Lenguaje C# - Guía N° 3

Arrays, estructuras y cadenas de texto

Una tabla, vector o array ("arreglo") es un conjunto de elementos, todos del mismo tipo. Estos elementos tendrán todos los mismos nombres, y ocuparán un espacio contiguo en la memoria.

Por ejemplo, si queremos definir un grupo de números enteros, el tipo de datos que usaremos para declararlo será "int[]". Si sabemos desde el principio cuantos datos tenemos (por ejemplo 4), les reservaremos espacio con "= new int[4]", así

```
<tipo> [] <nombre>  = new <tipo> [<tamaño>];
```

```
int[] ejemplo = new int[4];  // array unidimensional de 4 números enteros
```

En el caso anterior tendríamos 4 elementos que serían: ejemplo[0], ejemplo[1], ejemplo[2], ejemplo[3].

Ejercicio 17: Realice el siguiente arreglo de 5 números enteros y hallar su suma:

```
int[] num = new int[5];  /* Un array de 5 números enteros */
int suma;
num[0] = 200;
num[1] = 150;
num[2] = 100;
num[3] = -50;
suma = num[0] + num[1] + num[2] + num[3];
Console.WriteLine("Su suma es {0}", suma);
Console.Read();
```

Valor inicial de un array

Se puede dar valor a los elementos de una tabla al principio del programa; se indican entre llave separados por comas.

Ejercicio 18:

```
int[] num = { 200, 150, 100, -50, 300 };
int suma;
suma = num[0] + num[1] + num[2] + num[3] + num[4];
Console.WriteLine("La suma del 1er arreglo es {0}", suma);

int[] elementos = new int[] { 5, 4, 3, 2, 1 };
int x = elementos[2];
Console.WriteLine("El elemento [2] del 2do arreglo es:" + x); // imprime 3

Console.Read();
```

Recorriendo los elementos de una tabla

Es de esperar que exista una forma más cómoda de acceder a varios elementos de un array, sin tener siempre que repetirlos todos, como hemos hecho en

```
suma = numero[0] + numero[1] + numero[2] + numero[3] + numero[4];
```

El "truco" consistirá en emplear cualquiera de las estructuras repetitivas que ya hemos visto (while, do..while, for), por ejemplo así:

```
int suma = 0;          /* Valor inicial de la suma */
for (i=0; i<=4; i++) /* Y hallamos la suma repetitiva */
  suma += numero[i];
```

Ejercicio 19:

```
int[] numero = { 200, 150, 100, -50, 300 };
int suma = 0;
int i;

for (i = 0; i <= 4; i++) // recorriendo y sumando
   {
    suma += numero[i];
   }
Console.WriteLine("Su suma es {0}", suma);
Console.ReadKey();
```

Datos repetitivos introducidos por el usuario

Si queremos que sea el usuario el que introduzca datos a un array, se usa un ciclo for.

Ejercicio 20:

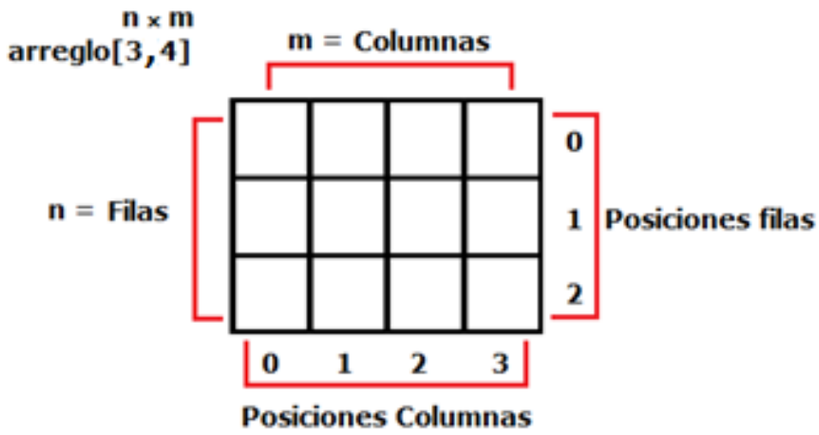
```
int[] numero = new int[5]; /* array de 5 números enteros */
int suma;                  /* Un entero que será la suma */
int i;
for (i = 0; i <= 4; i++)   /* recorre y Pide los datos */
    {
     Console.Write("Introduce el dato numero {0}: ", i + 1);
     numero[i] = Convert.ToInt32(Console.ReadLine());
    }
suma = 0;                /* Valor inicial de la suma */
for (i = 0; i <= 4; i++) /* Y hallamos la suma repetitiva */
      suma += numero[i];

Console.WriteLine("Su suma es {0}", suma);
Console.Read ();
```

Tablas bidimensionales

Podemos declarar tablas de dos o más dimensiones. Por ejemplo, si queremos guardar datos de dos grupos de alumnos, cada uno de los cuales tiene 20 alumnos, tenemos la siguiente opción:

- **Podemos emplear int datosAlumnos[2,20]** y entonces sabemos que los datos de la forma datosAlumnos[0,i] son los del primer grupo, y los datosAlumnos[1,i] son los del segundo.
- Una alternativa, que puede sonar más familiar a quien ya haya programado en C es emplear int datosAlumnos[2][20] pero en C# esto no tiene exactamente el mismo significado que [2,20], sino que se trata de dos arrays, cuyos elementos a su vez son arrays de 20 elementos. De hecho, podrían ser incluso dos arrays de distinto tamaño, como veremos en el segundo ejemplo.

Leyendo una matriz



Ejercicio 21: Uso con arrays de la forma [n,m], usemos arrays con valores prefijados, y arrays para los que reservemos espacio con "new" y a los que demos valores más tarde:

```
int[,] notas1 = new int[2, 2]; // 2 bloques de 2 datos
notas1[0, 0] = 1;  // [0,0] alumno 1
notas1[0, 1] = 2;  // [0,1] alumno 2
notas1[1, 0] = 3;  // [1,0] alumno 3
notas1[1, 1] = 4;  // [1,1] alumno 4

int[,] notas2 = {  // 2 bloques de 10 datos
    {1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
    {11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
};

Console.WriteLine("La nota1 del 2° alumno del grupo 1 es: " + notas1[0, 1]);
Console.WriteLine("La nota2 del 3° alumno del grupo 1 es: " + notas2[0, 2]);
Console.Read();
```

Este tipo de tablas de varias dimensiones son las que se usan también para guardar matrices, cuando se trata de resolver problemas matemáticos más complejos

Otra forma de tener arrays multidimensionales son los "arrays de arrays", pueden tener elementos de distinto tamaño. Para saber su longitud, podemos usar "a.Length":

Ejercicio 22

```
int[][] notas;         // Array de dos dimensiones
notas = new int[3][];  // 3 bloques de datos
notas[0] = new int[10];  // 10 notas en un grupo
notas[1] = new int[15];  // 15 notas en otro grupo
notas[2] = new int[12];  // 12 notas en el ultimo

Console.WriteLine("largo del arreglo [3,0] es: " + notas[0].Length);
Console.WriteLine("largo del arreglo [3,1] es: " + notas[1].Length);
Console.WriteLine("largo del arreglo [3,3] es: " + notas[2].Length);

// Damos valores de ejemplo
for (int i = 0; i < notas.Length; i++)
{
      for (int j = 0; j < notas[i].Length; j++)
      {
            notas[i][j] = i + j;
     }
}

// Y mostramos esos valores
for (int i = 0; i < notas.Length; i++)
{
     for (int j = 0; j < notas[i].Length; j++)
       {
          Console.Write(" {0}", notas[i][j]);
       }
       Console.WriteLine();
}
```

Estructuras o registros

Un registro es una agrupación de datos, los cuales no necesariamente son del mismo tipo. Se definen con la palabra "struct". La serie de datos que van a formar

En C# (al contrario que en C), primero deberemos declarar cual va a ser la estructura de nuestro registro, lo que no se puede hacer dentro de "Main". Más adelante, ya dentro de "Main", podremos declarar variables de ese nuevo tipo.

Los datos que forman un "struct" pueden ser públicos o privados. Por ahora, a nosotros nos interesará que sean accesibles desde el resto de nuestro programa, por lo que les añadiremos delante la palabra "public" para indicar que queremos que sean públicos.

Ya desde el cuerpo del programa, para acceder a cada uno de los datos que forman el registro, tanto si queremos leer su valor como si queremos cambiarlo, se debe indicar el nombre de la variable y el del dato (o campo) separados por un punto:

Ejercicio 23

```
public void Mostrar()
{
   tipoPersona persona;
   persona.nombre = "Juan Gabriel";
   persona.inicial = 'J';
   persona.edad = 66;
   persona.nota = 7.5f;
   Console.WriteLine(persona.nombre + " murió a los {0}", persona.edad + " años");
   Console.ReadKey();
}

struct tipoPersona
{
   public string nombre;
   public char inicial;
   public int edad;
   public float nota;
}
```

La notación 7.5f se usa para detallar que se trata de un número real de simple precisión (un "float")

Funciones

En C#, al igual que en C y los demás lenguajes derivados de él, todos los "trozos de programa" son funciones, incluyendo el propio cuerpo de programa, Main. De hecho, la forma básica de definir una función será indicando su nombre seguido de unos paréntesis vacíos, como hacíamos con "Main", y precediéndolo por ciertas palabras reservadas, como "public static void"

Por ejemplo, podríamos crear una función llamada "saludar", que escribiera varios mensajes en la pantalla:

```
public static void saludar()
{
  Console.WriteLine("Bienvenido al programa");
  Console.WriteLine(" de ejemplo");
  Console.WriteLine("Espero que estés bien");
}
```

Ahora desde dentro del cuerpo de nuestro programa, podríamos "llamar" a esa función:

```
public static void Main()
{
  saludar();
  …
}
```

Ejercicio 24

```
public void Mostrar()
{
    Saludar();
    Console.WriteLine("nos vemos...");
}

public static void Saludar()
{
    Console.WriteLine("La prueba se acerca");
    Console.WriteLine("a pasos agigantados");
    Console.WriteLine("Recuerden estudiar y practicar");
}
```

Parámetros de una función

Es muy frecuente que nos interese además indicarle a nuestra función ciertos datos especiales con los que queremos que trabaje.

Estos datos adicionales que indicamos a la función es lo que llamaremos sus "parámetros". Hay que indicar un nombre para cada parámetro (puede haber varios) y el tipo de datos que corresponde a ese parámetro. Por ejemplo:

```
public static void sumar ( int x, int y ) {
  ...
}
```

Ejercicio 25: método que reciba dos enteros y luego imprima de uno en uno desde el valor menor hasta el valor mayor.

```
7  namespace Ejercicios_Clase_3
8  {
9      class Class3
10     {
11
12         public void Mostrar()
13         {
14             Class3 p = new Class3();
15             p.MostrarRango(1, 25);
16             Console.ReadKey();
17         }
18
19         public void MostrarRango(int menor, int mayor)
20         {
21             for (var x = menor; x <= mayor; x++)
22             {
23                 Console.Write(x + " ");
24             }
25         }
26
27
28     }
29 }
30
```

Nota: Considere que el ejercicio fue hecho en la clase 3

Ejercicio 26: Función que recibe reciba como parámetros tres valores enteros y retorne el mayor de los mismos.

```
7  namespace Ejercicios_Clase_3
8  {
9      class Class3
10     {
11         public void Mostrar()
12         {
13             Class3 p = new Class3();
14             Console.Write("Ingrese primer valor:");
15             int x1 = int.Parse(Console.ReadLine());
16             Console.Write("Ingrese segundo valor:");
17             int x2 = int.Parse(Console.ReadLine());
18             Console.Write("Ingrese tercer valor:");
19             int x3 = int.Parse(Console.ReadLine());
20             Console.Write("El mayor valor de los tres es: " + p.Mayor(x1, x2, x3));
21             Console.ReadKey();
22         }
23
24         public int Mayor(int v1, int v2, int v3)
25         {
26             if (v1 >= v2 && v1 >= v3) {
27                 return v1;
28             }
29             else {
30                 if (v2 >= v3) {
31                     return v2;
32                 }
33                 else {
34                     return v3;
35                 }
36             }
37         }
38
39     }
40 }
```

Nota: Considere que el ejercicio fue hecho en la clase 3

Número variable de parámetros

C# permite implementar métodos con una cantidad variable de parámetros (es decir que llamemos al método pasando en algunas circunstancias pasándole dos parámetros y en otras pasándole 5)

Para ello se usa la palabra clave **params** y seguidamente un vector.

Ejercicio 27: Método que le envíe una cantidad n de enteros y me retorne la suma de los mismos.

```
7  namespace Ejercicios_Clase_3
8  {
9      class Class3
10     {
11         public void Mostrar()
12         {
13             Class3 p = new Class3();
14             Console.Write("La suma de 3,4,5 es ");
15             Console.WriteLine(p.Sumar(3, 4, 5));
16             Console.ReadKey();
17         }
18
19         public int Sumar(params int[] p)
20         {
21             int su = 0;
22             for (var f = 0; f < p.Length; f++)
23             {
24                 su = su + p[f];
25             }
26             return su;
27         }
28
29
30     }
31 }
```

Nota: Considere que el ejercicio fue hecho en la clase 3

Ejercicio 28: Método que reciba un string con la cadena "suma" o "producto" y seguidamente una lista de enteros. El método debe retornar la suma o producto de todos los valores enviados

```
7  namespace Ejercicios_Clase_3
8  {
9      class Class3
10     {
11         public void Mostrar()
12         {
13             Class3 p = new Class3();
14             Console.Write("El producto de 2*3*4 es ");
15             Console.WriteLine(p.Operar("producto", 2, 3, 4));
16             Console.Write("La suma de 5+10 es ");
17             Console.WriteLine(p.Operar("suma", 5, 10));
18             Console.ReadKey();
19         }
20
21         public int Operar(string operacion, params int[] v)
22         {
23             if (operacion == "suma")
24             {
25                 int suma = 0;
26                 for (var f = 0; f < v.Length; f++)
27                 {
28                     suma = suma + v[f];
29                 }
30                 return suma;
31             }
32             if (operacion == "producto")
33             {
34                 int producto = 1;
35                 for (var f = 0; f < v.Length; f++)
36                 {
37                     producto = producto * v[f];
38                 }
39                 return producto;
40             }
41             return int.MaxValue;
42         }
43
44     }
45 }
```

Nota: Considere que el ejercicio fue hecho en la clase 3

TAREA 9: Modifique el ejercicio 28, tal que ingrese por teclado:

3 datos para la multiplicación
4 datos para la suma

TAREA 10: En un proyecto Nuevo cree un menú simple en el que aparezcan sus datos (Nombre, apellido, curso) y realice estos ejercicios:

1.- Matriz bidimensional que solicite filas y columnas y que muestre según los ejemplos:

Si ingreso 3 filas y 5 columnas que muestre así:
01234
12345
23456

Si ingreso 4 filas x 3 columnas que muestre así:
012
123
234
345

2.- Solicite un nombre, edad y su rut, luego muestre por pantalla sus datos (cree una función que verifique el digito verificador). (15 puntos) Considere lo siguiente:

RUT	**1**	**6**	**4**	**9**	**1**	**7**	**4**	**3**
Números fijos	3	2	7	6	5	4	3	2
Multiplicar	**1** x 3	**6** x 2	**4** x 7	**9** x 6	**1** x 5	**4** x 7	**3** x 4	**2** x 3
Productos	**3**	**12**	**28**	**54**	**5**	**28**	**12**	**6**
Sumarlos	3 + 12 + 28 + 54 + 5 + 28 + 12 + 6							
Resultado	**148**							
Dividir el resultado anterior por 11	14'8' : **11** = 13 3 8 **5**//							
A 11 restarle el resto de la división	11 – 5 = **6** es el dígito verificador							

NOTA: cuando la diferencia es 11 el digito verificador es 0
y cuando la diferencia es 10, el digito es k

3.- Salir