ARRAYS en C++

Programación II

Facultad de Estudios Estadísticos

Universidad Complutense de Madrid

Olga Marroquín Alonso

ARRAYS EN C++: ARRAYS UNIDIMENSIONALES

❖ Un **array** es una colección **homogénea** de datos relacionados que se almacenan en memoria de manera contigua con el mismo nombre.

↳**Ejemplo:** Ventas de cada día de la semana o temperaturas mínimas de cada día del mes.

- En lugar de declarar N variables…

ventasL	ventasM	ventasX	ventasJ	ventasV	ventasS	ventasD
325	490	123	654	284	920	0

…declaramos una tabla de N valores.

ventas	325	490	123	654	284	920	0
índices →	0	1	2	3	4	5	6

❖ Cada elemento del array se encuentra en una casilla identificada por un **índice** (posición del elemento).

- Los índices son enteros positivos ➔ el primer elemento se encuentra en la casilla 0.
- A cada elemento se accede a través de su índice.

↳**Ejemplo:** `ventas[1]` accede al segundo elemento, que contiene el valor `490`.

DECLARACIÓN DE UN ARRAY

❖ La declaración de un array es de la forma siguiente:

```
tipoElementos nombre[capacidad];
```

- **`tipoElementos:`** Tipo de los valores almacenados en el array.
- **`nombre:`** Identificador del array.
- **`capacidad:`** Cantidad de casillas reservadas para el array.
  - → La capacidad de un array no puede ser alterada durante la ejecución.
  - → Debe ser un entero constante conocido en tiempo de compilación.

↳**Ejemplo:** `int ventas[7];`

ARRAYS EN C++: ARRAYS UNIDIMENSIONALES

INICIALIZACIÓN DE UN ARRAY

❖ Los arrays pueden ser inicializados en la declaración.

Opción 1: `tipoElementos nombre[capacidad] = {valor0, valor1,…, valorN};`

- La cantidad de valores entre llaves debe ser menor o igual que la capacidad del array.
  - → Si es menor que la capacidad, el resto de casillas se quedan sin inicializar.

Opción 2: `tipoElementos nombre[] = {valor0, valor1,…, valorN};`

- La capacidad no está indicada, así que se reservan tantas casillas como elementos hay entre llaves.

Ejemplo: `int ventas[7] = {325, 490, 123, 654, 284, 920, 0};`

❖ Los arrays pueden ser inicializados durante su recorrido con un bucle.

Ejemplo:

```
const int CANTIDAD = 7 ;
int ventas[CANTIDAD];
for (int i=0; i<CANTIDAD; i=i+1)
    ventas[i] = 0;
```

❖ Cada elemento es accesible a través de su índice y puede utilizarse como cualquier variable de su tipo.

Importante: No se comprueba si el índice es correcto. ¡ES RESPONSABILIDAD DEL PROGRAMADOR!

Ejemplo:

```
cout << ventas[6]; ventas[0] = 100;
if (ventas[3] < 50)
    cout << "Ventas inferiores al límite";
```

❖ **NO** se puede copiar el contenido de un array en otro del mismo tipo con el operador de asignación.

- **NO** existen funciones de E/S ni operadores de comparación en bloque.

ARRAYS EN C++: ARRAYS UNIDIMENSIONALES

RECORRIDO DE UN ARRAY

Ejemplo: Cálculo de la media de ventas.

```
const int CANTIDAD = 7 ;
int ventas[CANTIDAD];
// Inicialización del array ventas
int total = 0; double media;
for (int i=0; i<CANTIDAD; i=i+1)
    total = total + ventas[i];
media = total / CANTIDAD;
```

Ejemplo: Generar una copia exacta de un array.

→ No se puede generar una copia de un array empleando la asignación.

```
copiaArray = array  ¡NO REALIZA LA COPIA!
```

→ Debemos copiar los elementos del array uno a uno.

```
const int CANTIDAD = 7;
int ventas[CANTIDAD];
// Inicialización del array ventas
int copiaVentas[CANTIDAD];
for (int i=0; i<CANTIDAD; i=i+1)
    copiaVentas[i] = ventas[i];
```

ARRAYS Y FUNCIONES

❖ Para declarar un array como parámetro de una función, podemos emplear la sintaxis siguiente:

```
tipoRet nombreFun(listaParam, tipoElementos nombre[capacidad])
```

ARRAYS EN C++: ARRAYS UNIDIMENSIONALES

❖ Para invocar una función que tiene como parámetro un array, podemos emplear la sintaxis siguiente:

```
tipoElementos nombreR[capacidad];
tipoRet variable = nombreFun(listaParamR, nombreR);
```

↳**Ejemplo:**

```
const int CANTIDAD = 10;
void initArrayCero(int vector[CANTIDAD]) {
    for (int i=0; i<CANTIDAD; i=i+1)
        vector[i]=0;
}
int ventas[CANTIDAD];
initArrayCero(ventas);
```

❖ Cuando se invoca una función que tiene un array como parámetro, la función recibe la dirección de memoria del primer elemento del array.

- Las modificaciones del array dentro de la función son permanentes ➔ **simulación de paso de paramétro por referencia** (no es necesario poner `&` en la declaración del parámetro).
- Las modificaciones en el array se impiden usando el modificador `const` en la declaración.

```
tipoRet nombreFun(listaParam, const tipoElementos nombre[capacidad])
```

→ El parámetro se trata como un array de constantes.

→ Si alguna instrucción de la función intenta modificar un elemento del array, entonces se produce un error de compilación.

❖ C++ **NO** permite que las funciones retornen arrays completos.

ARRAYS EN C++: ARRAYS BIDIMENSIONALES

❖ Los arrays estudiados se denominan **arrays unidimensionales**. Los **arrays multidimensionales** tienen varias dimensiones, tantas como capacidades se indican.

```
tipoElementos nombre[capacidad1][capacidad2]…[capacidadN];
```

- Los arrays multidimensionales más utilizados son los **arrays bidimensionales** (matrices).

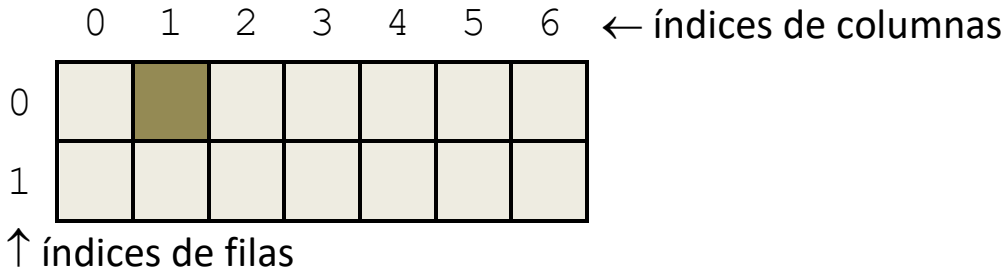
```
tipoElementos nombre[capacidad1][capacidad2];
```

- Los elementos se almacenan en posiciones contiguas de memoria, comenzando por los elementos de la primera fila, luego los elementos de la segunda fila y así sucesivamente.

↳**Ejemplo:** Temperaturas mínimas y máximas de cada día de la semana.

- Las temperaturas mínimas se almacenan en la primera fila y las máximas en la segunda.
- Los días de la semana están representados por las columnas.

```
float matriz[2][7];
```



❖ Cada elemento del array se encuentra en una casilla identificada por dos **índices**, uno correspondiente a la fila y otro correspondiente a la columna.

↳**Ejemplo:** `matriz[0][1]` accede a la temperatura mínima del martes (segundo día de la semana).

❖ Los arrays bidimensionales también pueden ser inicializados en la declaración.

```
tipoElementos nombre[capacidad1][capacidad2] = {valor0,…, valorN};
```

- La cantidad de valores entre llaves debe ser menor o igual que la capacidad total del array.

ARRAYS EN C++: ARRAYS BIDIMENSIONALES

→ Si es menor que la capacidad, el resto de casillas se quedan sin inicializar.

• Los valores se asignan en el orden en que aparecen, completando la matriz fila a fila.

Ejemplo: `float matriz[2][7] = {-2,0,-1,2,3,0,1, 12,13,14,10,17,9,9};`

	0	1	2	3	4	5	6
0	-2	0	-1	2	3	0	1
1	12	13	14	10	17	9	9

❖ Los arrays pueden ser inicializados durante su recorrido con un par de bucles.

Ejemplo:

```
const int NUM_FILAS = 2;
const int NUM_COLUMNAS = 7;
int matriz[NUM_FILAS][NUM_COLUMNAS];
for (int i=0; i<NUM_FILAS; i=i+1)
    for (int j=0; i<NUM_COLUMNAS; j=j+1)
        matriz[i][j] = 0;
```

RECORRIDO DE UN ARRAY BIDIMENSIONAL

Ejemplo: Cálculo de la media de temperaturas.

```
const int NUM_FILAS = 2;
cons tint NUM_COLUMNAS = 7;
int matriz[NUM_FILAS][NUM_COLUMNAS];
// Inicialización del array matriz
float total = 0; double media;
```

ARRAYS EN C++: ARRAYS BIDIMENSIONALES

```
for (int i=0; i<NUM_FILAS; i=i+1)
    for (int j=0; i<NUM_COLUMNAS; j=j+1)
        total = total + matriz[i][j];
media = total / (NUM_FILAS*NUM_COLUMNAS);
```

ARRAYS BIDIMENSIONALES Y FUNCIONES

❖ Para declarar un array como parámetro de una función, podemos emplear la sintaxis siguiente:

```
tipoRet nombreFun(listaParam, tipoElementos nombre[capacidad1][capacidad2])
```

❖ Para invocar una función que tiene como parámetro un array, podemos emplear la sintaxis siguiente:

```
tipoElementos nombreR[capacidad1][capacidad2];
tipoRet variable = nombreFun(listaParamR, nombreR);
```

Ejemplo:

```
const int NUM_FILAS = 2;
cons tint NUM_COLUMNAS = 7;
void initArrayCero(int tabla[NUM_FILAS][NUM_COLUMNAS]) {
    for (int i=0; i<NUM_FILAS; i=i+1)
        for (int j=0; i<NUM_COLUMNAS; j=j+1)
            tabla[i][j] = 0;
}
int matriz[NUM_FILAS][NUM_COLUMNAS];
initArrayCero(matriz);
```

❖ C++ **no** permite que las funciones retornen arrays completos.

Observación: Las consideraciones realizadas para el tratamiento de los arrays unidimensionales son válidas para el tratamiento de los arrays bidimensionales.