Amplificación 15032022 LS 8CM11

Implementación

Elementos de c#

Arreglos.

Sea un conjunto de datos

$d = \{d_1, d_2, \ldots, d_n\}$ → [d0 | d1 | d2 | ... | dn-1] →

d = {d[0]

d[0]
d[1]
⋮
d[n-1]

$a = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$ →

[a00 a01 ... a0 n-1
a10 a11 ...
⋮
am-1 0 am-1 1 ... am-1 n-1]

a = {a[0]

a → a[0,0]
a[0,1]
⋮
a[0, n-1]
a[1,0]
⋮
a[1, n-1]
⋮
a[m-1, 0]
⋮
a[m-1, n-1]

fila 0

fila 1

1.- Un arreglo siempre es adyacente en sus elementos
2.- Todos los elementos son de un solo tipo
3.- El nombre del arreglo es la referencia al inicio del arreglo
4.- En c# los arreglos son dinámicos

Definición de un arreglo

```
tipo [] d;      // 1 dim
tipo [,] a;     // 2 dim
tipo [,,] b;    // 3 dim
tipo [][] c;    // arreglo irregular
```

Instancia de arreglos (apartar memoria)

```
d = new tipo[n];        n, m ∈ Z+
a = new tipo[m,n];
b = new tipo[m,n,k];
c = new {new c[3] {n}
         new c[3] {m}
         ⋮
        }
```

$\begin{bmatrix} 1 & 2 & 3 \\ 1 & & \\ 1 & 2 & 3 & 4 \end{bmatrix}$

Para manejar los arreglos, se utiliza una iteración

$\forall\ i = 0, n-1$
  d[i]

$\forall\ i = 0, m-1$
$\wedge\ j = 0, n-1$
  a[i,j]
  ⋮

int[] x = new [5]
double[] y = new [3]

double [,] a = new

int [][] b = n

PRACTICA
Terminar poniendo salida los valores utilizando Con

→ for(int i = 0
  {
    Console.
  }

Ejemplos

```
int[] x = new [] {1, 3, -2, 4};
double[] y = new [] {1.1, -2.5, 4.3};

double[,] a = new [,] { {-1.1, 2.1, 3.1},
                        {2.1, 1.1, 3.1} };

int[][] b = new [] {new [] {1, 2, 3},
                    new [] {1},
                    new [] {1, 2, 3, 4},
                   };
```

$\{1, 3, -2, 4\}$

$\{1.1, -2.5, 4.3\}$

$\begin{bmatrix} -1.1 & 2.1 & 3.1 \\ 2.1 & 1.1 & 3.1 \end{bmatrix}$

$\begin{bmatrix} 1 & 2 & 3 \\ 1 & & \\ 1 & 2 & 3 & 4 \end{bmatrix}$

PRACTICA
Teniendo poniendo a la salida los valores de los arreglos utilizando Console.Write

```
for(int i = 0; i<n; i++)
{
    Console.WriteLine( x[i] );
}
```

Definición de multitipos:

```
var variable = cte;
```

+ donde cte de un tipo definido, le asigna el tipo a variable
+ Si se reasigna otra constante de diferente tipo, este define a variable con el tipo

```
var x = 2.5; // x; es double
x = 2      ; // x, es entero
```

+ El tipo asignado inicialmente debe ser mayor que los tipos reasignados.

PRACTICA
obtener a la salida, los valores asignado utilizando Console.WriteLine( )

El tipo object:

System define a los tipos como objetos, por lo tanto existe un tipo llamado object

donde este tipo es multitipo en compilación

```
object a;
a = 3;
a = 1.5F;
a = 2.6;
a = 'A';
a = "ESIME"
```