

Arrays multidimensionales



Arrays multidimensionales

Arrays de varias dimensiones

Podemos indicar varios tamaños en la declaración de un array.
Cada uno en su par de corchetes.



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

El array tendrá tantas como tamaños se indiquen.

Podemos definir tantas dimensiones como necesitemos.

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

Necesitamos tantos índices como dimensiones:

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



Arrays multidimensionales

Inicialización de arrays multidimensionales

```
typedef double tMatriz[3][4][2][3];  
tMatriz matriz;  
for (int n1 = 0; n1 < 3; ++n1)  
    for (int n2 = 0; n2 < 4; ++n2)  
        for (int n3 = 0; n3 < 2; ++n3)  
            for (int n4 = 0; n4 < 3; ++n4)  
                matriz[n1][n2][n3][n4] = 0;
```

	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



Arrays multidimensionales

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;
```

Anidamiento de bucles 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



Ventas de todos los meses de un año (bisiesto)



Ejemplo: Ventas



Ventas de todos los meses de un año (bisiesto)

Array bidimensional de meses y días:

```
const int Meses = 12;  
const int MaxDias = 31;  
typedef double tVentas[Meses][MaxDias];  
typedef int tDiasMes[Meses];
```

```
tVentas ventas; // Ventas de todo el año  
tDiasMes diasMes;
```

```
inicializa(diasMes); // Asigna a cada mes su nº de días  
// Leemos las ventas de cada día del año...  
for (int mes = 0; mes < Meses; ++mes)  
    for (int dia = 0; dia < diasMes[mes]; ++dia) {  
        ent >> ventas[mes][dia];  
    }
```



Ejemplo: Ventas



Ventas de todos los meses de un año (bisiesto)

		Días									
		0	1	2	3	4	...	29	30	31	
Meses	0	201	125	234	112	156	...	234	543	667	
	1	323	231	675	325	111	...	342			
	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



Ventas diarias de cada sucursal

Para 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 MESES = 12;
const int DIAS = 31;
const int SUCURSALES = 4;
typedef double tVMes[DIAS][SUCURSALES];
typedef struct {
    tVMes ventas;
    int dias;
} tVentasMes;
typedef tVentasMes
    tVentaAnual[MESES];
tVentaAnual anual;
```

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



Ejemplo: Ventas



Ventas diarias de cuatro sucursales

```
...
typedef double tVMes[DIAS][SUCURSALES];
typedef struct {
    tVMes ventas;
    int dias;
} tVentasMes;
typedef tVentasMes tVentaAnual[MESES];
```

Cálculo del total de las ventas del año:

```
double totalVentas(const 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];
    return total;
}
```



Ejemplo: Ventas



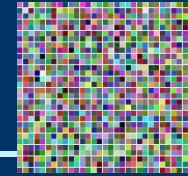
Primer valor por encima de un umbral

```
double umbral;
cout << "Valor umbral: ";
cin >> 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) ... // En anual[mes].ventas[dia][suc]
```

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



Ejemplo: Aplicar filtro de convolución



Se quiere desarrollar un subprograma `convolucion()` que, dada una imagen y una máscara de convolución (matriz de 3x3 enteros), obtenga la imagen resultante de aplicar el filtro.

Aplicar un filtro de convolución consiste en sustituir cada valor de la imagen `imagen[f][c]`, por la media de los 3x3 vecinos, ponderada por los pesos de la máscara centrada en ese elemento.

```
typedef uint tM3x3[3][3]; // matriz 3x3 enteros (máscara)

void convolucion(tImagen const& imagen, tM3x3 const filtro,
                tImagen & /*sal*/ resultado);
```



Ejemplo: Aplicar filtro de convolución

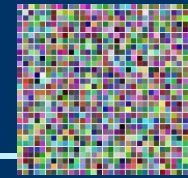
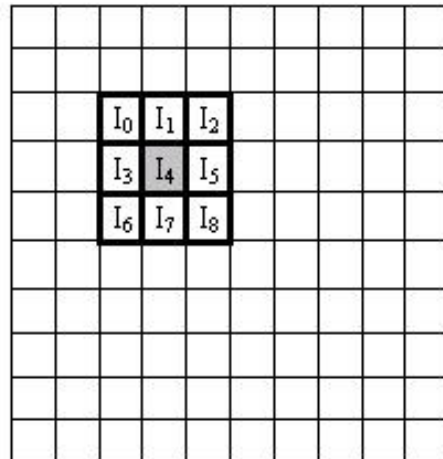


Imagen de entrada



Ventana de convolución

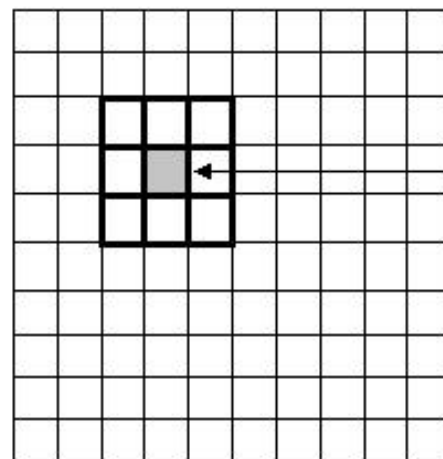
I_0	I_1	I_2
I_3	I_4	I_5
I_6	I_7	I_8

Máscara de convolución

M_0	M_1	M_2
M_3	M_4	M_5
M_6	M_7	M_8

×

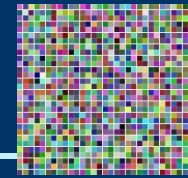
Imagen de salida



$$\begin{aligned} \text{Nuevo píxel} = & I_0 \times M_0 + I_1 \times M_1 + I_2 \times M_2 + \\ & I_3 \times M_3 + I_4 \times M_4 + I_5 \times M_5 + \\ & I_6 \times M_6 + I_7 \times M_7 + I_8 \times M_8 \end{aligned}$$



Ejemplo: Aplicar filtro de convolución

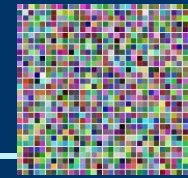


```
void convolucion(tImagen const& imagen, tM3x3 const filtro,
                tImagen & /*sal*/ resultado) {
    // iniciar dimensiones
    resultado.numFilas = imagen.numFilas;
    resultado.numCols = imagen.numCols;
    // copiar el borde
    ... →
    // calcular la suma del filtro
    int sum = suma(filtro);
    // rellenar el resto de la imagen calculando la convolución por posición
    int ultF = imagen.numFilas-1;
    int ultC = imagen.numCols-1;
    for (usint f = 1; f < ultF; ++f)
        for (usint c = 1; c < ultC; ++c)
            resultado.bmp[f][c] = convolucion(imagen,{f,c},filtro) / sum;
}
tRGB convolucion(tImagen const& imagen, tCoor pos, tM3x3 const filtro)
```

Diagrammatic annotations: A yellow arrow points from the `{f,c}` argument in the `convolucion` call inside the loop to the `tCoor pos` parameter of the `convolucion` function signature below. Another yellow arrow points from the `convolucion` function call inside the loop to the `tCoor pos` parameter of the `convolucion` function signature below.



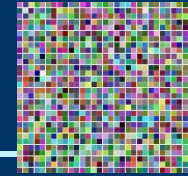
Ejemplo: Aplicar filtro de convolución



```
void convolucion(tImagen const& imagen, tM3x3 const filtro,
                tImagen & /*sal*/ resultado) {
    // iniciar dimensiones
    ...
    // copiar el borde
    int ultF = imagen.numFilas-1; // primera y última fila
    for (usint c = 0; c < imagen.numCols; ++c) {
        resultado.bmp[0][c] = imagen.bmp[0][c];
        resultado.bmp[ultF][c] = imagen.bmp[ultF][c];
    }
    int ultC = imagen.numCols-1; // primera y última columna
    for (usint f = 1; f < ultF; ++f) {
        resultado.bmp[f][0] = imagen.bmp[f][0];
        resultado.bmp[f][ultC] = imagen.bmp[f][ultC];
    }
    // calcular la suma del filtro
    // rellenar el resto de la imagen calculando la convolución por posición
}
```



Ejemplo: Aplicar filtro de convolución



```
const int incF[] = {-1, -1, -1, 0, 0, 0, 1, 1, 1}; // vecinos 3x3
const int incC[] = {-1, 0, 1, -1, 0, 1, -1, 0, 1};

tRGB convolucion(tImagen const& imagen, tCoor pos,
                 tM3x3 const filtro) {
    tRGB rgb = { 0, 0, 0 };
    tCoor posCF = { 1, 1 }; // centro del filtro
    for (usint d = 0; d < 9; ++d)
        rgb += imagen.bmp[pos.fila + incF[d]][pos.col + incC[d]] *
               filtro[posCF.fila + incF[d]][posCF.col + incC[d]];
    return rgb;
}
```



Ejemplo: Aplicar filtro de convolución



```
usint suma(tM3x3 const filtro) {  
    usint sum = 0;  
    for (usint f = 0; f < 3; f++)  
        for (usint c = 0; c < 3; c++)  
            sum += filtro[f][c];  
    return sum;  
}
```

```
tRGB operator * (tRGB rgb, int mul) {  
    rgb.rojo *= mul;  
    rgb.verde *= mul;  
    rgb.azul *= mul;  
    return rgb;  
}
```

Parámetro por valor:
variable local iniciada
con el argumento de llamada



Ejemplo: Aplicar filtro de convolución



```
tRGB operator / (tRGB rgb, int div) {  
    rgb.rojo /= div;  
    rgb.verde /= div;  
    rgb.azul /= div;  
    return rgb;  
}
```

```
void operator += (tRGB & /*ent/sal*/ rgb, tRGB const& rgb2) {  
    rgb.rojo += rgb2.rojo;  
    rgb.verde += rgb2.verde;  
    rgb.azul += rgb2.azul;  
}
```






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.

