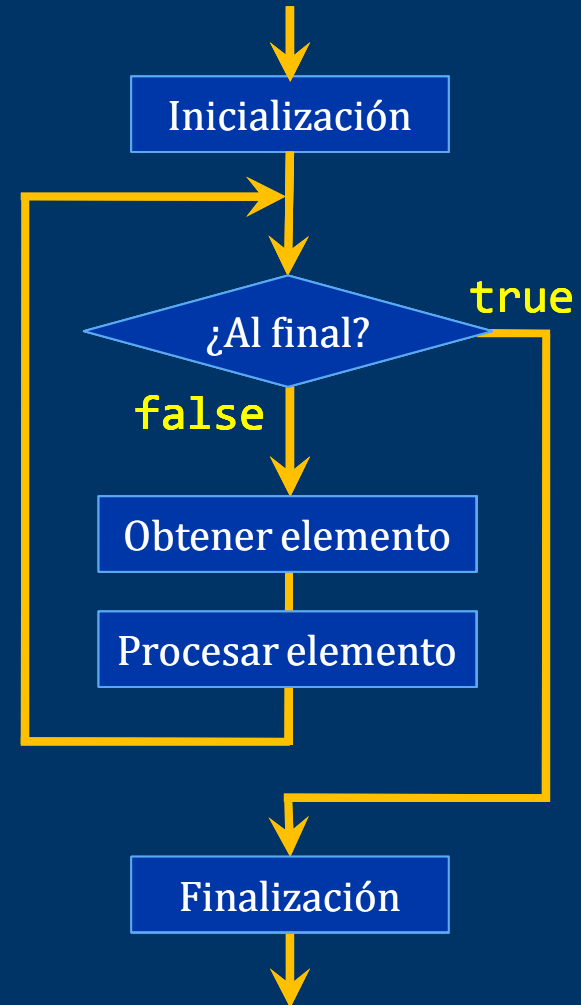
Inicialización

Mientras no al final de la secuencia:

Obtener el siguiente elemento

Procesar el elemento

Finalización



# Recorrido de arrays

---

## *Recorrido de secuencias en arrays*

- ✓ Todas las posiciones ocupadas:  
Tamaño del array = longitud de la secuencia  
N elementos en un array de N posiciones:  
Recorrer el array desde la primera posición hasta la última
- ✓ Posiciones libres al final del array:  
Tamaño del array > longitud de la secuencia
  - Con centinela:  
Recorrer el array hasta encontrar el valor centinela
  - Con *contador* de elementos:  
Recorrer el array hasta el índice *contador* – 1



# Recorrido de arrays

## *Recorrido de arrays completos*

Todas las posiciones del array ocupadas

```
const int N = 10;  
typedef double tVentas[N];  
tVentas ventas;
```

...

ventas	125.40	76.95	328.80	254.62	435.00	164.29	316.05	219.99	93.45	756.62
	0	1	2	3	4	5	6	7	8	9

```
double elemento;  
for (int i = 0; i < N; i++) {  
    elemento = ventas[i];  
    // Procesar el elemento ...  
}
```



# Recorrido de arrays

## *Recorrido de arrays no completos – con centinela*

No todas las posiciones del array están ocupadas

```
const int N = 10;
typedef double tArray[N];
tArray datos; // Datos positivos: centinela = -1
...
```

datos	125.40	76.95	328.80	254.62	435.00	164.29	316.05	-1.0		
	0	1	2	3	4	5	6	7	8	9

```
int i = 0;
double elemento = datos[i];
while (elemento != -1) {
    // Procesar el elemento ...
    i++;
    elemento = datos[i];
}
```

```
int i = 0;
double elemento;
do {
    elemento = datos[i];
    if (elemento != -1) {
        // Procesar el elemento...
        i++;
    }
} while (elemento != -1);
```



# Recorrido de arrays

## *Recorrido de arrays no completos – con contador*

Array y contador íntimamente relacionados: estructura

```
const int N = 10;  
typedef double tArray[N];  
typedef struct {  
    tArray elementos;  
    int contador;  
} tLista;
```

Listas de elementos de longitud variable

elementos									
125.40	76.95	328.80	254.62	435.00	164.29	316.05			
0	1	2	3	4	5	6	7	8	9
contador		7							

Nº de elementos (primer índice sin elemento)





# Recorrido de arrays

---

## *Recorrido de arrays no completos – con contador*

```
const int N = 10;
typedef double tArray[N];
typedef struct {
    tArray elementos;
    int contador;
} tLista;
tLista lista;

...
double elemento;
for (int i = 0; i < lista.contador; i++) {
    elemento = lista.elementos[i];
    // Procesar el elemento...
}
```



# Fundamentos de la programación

---

## Ejemplos



## *Array con los N primeros números de Fibonacci*

```
const int N = 50;
typedef long long int tFibonacci[N]; // 50 números
tFibonacci fib;
fib[0] = 1;
fib[1] = 1;
for (int i = 2; i < N; i++) {
    fib[i] = fib[i - 1] + fib[i - 2];
}
for (int i = 0; i < N; i++) {
    cout << fib[i] << "    ";
}
```



# Ejemplos

## *Cuenta de valores con k dígitos*

Recorrer una lista de N enteros contabilizando cuántos son de 1 dígito, cuántos de 2 dígitos, etcétera (hasta 5 dígitos)

2 arrays: array con los números y array de contadores

```
const int NUM = 100;  
typedef int tNum[NUM]; // Exactamente 100 números  
tNum numeros;  
const int DIG = 5;  
typedef int tDig[DIG]; // i --> números de i+1 dígitos  
tDig numDig = { 0 };
```

numeros	123	2	46237	2345	236	11234	33	999	...	61
	0	1	2	3	4	5	6	7		99
numDig	0	0	0	0	0	0				
	0	1	2	3	4	5				



# Ejemplos

---

## *Cuenta de valores con $k$ dígitos*

Función que devuelve el número de dígitos de un entero:

```
int digitos(int dato) {  
    int n_digitos = 1; // Al menos tiene un dígito  
    // Recorremos la secuencia de dígitos...  
    while (dato >= 10) {  
        dato = dato / 10;  
        n_digitos++;  
    }  
    return n_digitos;  
}
```



# Ejemplos

## *Generación de números pseudoaleatorios*

Probemos con una secuencia de enteros generada aleatoriamente

Función `rand()` (`cstdlib`): entero aleatorio entre 0 y 32766

`srand()` (`cstdlib`): inicia la secuencia de números aleatorios

Acepta un entero que usa como semilla para iniciar la secuencia

¿Qué valor usar? Uno distinto en cada ejecución

→ El instante de tiempo actual (diferente cada vez)

Función `time()` (`ctime`): segundos transcurridos desde 1970

Requiere un argumento, que en nuestro caso será NULL

```
srand(time(NULL)); // Inicia la secuencia
```

```
...
```

```
numeros[0] = rand(); // Entre 0 y 32766
```



## *Cuenta de valores con k dígitos*

```
#include <iostream>
using namespace std;
#include <cstdlib> // srand() y rand()
#include <ctime> // time()

int digitos(int dato);

int main() {
    const int NUM = 100;
    typedef int tNum[NUM]; // Exactamente 100 números
    const int DIG = 5;
    typedef int tDig[DIG];
    tNum numeros;
    tDig numDig = { 0 }; // Inicializa todo el array a 0

    srand(time(NULL)); // Inicia la secuencia aleatoria
    ...
}
```



# Ejemplos

```
for (int i = 0; i < NUM; i++) { // Creamos la secuencia
    numeros[i] = rand(); // Entre 0 y 32766
}
```

```
for (int i = 0; i < NUM; i++) {
    // Recorremos la secuencia de enteros
    numDig[digitos(numeros[i]) - 1]++;
}
```

```
for (int i = 0; i < DIG; i++) {
    // Recorremos la secuencia de contadores
    cout << "De " << i + 1 << " dígs. = " << numDig[i]
        << endl;
}
return 0;
}
```

```
int digitos(int dato) {
    ...
}
```





# Fundamentos de la programación

---

## Búsquedas en arrays



# Búsquedas en arrays

## *Esquema de búsqueda*

Inicialización

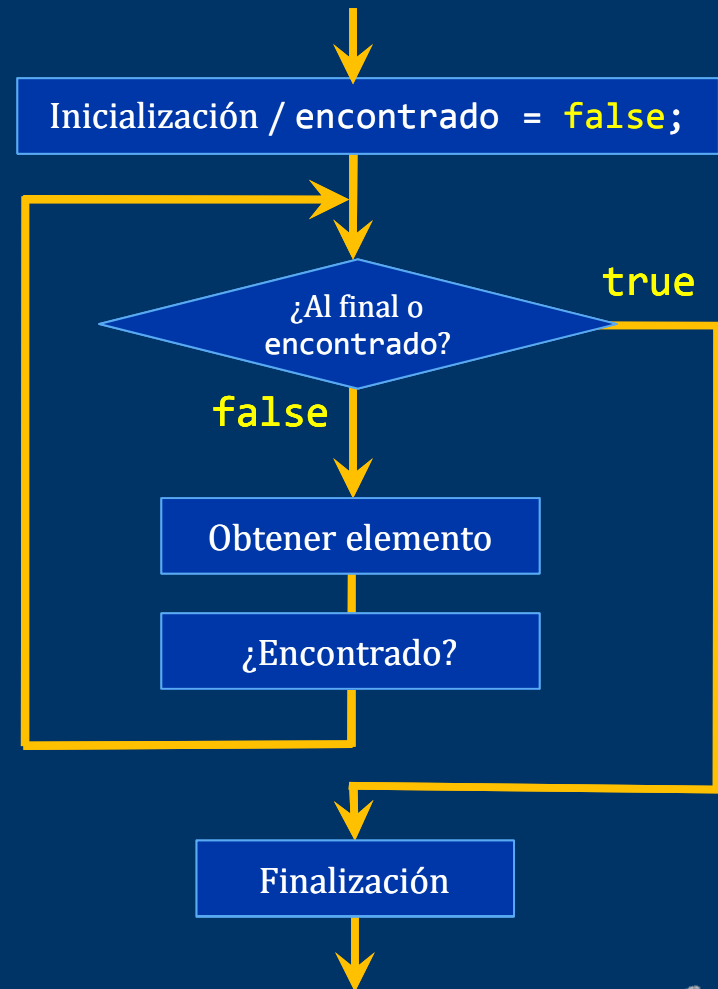
Mientras no se encuentre el elemento  
y no se esté al final de la secuencia:

Obtener el siguiente elemento

Comprobar si el elemento  
satisface la condición

Finalización

(tratar el elemento encontrado  
o indicar que no se ha encontrado)



# Búsquedas en arrays completos

Todas las posiciones ocupadas

```
int buscado;
bool encontrado = false;
cout << "Valor a buscar: ";
cin >> buscado;
int pos = 0;
while ((pos < N) && !encontrado) {
    // Mientras no se llegue al final y no encontrado
    if (lista[pos] == buscado) {
        encontrado = true;
    }
    else {
        pos++;
    }
}
if (encontrado) // ...
```

```
const int N = 100;
typedef int tArray[N];
tArray lista;
```



# Búsquedas en arrays incompletos

## *Con centinela*

```
int buscado;
cout << "Valor a buscar: ";
cin >> buscado;
int pos = 0;
bool encontrado = false;
while ((array[pos] != centinela) && !encontrado) {
    if (array[pos] == buscado) {
        encontrado = true;
    }
    else {
        pos++;
    }
}
if (encontrado) // ...
```

```
const int N = 10;
typedef int tArray[N];
tArray array;
const int centinela = -1;
```



# Búsquedas en arrays incompletos

## *Con contador*

```
int buscado;
cout << "Valor a buscar: ";
cin >> buscado;
int pos = 0;
bool encontrado = false;
while ((pos < miLista.contador)
        && !encontrado) {
    // Mientras no al final y no encontrado
    if (miLista.elementos[pos] == buscado) {
        encontrado = true;
    }
    else {
        pos++;
    }
}
if (encontrado) // ...
```

```
const int N = 10;
typedef double tArray[N];
typedef struct {
    tArray elementos;
    int contador;
} tLista;
tLista miLista;
```



# Búsquedas por posición

## *Acceso directo a cualquier posición*

Acceso directo: *array[posición]*

Si se puede calcular la posición del elemento, su acceso es directo

```
typedef double tVentaMes[DIAS][SUCURSALES];  
typedef struct {  
    tVentaMes ventas;  
    int dias;  
} tMes;  
typedef tMes tVentaAnual[MESES];  
tVentaAnual anual;
```

*Ventas del cuarto día del tercer mes en la primera sucursal:*

`anual[2].ventas[3][0]`



# Fundamentos de la programación

---

## Ejemplo



# Primer valor por encima de un umbral

umbral.cpp

```
#include <iostream>
using namespace std;
#include <fstream>

const int N = 100;
typedef double tArray[N];
typedef struct {
    tArray elementos;
    int contador;
} tLista;

void cargar(tLista &lista, bool &ok);

int main() {
    tLista lista;
    bool ok;
    cargar(lista, ok);
    if (!ok) {
        cout << "Error: no hay archivo o demasiados datos"
              << endl;
    }
}
```





# Primer valor por encima de un umbral

```
else {
    double umbral;
    cout << "Valor umbral: "; cin >> umbral;
    bool encontrado = false;
    int pos = 0;
    while ((pos < lista.contador) && !encontrado) {
        if (lista.elementos[pos] > umbral) {
            encontrado = true;
        }
        else {
            pos++;
        }
    }
    if (encontrado) {
        cout << "Valor en pos. " << pos + 1 << " ("
            << lista.elementos[pos] << ")" << endl;
    }
    else {
        cout << "¡No encontrado!" << endl;
    }
}
return 0;
}
```



# Primer valor por encima de un umbral

```
void cargar(tLista &lista, bool &ok) {
    ifstream archivo;
    double dato;
    bool abierto = true, overflow = false;
    lista.contador = 0;
    archivo.open("datos.txt");
    if (!archivo.is_open()) {
        abierto = false;
    }
    else {
        archivo >> dato;
        while ((dato >= 0) && !overflow) {
            if (lista.contador == N) {
                overflow = true; // ¡Demasiados!
            }
            else {
                lista.elementos[lista.contador] = dato;
                lista.contador++;
                archivo >> dato;
            }
        }
        archivo.close();
    }
    ok = abierto && !overflow;
}
```



# Fundamentos de la programación

---

## Recorridos y búsquedas en cadenas de caracteres



## *Recorridos y búsquedas en cadenas de caracteres*

Longitud de la cadena: `size()` o `length()`

Caso similar a los arrays con contador de elementos

Ejemplo: Recorrido de una cadena generando otra invertida

```
string cadena, inversa = "";
int pos;
char car;
// ... (lectura de cadena)
pos = 0;
while (pos < cadena.size()) {
    // Mientras no se llegue al final de la cadena
    car = cadena.at(pos);
    inversa = car + inversa; // Inserta car al principio
    pos++;
} // ...
```



## *Búsqueda de un carácter en una cadena*

```
string cadena;  
char buscado;  
int pos;  
bool encontrado;  
// ... (lectura de cadena)  
cout << "Introduce el carácter a buscar: ";  
cin >> buscado;  
pos = 0;  
encontrado = false;  
while ((pos < cadena.size()) && !encontrado) {  
    if (cadena.at(pos) == buscado) {  
        encontrado = true;  
    }  
    else {  
        pos++;  
    }  
}  
if (encontrado) // ...
```



# Fundamentos de la programación

---

## Más ejemplos de manejo de arrays



# Manejo de vectores

Tipo tVector para representar secuencias de N enteros:

```
const int N = 10;  
typedef int tVector[N];
```

Subprogramas:

- ✓ Dado un vector, mueve sus componentes un lugar a la derecha; el último componente se moverá al 1<sup>er</sup> lugar
- ✓ Dado un vector, calcula y devuelve la suma de los elementos que se encuentran en las posiciones pares del vector
- ✓ Dado un vector, encuentra y devuelve la componente mayor
- ✓ Dados dos vectores, devuelve un valor que indique si son iguales
- ✓ Dado un vector, determina si alguno de los valores almacenados en el vector es igual a la suma del resto de los valores del mismo; devuelve el índice del primero encontrado o -1 si no se encuentra
- ✓ Dado un vector, determina si alguno de los valores almacenados en el vector está repetido



```
void desplazar(tVector v) {
    int aux = v[N - 1];

    for (int i = N - 1; i > 0; i--) {
        v[i] = v[i - 1];
    }
    v[0] = aux;
}

int sumaPares(const tVector v) {
    int suma = 0;

    for (int i = 0; i < N; i = i + 2) {
        suma = suma + v[i];
    }

    return suma;
}
```





# Manejo de vectores

```
int encuentraMayor(const tVector v) {  
    int max = v[0], posMayor = 0;  
    for (int i = 1; i < N; i++) {  
        if (v[i] > max) {  
            posMayor = i;  
            max = v[i];  
        }  
    }  
    return posMayor;  
}
```

```
bool sonIguales(const tVector v1, const tVector v2) {  
    //Implementación como búsqueda del primer elemento distinto  
    bool encontrado = false;  
    int i = 0;  
    while ((i < N) && !encontrado) {  
        encontrado = (v1[i] != v2[i]);  
        i++;  
    }  
    return !encontrado;  
}
```



# Manejo de vectores

```
int compruebaSuma(const tVector v) {  
    // ¿Alguno igual a la suma del resto?  
    bool encontrado = false;  
    int i = 0;  
    int suma;  
    while ((i < N) && !encontrado) {  
        suma = 0;  
        for (int j = 0; j < N; j++) {  
            if (j != i) {  
                suma = suma + v[j];  
            }  
        }  
        encontrado = (suma == v[i]);  
        i++;  
    }  
    if (!encontrado) {  
        i = 0;  
    }  
    return i - 1;  
}
```



# Manejo de vectores

```
bool hayRepetidos(const tVector v) {  
    bool encontrado = false;  
    int i = 0, j;  
  
    while ((i < N) && !encontrado) {  
        j = i + 1;  
        while ((j < N) && !encontrado) {  
            encontrado = (v[i] == v[j]);  
            j++;  
        }  
        i++;  
    }  
  
    return encontrado;  
}
```



# Más vectores

Dado un vector de  $N$  caracteres  $v1$ , en el que no hay elementos repetidos, y otro vector de  $M$  caracteres  $v2$ , donde  $N \leq M$ , se quiere comprobar si todos los elementos del vector  $v1$  están también en el vector  $v2$

Por ejemplo, si:

$v1 =$  'a' 'h' 'i' 'm'

$v2 =$  'h' 'a' 'x' 'x' 'm' 'i'

El resultado sería cierto, ya que todos los elementos de  $v1$  están en  $v2$



# Manejo de vectores

incluidos.cpp

```
#include <iostream>
using namespace std;

const int N = 3;
const int M = 10;
typedef char tVector1[N];
typedef char tVector2[M];

bool esta(char dato, const tVector2 v2);
bool vectorIncluido(const tVector1 v1, const tVector2 v2);

int main() {
    tVector1 v1 = { 'a', 'b', 'c' };
    tVector2 v2 = { 'a', 'r', 'e', 't', 'z', 's', 'a', 'h', 'b', 'x' };
    bool ok = vectorIncluido(v1, v2);
    if (ok) {
        cout << "OK: v1 esta incluido en v2" << endl;
    }
    else {
        cout << "NO: v1 no esta incluido en v2" << endl;
    }
    return 0;
}
```



# Manejo de vectores

```
bool esta(char dato, const tVector2 v2) {
    int i = 0;
    bool encontrado = (dato == v2[0]);

    while (!encontrado && (i < M - 1)) {
        i++;
        encontrado = (dato == v2[i]);
    }

    return encontrado;
}

bool vectorIncluido(const tVector1 v1, const tVector2 v2) {
    int i = 0;
    bool encontrado = esta(v1[0], v2);

    while (encontrado && (i < N - 1)) {
        i++;
        encontrado = esta(v1[i], v2);
    }

    return encontrado;
}
```



# Anagramas

---

Un programa que lea dos cadenas del teclado y determine si una es un anagrama de la otra, es decir, si una cadena es una permutación de los caracteres de la otra.

Por ejemplo, "acre" es un anagrama de "cera" y de "arce". Ten en cuenta que puede haber letras repetidas ("carro", "llave").



# Anagramas

anagramas.cpp

```
#include <iostream>
#include <string>
using namespace std;

int buscaCaracter(string cad, char c); // Índice o -1 si no está

int main() {
    string cad1, cad2;
    bool sonAnagramas = true;
    int numCar, posEnCad2;

    cout << "Introduce la primera cadena: ";
    getline(cin, cad1);
    cout << "Introduce la segunda cadena: ";
    getline(cin, cad2);
    if (cad1.length() != cad2.length()) { // No son anagramas
        sonAnagramas = false;
    }
    else {
        numCar = 0; // Contador de caracteres de la primera cadena
        while (sonAnagramas && (numCar < cad1.length())) {
            posEnCad2 = buscaCaracter(cad2, cad1.at(numCar));
```





# Anagramas

```
        if (posEnCad2 == -1) { //No se ha encontrado el caracter
            sonAnagramas = false;
        }
        else {
            cad2.erase(posEnCad2, 1);
        }
        numCar++;
    }
}

if (sonAnagramas) {
    cout << "Las palabras introducidas son anagramas" << endl;
}
else {
    cout << "Las palabras introducidas NO son anagramas" << endl;
}

return 0;
}
```



# Anagramas

```
int buscaCaracter(string cad, char c) {  
    int pos = 0, lon = cad.length();  
    bool encontrado = false;  
  
    while ((pos < lon) && !encontrado) {  
        if (cad.at(pos) == c) {  
            encontrado = true;  
        }  
        else {  
            pos++;  
        }  
    }  
    if (!encontrado) {  
        pos = -1;  
    }  
  
    return pos;  
}
```



## Arrays multidimensionales



# Arrays multidimensionales

## *Arrays de varias dimensiones*

Varios tamaños en la declaración: cada uno con sus corchetes

```
typedef tipo_base nombre[tamaño1][tamaño2][...][tamañoN];
```

Varias dimensiones, tantas como tamaños se indiquen

```
typedef double tMatriz[50][100];  
tMatriz matriz;
```

Tabla bidimensional de 50 filas por 100 columnas:

	0	1	2	3	...	98	99
0					...		
1					...		
2					...		
...	...	...	...	...	...	...	...
48					...		
49					...		



# Arrays multidimensionales

## *Arrays de varias dimensiones*

```
typedef double tMatriz[50][100];  
tMatriz matriz;
```

Cada elemento se localiza con dos índices, uno por dimensión

```
cout << matriz[2][98];
```

	0	1	2	3	...	98	99
0					...		
1					...		
2					...		
...	...	...	...	...	...	...	...
48					...		
49					...		



# Arrays multidimensionales



## *Arrays de varias dimensiones*

Podemos definir tantas dimensiones como necesitemos

```
typedef double tMatriz[5][10][20][10];  
tMatriz matriz;
```

Necesitaremos tantos índices como dimensiones:

```
cout << matriz[2][9][15][6];
```



# Arrays multidimensionales

## *Ejemplo de array bidimensional*

Temperaturas mínimas y máximas

Matriz bidimensional de días y mínima/máxima:

```
const int MaxDias = 31;  
const int MED = 2; // N° de medidas  
typedef double tTemp[MaxDias][MED]; // Día x mín./máx.  
tTemp temp;
```

Ahora:

- ✓ `temp[i][0]` es la temperatura **mínima** del día `i+1`
- ✓ `temp[i][1]` es la temperatura **máxima** del día `i+1`



# Arrays multidimensionales

temp.cpp

```
int main() {
    const int MaxDias = 31;
    const int MED = 2; // N° de medidas
    typedef double tTemp[MaxDias][MED]; // Día x mín./máx.
    tTemp temp;
    double tMaxMedia = 0, tMinMedia = 0,
           tMaxAbs = -100, tMinAbs = 100;
    int dia = 0;
    double max, min;
    ifstream archivo;

    archivo.open("temp.txt");
    if (!archivo.is_open()) {
        cout << "No se ha podido abrir el archivo!" << endl;
    }
    else {
        archivo >> min >> max;
        // El archivo termina con -99 -99
        ...
    }
}
```





# Arrays multidimensionales

```
while (!(min == -99) && (max == -99))
    && (dia < MaxDias)) {
    temp[dia][0] = min;
    temp[dia][1] = max;
    dia++;
    archivo >> min >> max;
}
archivo.close();
for (int i = 0; i < dia; i++) {
    tMinMedia = tMinMedia + temp[i][0];
    if (temp[i][0] < tMinAbs) {
        tMinAbs = temp[i][0];
    }
    tMaxMedia = tMaxMedia + temp[i][1];
    if (temp[i][1] > tMaxAbs) {
        tMaxAbs = temp[i][1];
    }
}
...
```



# Arrays multidimensionales

```
tMinMedia = tMinMedia / dia;
tMaxMedia = tMaxMedia / dia;
cout << "Temperaturas mínimas.-" << endl;
cout << "    Media = " << fixed << setprecision(1)
    << tMinMedia << " C    Mínima absoluta = "
    << setprecision(1) << tMinAbs << " C" << endl;
cout << "Temperaturas máximas.-" << endl;
cout << "    Media = " << fixed << setprecision(1)
    << tMaxMedia << " C    Máxima absoluta = "
    << setprecision(1) << tMaxAbs << " C" << endl;
}

return 0;
}
```



# Inicialización de arrays multidimensionales

Podemos dar valores a los elementos de un array al declararlo

Arrays bidimensionales:

```
typedef int tArray[5][2];  
tArray cuads = {1,1, 2,4, 3,9, 4,16, 5,25};
```

Se asignan en el orden en el que los elementos están en memoria

La memoria es de una dimensión: secuencia de celdas

En memoria varían más rápidamente los índices de la derecha:

cuads[0][0] cuads[0][1] cuads[1][0] cuads[1][1] cuads[2][0]...

Para cada valor del primer índice: todos los valores del segundo



# Inicialización de arrays multidimensionales

## *Inicialización de un array bidimensional*

```
typedef int tArray[5][2];  
tArray cuads = {1,1, 2,4, 3,9, 4,16, 5,25};
```

	Memoria
cuads[0][0]	1
cuads[0][1]	1
cuads[1][0]	2
cuads[1][1]	4
cuads[2][0]	3
cuads[2][1]	9
cuads[3][0]	4
cuads[3][1]	16
cuads[4][0]	5
cuads[4][1]	25

	0	1
0	1	1
1	2	4
2	3	9
3	4	16
4	5	25



Si hay menos valores que elementos, el resto se inicializan a cero

Inicialización a cero de todo el array:

```
int cuads[5][2] = { 0 };
```



# Inicialización de arrays multidimensionales

```
typedef double tMatriz[3][4][2][3];  
tMatriz matriz =  
{1, 2, 3, 4, 5, 6,  
7, 8, 9, 10, 11, 12};
```

	Memoria
matriz[0][0][0][0]	1
matriz[0][0][0][1]	2
matriz[0][0][0][2]	3
matriz[0][0][1][0]	4
matriz[0][0][1][1]	5
matriz[0][0][1][2]	6
matriz[0][1][0][0]	7
matriz[0][1][0][1]	8
matriz[0][1][0][2]	9
matriz[0][1][1][0]	10
matriz[0][1][1][1]	11
matriz[0][1][1][2]	12
matriz[0][2][0][0]	0
...	0



# Recorrido de un array bidimensional

```
const int FILAS = 10;
const int COLUMNAS = 5;
typedef double tMatriz[FILAS][COLUMNAS];
tMatriz matriz;
```

Para cada *fila* (de 0 a FILAS – 1):

Para cada *columna* (de 0 a COLUMNAS – 1):

Procesar el elemento en *[fila][columna]*

```
for (int fila = 0; fila < FILAS; fila++) {
    for (int columna = 0; columna < COLUMNAS; columna++) {
        // Procesar matriz[fila][columna]
    }
}
```



# Ejemplo

## *Ventas de todos los meses de un año*

```
const int Meses = 12;
const int MaxDias = 31;
typedef double tVentas[Meses][MaxDias];
tVentas ventas; // Ventas de todo el año
typedef short int tDiasMes[Meses];
tDiasMes diasMes;
inicializa(diasMes); // Nº de días de cada mes
// Pedimos las ventas de cada día del año...

for (int mes = 0; mes < Meses; mes++) {
    for (int dia = 0; dia < diasMes[mes]; dia++) {
        cout << "Ventas del día " << dia + 1
              << " del mes " << mes + 1 << ": ";
        cin >> ventas[mes][dia];
    }
}
```



# Ejemplo

*Ventas de todos los meses de un año*

		Días								
		0	1	2	3	4	...	28	29	30
Meses	0	201	125	234	112	156	...	234	543	667
	1	323	231	675	325	111	...			
	2	523	417	327	333	324	...	444	367	437
	3	145	845	654	212	562	...	354	548	
	4	327	652	555	222	777	...	428	999	666
	5	854	438	824	547	175	...	321	356	
	6	654	543	353	777	437	...	765	678	555
	7	327	541	164	563	327	...	538	159	235
	8	333	327	432	249	777	...	528	529	
	9	524	583	333	100	334	...	743	468	531
	10	217	427	585	218	843	...	777	555	
	11	222	666	512	400	259	...	438	637	879

Celdas no utilizadas





# Recorrido de arrays N-dimensionales

```
const int DIM1 = 10;  
const int DIM2 = 5;  
const int DIM3 = 25;  
const int DIM4 = 50;
```

```
typedef double tMatriz[DIM1][DIM2][DIM3][DIM4];
```

```
tMatriz matriz;
```

Bucles anidados, desde la primera dimensión hasta la última:

```
for (int n1 = 0; n1 < DIM1; n1++) {  
    for (int n2 = 0; n2 < DIM2; n2++) {  
        for (int n3 = 0; n3 < DIM3; n3++) {  
            for (int n4 = 0; n4 < DIM4; n4++) {  
                // Procesar matriz[n1][n2][n3][n4]  
            }  
        }  
    }  
}
```



# Ejemplo

## *Ventas diarias de cuatro sucursales*

Cada mes del año: ingresos de cada sucursal cada día del mes  
Meses con distinto nº de días → junto con la matriz de ventas mensual guardamos el nº de días del mes concreto → estructura

```
const int DIAS = 31;
const int SUCURSALES = 4;
typedef double tVentaMes[DIAS][SUCURSALES];
typedef struct {
    tVentaMes ventas;
    int dias;
} tMes;
```

```
const int MESES = 12;
typedef tMes tVentaAnual[MESES];
tVentaAnual anual;
```

```
anual → tVentaAnual
anual[i] → tMes
anual[i].dias → int
anual[i].ventas → tVentaMes
anual[i].ventas[j][k] → double
```



# Ejemplo

Cálculo de las ventas  
de todo el año:

Para cada mes...

Para cada día del mes...

Para cada sucursal...

Acumular las ventas

```
const int DIAS = 31;
const int SUCURSALES = 4;
typedef double
tVentaMes[DIAS][SUCURSALES];
typedef struct {
    tVentaMes ventas;
    int dias;
} tMes;

const int MESES = 12;
typedef tMes tVentaAnual[MESES];
tVentaAnual anual;
```

```
double total = 0;
for (int mes = 0; mes < MESES; mes++) {
    for (int dia = 0; dia < anual[mes].dias; dia++) {
        for (int suc = 0; suc < SUCURSALES; suc++) {
            total = total + anual[mes].ventas[dia][suc];
        }
    }
}
```



# Búsqueda en un array multidimensional

Primer valor > umbral

```
bool encontrado = false;
int mes = 0, dia, suc;
while ((mes < MESES) && !encontrado) {
    dia = 0;
    while ((dia < anual[mes].dias) && !encontrado) {
        suc = 0;
        while ((suc < SUCURSALES) && !encontrado) {
            if (anual[mes].ventas[dia][suc] > umbral) {
                encontrado = true;
            }
            else {
                suc++;
            }
        }
        if (!encontrado) {
            dia++;
        }
    }
    if (!encontrado) {
        mes++;
    }
}
if (encontrado) { ...
```






# Acerca de *Creative Commons*



## Licencia CC (*Creative Commons*)

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.

