Arreglos Bidimensionales

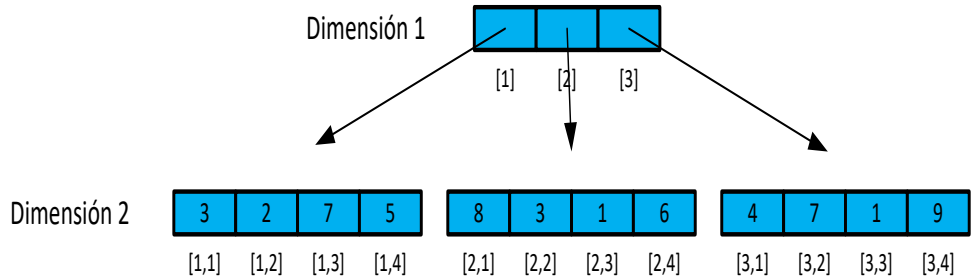
Programación Lógica

UNIVERSIDAD SIGLO 21 | MIEMBRO DE LA RED ILUMNO

1. Arreglos bidimensionales

Existen escenarios en los que la información es mejor organizarla en forma de tabla o matriz con dos subíndices: uno para recorrer las filas y otro para recorrer las columnas. Un arreglo bidimensional es un vector de vectores cuya representación gráfica puede ser representada mediante una tabla.

Figura 1: Representación de arreglo bidimensional



Fuente: elaboración propia.

La figura anterior representa un arreglo bidimensional cuya primera dimensión contiene tres elementos que almacenan nuevos arreglos de cuatro elementos (segunda dimensión).

La siguiente figura representa el mismo arreglo definido anteriormente, pero para su representación se utiliza una matriz:

Figura 2: Representación de arreglo bidimensional

Columnas

Filas	1	2	3	4
1	**3**	**2**	**7**	**5**
2	**8**	**3**	**1**	**6**
3	**4**	**7**	**1**	**9**

Fuente: elaboración propia.

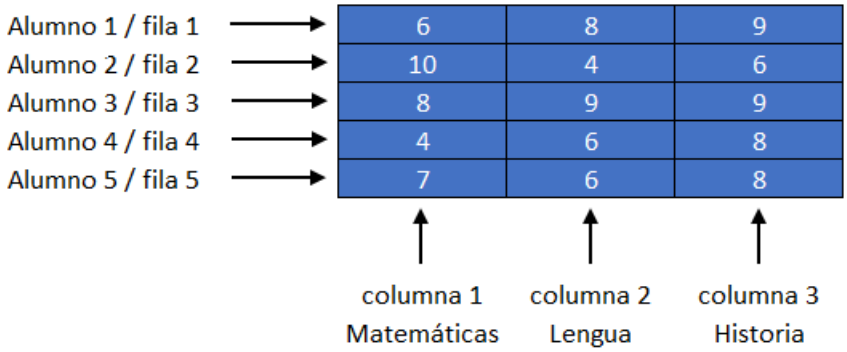
Cada elemento de un arreglo bidimensional se referencia con un mismo nombre y dos subíndices. En notación estándar el primer subíndice hace referencia a las filas y el segundo a las columnas del arreglo. Es decir, si el arreglo bidimensional de la Figura 2 se llama *tabla*, entonces *tabla[2,3]* es el elemento que ocupa la posición de la segunda fila y tercera columna, cuyo valor es 1.

En pseudocódigo, los arreglos bidimensionales se declaran de la siguiente forma:

```
Algoritmo declaracion_arreglo_bidimensional
tipo
    array[límite inferior filas...límite superior filas] [límite inferior columnas...límite superior columnas] de <tipo de dato>: <nombre estructura arreglo>
    var
        <nombre estructura arreglo>: <nombre variable de tipo arreglo>
inicio
……
```

Para mencionar un ejemplo, supongamos que queremos almacenar las calificaciones de los alumnos en tres materias: matemáticas, lengua e historia. Para ello, se podría construir un arreglo bidimensional para representar una tabla en la cual las filas hacen referencia a cada alumno y las columnas hacen referencia a cada una de las materias: en la columna 1, se indican las calificaciones de matemáticas; en la columna 2, las calificaciones de lengua, y en la columna 3, las calificaciones de historia. La representación de esta tabla para 5 alumnos se muestra en la Figura 3.

Figura 3: Representación de arreglo bidimensional



Fuente: elaboración propia.

Si se hace una lectura horizontal, se puede decir que el alumno de la fila 1 tiene 6 en matemáticas, 8 en lengua y 9 en historia. Si se quiere referenciar la nota del cuarto alumno en historia, se indica con la expresión *notas[4][3]*.

Para recorrer arreglos bidimensionales, es necesario tener dos estructuras repetitivas anidadas. Las estructuras que mejor se adaptan a estos casos son las estructuras *desde/fin-desde*. Por ejemplo, para leer datos desde el teclado, guardarlos en un arreglo bidimensional y luego recorrerlo para mostrar los elementos en pantalla, se puede utilizar el siguiente pseudocódigo:

```
Algoritmo recorrer_arreglo_bidimensional
tipo
    array[1...5][1...3] de entero: lista
```

```
var
        lista: notas
        entero: i, j
inicio
        // cargar arreglo bidimensional
        desde i ← 1 hasta 5 hacer
                desde j ← 1 hasta 3 hacer
                    leer(notas[i,j])
                fin-desde
        fin-desde

        // recorrer y mostrar elementos del arreglo bidimensional
        desde i ← 1 hasta 5 hacer
                desde j ← 1 hasta 3 hacer
                    mostrar(notas[i,j])
                fin-desde
        fin-desde
fin
```

La estructura repetitiva *desde* externa recorre las filas y, para cada una de ellas, la estructura *desde* interna recorre cada una de las columnas.

Ejercicios de aplicación

1) Implementar un algoritmo que permita ingresar las notas de 10 alumnos en las materias de física y matemáticas correspondientes a un curso. Se desea mostrar el promedio general del curso y por materia.

```
algoritmo arreglo_promedio_curso
tipo
    array[1...10][1…2] de real: tabla_notas
var
    entero: cantidadAlumnos, i, j
    real: sumaNotasCurso, sumaNotasFisica, sumaNotasMatematicas,
    promedioNotasCurso, promedioNotasFisica, promedioNotasMatematicas
    tabla_notas: notas
inicio
    cantidadAlumnos ← 10

    mostrar("Carga de notas...")
    mostrar("")
    para i ← 1 hasta cantidadAlumnos hacer
        mostrar("Ingrese las notas del alumno ", i)
        para j ← 1 hasta 2 hacer
            si j==1 entonces
                mostrar("Física: ")
            si-no
```

```
                mostrar("Matemáticas: ")
            fin-si
            leer notas[i,j]
        fin-para
        mostrar("")
    fin-para

    // Cálculo de promedios
    sumaNotasCurso ← 0
    sumaNotasFisica ← 0
    sumaNotasMatematicas ← 0
    para i ← 1 hasta cantidadAlumnos hacer
        sumaNotasCurso ← sumaNotasCurso + notas[i,1] + notas[i,2]
        sumaNotasFisica ← sumaNotasFisica + notas[i,1]
        sumaNotasMatematicas ← sumaNotasMatematicas + notas[i,2]
    fin-para

    promedioNotasCurso ← sumaNotasCurso / (cantidadAlumnos*2)
    promedioNotasFisica ← sumaNotasFisica / cantidadAlumnos
    promedioNotasMatematicas ← sumaNotasMatematicas / cantidadAlumnos

    mostrar("Promedio de notas del curso: ", promedioNotasCurso)
    mostrar("Promedio de notas de Física: ", promedioNotasFisica)
    mostrar("Promedio de notas de Matemáticas: ", promedioNotasMatematicas)
fin
```

2) Implementar un algoritmo que permita cargar una tabla con temperaturas. La tabla debe tener 5 filas correspondientes a 5 días y 3 columnas correspondientes a 3 momentos del día. El algoritmo debe mostrar la temperatura mínima y máxima de la tabla completa.

```
algoritmo matriz_temperaturas
tipo
    array[1...5][1…3] de real: tabla_5x3
var
    entero: i, j
    real: tempMin, tempMax
    tabla_5x3: temperaturas
inicio
    mostrar("Carga de temperaturas...")
    mostrar("")
    para i ← 1 hasta 5 hacer
        para j ← 1 hasta 3 hacer
            mostrar("Ingrese la temperatura ", j, " del día ", i)
            leer(temperaturas[i,j])
        fin-para
        mostrar("")
    fin-para

    tempMin ← temperaturas[1,1]
    tempMax ← temperaturas[1,1]
    para i ← 1 hasta 5 hacer
        para j ← 1 hasta 3 hacer
```

```
            si temperaturas[i,j] < tempMin entonces
                tempMin ← temperaturas[i,j]
            fin-si
            si temperaturas[i,j] > tempMax entonces
                tempMax ← temperaturas[i,j]
            fin-si
        fin-para
    fin-para

    mostrar("La temperatura mínima registrada es: ", tempMin)
    mostrar("La temperatura máxima registrada es: ", tempMax)
fin
```

3) Implementar un algoritmo que permita cargar una matriz de $M \times N$ números enteros y muestre la cantidad de números positivos.

```
algoritmo matriz_cantidad_positivos
tipo
    array[1...50][1…50] de entero: matriz
var
    entero: filas, columnas, cantidadPositivos, i, j
    matriz: datos
inicio
    mostrar("Ingrese la cantidad de filas que tendrá la matriz (menor a 50)")
    leer(filas)

    mostrar("Ingrese la cantidad de columnas que tendrá la matriz (menor a 50)")
    leer(columnas)

    mostrar("Carga de matriz...")
    mostrar("")
    para i ← 1 hasta filas hacer
        para j ← 1 hasta columnas hacer
            mostrar("Ingrese el valor para la posición [",i,",",j,"]")
            leer(matriz[i,j])
        fin-para
        mostrar("")
    fin-para

    cantidadPositivos ← 0
    para i ← 1 hasta filas hacer
        para j ← 1 hasta columnas hacer
            si matriz[i,j] > 0 entonces
                cantidadPositivos ← cantidadPositivos + 1
            fin-si
        fin-para
    fin-para

    mostrar("La cantidad de números positivos en la matriz es: ", cantidadPositivos)
fin
```

4) Implementar un algoritmo que permita cargar dos matrices de 3 × 3. El algoritmo debe realizar la suma de ambas matrices y mostrar la matriz resultante.

```
algoritmo suma_matrices
tipo
    array[1...3][1…3] de entero: matriz
var
    entero: i, j
    matriz: matriz1, matriz2, matrizSuma
inicio
    mostrar("Carga de matriz 1...")
    mostrar("")
    para i ← 1 hasta 3 hacer
        para j ← 1 hasta 3 hacer
            mostrar("Ingrese el valor para la posición [",i,",",j,"]")
            leer(matriz1[i,j])
        fin-para
        mostrar("")
    fin-para

    mostrar("Carga de matriz 2...")
    mostrar("")
    para i ← 1 hasta 3 hacer
        para j ← 1 hasta 3 hacer
            mostrar("Ingrese el valor para la posición [",i,",",j,"]")
            leer(matriz2[i,j])
        fin-para
        mostrar("")
    fin-para

    // Suma de matrices
    para i ← 1 hasta 3 hacer
        para j ← 1 hasta 3 hacer
            matrizSuma[i,j] ← matriz1[i,j] + matriz2[i,j]
        fin-para
    fin-para

    mostrar("Matriz resultante (matriz1 + matriz2)")
    para i ← 1 hasta 3 hacer
        para j ← 1 hasta 3 hacer
            mostrar sin saltar (" ", matrizSuma[i,j]) // función “sin saltar” utilizada en el programa
            PSeInt para evitar el salto de línea cuando se imprime un mensaje por pantalla
        fin-para
        mostrar("")
    fin-para
fin
```

5) Implementar un algoritmo que permita guardar 16 caracteres tomados desde el teclado en un arreglo bidimensional de 4 × 4. El algoritmo solicitará al usuario el ingreso de un número de fila y el programa debe mostrar todos los elementos de dicha fila.

```
algoritmo matriz_mostrar_fila
tipo
    array[1...4][1…4] de caracter: matriz
var
    entero: i, j, fila
    matriz: datos
inicio
    mostrar("Carga de matriz...")
    mostrar("")
    para i ← 1 hasta 4 hacer
        para j ← 1 hasta 4 hacer
            mostrar("Ingrese el valor para la posición [",i,",",j,"]")
            leer(datos[i,j])
        fin-para
        mostrar("")
    fin-para

    repetir
        mostrar("Ingrese la fila que desea mostrar (entre 1 y 4)")
        leer(fila)
    hasta-que fila >= 1 y fila <= 4

    mostrar("")
    mostrar("Mostrando contenido de la fila ",fila)
    para i ← 1 hasta 4 hacer
        mostrar sin saltar (" ", datos[fila,i])
    fin-para
fin
```