Benemérita Universidad Autónoma de Puebla

Materia: *Sistemas Operativos II*

Profesor: *Gilberto López Poblano*

Alumno: *Christian Amauri Amador Ortega - 201927821*

- Archivo txt de configuración -

Para esta serie de prácticas con código utilizamos el lenguaje C# en el IDE visual studio (la versión más reciente en septiembre de 2022). Puede haber variaciones en el funcionamiento de exactamente los mismos códigos pero con otra versión de visual studio u otro IDE. De todas formas estas variaciones deben de ser mínimas e insignificantes.

!!! Documentaremos el funcionamiento de las aplicaciones (en su ejecución) y haremos algunos comentarios / notas / consideraciones / advertencias al respecto. Para ahorrar tiempo y esfuerzo, trataremos de no ahondar demasiado en la estructura del código, sino simplemente explicar brevemente cómo funciona ***(se asume que el lector tiene ya conocimientos suficientes sobre conceptos varios de programación estructurada, ambiente gráfico, variables, datos, y demás…)***

Finalmente hay que tomar en cuenta que C# y Visual studio son herramientas pesadas y caprichosas. Y que el equipo que vayamos a utilizar debe tener cierta capacidad de rendimiento superior a sólo básica.

Primero, el objetivo del proyecto es crear un programa en C# con el que podamos escribir y leer documentos con extensión .txt en los que podamos editar ciertas características básicas del texto a trabajar (tales como fuente, tamaño, color, etc….)

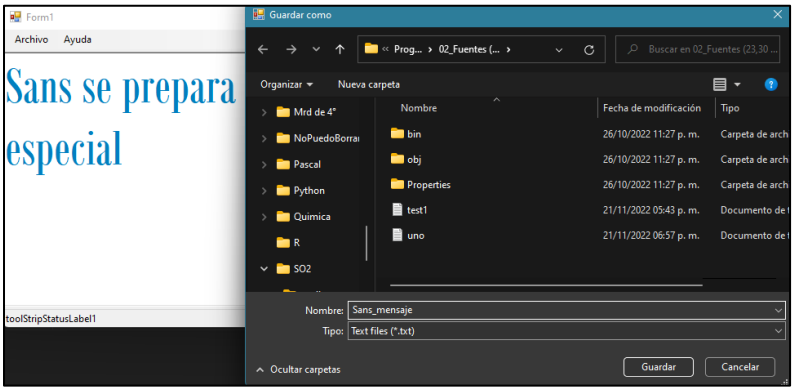
En cualquier caso, el repositorio en el que este código se encuentra (y otros códigos en C# del curso de Sistemas operativos 2, BUAP FCC, Otoño 2022) está disponible haciendo click en el siguiente enlace:

https://drive.google.com/drive/folders/1yZujMI51XAnEZBMj-4ykUWMfHXBC9uYp?usp=sharing

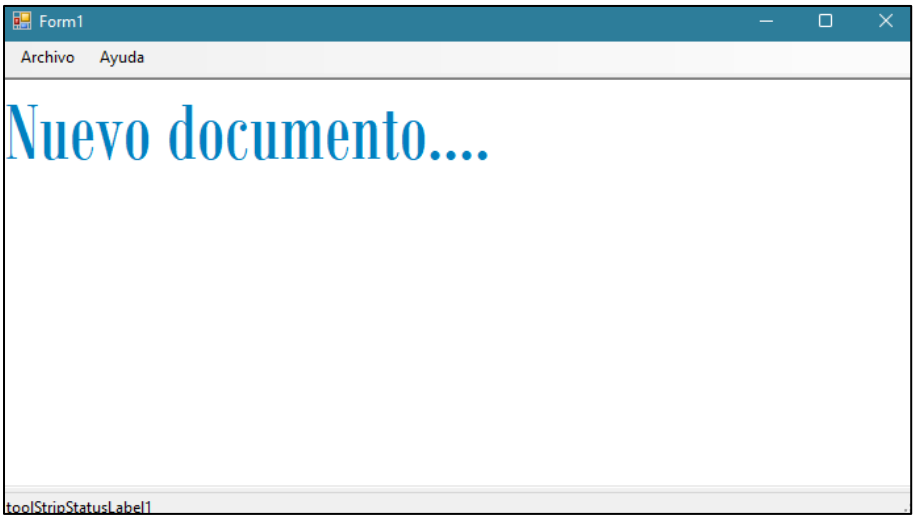
Y estará disponible al menos durante el resto de 2022. Gracias. 😊

Ejemplo rápido del funcionamiento general del proyecto, para resumir:

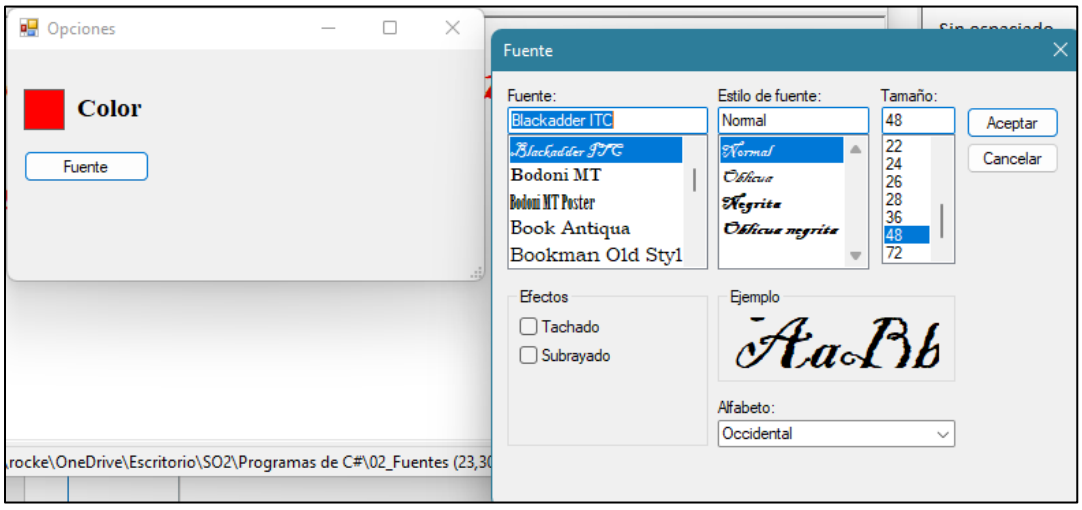
- Podemos escribir cosas y guardarlas en documentos de texto:

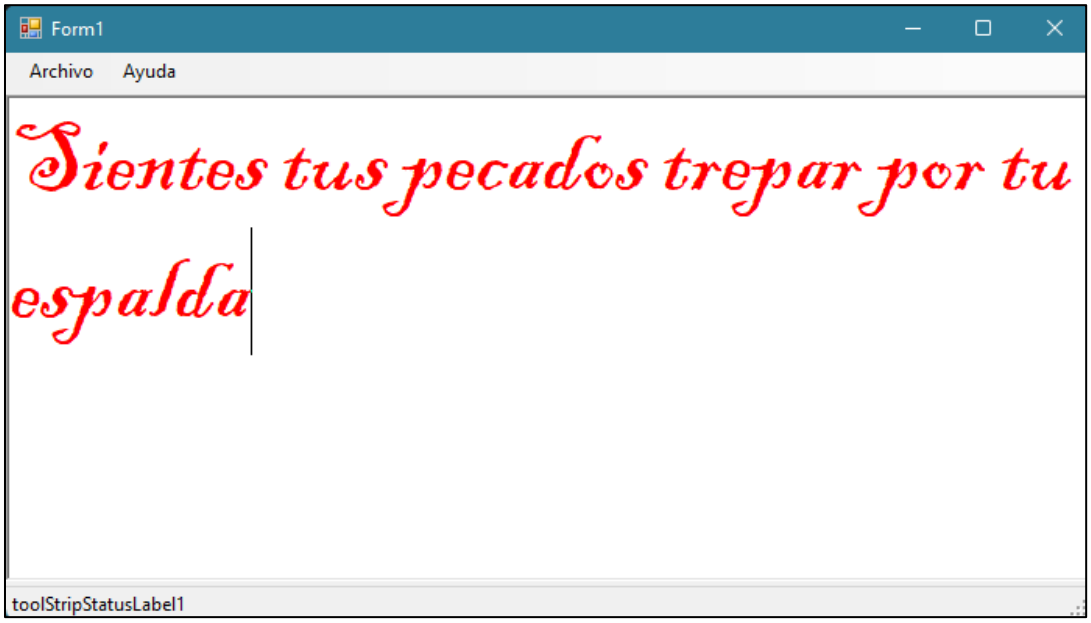


- Cuando volvamos a abrir el programa, se restaurarán las características de la fuente que definimos por ultima vez:



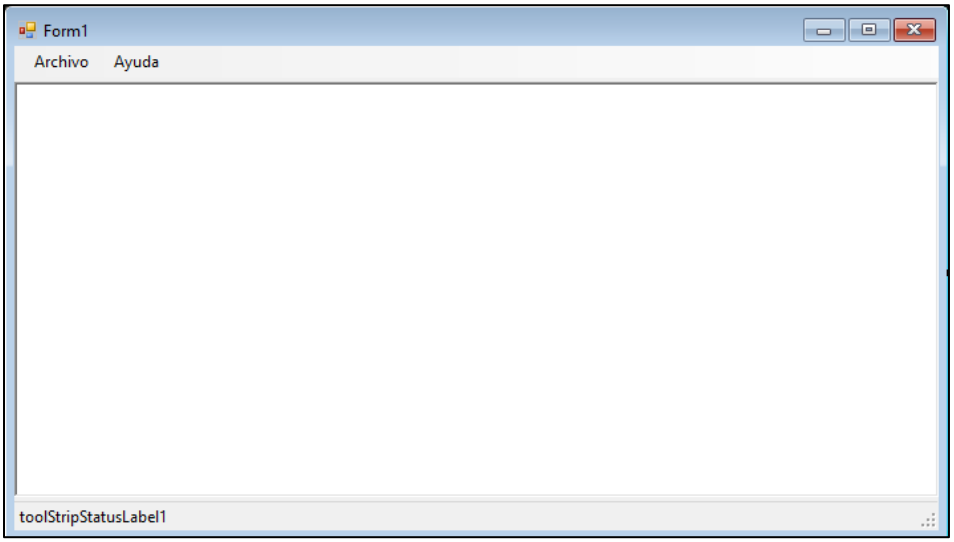
- Cambiamos el estilo de la fuente…





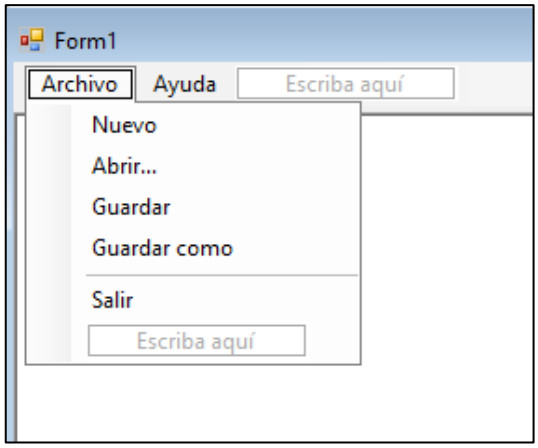
Ahora, los archivos .cs del proyecto (que nosotros creamos) corresponden a 3 diseños y 2 scripts. Comencemos por los diseños…

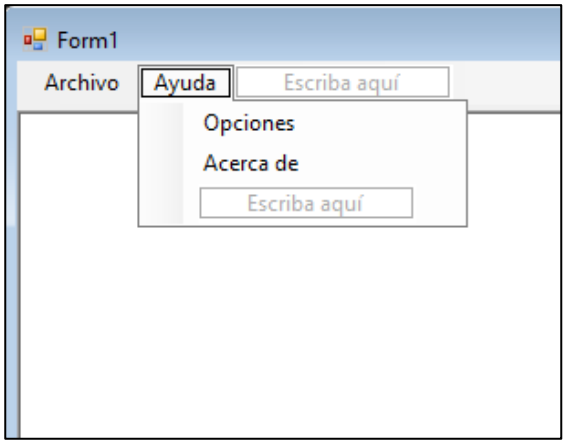
- Diseños -

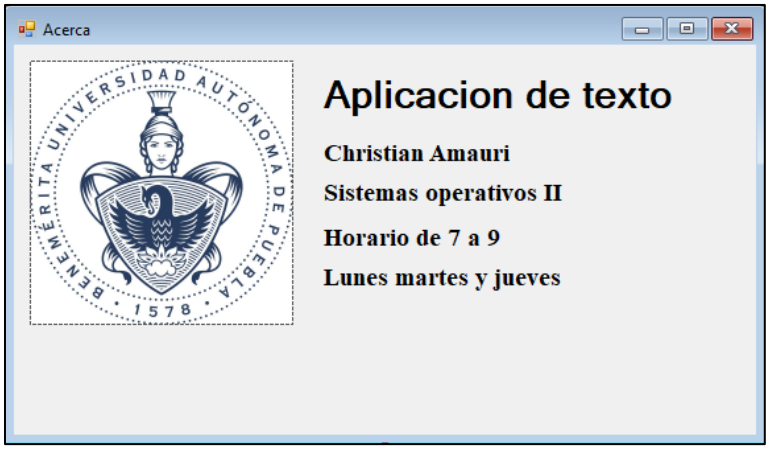


#1 **Form1** (Llamémoslo Form main)

Contiene el panel en el que escribiremos el texto cuyas características queremos editar al gusto, y un menu strip con las opciones: "**Archivo**" y "**Ayuda**". En el apartado de "Archivo" tenemos las opciones: Nuevo, Abrir, Guardar, Guardar como y Salir. En el apartado de "ayuda" tenemos las opciones: Opciones y Acerca de.

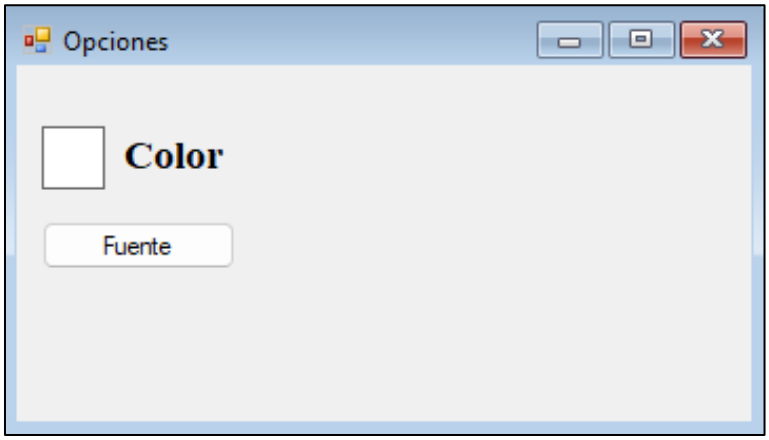






#2 **Acerca**

Esta es la ventana / diseño que la ejecución del programa nos mostrará en caso de acceder a la opción “Acerca de” de la opción “Ayuda” del menú strip del Form main. Esta ventana simplemente **contiene información sobre el autor del código** (no posee funcionalidad extra).



#3 **Opciones**

Esta es la ventana / diseño que la ejecución del programa nos mostrará en caso de acceder a la opción “Opciones” de la opción “Ayuda” del menú strip del Form main. **Contiene un pequeño panel en el que se muestra el color en el cual el texto está siendo escrito, y un botón para escoger la fuente en la que el texto está escrito.** Los cambios se guardan inmediatamente después de definirlos (en caso de que se haya hecho bien ese proceso) mediante funciones pre-definidas por el lenguaje/IDE.

- Scripts -

#1 ***Opciones.cs***

La estructura inicial de este script consta de:

- 10 librerías
- namespace definido como: _03_Fuentes
- public partial class Opciones : Form

```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9   using System.Windows.Forms;
10  using System.IO;
11
12  namespace _03_Fuentes
13  {
        5 referencias
14      public partial class Opciones : Form
15      {
16          public Form1 f1;
            0 referencias
```

Dentro de Opciones : Form. se encuentran los siguientes componentes:

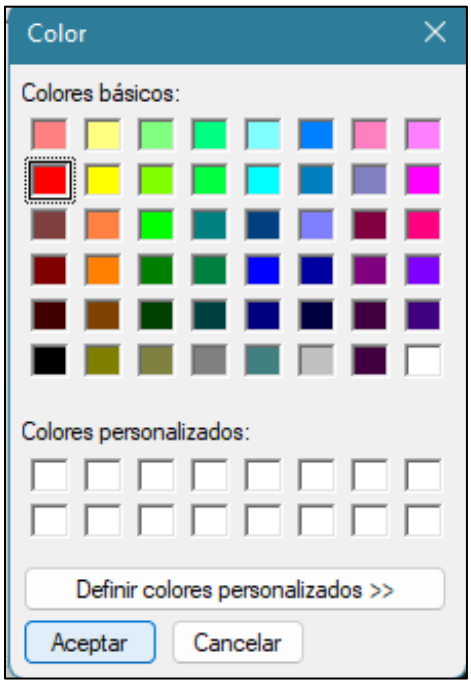
```
        5 referencias
14      public partial class Opciones : Form
15      {
16          public Form1 f1;
            0 referencias
17          public Opciones()...
22
            1 referencia
23          public Opciones(Form1 f1)...
28
            1 referencia
29          private void panel1_Click(object sender, EventArgs e)...
45
            1 referencia
46          public void SaveFile(String nombre, String fuente, float size, FontStyle style)...
53
            1 referencia
54          private void button1_Click(object sender, EventArgs e)...
67      }
68  }
```

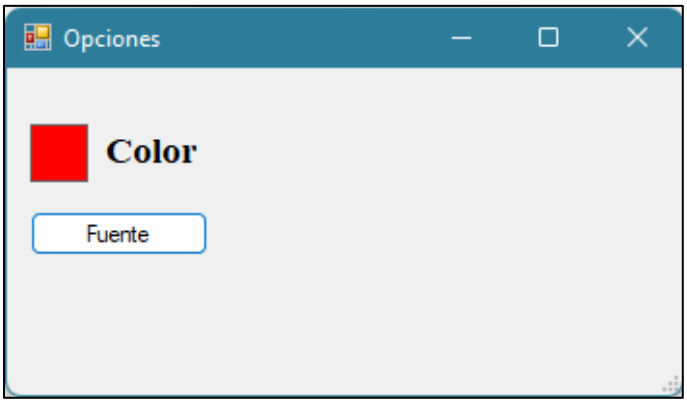
- una variable (tipo Form1)
- 2 constructores
- 3 métodos

La función `panel1_Click()` Sirve para que al presionar el pequeño panel ubicado en la ventana "Opciones" se nos abra un pequeño menú para seleccionar el color del que queremos pintar el texto.

```
1 referencia
private void panel1_Click(object sender, EventArgs e)
{
    if (this.colorDialog1.ShowDialog() == DialogResult.OK)
    {
        this.panel1.BackColor = colorDialog1.Color;
        f1.colorTexto(colorDialog1.Color);
        f1.newColor(colorDialog1.Color.R, colorDialog1.Color.G, colorDialog1.Color.B);

        String linea = Path.GetFullPath("color.txt");
        f1.notificacion(linea);
        StreamWriter sw = new StreamWriter(linea, false);

        sw.Write(colorDialog1.Color.R +";"+ colorDialog1.Color.G + ";" + colorDialog1.Color.B);
        sw.Close();
    }
}
```

Al escoger el color y presionar aceptar, se indica en el panel, el color en el que estamos escribiendo:



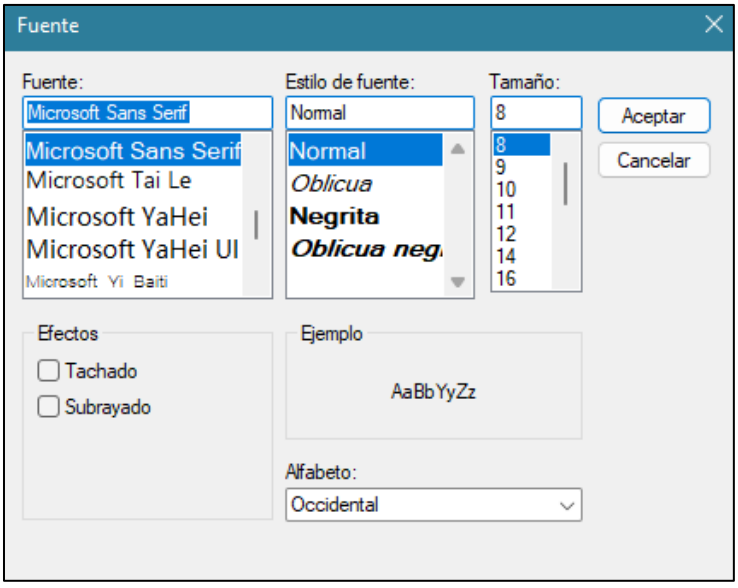


La función `SaveFile()` sirve para guardar las características que definimos para el texto (fuente, tamaño, estilo). estas características se guardan para el programa, no para el texto:

```
1 referencia
public void SaveFile(String nombre, String fuente, float size, FontStyle style)
{
    String linea = Path.GetFullPath(nombre);
    StreamWriter sw = new StreamWriter(linea, false);
    sw.Write(fuente + ";" + size.ToString() + ";" + Convert.ToInt32(style));
    sw.Close();
}
```

La función `button1_Click()` corresponde al botón “Fuente” en la ventana “Opciones” sirve para cambiar la fuente del texto mediante un fontDialog:

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{

    if (fontDialog1.ShowDialog() == DialogResult.OK)
    {
        String name = fontDialog1.Font.Name;
        float size = fontDialog1.Font.Size;
        FontStyle stilo = fontDialog1.Font.Style;

        f1.fuente(name, size, stilo);
        SaveFile("fuente.data",name, size, stilo);
    }
}
```



#2 ***Form1.cs***

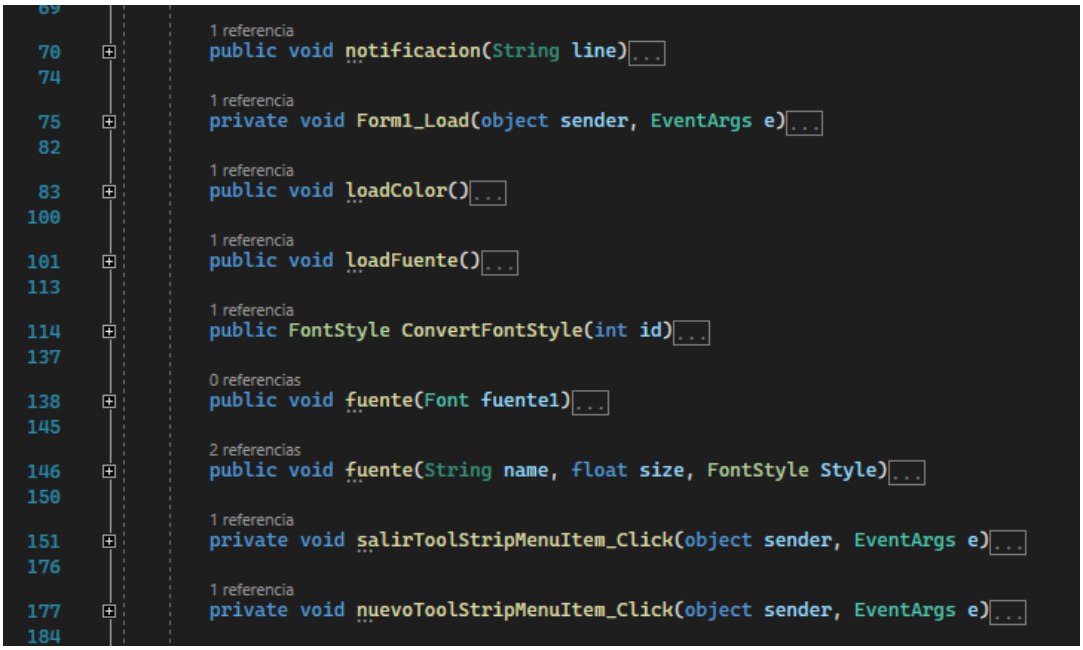
La estructura inicial de este script consta de:

- 11 librerías
- namespace definido como: _03_Fuentes
- public partial class Form1 : Form

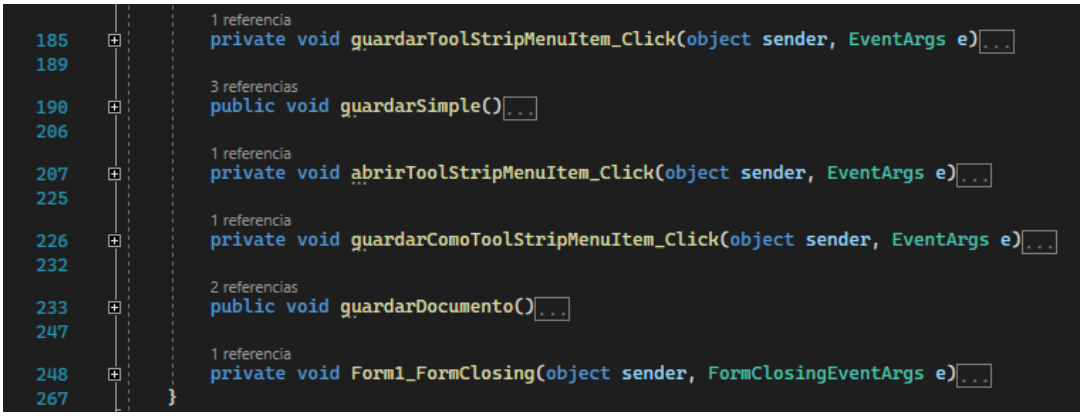
```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.IO;
7   using System.Linq;
8   using System.Text;
9   using System.Threading.Tasks;
10  using System.Windows.Forms;
11  using System.Xml.Linq;
12
13  namespace _03_Fuentes
14  {
        5 referencias
15      public partial class Form1 : Form
16      {
```

Dentro de Form1 : Form. se encuentran los siguientes componentes:

```
        5 referencias
15      public partial class Form1 : Form
16      {
17          String Documento;
18          int estados;
19          String Direccion;
20          Acerca acerca;
21          Opciones op;
22
            1 referencia
23          public Form1()...
27
            1 referencia
28          private void acercaDeToolStripMenuItem_Click(object sender, EventArgs e)...
40
            1 referencia
41          private void opcionesToolStripMenuItem_Click(object sender, EventArgs e)...
53
            1 referencia
54          public void colorTexto(Color c)...
58
            2 referencias
59          public void newColor(int red, int green, int blue)...
64
            1 referencia
65          private void toolStripStatusLabel1_Click(object sender, EventArgs e)...
69
```



```
1 referencia
public void notificacion(String line)...

1 referencia
private void Form1_Load(object sender, EventArgs e)...

1 referencia
public void loadColor()...

1 referencia
public void loadFuente()...

1 referencia
public FontStyle ConvertFontStyle(int id)...

0 referencias
public void fuente(Font fuente1)...

2 referencias
public void fuente(String name, float size, FontStyle Style)...

1 referencia
private void salirToolStripMenuItem_Click(object sender, EventArgs e)...

1 referencia
private void nuevoToolStripMenuItem_Click(object sender, EventArgs e)...
```



```
1 referencia
private void guardarToolStripMenuItem_Click(object sender, EventArgs e)...

3 referencias
public void guardarSimple()...

1 referencia
private void abrirToolStripMenuItem_Click(object sender, EventArgs e)...

1 referencia
private void guardarComoToolStripMenuItem_Click(object sender, EventArgs e)...

2 referencias
public void guardarDocumento()...

1 referencia
private void Form1_FormClosing(object sender, FormClosingEventArgs e)...
}
```

- 5 variables para uso general (tipo String, Int, Acerca y Opciones)
- un constructor
- 20 métodos

Muchas de estas funciones/métodos realizan tareas bastante simples que pueden ser entendidas por un programador con apenas unos vistazos, tales como: `acercaDeToolStripMenuItem_Click()` y `opcionesToolStripMenuItem_Click()` : Las cuales al verlas se entiende que crean ventanas de tipo ***Acerca*** y ***Opciones*** respectivamente.

```
1 referencia
private void acercaDeToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (acerca == null)
    {
        acerca = new Acerca();
        acerca.ShowDialog();
    }
    else
    {
        acerca.ShowDialog();
    }
}
```

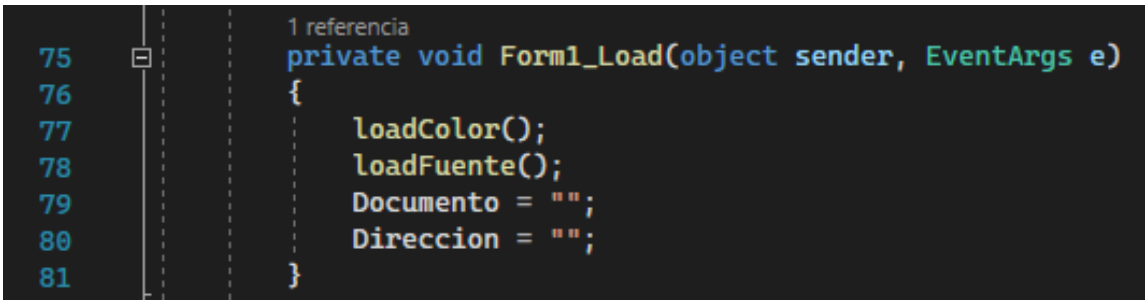
```
1 referencia
private void opcionesToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (op == null)
    {
        op = new Opciones(this);
        op.ShowDialog();
    }
    else
    {
        op.ShowDialog();
    }
}
```

O la función `loadColor()` cuyo propósito es obtener información del color de un texto previamente guardado:

```
1 referencia
public void loadColor()
{
    String archivo = Path.GetFullPath("color.txt");
    if (File.Exists(archivo))
    {
        StreamReader sr = new StreamReader(archivo);
        String Scolor = sr.ReadToEnd();
        sr.Close();
        String[] rgb = Scolor.Split(';');

        int r = Convert.ToInt32(rgb[0]);
        int g = Convert.ToInt32(rgb[1]);
        int b = Convert.ToInt32(rgb[2]);
        newColor(r, g, b);

    }
}
```

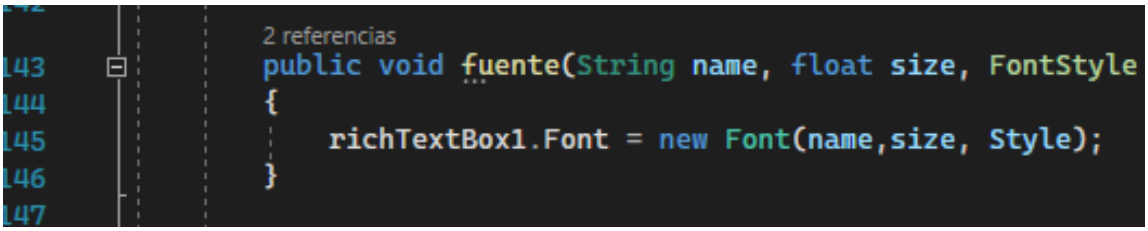
(Concepto que se repite con la función `loadFuente()`)

Hay funciones que llaman a otras funciones para agilizar todo un proceso:



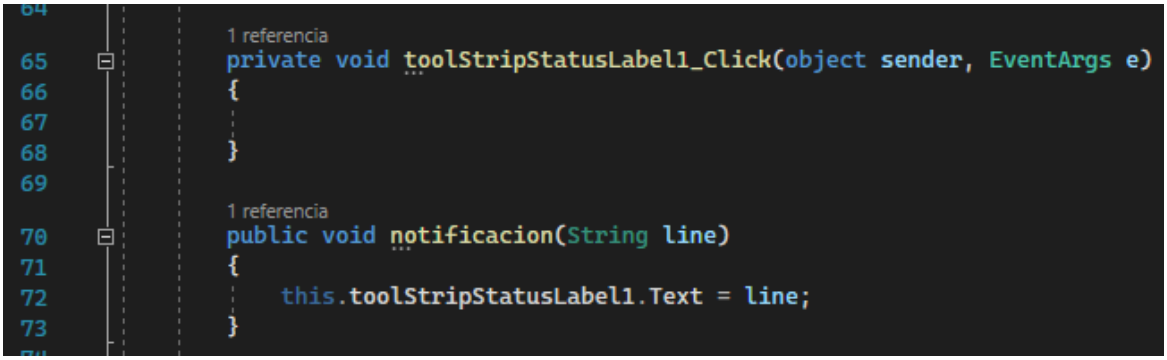
```
1 referencia
private void Form1_Load(object sender, EventArgs e)
{
    loadColor();
    loadFuente();
    Documento = "";
    Direccion = "";
}
```

Hay funciones que ejecutan una sola línea de comandos:



```
2 referencias
public void fuente(String name, float size, FontStyle
{
    richTextBox1.Font = new Font(name,size, Style);
}
```

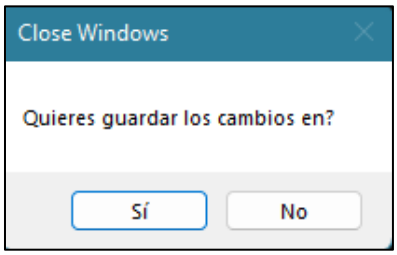
Y hay funciones que no se utilizan, ya sea porque al final quedaron vacías como prototipo, porque fueron reemplazadas por una versión mejorada o adaptada de ellas mismas, o porque simplemente no se llaman en la ejecución real (pero las dejamos ahí con motivos didácticos o para un potencial futuro/hipotético uso):



```
1 referencia
private void toolStripStatusLabel1_Click(object sender, EventArgs e)
{

}

1 referencia
public void notificacion(String line)
{
    this.toolStripStatusLabel1.Text = line;
}
```

Por lo tanto, simplemente mencionaremos rápidamente las funciones más especializadas del programa

`salirToolStripMenuItem_Click()` Verifica si al intentar detener la aplicación, el documento y sus datos han sido guardados (tal como haría cualquier otro editor de texto).

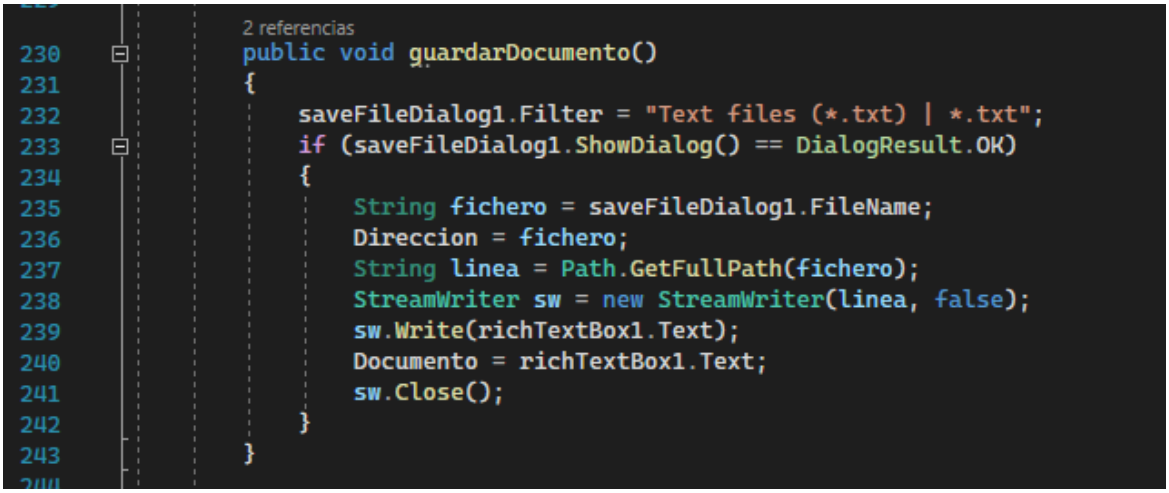



```
1 referencia
private void salirToolStripMenuItem_Click(object sender, EventArgs e)
{
    String cadena = richTextBox1.Text;
    if (Documento == cadena)
    {
        Application.Exit();
    }
    else
    {
        String message = "Quieres guardar los cambios en?";
        String titulo = "Close Windows";
        MessageBoxButtons btn = MessageBoxButtons.YesNo;
        DialogResult result = MessageBox.Show(message, titulo, btn);
        if (result == DialogResult.Yes)
        {
            guardarSimple();
            Application.Exit();
        }
        else
        {
            Application.Exit();
        }
    }

}
```

`nuevoToolStripMenuItem_Click()` limpia los datos del documento sobre el que estamos trabajando. (Hay que tener cuidado porque este no pregunta por confirmación, simplemente lo hace). Creando así, un nuevo documento:

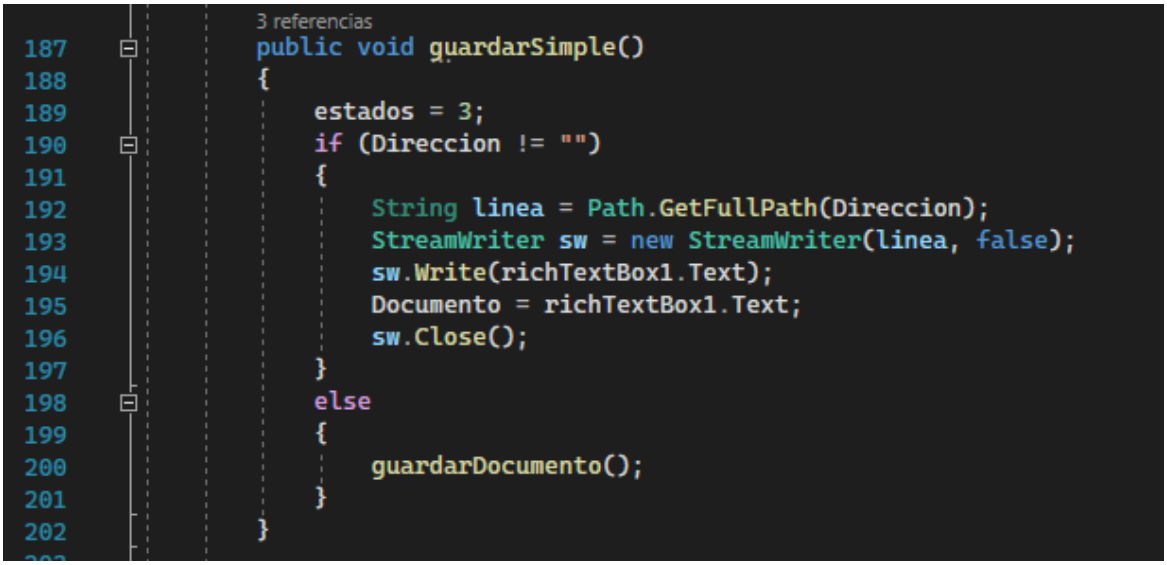
```
1 referencia
private void nuevoToolStripMenuItem_Click(object sender, EventArgs e)
{
    estados = 1;
    Documento = "";
    Direccion = "";
    richTextBox1.Text = "";
}
```

`guardarDocumento()` Corresponde a guardar el texto que se está trabajando, esta función puede ser llamada directamente desde la opción “Guardar” del menú de “Archivo” en la ventana principal, o mediante la función `guardarSimple()` en un caso específico de ese algoritmo. Lo primero que hace es verificar con un filtro, que el archivo que se va a guardar sea de tipo txt, luego usa un saveFileDialog para realizar la operación.



```
2 referencias
230  public void guardarDocumento()
231  {
232      saveFileDialog1.Filter = "Text files (*.txt) | *.txt";
233      if (saveFileDialog1.ShowDialog() == DialogResult.OK)
234      {
235          String fichero = saveFileDialog1.FileName;
236          Direccion = fichero;
237          String linea = Path.GetFullPath(fichero);
238          StreamWriter sw = new StreamWriter(linea, false);
239          sw.Write(richTextBox1.Text);
240          Documento = richTextBox1.Text;
241          sw.Close();
242      }
243  }
```

`guardarSimple()` Corresponde a la función asociada al apartado “Guardar” del menú de “Archivo”, que como ya mencionamos, guarda los cambios realizados sobre el archivo específico que se está trabajando en el momento de la ejecución. Si el archivo es nuevo (`if (Direccion != "")`) primero se define un nombre para éste



```
3 referencias
187  public void guardarSimple()
188  {
189      estados = 3;
190      if (Direccion != "")
191      {
192          String linea = Path.GetFullPath(Direccion);
193          StreamWriter sw = new StreamWriter(linea, false);
194          sw.Write(richTextBox1.Text);
195          Documento = richTextBox1.Text;
196          sw.Close();
197      }
198      else
199      {
200          guardarDocumento();
201      }
202  }
```

`abrirToolStripMenuItem_Click()` Usa un OpenFileDialog para leer un archivo de texto previamente guardado. Esta función corresponde a la opción “Abrir” del menú “Archivo” de la ventana principal:

```
203
        1 referencia
204     private void abrirToolStripMenuItem_Click(object sender, EventArgs e)
205     {
206         estados = 2;
207         openFileDialog1.FileName = "Seleccione el archivo";
208         openFileDialog1.Filter = "Text files (*.txt) | *.txt";
209         openFileDialog1.Title = "Abrir archivo";
210 
211         if (openFileDialog1.ShowDialog() == DialogResult.OK)
212         {
213             String url = openFileDialog1.FileName;
214             Direccion = url;
215             StreamReader sr = new StreamReader(url);
216             String texto = sr.ReadToEnd();
217             sr.Close();
218             richTextBox1.Text = texto;
219             Documento = texto;
220         }
221     }
222 
```