Principios de Programación

Arreglos multidimensionales

Matrices en C: declaración, inicialización, acceso por fila y columna, recorridos con bucles anidados y errores frecuentes.

Recordando la semana anterior

Base previa

- Ya se trabajó con arreglos unidimensionales
- Ya se accedió a posiciones con un índice
- Ya se recorrieron colecciones con `for`

Necesidad de una tabla de datos

SIN MATRIZ

```c
int notas_ana[4];
int notas_bruno[4];
int notas_carla[4];

printf("Ana, materia 1: ");
scanf("%d", &notas_ana[0]);
printf("Bruno, materia 1: ");
scanf("%d", &notas_bruno[0]);
// ...
```

Problema

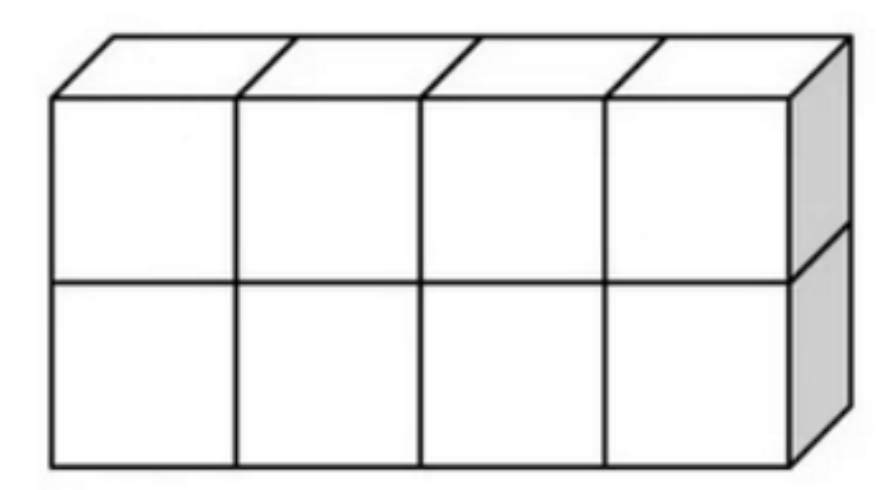
- Cada estudiante queda en un arreglo distinto
- El patrón se repite al agregar más estudiantes
- Procesar por fila y por columna resulta incómodo

Pregunta

Si las filas representan estudiantes y las columnas representan materias, conviene modelar la información como una sola tabla.

Necesidad de una tabla de datos

	A	B	C	D	E	F
1		Nota 1	Nota 2	Nota 3	Nota 4	Nota N
2	Pedro	12	2	23	6	23
3	Carlos	32	3	23	2	54
4	Luisina	54	4	43	6	56
5	Alumno N	7	6	41	1	77

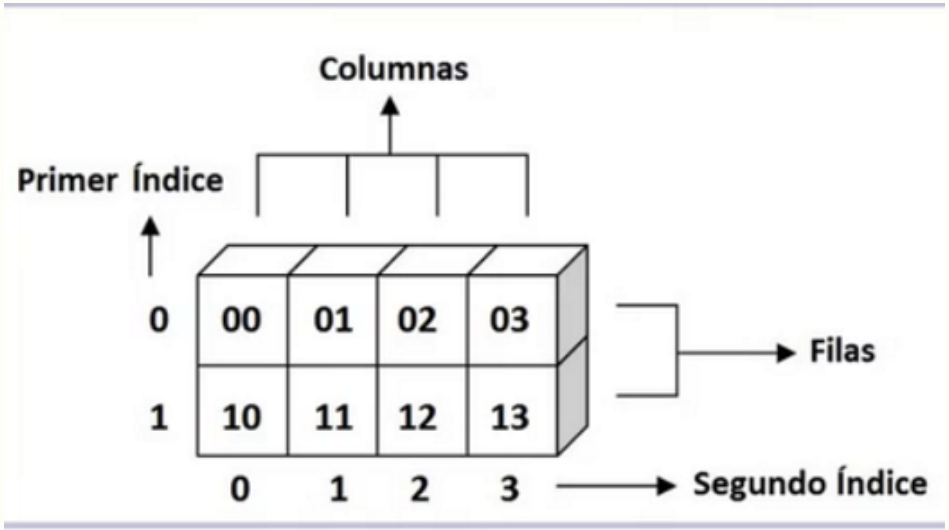


Definición

Un arreglo bidimensional/matriz: una estructura homogénea que se entiende como filas y columnas. Cada celda se identifica combinando un índice de fila y un índice de columna.

- Extiende la idea del arreglo a una tabla completa
- Permite representar varios grupos de datos relacionados
- Se usa cuando una sola *lista* ya no alcanza para modelar el problema
- Cada celda queda determinada por la combinación de una fila y una columna

¿Qué es un arreglo multidimensional?



Array 2D con 8 elementos (2*4)

Características

- Homogéneo: todos los elementos tienen el mismo tipo
- Ordenado: cada posición usa dos índices
- Finito: tiene una cantidad determinada de filas y columnas
- De tamaño fijo: sus dimensiones quedan definidas al declararlo

Lectura

- El primer índice selecciona la fila
- El segundo índice selecciona la columna
- Ambos índices, combinados, determinan la posición de cada celda
- En el ejemplo, `2 * 4 = 8` elementos

Declaración de una matriz

SINTAXIS

```
tipo nombre[filas][columnas];
```

EJEMPLOS

```
int notas[3][4];
float temperaturas[7][24];
char tablero[3][3];
```

Componentes

- `tipo`: clase de dato de cada elemento
- `nombre`: identificador de la matriz
- `filas`: cantidad de filas
- `columnas`: cantidad de columnas

Inicialización de matrices

DECLARACIÓN EN CERO

```
int notas[3][4] = {0};
```

DECLARACIÓN CON VALORES

```
int notas[2][3] = {
    {85, 92, 78},
    {90, 88, 95}
};
```

Lectura

- `{0}` deja todas las posiciones inicializadas en cero
- Con llaves anidadas se cargan valores por fila
- La cantidad de valores debe ser coherente con las dimensiones declaradas

Acceso a elementos individuales

LECTURA Y ESCRITURA

```
int notas[3][4] = {
    {85, 92, 78, 90},
    {70, 81, 76, 88},
    {95, 89, 91, 84}
};

int primera = notas[0][0]; // accede al valor: "85"
notas[1][2] = 100; // modifica el valor: "76" por "100"
int suma = notas[0][0] + notas[0][1];  // 85 + 92 = 177
```

Reglas

- `matriz[fila][columna]` sirve para **leer** y también para **modificar**
- Cada índice puede ser una constante, una variable o una expresión
- Ambos índices deben quedar dentro del rango válido

Acceder a una fila o columna inexistente produce comportamiento indefinido.

Ejemplo: leer, mostrar y modificar

CÓDIGO EN C

```c
#include <stdio.h>

int main() {
    int notas[3][4];
    int i, j;

    for (i = 0; i < 3; i++) {
        for (j = 0; j < 4; j++) {
            printf("Estudiante %d, materia %d: ", i + 1, j + 1);
            scanf("%d", &notas[i][j]);
        }
    }

    printf("Primera nota: %d\n", notas[0][0]);
    printf("Ultima nota: %d\n", notas[2][3]);

    notas[1][2] = 100;
    printf("Nota modificada: %d\n", notas[1][2]);

    return 0;
}
```

Lectura

Los bucles cargan la matriz completa usando `notas[i][j]`.

Accesos puntuales

Después de la carga se consultan dos posiciones y se modifica una tercera.

Recorrer una matriz con `for`

PATRÓN GENERAL

```c
for (int i = 0; i < filas; i++) {
    for (int j = 0; j < columnas; j++) {
        // Trabajar con matriz[i][j]
    }
}

for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
        printf("%d ", notas[i][j]);
    }
    printf("\n");
}
```

Lectura

- `i` recorre las filas
- `j` recorre las columnas de cada fila
- La condición de cada `for` debe respetar su dimensión
- Cada iteración procesa exactamente una posición `matriz[i][j]`

Ejemplo: suma de cada fila

CÓDIGO EN C

```c
int matriz[3][4] = {
    {10, 20, 30, 40},
    {15, 25, 35, 45},
    { 5, 10, 15, 20}
};
int suma;
int i, j;

for (i = 0; i < 3; i++) {
    suma = 0;
    for (j = 0; j < 4; j++) {
        suma = suma + matriz[i][j];
    }
    printf("Suma fila %d: %d\n", i + 1, suma);
}
```

RESULTADO

```
Suma fila 1: 100
Suma fila 2: 120
Suma fila 3: 50
```

Idea central

La variable `suma` se reinicia al comenzar cada fila y acumula solo los valores de esa fila mientras avanza `j`.

Ejemplo: encontrar el mayor valor

CÓDIGO EN C

```c
int tabla[3][4] = {
    {10, 50, 30, 20},
    {45, 15, 60,  5},
    {25, 80, 35, 40}
};
int mayor = tabla[0][0];
int i, j;

for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++) {
        if (tabla[i][j] > mayor) {
            mayor = tabla[i][j];
        }
    }
}

printf("El mayor valor es: %d\n", mayor);
```

RESULTADO

```
El mayor valor es: 80
```

Estrategia

- Tomar una posición válida como valor inicial
- Comparar cada elemento de la matriz con `mayor`
- Reemplazarlo cuando aparece un valor superior

Ejemplo: llenar una matriz con valores

CÓDIGO EN C

```c
int matriz[3][4];
int i, j;

for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++) {
        matriz[i][j] = i * 10 + j;
    }
}

for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++) {
        printf("%d ", matriz[i][j]);
    }
    printf("\n");
}
```

SALIDA

```
0 1 2 3
10 11 12 13
20 21 22 23
```

Lectura

- `i` y `j` participan en el valor calculado
- Índices y datos no tienen por qué coincidir
- El patrón permite llenar toda la matriz sin cargar posición por posición a mano

Índices inválidos

CÓDIGO CON ERROR

```c
int notas[3][4] = {0};

printf("%d\n", notas[3][0]);
printf("%d\n", notas[1][4]);
```

MENSAJE / PROBLEMA

- Para `notas[3][4]`, las filas válidas son `0`, `1` y `2`.
- Las columnas válidas son `0`, `1`, `2` y `3`.
- `notas[3][0]` y `notas[1][4]` quedan fuera del rango de la matriz.

CORRECCIÓN

- Verificar siempre que `0 <= fila < filas`.
- Verificar siempre que `0 <= columna < columnas`.
- Si se quiere la última posición válida, en este caso es `notas[2][3]`.

Valores indefinidos

CÓDIGO CON ERROR

```
int notas[3][4];
int suma = 0;
int i, j;

for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++) {
        suma = suma + notas[i][j];
    }
}
```

MENSAJE / PROBLEMA

- `notas[i][j]` no tiene un valor definido si antes no se inicializa o se lee.
- Sumar esos datos produce resultados incorrectos.

CORRECCIÓN

```
int notas[3][4] = {0};
```

o bien leer cada posición antes de usarla.

Condiciones de recorrido

CÓDIGO CON ERROR

```c
int notas[3][4];
int i, j;

for (i = 0; i <= 3; i++) {
    for (j = 0; j < 4; j++) {
        printf("%d ", notas[i][j]);
    }
}
```

MENSAJE / PROBLEMA

- La condición `i <= 3` permite que `i` llegue a `3`.
- Esa iteración intenta acceder a una fila que no existe.
- En matrices hay que revisar el límite de cada `for` por separado.

CORRECCIÓN

```c
for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++) {
        printf("%d ", notas[i][j]);
    }
}
```

Ejemplo: estadísticas por estudiante

Problema

Leer las notas de `3` estudiantes en `4` materias y calcular:

- promedio de cada estudiante
- cantidad de estudiantes con promedio mayor o igual a `60`

Método

- Cargar la matriz completa
- Recorrer fila por fila
- Reiniciar `suma` al comenzar cada estudiante
- Calcular el promedio y actualizar el contador

Ejemplo: estadísticas por estudiante

CÓDIGO EN C

```c
#include <stdio.h>

int main() {
    int notas[3][4];
    int i, j;
    int suma;
    float promedio;
    int aprobados = 0;

    for (i = 0; i < 3; i++) {
        for (j = 0; j < 4; j++) {
            printf("Estudiante %d, materia %d: ", i + 1, j + 1);
            scanf("%d", &notas[i][j]);
        }
    }

    for (i = 0; i < 3; i++) {
        suma = 0;

        for (j = 0; j < 4; j++) {
            suma = suma + notas[i][j];
        }

        promedio = suma / 4.0;
        printf("Promedio estudiante %d: %.2f\n", i + 1, promedio);

        if (promedio >= 60) {
            aprobados = aprobados + 1;
        }
    }

    printf("Aprobados: %d\n", aprobados);
    return 0;
}
```

Observaciones

- Cada fila representa un estudiante
- El promedio se calcula al terminar cada fila
- Un mismo recorrido puede producir varios resultados

Ejercicio 1

Leer una matriz `3 x 3` de enteros y mostrar la suma de cada fila y la suma de cada columna.

Ejercicio 2

Leer las notas de `4` estudiantes en `3` materias, calcular el promedio por estudiante y contar cuántos aprobaron con promedio mayor o igual a `60`.

Ejercicio 3

Leer una matriz `3 x 4`, encontrar el mayor valor y mostrar también en qué fila y en qué columna aparece.