

C# .Net Programming: A graphical approach

Class 4

Ing. Hazael Fernando Mojica García

- Methods
- Access Modifiers
- Arrays

Methods

To solve real life problems we need more than a couple of lines of code, a real system could go from 100,000 lines of code to millions. (See Linux [kernel](#)).

In order to keep and update the code we need to modularize it, it means, divide the code in blocks that could be used independently, this improves the code readability and help the comprehension of itself.

This blocks, and the way we use this blocks in our program are called Methods, and their a basic component in programming. (In other languages they call then “functions”).

Syntax

Access-modifier return-type method_name(arguments)

```
{  
    //Instructions to execute  
    return return-variable;  
}
```

Examples

```
public int ageOfEmployee  
{  
    int age = 50;  
    return age;  
}
```

```
public int calculateIncome(int daysWorked)  
{  
    int dailySalary = 200;  
    int income = daysWorked * dailySalary;  
    return income;  
}
```

```
public void showMessage(string msg)  
{  
    MessageBox.Show(msg);  
}
```

Example 1 - Methods

Net Salary Calculator.

This is going to be a fictitious example, here we consider the ISR to be 15% of the Nominal Salary always.

Further when we see Arrays, we will make a completely functional ISR calculator.

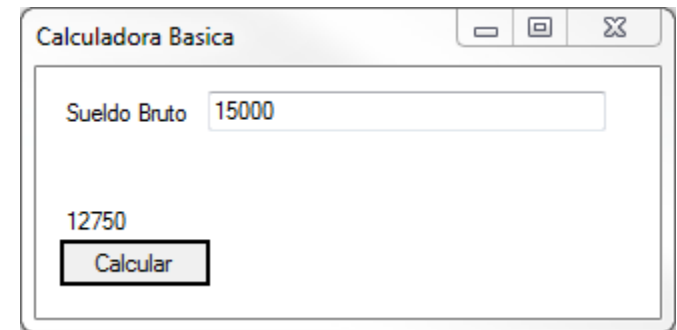
```
private void button_Calcular_Click(object sender, EventArgs e)
{
    double bruto = 0;
    double neto = 0;

    //Obtenemos el sueldo bruto a calcular de la TextBox
    bruto = Convert.ToDouble(this.textBox_Val1.Text);

    //Calculamos el Sueldo Neto usando un Metodo
    neto = sueldoNeto(bruto);

    //Desplegamos el resultado
    this.label_Resultado.Text = neto.ToString();
}

private double sueldoNeto(double sueldoBruto)
{
    double sNeto = 0;
    double descuento = sueldoBruto * 0.15;
    sNeto = sueldoBruto - descuento;
    return sNeto;
}
```



4.1-CalculadoraBasica-EjemploISR

Access modifiers

The access modifiers are reserved words that tell the compiler what level of visibility will have a variable, method or inner class inside or application.

The access modifiers were born from the necessity of having work done correctly inside a big team and that the fact that other people is going to use our code.

These are the access modifiers used in C#:

Modificador	Descripción
Public	The type or member can be accessed by any other code in the same assembly or another assembly that references it.
Private	The type or member can be accessed only by code in the same class or struct.
Protected	The type or member can be accessed only by code in the same class or struct, or in a class that is derived from that class.
Internal	The type or member can be accessed by any code in the same assembly, but not from another assembly.

Example 2 - Methods

Using the code we already have for “Calculadora Básica”, rewrite the code and encapsulate each operation inside a method.

```
private void button_Calcular_Click(object sender, EventArgs e)
{
    String operador = this.comboBox1.Text;
    int resultado = 0;
    int valor1 = 0;
    int valor2 = 0;

    //Obtenemos los valores de los TextBox's
    valor1 = Convert.ToInt32(this.textBox_Val1.Text);
    valor2 = Convert.ToInt32(this.textBox_Val2.Text);

    switch (operador)
    {
        case "+": //SUMA
            resultado = doSuma(valor1, valor2);
            break;

        case "-": //RESTA
            resultado = doResta(valor1, valor2);
            break;

        case "*": //MULTIPLICACION
            resultado = doMult(valor1, valor2);
            break;

        case "/": //DIVISION
            resultado = doDiv(valor1, valor2);
            break;

        case "pow": //POTENCIA
            resultado = doPotencia(valor1, valor2);
            break;

        default: //SIN OPERADOR
            MessageBox.Show("Selecciona un operador");
            break;
    }

    //Colocamos el resultado en el Label
    this.label_Resultado.Text = resultado.ToString();
}
```

```
private int doSuma(int valor1, int valor2)
{
    int resultado = valor1 + valor2;
    return resultado;
}

private int doResta(int valor1, int valor2)
{
    int resultado = valor1 - valor2;
    return resultado;
}

private int doMult(int valor1, int valor2)
{
    int resultado = valor1 * valor2;
    return resultado;
}

private int doDiv(int valor1, int valor2)
{
    int resultado = valor1 / valor2;
    return resultado;
}

private int doPotencia(int valor1, int valor2)
{
    int resultado = valor1;
    int i = 1;
    for (i = 1; i < valor2; i++)
    { //Repetir tantas veces como valor2
        resultado = resultado * valor1;
    }
    return resultado;
}
```

The Math class

In C# the [Math class](#) allow us to perform complex mathematical operations without the need for us to write our own algorithm.

The Math class lives inside the System *namespace*.

Until this moment you don't need to understand what a *class* or *namespace* is, we will catch this subject later.

The Math class has many variables and methods very well structured and I will use this class as an example for modularization.

```
double result = Math.Pow(5, 2);  
result = 25
```

```
double result = Math.Round(5.7356, 2);  
result = 5.74
```

Round to #

of decimal places

Most used methods of the Math class.

Method	Description	Example
Abs (x)	absolute value of x	Abs (23.7) is 23.7 Abs (0) is 0 Abs (-23.7) is 23.7
Ceiling (x)	rounds x to the smallest integer not less than x	Ceiling (9.2) is 10.0 Ceiling (-9.8) is -9.0
Cos (x)	trigonometric cosine of x (x in radians)	Cos (0.0) is 1.0
Exp (x)	exponential method e^x	Exp (1.0) is approximately 2.7182818284590451 Exp (2.0) is approximately 7.3890560989306504
Floor (x)	rounds x to the largest integer not greater than x	Floor (9.2) is 9.0 Floor (-9.8) is -10.0
Log (x)	natural logarithm of x (base e)	Log (2.7182818284590451) is approximately 1.0 Log (7.3890560989306504) is approximately 2.0
Max (x, y)	larger value of x and y (also has versions for float , int and long values)	Max (2.3, 12.7) is 12.7 Max (-2.3, -12.7) is -2.3
Min (x, y)	smaller value of x and y (also has versions for float , int and long values)	Min (2.3, 12.7) is 2.3 Min (-2.3, -12.7) is -12.7
Pow (x, y)	x raised to power y (x^y)	Pow (2.0, 7.0) is 128.0 Pow (9.0, .5) is 3.0
Sin (x)	trigonometric sine of x (x in radians)	Sin (0.0) is 0.0
Sqrt (x)	square root of x	Sqrt (900.0) is 30.0 Sqrt (9.0) is 3.0
Tan (x)	trigonometric tangent of x (x in radians)	Tan (0.0) is 0.0

Fig. 6.2 Commonly used **Math** class methods.

Example 3 - Methods

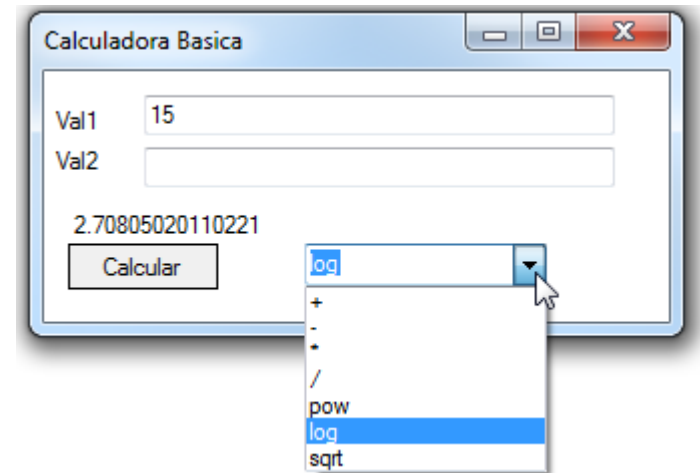
Using the code we already have for the project “Calculadora Basica” adapt the code to add two new operations (We will use the Math class inside of them).

Natural logarithm and Square root.

Math.Log(x) y Math.Sqrt(x)

```
case "log"://LOGARITMO NATURAL
    resultado = Math.Log(valor1);
    break;
```

```
case "sqrt"://RAIZ CUADRADA
    resultado = Math.Sqrt(valor1);
    break;
```



```

private void button_Calcular_Click(object sender, EventArgs e)
{
    String operador = this.comboBox1.Text;
    double resultado = 0;
    double valor1 = 0;
    double valor2 = 0;

    //Obtenemos los valores de los TextBox's
    if (this.textBox_Val1.Text != "")
        valor1 = Convert.ToDouble(this.textBox_Val1.Text);
    else
        valor1 = 0;

    if (this.textBox_Val2.Text != "")
        valor2 = Convert.ToDouble(this.textBox_Val2.Text);
    else
        valor2 = 0;

    switch (operador)
    {
        case "+"://SUMA
            resultado = doSuma(valor1, valor2);
            break;

        case "-"://RESTA
            resultado = doResta(valor1, valor2);
            break;

        case "*"://MULTIPLICACION
            resultado = doMult(valor1, valor2);
            break;

        case "/"://DIVISION
            resultado = doDiv(valor1, valor2);
            break;

        case "pow"://POTENCIA
            resultado = doPotencia(valor1, valor2);
            break;
    }
}

```

```

        case "log"://LOGARITMO NATURAL
            resultado = Math.Log(valor1);
            break;

        case "sqrt"://RAIZ CUADRADA
            resultado = Math.Sqrt(valor1);
            break;

        default://SIN OPERADOR
            MessageBox.Show("Selecciona un operador");
            break;
    }

    //Colocamos el resultado en el Label
    this.label_Resultado.Text = resultado.ToString();
}

private double doSuma(double valor1, double valor2)
{
    double resultado = valor1 + valor2;
    return resultado;
}

private double doResta(double valor1, double valor2)
{
    double resultado = valor1 - valor2;
    return resultado;
}

private double doMult(double valor1, double valor2)
{
    double resultado = valor1 * valor2;
    return resultado;
}

private double doDiv(double valor1, double valor2)
{
    double resultado = valor1 / valor2;
    return resultado;
}

private double doPotencia(double valor1, double valor2)
{
    double resultado = Math.Round(5.7356, 2);
    return resultado;
}

```

Example 4 - Methods

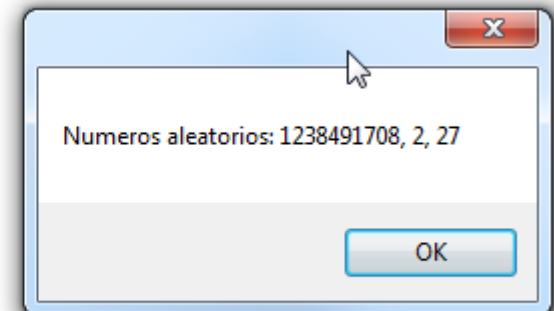
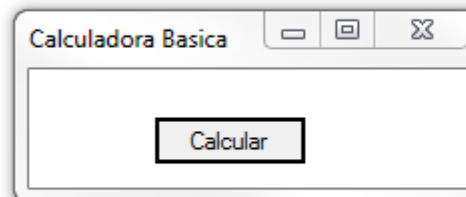
Using the code we already have adapt it so it will be capable of show 3 random numbers.

To solve this we will use a method called **Random**.

```
private void button_Calcular_Click(object sender, EventArgs e)
{
    //Creamos una instancia de la clase Random
    Random randomObject = new Random();

    int random = randomObject.Next();
    int r0y5 = randomObject.Next(5);
    int r20y30 = randomObject.Next(20, 30);

    string text = "Numeros aleatorios: " + random + ", " + r0y5 + ", " + r20y30;
    MessageBox.Show(text);
}
```



Arrays

The arrays are a set of data of the same type, you treat an array as a collection of objects with a defined and static size (the size of an array can't change once it's created).

A more low level approach will say that an array is an arrangement of memory blocks (the size of the blocks depends of the data type of the array).

If we have a collection of data arranged in a sequential way we can use this collection to perform series of operations. The way you access data inside an array in **C#** is using the square brackets `[]`.

For example, if we have an array (you can consider it as a variable) called *data*, then you can reference to the fourth element of the array using: *data[3]*

In this other example, the array called *c* has a size of 12.

We should remember that in C# the enumeration of the “places” inside an array begins in zero.

The data type of the array in the example is integer.

```
sum = c[ 0 ] + c[ 1 ] + c[ 2 ];
```

```
c.Length
```

```
x = c[ 6 ] / 2;
```

Name of array (Note
that all elements of
this array have the
same name, *c*)

Position number (index
or subscript) of the
element within array *c*

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Declaring Arrays

In order to declare an array the programmer must now beforehand the data type and the maximum numbers of item the array will handle. In order to declare an array the *new* operator must be used.

```
int[] c = new int[ 12 ];
```

That declarations creates an integer array with 12 elements. Another way to do that is in two different lines like:

```
int[] c; // declares the array  
c = new int[ 12 ]; // allocates the reference to the array
```

In a similar way we can declare other arrays with different data type.

```
double[] array1 = new double[ 10 ];  
string[] array2 = new string[ 20 ];
```

When the arrays are created they are automatically initialized. For primitive data types (int, float, double, bool...) all the elements are initialized to zero (false). For any other data type they are initialized to **null**.

Example 1 - Arrays

Create a program using two arrays, one for prices and other for products.

When the user clicks a button a message will pop up listing all the products with their respective prices.

```
private void button_Calcular_Click(object sender, EventArgs e)
{
    arregloManual();
}

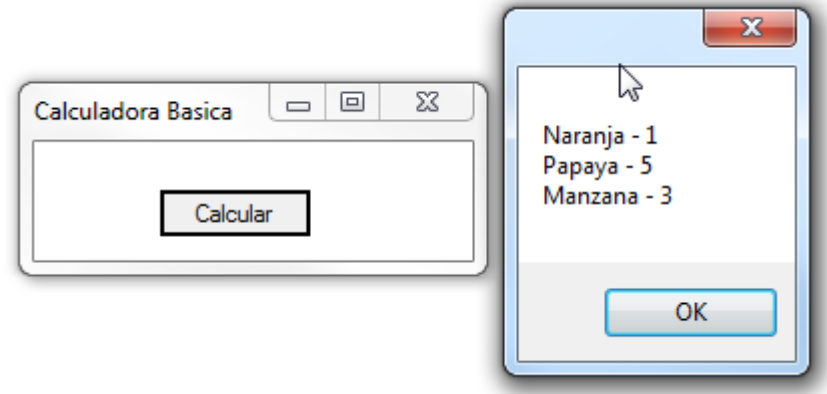
private void arregloManual()
{
    string[] productos = new string[3];
    int[] precios = new int[3];

    productos[0] = "Naranja";
    productos[1] = "Papaya";
    productos[2] = "Manzana";

    precios[0] = 1;
    precios[1] = 5;
    precios[2] = 3;

    string text = productos[0] + " - " + precios[0] + "\n";
    text += productos[1] + " - " + precios[1] + "\n";
    text += productos[2] + " - " + precios[2];

    MessageBox.Show(text);
}
```



Now rewrite the previous program, and instead of accessing manually the array, use a *for loop*.

Here you can see the potential of this flow control structure.

```
private void button_Calcular_Click(object sender, EventArgs e)
{
    arregloAuto();
}

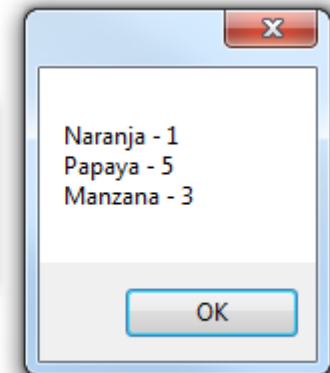
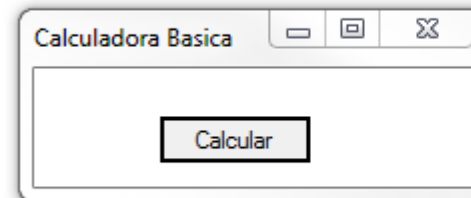
private void arregloAuto()
{
    string[] productos = new string[3];
    int[] precios = new int[3];

    productos[0] = "Naranja";
    productos[1] = "Papaya";
    productos[2] = "Manzana";

    precios[0] = 1;
    precios[1] = 5;
    precios[2] = 3;

    string text = "";
    for (int i = 0; i < 3; i++)
    {
        text += productos[i] + " - " + precios[i] + "\n";
    }

    MessageBox.Show(text);
}
```



Example 2 - Arrays

We are going to update the code we have for the ISR calculator in order to perform a better calculation.

You can use the monthly table <http://losimpuestos.com.mx/tablas-isr-2016>

Remember, the formula to calculate the Net Salary is:

$$\mathbf{SueldoNeto = SueldoBruto - ISR}$$

Where

$$\mathbf{ISR = CuotaFija + (ExcedenteLimtInf * p)}$$

$$\mathbf{ExcedenteLimtInf = SueldoBruto - LimiteInferior}$$

All the other variables (*CuotaFija*, *LimiteInferior* y *p*) can be extracted from the table using the Nominal Salary (Salario Bruto).

```

namespace CalculadoraBasica
{
    public partial class main : Form
    {
        //Creamos los arreglos con los datos de la tabla de ISR
        private double[] limitInfData;
        private double[] limitSupData;
        private double[] cuotaFixData;
        private double[] percentData;

        public main()
        {
            InitializeComponent();
        }

        private void button_Calcular_Click(object sender, EventArgs e)
        {
            if (this.textBox1.Text != "")
            {
                //Obtenemos el sueldo Neto introducido por el usuario
                double sueldoBruto = Convert.ToDouble(this.textBox1.Text);

                if (sueldoBruto > 0)
                {
                    //Si el sueldo neto es mayor a cero, continuar
                    //Creamos los arreglos de datos
                    createISRTTable();
                    int tam = this.limitInfData.Length;
                    double sueldoNeto, exclimitInf, ISR, limitInf, limitSup, cuotaFix, percent;

                    for (int i = 0; i < tam; i++)
                    {
                        //Iteramos tantas veces como el tamaño del arreglo
                        if ((sueldoBruto >= limitInfData[i]) && (sueldoBruto <= limitSupData[i]))
                        {
                            limitInf = limitInfData[i];
                            limitSup = limitSupData[i];
                            cuotaFix = cuotaFixData[i];
                            percent = percentData[i]/100.0;

                            //Aplicamos las operaciones
                            exclimitInf = sueldoBruto - limitInf;
                            ISR = cuotaFix + (exclimitInf * percent);
                            sueldoNeto = sueldoBruto - ISR;

                            //Mandamos mensaje
                            string text = "ISR: " + ISR.ToString() + "\n";
                            text += "Sueldo Neto: " + sueldoNeto.ToString() + "\n";
                            MessageBox.Show(text);

                            break; //Rompe la iteración
                        }
                    }
                }
            }
        }
    }
}

```

```
//Creamos los arreglos de datos y los llenamos
```

```
private void createISRTTable()
```

```
{
```

```
    //Data from: http://losimpuestos.com.mx/tablas-isr-2015/
```

```
    limitInfData = new double[11];
```

```
    limitSupData = new double[11];
```

```
    cuotaFixData = new double[11];
```

```
    percentData = new double[11];
```

```
    //Datos de limite inferior
```

```
    limitInfData[0] = 0.01;
```

```
    limitInfData[1] = 496.08;
```

```
    limitInfData[2] = 4210.42;
```

```
    limitInfData[3] = 7399.43;
```

```
    limitInfData[4] = 8601.51;
```

```
    limitInfData[5] = 10298.36;
```

```
    limitInfData[6] = 20770.30;
```

```
    limitInfData[7] = 32736.84;
```

```
    limitInfData[8] = 62500.01;
```

```
    limitInfData[9] = 83333.34;
```

```
    limitInfData[10] = 250000.01;
```

```
    //Datos de limite superior
```

```
    limitSupData[0] = 496.07;
```

```
    limitSupData[1] = 4210.41;
```

```
    limitSupData[2] = 7399.42;
```

```
    limitSupData[3] = 8601.50;
```

```
    limitSupData[4] = 10298.35;
```

```
    limitSupData[5] = 20770.29;
```

```
    limitSupData[6] = 32736.83;
```

```
    limitSupData[7] = 62500.00;
```

```
    limitSupData[8] = 83333.33;
```

```
    limitSupData[9] = 250000.00;
```

```
    limitSupData[10] = Double.PositiveInfinity;
```

```
    //Cuota Fija
```

```
    cuotaFixData[0] = 0.0;
```

```
    cuotaFixData[1] = 9.52;
```

```
    cuotaFixData[2] = 247.24;
```

```
    cuotaFixData[3] = 594.21;
```

```
    cuotaFixData[4] = 786.54;
```

```
    cuotaFixData[5] = 1090.61;
```

```
    cuotaFixData[6] = 3327.42;
```

```
    cuotaFixData[7] = 6141.95;
```

```
    cuotaFixData[8] = 15070.90;
```

```
    cuotaFixData[9] = 21737.57;
```

```
    cuotaFixData[10] = 78404.23;
```

```
    //Porcentaje sobre el Excedente del Limite Inferior
```

```
    percentData[0] = 1.92;
```

```
    percentData[1] = 6.4;
```

```
    percentData[2] = 10.88;
```

```
    percentData[3] = 16.0;
```

```
    percentData[4] = 17.92;
```

```
    percentData[5] = 21.36;
```

```
    percentData[6] = 23.52;
```

```
    percentData[7] = 30.0;
```

```
    percentData[8] = 32.0;
```

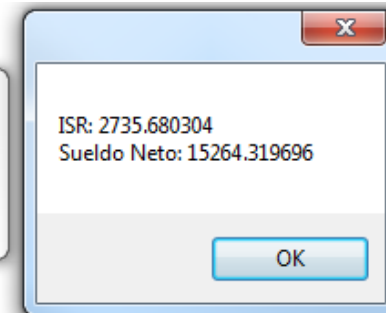
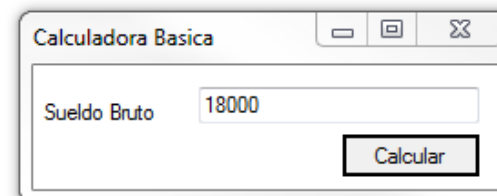
```
    percentData[9] = 34.0;
```

```
    percentData[10] = 35.0;
```

```
}
```

```
}
```

```
}
```



Jagged Arrays

A [jagged array](#) is an array whose elements are arrays. The elements of a jagged array can be of different dimensions and sizes. A jagged array is sometimes called an "array of arrays."

We can see the this jagged array as a table, where each element is a cell.

You can declare a jagged array as follow:

```
int[][] jaggedArray = new int[3][];
```

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

NOTE: There are multidimensional arrays, but they have poor performance compared to jagged arrays. In this course we will learn only about jagged arrays.

Example 3 - Arrays

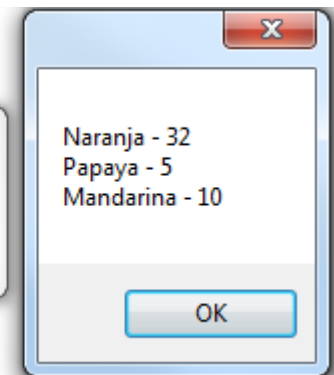
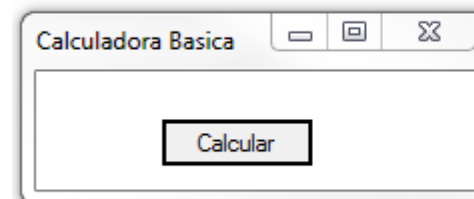
Using the Fruit – Price example, build a jagged array to hold all the values.

```
private void button_Calcular_Click(object sender, EventArgs e)
{
    string[][] data = new string[3][];
    data[0] = new string[2];
    data[1] = new string[2];
    data[2] = new string[2];

    data[0][0] = "Naranja";
    data[0][1] = "32";
    data[1][0] = "Papaya";
    data[1][1] = "5";
    data[2][0] = "Mandarina";
    data[2][1] = "10";

    string text = data[0][0] + " - " + data[0][1] + "\n";
    text += data[1][0] + " - " + data[1][1] + "\n";
    text += data[2][0] + " - " + data[2][1];

    MessageBox.Show(text);
}
```



4.7-CalculadoraBasica-BidimensionalArrays

Ejercicios de Apoyo

1. Crea un programa que llene un arreglo sencillo de 30 elementos de tipo entero con números aleatorios, y que después mande un mensaje enumerando dichos elementos.
2. Crea arreglo sencillo tipo entero de 10 elementos, llénalo con cualquier información, escribe el código para que al momento de hacer click a un botón lea un número de un TextBox y mande un mensaje indicando si dicho número se encuentra en el arreglo o no y en que posición.
3. Modifica el **Ejemplo 2 - Arrays** para que en lugar de tener 4 variables de arreglos sencillos conteniendo la información solo se tenga un “jagged array” (array de arrays). Por ejemplo:

```
private double[] [] data = new double[11] [];
```

HOMEWORK

1. Build a program that fills an array of 30 elements using random numbers, then display a message listing all the elements.
2. Build a program that has an array of size 10 with some numbers inside (integer data type), the user should be able to type a number in a TextBox and when the button is clicked the program should tell if the number typed is inside or not the array.
3. Update the code of the **Example 2 – Methods**, instead of having 4 variables use a jagged array. Something like

```
private double[] [] data = new double[11] [];
```