SENSOR SUBMARINO

Sistema de Gestión de Lecturas de Temperatura

Romero Rodríguez Uriel Augusto
DS02SM-24

Introducción

Se desarrolló la aplicación **Sensor Submarino** para llevar el registro de las temperaturas de un sensor ubicado en la costa de Guerrero. Este dispositivo registra la temperatura del agua cada hora durante toda la semana. Esto permite recolectar un volumen considerable de datos que es necesario analizar para obtener promedios, máximos y mínimos.

Para el almacenamiento de estos datos, fue necesario decidir cómo manejar la memoria. Se eligió utilizar **memoria estática**.

La razón es simple: se conoce exactamente la cantidad de datos que se obtendrán. Dado que el sensor opera 24 horas por 7 días, siempre se tendrán 168 lecturas, ni una más ni una menos. Al no cambiar el tamaño de los datos, no es necesario utilizar memoria dinámica. Un arreglo fijo resulta adecuado y más sencillo de manejar para este problema.

Desarrollo de la Aplicación

El elemento principal del programa es un arreglo de dos dimensiones (una matriz) donde se guardan las horas y los días. Esto facilita la localización de cualquier dato, por ejemplo, la temperatura del martes a las 10 am. El código para crear esta estructura es el siguiente:

```
int[,] temp = new int[24, 7];
```

El 24 corresponde a las filas (horas del día) y el 7 a las columnas (días de la semana).

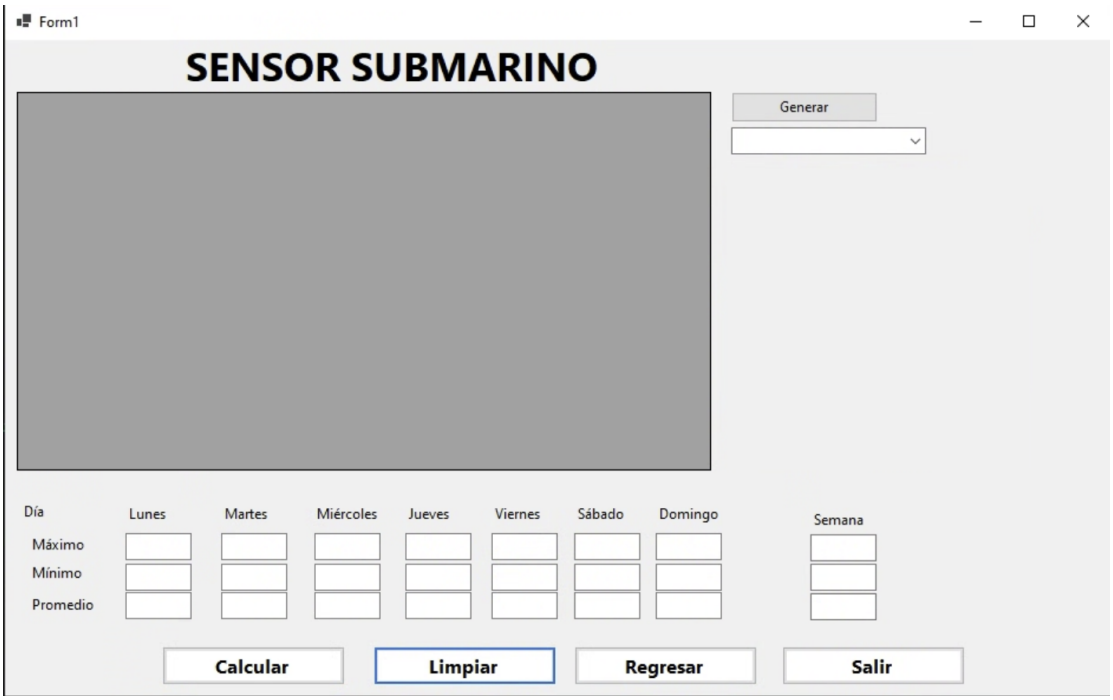
Interfaz de Usuario

Se utilizó Windows Forms para el diseño de la pantalla. Se colocó una tabla (‘DataGridView’) para visualizar los valores numéricos, un cuadro de imagen (‘PictureBox’) donde se dibuja la gráfica, y varios paneles con cajas de texto para mostrar los resultados correspondientes a cada día y a la semana completa.

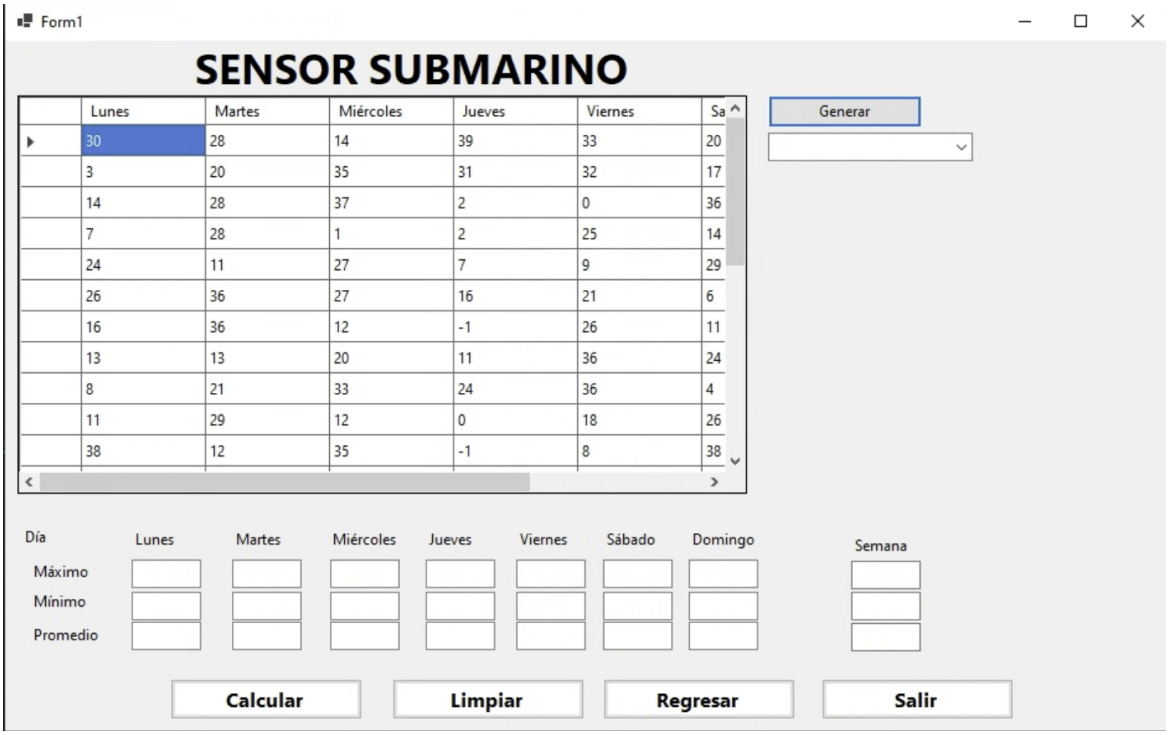
Generación de Datos

Para verificar el funcionamiento, se implementó un botón que llena la tabla con números aleatorios. Se configuró para generar temperaturas entre -2 y 40 grados para simular un escenario real.

```
private void btnGenerar_Click(object sender, EventArgs e)
{
    Random r = new Random();
```




```
    DataTable dt = new DataTable();

    // Se agregan las columnas de los días
    dt.Columns.Add("Lunes"); dt.Columns.Add("Martes");
    dt.Columns.Add("Miércoles"); dt.Columns.Add("Jueves");
    dt.Columns.Add("Viernes"); dt.Columns.Add("Sabado");
    dt.Columns.Add("Domingo");

    // Se generan 24 filas, una por hora
    for (int i = 0; i < 24; i++)
    {
        DataRow fila = dt.NewRow();

        for (int j = 0; j < 7; j++)
        {
            // Se genera una temperatura entre -2 y 40
            temp[i,j] = r.Next(-2, 40);

            // Se almacena para visualización en la tabla
            fila[j] = temp[i,j];

            // Y también en el combo box
            cmbTemp.Items.Add(temp[i,j]);
        }
        dt.Rows.Add(fila);
    }

    // Se muestra todo en la cuadrícula
    dtgTemp.DataSource = dt;
}
```



Cálculos

En esta sección se realizan las operaciones. Se recorre toda la matriz mediante ciclos para encontrar la temperatura más alta, la más baja y calcular el promedio de cada día.

```
private void btnCalcular_Click(object sender, EventArgs e)
{
    int maxDia, minDia;
    double promDia = 0;

    // Valores iniciales para la semana
    maxSem = temp[0, 0];
    minSem = temp[0, 0];

    // Se recorre columna por columna (días)
    for (int columnas = 0; columnas < 7; columnas++)
    {
        promDia = 0;
        maxDia = temp[0, columnas];
        minDia = temp[0, columnas];

        // Se recorre fila por fila (horas)
        for (int filas = 0; filas < 24; filas++)
        {
            // Se verifica si es el nuevo máximo o mínimo del día
            if (temp[filas, columnas] > maxDia) maxDia = temp[filas, columnas];
            if (temp[filas, columnas] < minDia) minDia = temp[filas, columnas];

            // Se acumula para los promedios
            promDia += temp[filas, columnas];
            promSem += temp[filas, columnas];
```

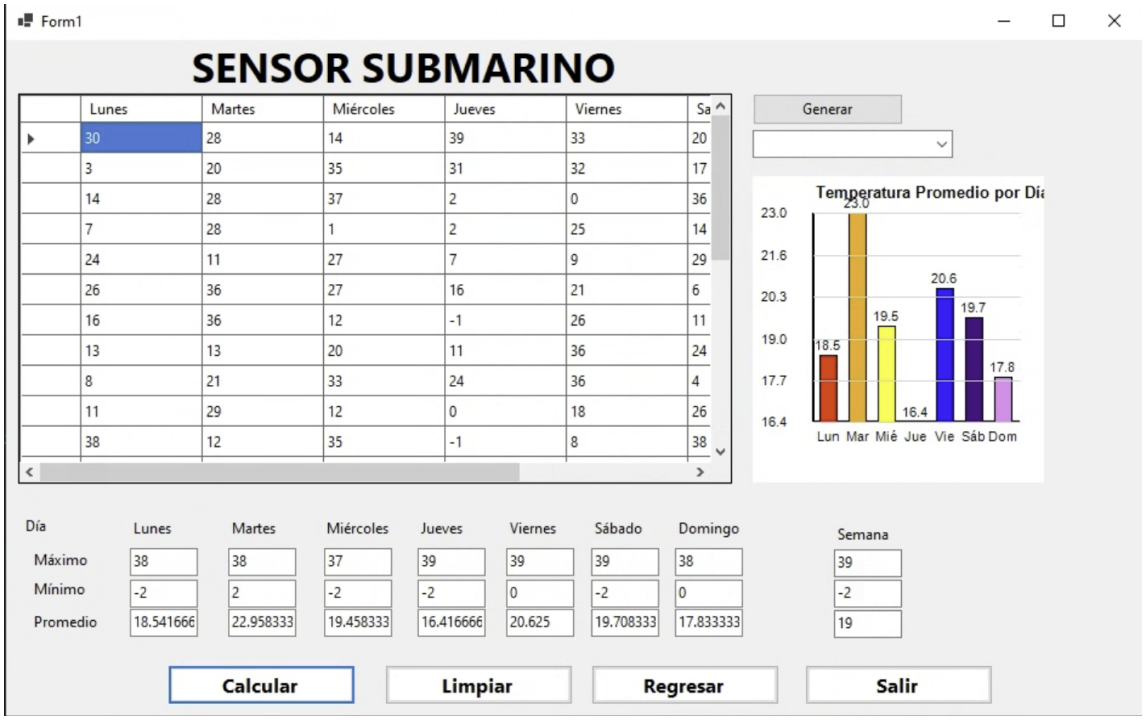
```
            // Se verifican máximos y mínimos de toda la semana
            if (temp[filas, columnas] > maxSem) maxSem = temp[filas, columnas];
            if (temp[filas, columnas] < minSem) minSem = temp[filas, columnas];
        }

        // Se calcula el promedio del día dividiendo entre 24
        promDia = promDia / 24;
        // ... aquí se muestran los resultados en las cajas de texto ...
    }

    // Promedio de toda la semana (168 datos)
    promSem = promSem / 168;

    // Se muestran los resultados de la semana
    tprSem.Text = promSem.ToString();
    tmaxSem.Text = maxSem.ToString();
    tminSem.Text = minSem.ToString();

    GenerarGrafico(); // Se invoca a la función de graficado
}
```

Gráfica

Para mejorar la visualización, se dibuja una gráfica de barras. Se utiliza un 'Bitmap' como lienzo. Se calcula el tamaño de las barras en función del promedio para ajustarlas correctamente al cuadro.

```
// Paso 1: Se prepara el dibujo en blanco
Bitmap bmp = new Bitmap(pictureBox1.Width, pictureBox1.Height);
Graphics g = Graphics.FromImage(bmp);
g.Clear(Color.White);

// Paso 2: Se dibujan las líneas de los ejes
g.DrawLine(penEjes, margenIzq, margenSup + altoGrafico,
           margenIzq + anchoGrafico, margenSup + altoGrafico); // Eje X
g.DrawLine(penEjes, margenIzq, margenSup, margenIzq, margenSup + altoGrafico); // E
g.DrawString("Temperatura Promedio por Día (°C)", fontTitulo, Brushes.Black, margen

// Paso 3: Se dibujan las barras
for (int i = 0; i < 7; i++)
{
    // Se calcula la altura de la barra
    int alturaBarra = (int)((promediosDiarios[i] - minTemp) / rangoTemp * altoGraf

    int x = margenIzq + (i * anchoBarra) + 5;
    int y = margenSup + altoGrafico - alturaBarra;

    // Se dibuja el rectángulo con color
    g.FillRectangle(new SolidBrush(colores[i]), x, y, anchoBarra - 10, alturaBarra
    g.DrawRectangle(Pens.Black, x, y, anchoBarra - 10, alturaBarra);

    // Se colocan los números y nombres
```

```
    g.DrawString(promediosDiarios[i].ToString("F1"), font, Brushes.Black, x, y - 15
    g.DrawString(diasSemana[i], font, Brushes.Black, x, margenSup + altoGrafico + 5
}
```



Botón de Limpiar

Finalmente, se creó un botón para borrar la información y reiniciar el proceso. Aunque el arreglo permanece en memoria, se asignan ceros a todos los elementos para limpiar los datos.

```
private void btnLimpiar_Click(object sender, EventArgs e)
{
    dtgTemp.DataSource = null; // Se limpia la tabla
    cmbTemp.Items.Clear();     // Se vacía la lista

    // Se borra el texto de las cajas
    foreach (Control control in panel1.Controls)
    {
        if (control is TextBox) control.Text = "";
    }

    pictureBox1.Image = null; // Se elimina la gráfica

    // Se restablecen a cero los valores de la matriz
    for (int i = 0; i < 24; i++)
        for (int j = 0; j < 7; j++)
            temp[i, j] = 0;
}
```

Conclusiones

Este ejercicio permitió practicar el uso de matrices (arreglos bidimensionales) en memoria estática. Se observó que cumple con lo necesario para un funcionamiento eficiente y directo.

Al utilizar un arreglo fijo, el código resultó sencillo y no se presentaron problemas de rendimiento. Se confirmó que cuando se conoce el tamaño de los datos (como en este caso, donde eran fijos), esta es la opción más adecuada.