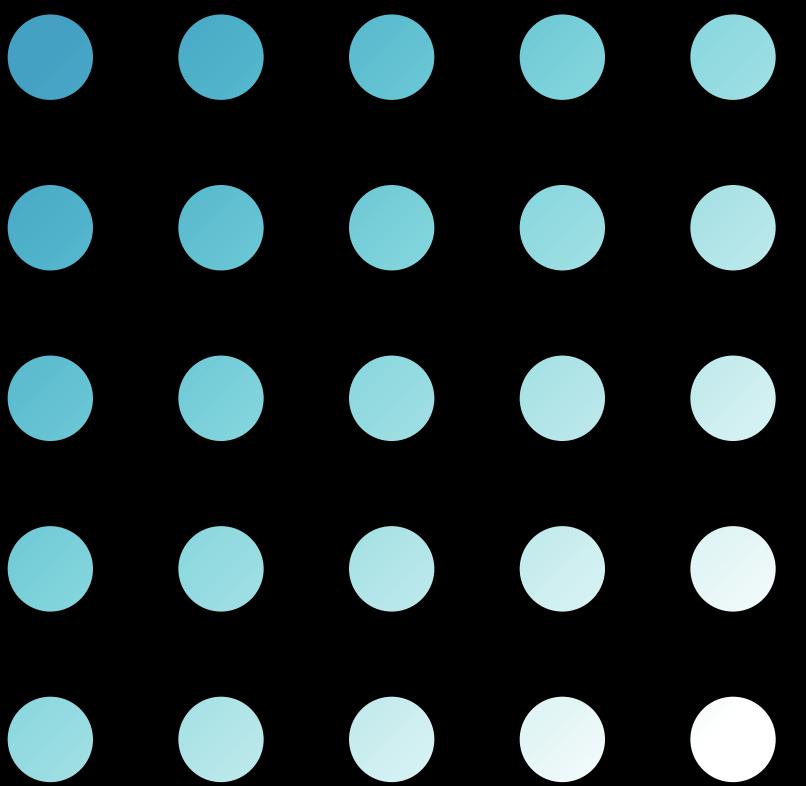


Arreglos



Arrays

Colecciones homogéneas

Un mismo tipo de dato para varios elementos:

- ✓ Notas de los estudiantes de una clase
- ✓ Ventas de cada día de la semana
- ✓ Temperaturas de cada día del mes

...

En lugar de declarar N variables...

vLun	vMar	vMie	vJue	vVie	vSab	vDom
125.40	76.95	328.80	254.62	435.00	164.29	0.00

... declaramos una tabla de N valores:

ventas	125.40	76.95	328.80	254.62	435.00	164.29	0.00
Índices →	0	1	2	3	4	5	6

Arrays

Estructura secuencial

Cada elemento se encuentra en una posición (*índice*):

- ✓ Los índices son enteros positivos
- ✓ El índice del primer elemento siempre es 0
- ✓ Los índices se incrementan de uno en uno

ventas	125.40	76.95	328.80	254.62	435.00	164.29	0.00
	0	1	2	3	4	5	6

ventas	125.40	76.95	328.80	254.62	435.00	164.29	0.00
	0	1	2	3	4	5	6

Acceso directo

A cada elemento se accede a través de su índice:

`ventas[4]` accede al 5º elemento (contiene el valor `435.00`)

```
cout << ventas[4];
```

```
ventas[4] = 442.75;
```



Datos de un mismo tipo base:
Se usan como cualquier variable

Tipos de datos

Clasificación de tipos

- ✓ Simples
 - ❖ Estándar: `int`, `float`, `double`, `char`, `bool`
Conjunto de valores predeterminado
 - ❖ Definidos por el usuario: *enumerados*
Conjunto de valores definido por el programador

- ✓ Estructurados
 - ❖ Colecciones homogéneas: *arrays*
Todos los elementos del mismo tipo
 - ❖ Colecciones heterogéneas: *estructuras*
Los elementos pueden ser de tipos distintos



Tipos arrays

Declaración de tipos de arrays

tipo nombre_tipo[tamaño];

Ejemplos:

```
double Temp[7];
short int DiasMes[12];
char Vocales[5];
double Ventas[31];
```



Recuerda: Adoptamos el convenio de comenzar los nombres de tipo con una t minúscula, seguida de una o varias palabras, cada una con su inicial en mayúscula

Manejo de vectores

Tipo **tVector** para representar secuencias de N enteros:

```
const int N = 10;  
int Vector[N];
```

Vector: Es una colección dinámica de elementos del mismo tipo que puede crecer o reducir su tamaño durante la ejecución del programa.

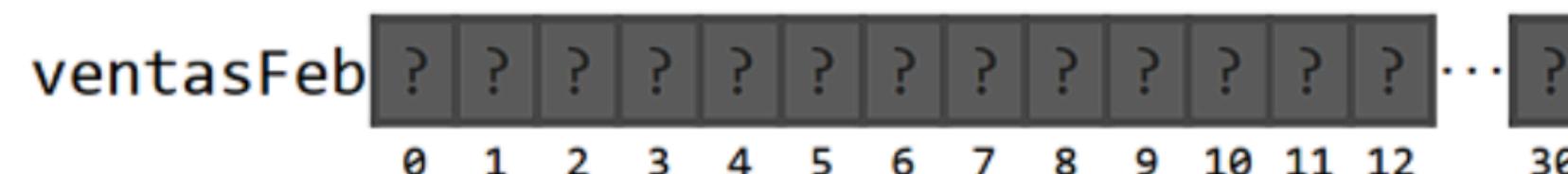
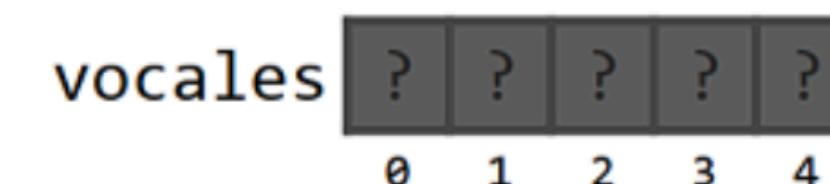
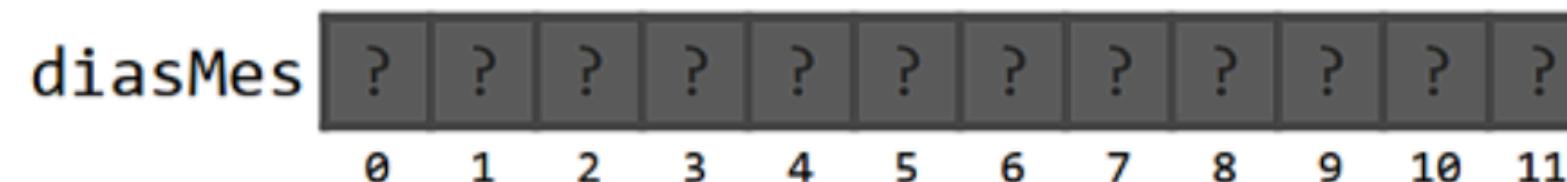
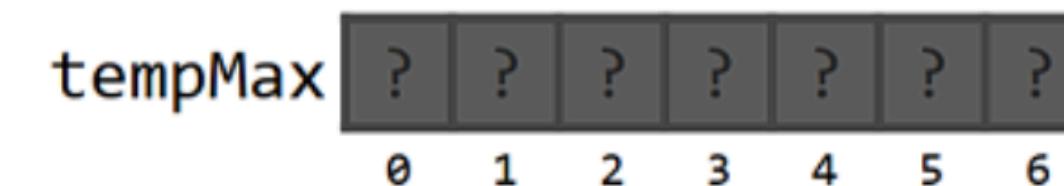
Variables arrays

Declaración de variables arrays

tipo nombre;

Ejemplos:

```
double Temp[7];
short int DiasMes[12];
char Vocales[5];
double Ventas[31];
```



NO se inicializan los elementos automáticamente

typedef

typedef en C++ es una palabra clave que se utiliza para crear alias (nombres alternativos) para tipos de datos existentes. Esto permite crear nombres más cortos y legibles para tipos de datos complejos o largos.

Tipo tVector para representar secuencias de N enteros:

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

¿Cómo crearían un arreglo utilizando la palabra clave typedef?

Variables arrays

typedef

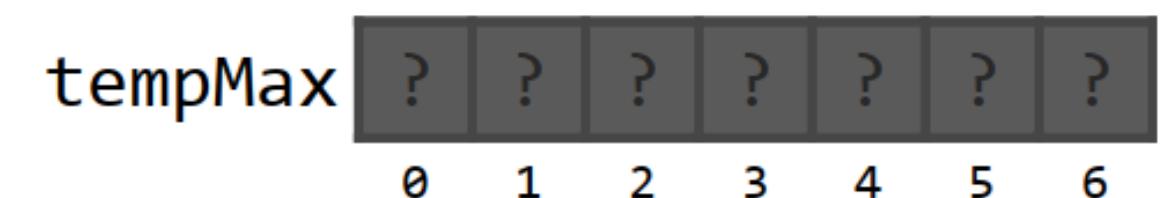
Declaración de variables arrays

tipo nombre;

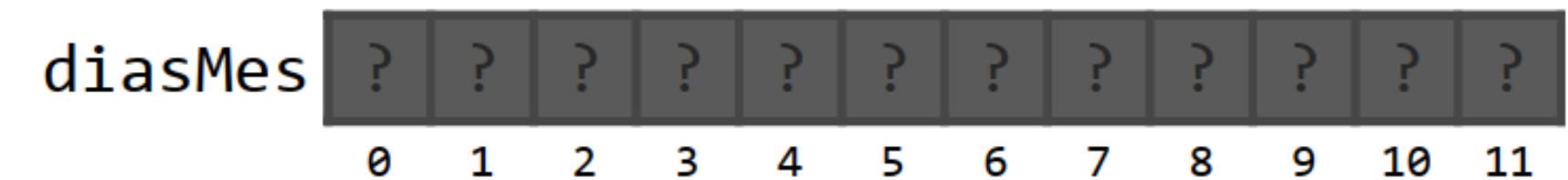
Ejemplos:

tTemp tempMax;

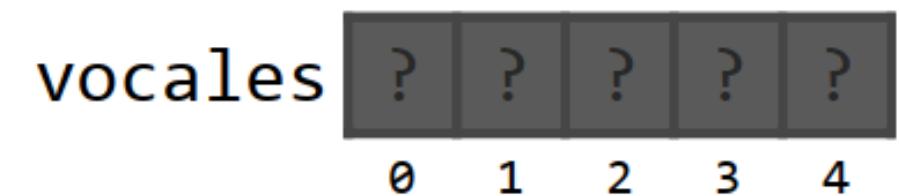
```
typedef double tTemp[7];
typedef short int tDiasMes[12];
typedef char tVocales[5];
typedef double tVentas[31];
```



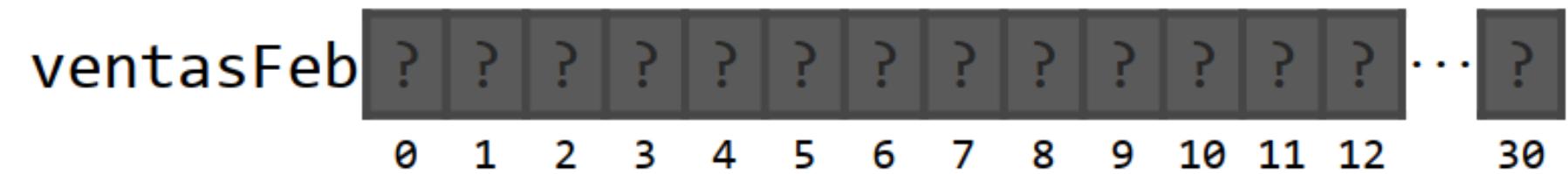
tDiasMes diasMes;



tVocales vocales;



tVentas ventasFeb;



NO se inicializan los elementos automáticamente

Uso de variables arrays

Acceso a los elementos de un array

nombre[índice]

Cada elemento se accede a través de su índice (posición en el array)

`char Vocales[5];`

vocales	'a'	'e'	'i'	'o'	'u'
	0	1	2	3	4

5 elementos, índices de 0 a 4:

`vocales[0] vocales[1] vocales[2] vocales[3] vocales[4]`

Procesamiento de cada elemento:

Como cualquier otra variable del tipo base

```
cout << vocales[4];
vocales[3] = 'o';
if (vocales[i] == 'e') ...
```

Acceso a los elementos de un array

¡IMPORTANTE!

¡No se comprueba si el índice es correcto!

;Es responsabilidad del programador!

```
const int Dim = 100;  
typedef double tVentas[Dim];  
tVentas ventas;
```

¿Cuál es la primera posición del arreglo?

Índices válidos: enteros entre 0 y Dim-1

ventas[0] ventas[1] ventas[2] ... ventas[98] ventas[99]

¿Qué es ventas[100]? ¿O ventas[-1]? ¿O ventas[132]?

¡Memoria de alguna otra variable del programa!



Define los tamaños de los arrays con constantes

Recorrido de arrays

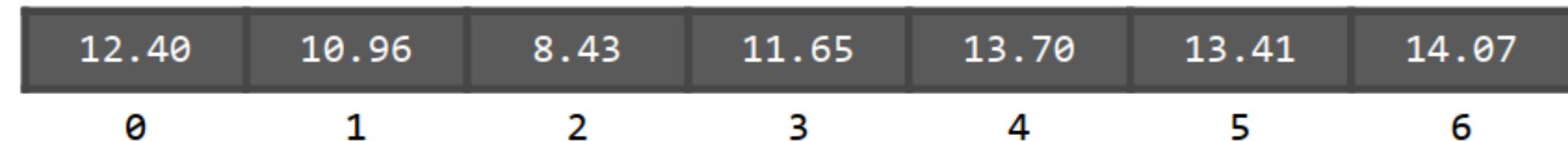
Recorrido de arrays

Arrays: tamaño fijo → Bucle de recorrido fijo (for)

Ejemplo: Media de un array de temperaturas

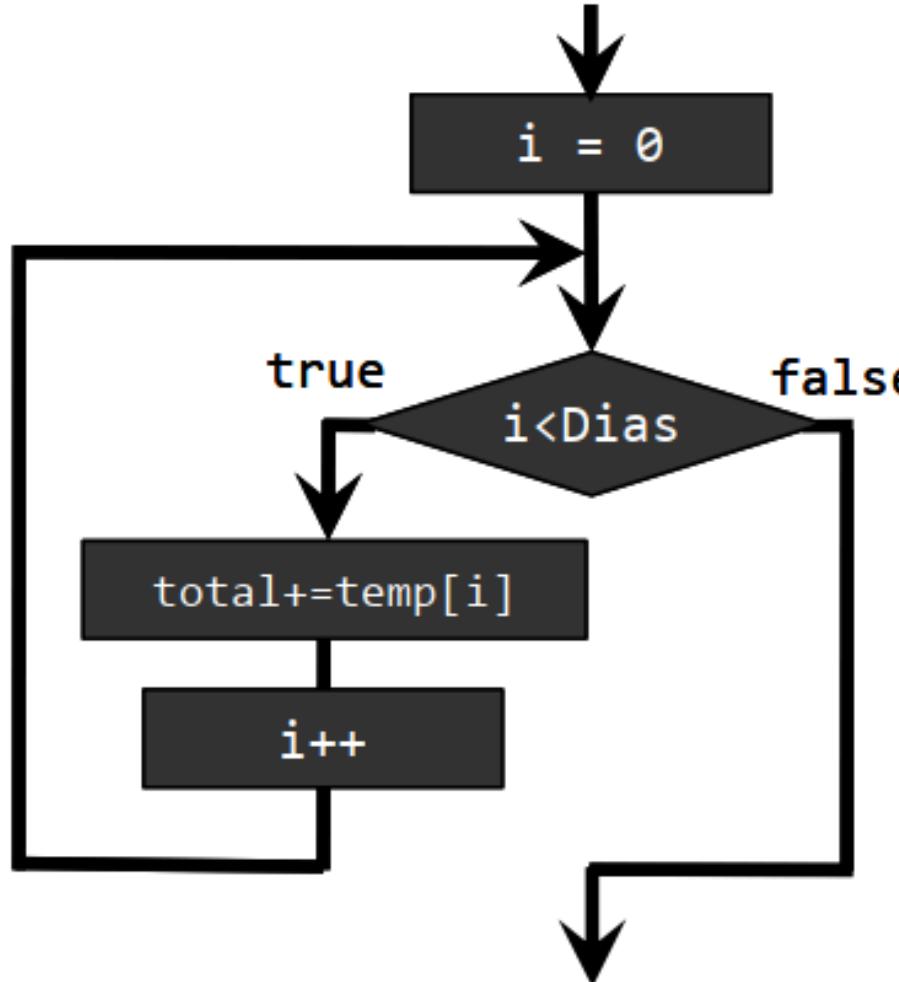
```
const int Dias = 7;  
typedef double *Temp[Dias];  
tTemp temp;  
double media, total = 0;  
  
...  
for (int i = 0; i < Dias; i++) {  
    total = total + temp[i];  
}  
media = total / Dias;
```

Recorrido de arrays



```
tTemp temp;
double media, total = 0;

...
for (int i = 0; i < Dias; i++) {
    total = total + temp[i];
}
```



Memoria

Dias	7
temp[0]	12.40
temp[1]	10.96
temp[2]	8.43
temp[3]	11.65
temp[4]	13.70
temp[5]	13.41
temp[6]	14.07
media	?
total	84.62
i	7

Recorrido de arrays

mediatemp.cpp

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

const int Dias = 7;
typedef double tTemp[Dias];

double media(const tTemp temp);

int main() {
    tTemp temp;
    for (int i = 0; i < Dias; i++) { // Recorrido del array
        cout << "Temperatura del día " << i + 1 << ": ";
        cin >> temp[i];
    }
    cout << "Temperatura media: " << media(temp) << endl;
    return 0;
}
...
```

Aún no lo vimos pero supongan que me traigo un arreglo de temp con valores de algun lado

Los usuarios usan de 1 a 7 para numerar los días
La interfaz debe aproximarse a los usuarios, aunque internamente se usen los índices de 0 a 6

Recorrido de arrays

```
double media(const tTemp temp) {  
    double med, total = 0;  
  
    for (int i = 0; i < Dias; i++) { // Recorrido del array  
        total = total + temp[i];  
    }  
    med = total / Dias;  
  
    return med;  
}
```



Los arrays se pasan a las funciones como constantes
Las funciones no pueden devolver arrays

Arrays de tipos enumerados

```
const int Cuantas = 15;  
enum define un tipo  
de datos enumerado,  
básicamente, es una  
lista de constantes  
con nombres  
simbólicos.  
typedef enum { centimo, dos_centimos, cinco_centimos,  
diez_centimos, veinte_centimos, medio_euro, euro } tMoneda;  
typedef tMoneda tCalderilla[Cuantas];  
string aCadena(tMoneda moneda);  
// Devuelve la cadena correspondiente al valor de moneda  
  
tCalderilla bolsillo; // Exactamente llevo Cuantas monedas  
bolsillo[0] = euro;  
bolsillo[1] = cinco_centimos;  
bolsillo[2] = medio_euro;  
bolsillo[3] = euro;  
bolsillo[4] = centimo;  
...  
for (int moneda = 0; moneda < Cuantas; moneda++)  
    cout << aCadena(bolsillo[moneda]) << endl;
```

Búsqueda en arrays

Búsqueda en arrays

buscaarray.cpp

;Qué día las ventas superaron los 1.000 €?

```
int main()
{ const int Dias = 365; // Año no bisiesto
    double Ventas[Dias];

    // Índice del primer elemento mayor que 1000 (-1 si no hay)
    bool encontrado = false;
    int ind = 0;
    while ((ind < Dias) && !encontrado) { // Esquema de búsqueda
        if (Ventas[ind] > 1000) {
            encontrado = true;
        }
        else {
            ind++;
        }
    }
    if (!encontrado) {
        ind = -1;
    }
    return 0 ;
}
```

Capacidad y copia de arrays

Capacidad de los arrays

La capacidad de un array no puede ser alterada en la ejecución

El tamaño de un array es una decisión de diseño:

- ✓ En ocasiones será fácil (días de la semana)
- ✓ Cuando pueda variar ha de estimarse un tamaño
Ni corto ni con mucho desperdicio (posiciones sin usar)

STL (*Standard Template Library*) de C++:

Colecciones más eficientes cuyo tamaño puede variar

Copia de arrays

No se pueden copiar dos arrays (del mismo tipo) con asignación:

```
array2 = array1; // iii NO COPIA LOS ELEMENTOS !!!
```

Han de copiarse los elementos uno a uno:

```
for (int i = 0; i < N; i++) {  
    array2[i] = array1[i];  
}
```

Arrays no completos

Arrays no completos

Puede que no necesitemos todas las posiciones de un array...

La dimensión del array será el máximo de elementos

Pero podremos tener menos elementos del máximo

Necesitamos un contador de elementos...

```
const int Max = 100;  
double Array[Max];  
  
int contador = 0;
```

Es lo mismo



```
const int Max = 100;  
typedef double tArray[Max];  
tArray lista;  
int contador = 0;
```

contador: indica cuántas posiciones del array se utilizan

Sólo accederemos a las posiciones entre 0 y contador-1

Las demás posiciones no contienen información del programa

Arrays no completos

```
#include <iostream>
using namespace std;
int main() {
    const int Max = 100;
    double lista[Max];

    int contador = 0;
    double valor, med;
    double med, total = 0;
```

```
while ((valor != -1) && (contador < Max)) {
    lista[contador] = valor;
    contador++;
```

```
for (int ind = 0; ind < cont; ind++) {
    total = total + lista[ind];
}
med = total / cont;
}
```

¿Qué le falta a este código ?

Sólo recorremos hasta cont-1

Más sobre arrays

Tipos estructurados

Colecciones o tipos aglomerados

Agrupaciones de datos (elementos):

- ✓ Todos del mismo tipo: *array* o *tabla*
- ✓ De tipos distintos: *estructura*, *registro* o *tupla*

Lista ordenada y finita de elementos

Arrays (tablas)

- Elementos organizados por posición: 0, 1, 2, 3, ...
- Acceso por índice: 0, 1, 2, 3, ...
- Una o varias dimensiones

Estructuras (tuplas, registros)

- Elementos (campos) sin orden establecido
- Acceso por nombre

Inicialización de arrays

Podemos inicializar los elementos de los arrays en la declaración

Asignamos una serie de valores al array:

```
const int DIM = 10;  
typedef int tTabla[DIM];  
tTabla i = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

Se asignan los valores por su orden:

i[0] i[1] i[2] i[3] i[4] ... i[9]
↑ ↑ ↑ ↑ ↑ ↑
1º 2º 3º 4º 5º ... 10º

Si hay menos valores que elementos, los restantes se ponen a 0

```
tTabla i = { 0 }; // Pone todos los elementos a 0
```

Enumerados como índices

```
const int Colores = 3,  
typedef enum { rojo, verde, azul } tRGB;  
typedef int tColor[Colores];  
tColor color;  
  
...  
cout << "Cantidad de rojo (0-255): ";  
cin >> color[rojo];  
cout << "Cantidad de verde (0-255): ";  
cin >> color[verde];  
cout << "Cantidad de azul (0-255): ";  
cin >> color[azul];
```

Recuerda que internamente se asignan enteros a partir de 0
a los distintos símbolos del enumerado

rojo ≡ 0 verde ≡ 1 azul ≡ 2

Implementación de listas

Implementación de listas con arrays

Listas con un número fijo de elementos

Array con el nº de elementos como dimensión

```
const int NUM = 100;  
typedef double tLista[NUM]; // Exactamente 100 double  
tLista lista;
```

Recorrido de la lista:

```
for (int i = 0; i < NUM; i++) {  
    ...
```

Búsqueda en la lista:

```
while ((i < NUM) && !encontrado) {  
    ...
```

Listas con un número variable de elementos

Array con un máximo de elementos + Contador de elementos

```
const int MAX = 100;  
typedef double tLista[MAX]; // Hasta 100 elementos  
tLista lista;  
int contador = 0; // Se incrementa al insertar
```

Recorrido de la lista:

```
for (int i = 0; i < contador; i++) {  
    ...
```

Búsqueda en la lista:

```
while ((i < contador) && !encontrado) {  
    ...
```

¿Array y contador por separado? → Estructuras

Cadenas de caracteres

Cadenas de caracteres

Arrays de caracteres

Cadenas: secuencias de caracteres de longitud variable

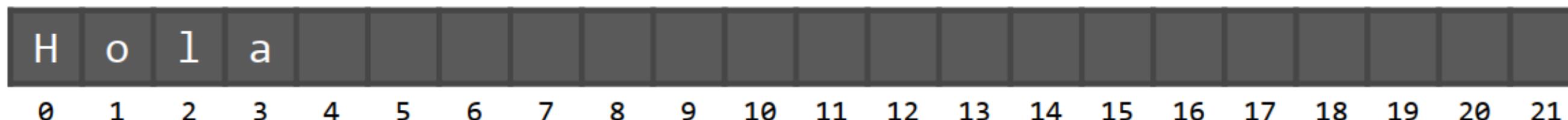
"Hola" "Adiós" "Supercalifragilístico" "1234 56 7"

Variables de cadena: contienen secuencias de caracteres

Se guardan en arrays de caracteres: tamaño máximo (dimensión)

No todas las posiciones del array son relevantes:

- ✓ Longitud de la cadena: número de caracteres, desde el primero, que realmente constituyen la cadena:



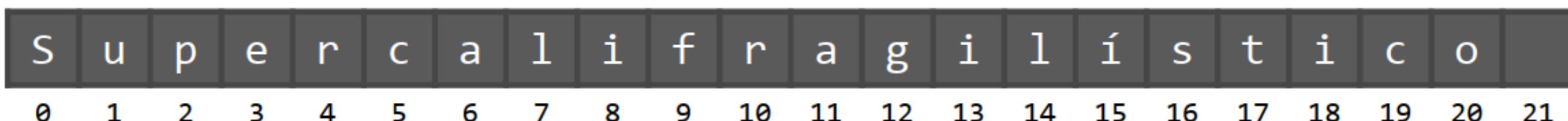
Longitud actual: 4

Cadenas de caracteres

Longitud de la cadena



Longitud: 5



Longitud: 21

Necesidad de saber dónde terminan los caracteres relevantes:

- ✓ Mantener la longitud de la cadena como dato asociado
- ✓ Colocar un carácter de terminación al final (*centinela*)



Cadenas de caracteres

Cadenas de caracteres en C++

Dos alternativas para el manejo de cadenas:

- ✓ Cadenas al estilo de C (*terminadas en nulo*)
- ✓ Tipo **string**

Cadenas al estilo de C

Anexo del tema

- ✓ Arrays de tipo **char** con una longitud máxima
- ✓ Un último carácter especial al final: '\0'

Tipo **string**

- ✓ Cadenas más sofisticadas
- ✓ Sin longitud máxima (gestión automática de la memoria)
- ✓ Multitud de funciones de utilidad (biblioteca **string**)

Cadenas de caracteres de tipo string

Cadenas de caracteres de tipo string

El tipo string

- ✓ El tipo asume la responsabilidad de la gestión de memoria
- ✓ Define operadores sobrecargados (+ para concatenar)
- ✓ Cadenas más eficientes y seguras de usar

Biblioteca string

Requiere establecer el espacio de nombres a std

- ✓ Se pueden inicializar en la declaración
- ✓ Se pueden copiar con el operador de asignación
- ✓ Se pueden concatenar con el operador +
- ✓ Multitud de funciones de utilidad

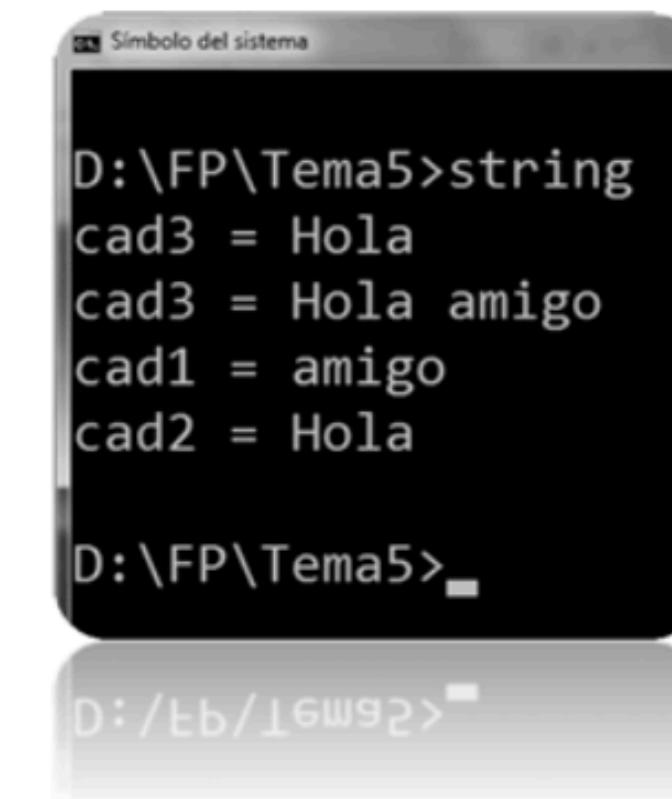
Cadenas de tipo string

string.cpp

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

int main() {
    string cad1("Hola");      // inicialización
    string cad2 = "amigo";   // inicialización
    string cad3;
    cad3 = cad1;             // copia
    cout << "cad3 = " << cad3 << endl;
    cad3 = cad1 + " ";       // concatenación
    cad3 += cad2;            // concatenación
    cout << "cad3 = " << cad3 << endl;
    cad1.swap(cad2);         // intercambio
    cout << "cad1 = " << cad1 << endl;
    cout << "cad2 = " << cad2 << endl;

    return 0;
}
```



Cadenas de tipo string

Longitud de la cadena:

`cadena.length()` o `cadena.size()`

Se pueden comparar con los operadores relacionales:

`if (cad1 <= cad2) { ... }`

Acceso a los caracteres de una cadena:

✓ Como array de caracteres: `cadena[i]`

Sin control de acceso a posiciones inexistentes del array

Sólo debe usarse si se está seguro de que el índice es válido

✓ Función `at(indice)`: `cadena.at(i)`

Error de ejecución si se accede a una posición inexistente

E/S con cadenas de tipo string

- ✓ Se muestran en la pantalla con `cout <<`
- ✓ Lectura con `cin >>`: termina con espacio en blanco (inc. Intro)
El espacio en blanco queda pendiente
- ✓ Descartar el resto de los caracteres del búfer:
`cin.sync();`
- ✓ Lectura incluyendo espacios en blanco:
`getline(cin, cadena)`
Guarda en la *cadena* los caracteres leídos hasta el fin de línea
- ✓ Lectura de archivos de texto:
Igual que de consola; `sync()` no tiene efecto
`archivo >> cadena` `getline(archivo, cadena)`

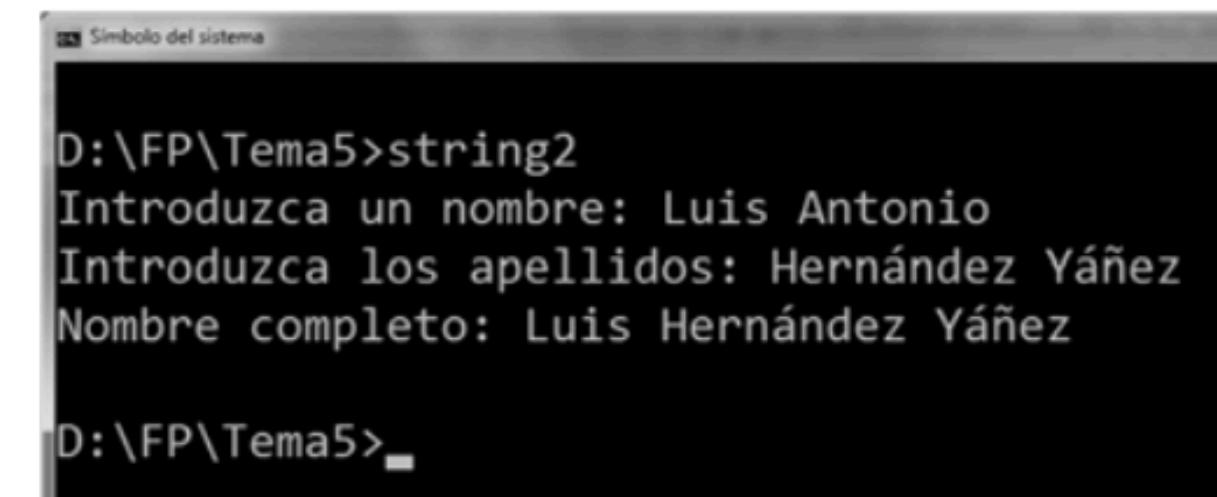
E/S con cadenas de tipo string

string2.cpp

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

int main() {
    string nombre, apellidos;
    cout << "Introduzca un nombre: ";
    cin >> nombre;
    cout << "Introduzca los apellidos: ";
    cin.sync();
    getline(cin, apellidos);
    cout << "Nombre completo: " << nombre << " "
        << apellidos << endl;

    return 0;
}
```



Operaciones con cadenas de tipo string

- ✓ *cadena.substr(posición, longitud)*

Subcadena de *longitud* caracteres desde *posición*

```
string cad = "abcdefg";  
cout << cad.substr(2, 3); // Muestra cde
```

- ✓ *cadena.find(subcadena)*

Posición de la primera ocurrencia de *subcadena* en *cadena*

```
string cad = "Olala";  
cout << cad.find("la"); // Muestra 1
```

(Recuerda que los arrays de caracteres comienzan con el índice 0)

- ✓ *cadena.rfind(subcadena)*

Posición de la última ocurrencia de *subcadena* en *cadena*

```
string cad = "Olala";  
cout << cad.rfind("la"); // Muestra 3
```

Operaciones con cadenas de tipo string

- ✓ *cadena.erase(i, num)*

Elimina *num* caracteres a partir de la posición *ini*

```
string cad = "abcdefghijklm";  
cad.erase(3, 4); // cad ahora contiene "abch"
```

- ✓ *cadena.insert(i, cadena2)*

Inserta *cadena2* a partir de la posición *ini*

```
string cad = "abcdefghijklm";  
cad.insert(3, "123"); // cad ahora contiene "abc123defgh"
```

<http://www.cplusplus.com/reference/string/string/>

Recorrido de arrays

Recorrido de arrays

Esquema de recorrido

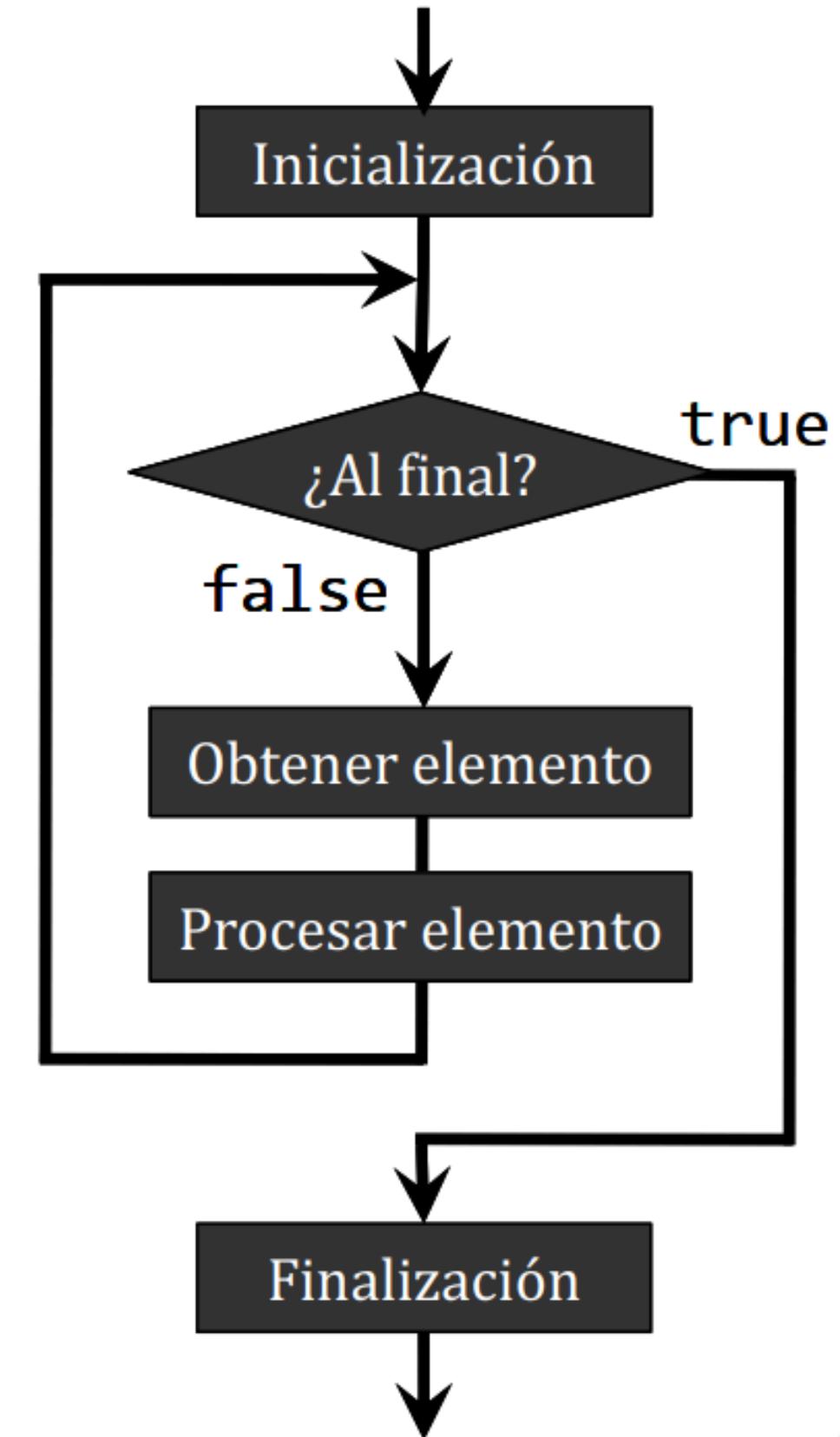
Inicialización

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

Una variable **centinela** es un tipo interruptor, puede ser calculada o sólo capturada, y dependiendo del valor que tome, le indica al ciclo (a través de la condición) si debe continuar la ejecución de instrucciones, o salir del mismo.

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

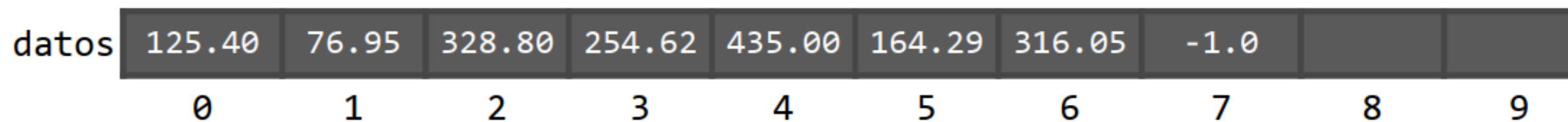
Recorrido de arrays no completos - con centinela

No todas las posiciones del array están ocupadas

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

```

¿Sigue recorriendo el
arreglo?



```
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);
```

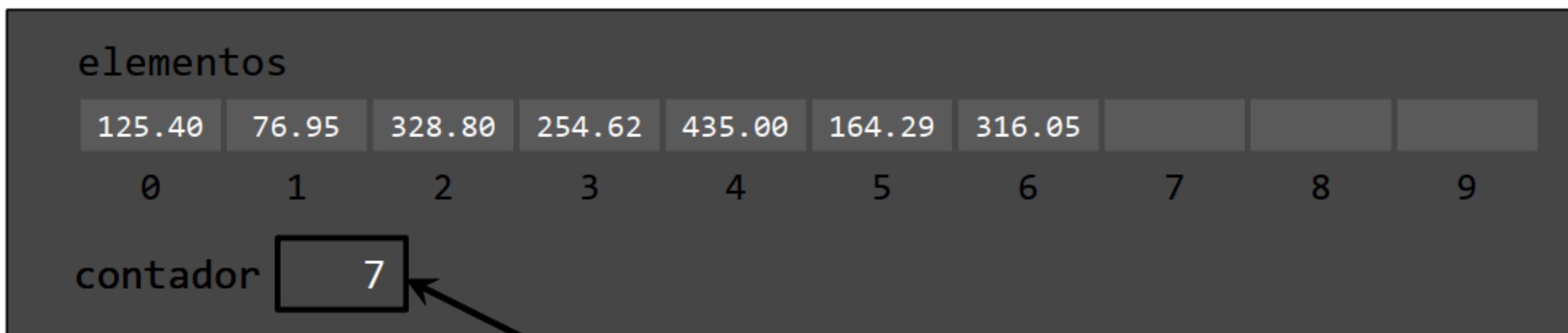
Recorrido de arrays

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



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...
}
```

Ejemplos

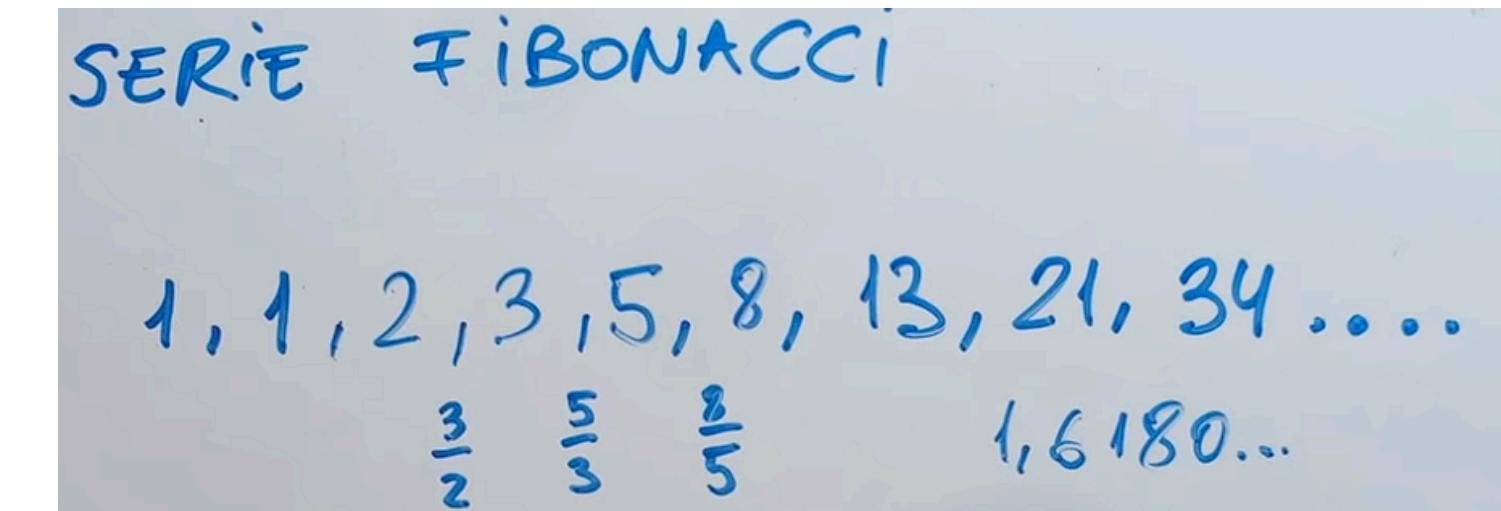
Ejemplos

fibonacci.cpp

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] << " ";  
}
```

Suma el ultimo y el anterior



Ejemplos

Tarea para casa

Cuenta de valores con k dígitos

Función que devuelve el número de dígitos de un entero:

```
int main() {    int dato = 168;  
    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;  
}
```

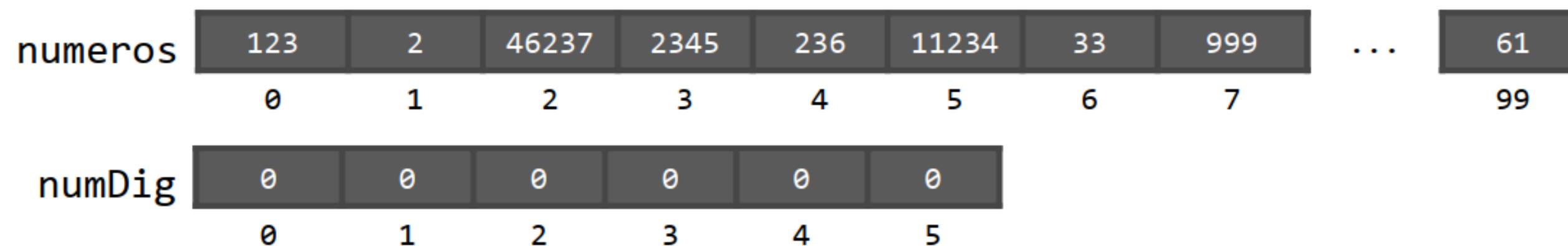
**Implementar esto mismo,
pero para un Arreglo de n posiciones**

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 };
```



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
```

Ejemplos

digitos.cpp

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 << " díg. = " << numDig[i]
        << endl;
}
return 0;
}

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

Más ejemplos de manejo de arrays

Para practicar en casa

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^{er} 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

Manejo de vectores

vectores.cpp

```
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

```
#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").

```
#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));
            if (posEnCad2 == -1) {
                sonAnagramas = false;
            }
            else {
                swap(cad2[posEnCad2], cad2.back());
                cad2.pop_back();
            }
            numCar++;
        }
    }
}
```

```
    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 de varias dimensiones

Varios tamaños en la declaración: cada uno con sus corchetes

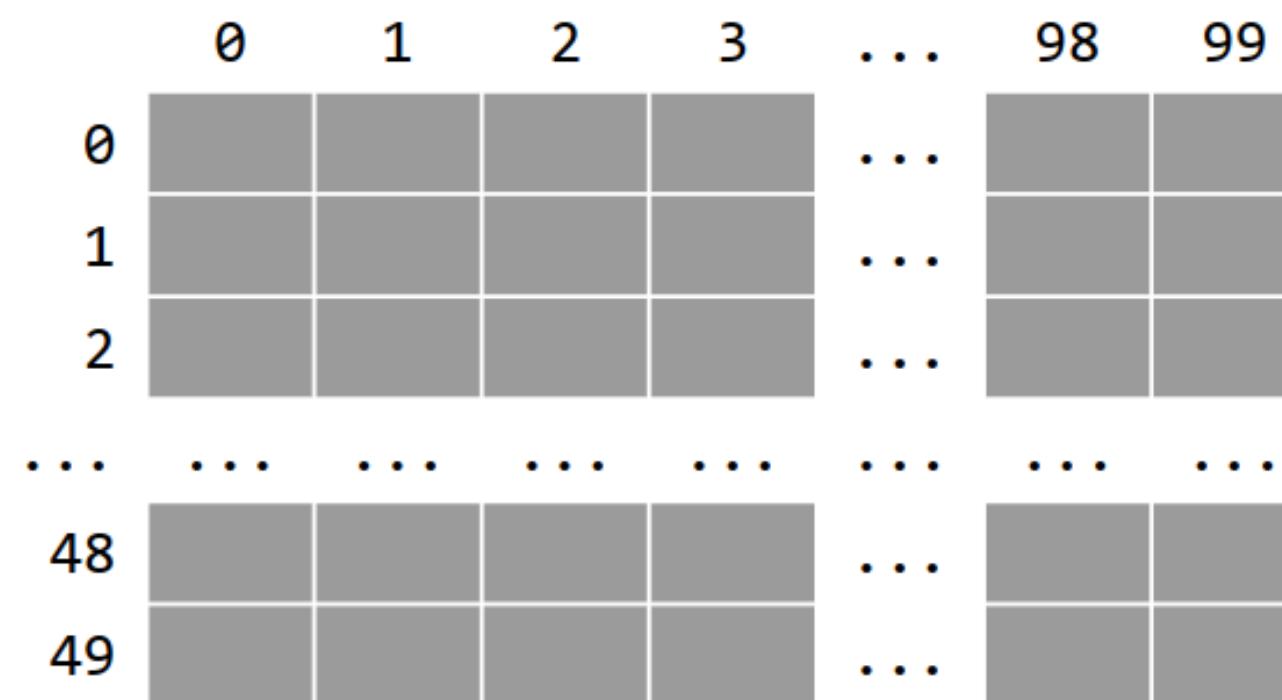
```
typedef tipo_base nombre[tamaño1][tamaño2]...[tamañoN];
```

Varias dimensiones, tantas como tamaños se indiquen

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

```
tMatriz matriz;
```

Tabla bidimensional de 50 filas por 100 columnas:



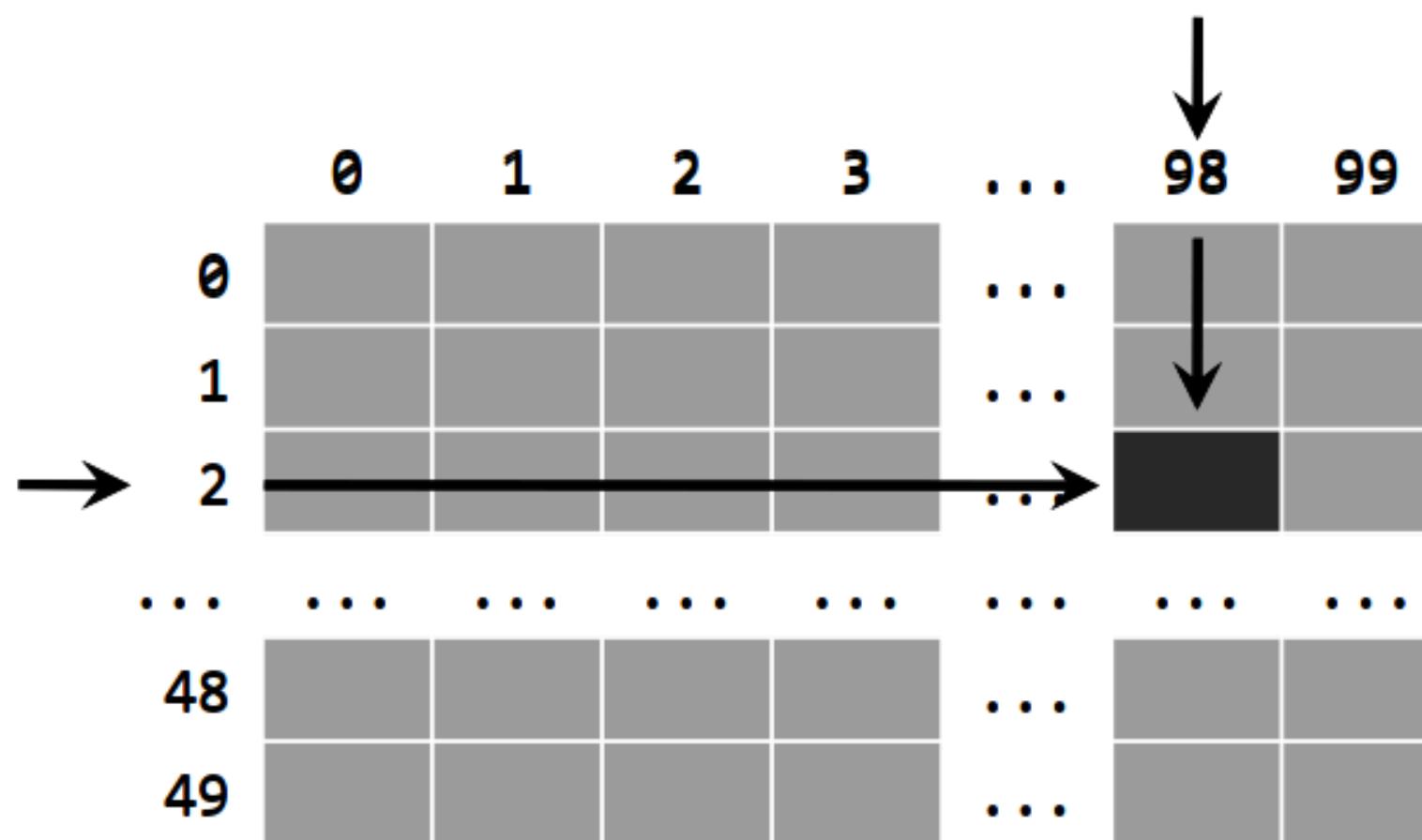
Arrays multidimensionales

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];
```



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; // Nº 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`

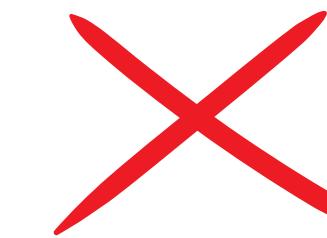
Arrays multidimensionales

temp.cpp

*|UTN

```
int main() {
    const int MaxDias = 31;
    const int MED = 2; // Nº 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;

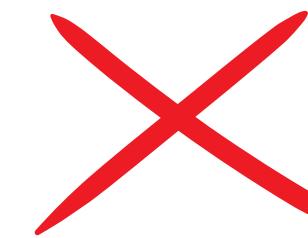
    archivo.open("temp.txt");
    if (!archivo.is_open()) {
        cout << "No se ha podido abrir el archivo!" << endl;
    }
    else {
        archivo >> min >> max;
        // El archivo termina con -99 -99
        ...
    }
}
```



Arrays multidimensionales

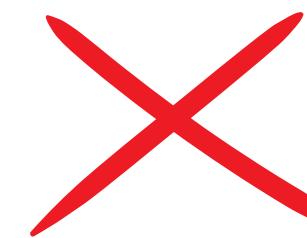
```
while (!((min == -99) && (max == -99))
      && (dia < MaxDias)) {
    temp[dia][0] = min;
    temp[dia][1] = max;
    dia++;
    archivo >> min >> max;
}
archivo.close();
for (int i = 0; i < dia; i++) {
    tMinMedia = tMinMedia + temp[i][0];
    if (temp[i][0] < tMinAbs) {
        tMinAbs = temp[i][0];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

¿Podría tener más dimensiones ?

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

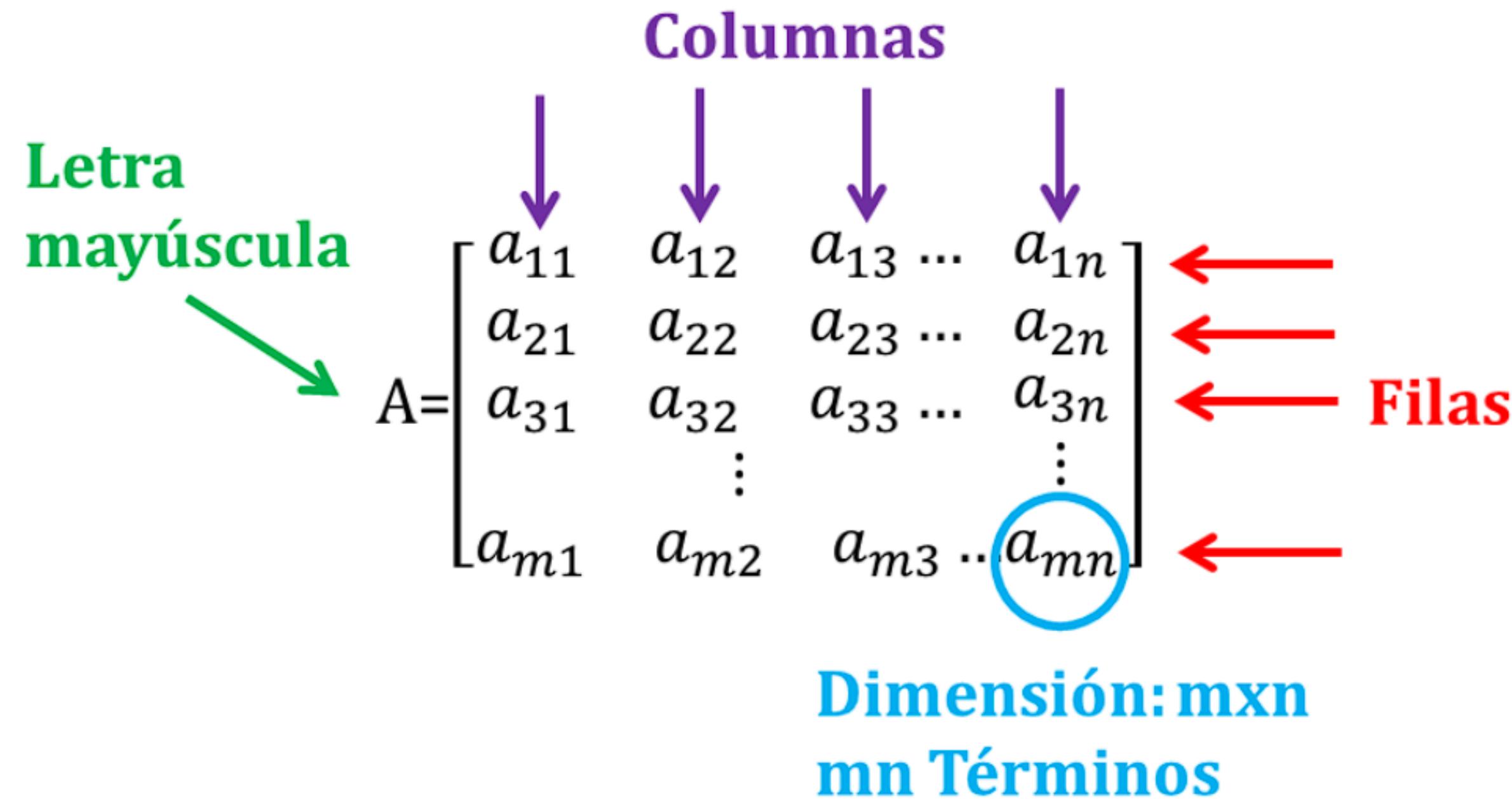


Si hay menos valores que elementos, el resto se inicializan a cero

Inicialización a cero de todo el array:

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

Elementos de una matriz



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

Días											
	0	1	2	3	4	...	28	29	30		
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

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];
    }
}
```

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

```
bool encontrado = false;                                Primer valor > umbral
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) { ... }
```