



Benemérita Universidad Autónoma de Puebla

Sistemas Operativos II

Christian Amauri Amador Ortega

Editor de texto en lenguaje C#

Contenido

<i>Consideraciones técnicas</i>	<i>1</i>
<i>Objetivo</i>	<i>1</i>
<i>Ejemplo del funcionamiento del proyecto</i>	<i>2</i>
<i>Librerías usadas</i>	<i>4</i>
<i>Diseños</i>	<i>5</i>
<i>Scripts</i>	<i>7</i>
<i>Conclusión</i>	<i>15</i>
<i>Referencias</i>	<i>16</i>

Consideraciones técnicas

Para esta serie de prácticas en lenguaje C#, usaremos el IDE: visual studio (la versión más reciente en septiembre de 2022 [2]). Puede haber variaciones en el funcionamiento del proyecto en otra versión de visual studio u otro IDE. Dependiendo de qué tan alejada sea la versión del IDE y del lenguaje con el que se consulte esta documentación, estas variaciones serán de mayor o menor impacto.

Documentaremos el funcionamiento de las aplicaciones (en su ejecución) y haremos algunos comentarios / notas / consideraciones / advertencias al respecto. Para ahorrar tiempo y esfuerzo, no ahondaremos 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, por lo menos lo básico: POO, programación funcional, programación estructurada).

Finalmente hay que tomar en cuenta que C# y Visual studio son herramientas pesadas. Y que el equipo que vayamos a utilizar debe tener cierta capacidad de rendimiento superior a sólo la básica. En particular, para el desarrollo de esta versión específica, se usó una laptop Lenovo Ideapad S145, con un Intel Core i3-8145U, 4 GB de RAM, y 1TB de almacenamiento en un disco duro mecánico. (Y con esas especificaciones técnicas mínimas, el rendimiento del desarrollo no fue cómodo).

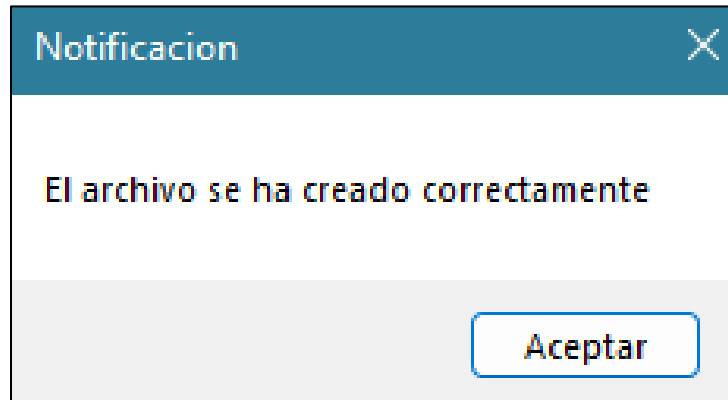
Objetivo

El objetivo es crear con C# un programa editor de texto con opciones para modificar el color, tamaño, tipo y alineado de los textos. Además de la opción de insertar imágenes, y guardar o abrir archivos con un formato definido por nosotros. La ventaja que tenemos en este proyecto es que va a estar construido principalmente con funciones predefinidas por el lenguaje, así que eso nos simplificará mucho el trabajo. Debemos usar y conocer las herramientas de las librerías que ya fueron hechas en algún momento en el pasado.

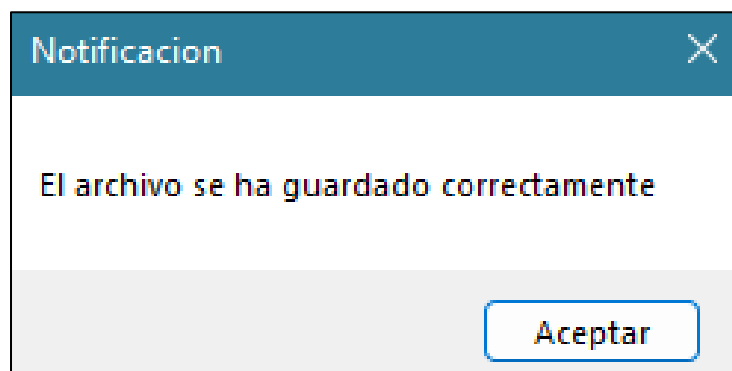
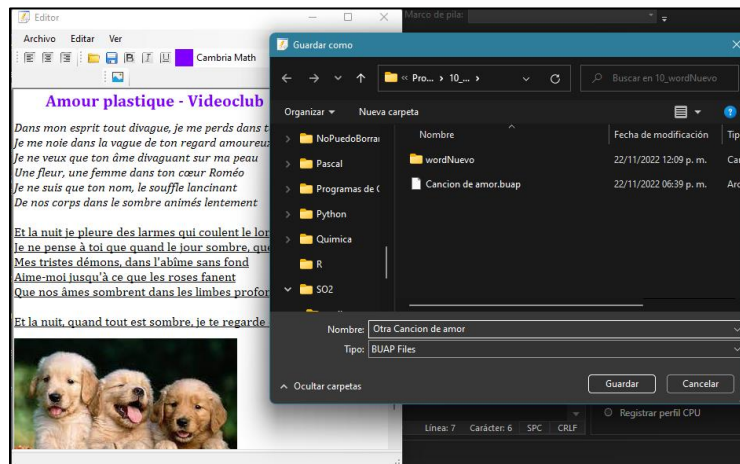
El repositorio en el que este código se encuentra (y otros códigos en C# del curso de Sistemas operativos II, BUAP FCC, otoño 2022) está disponible (y estará disponible al menos durante el resto del año 2022) haciendo click en el siguiente enlace de Google Drive: [1] (aunque no garantizamos su permanencia después de la fecha indicada).

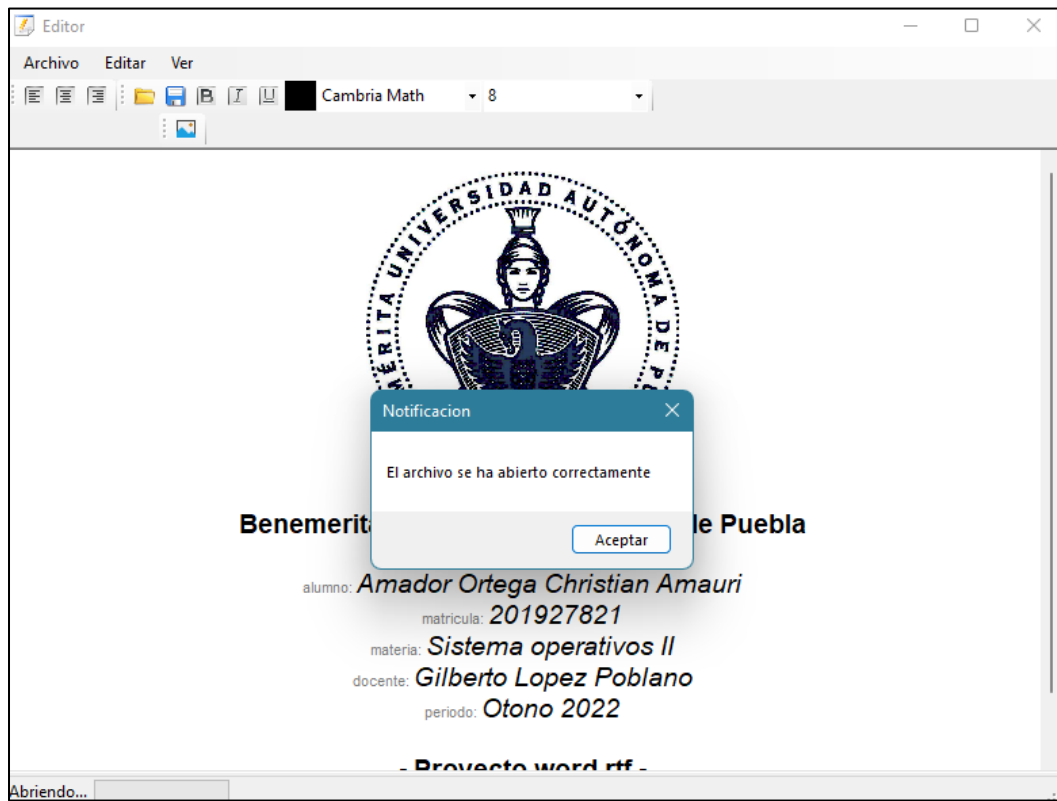
Ejemplo del funcionamiento del proyecto

- Podemos crear documentos con nuestro propio tipo de formato (.BUAP files)



- Tenemos la opción de guardar documentos, y de sobrescribirlos (extensión .BUAP y .txt).





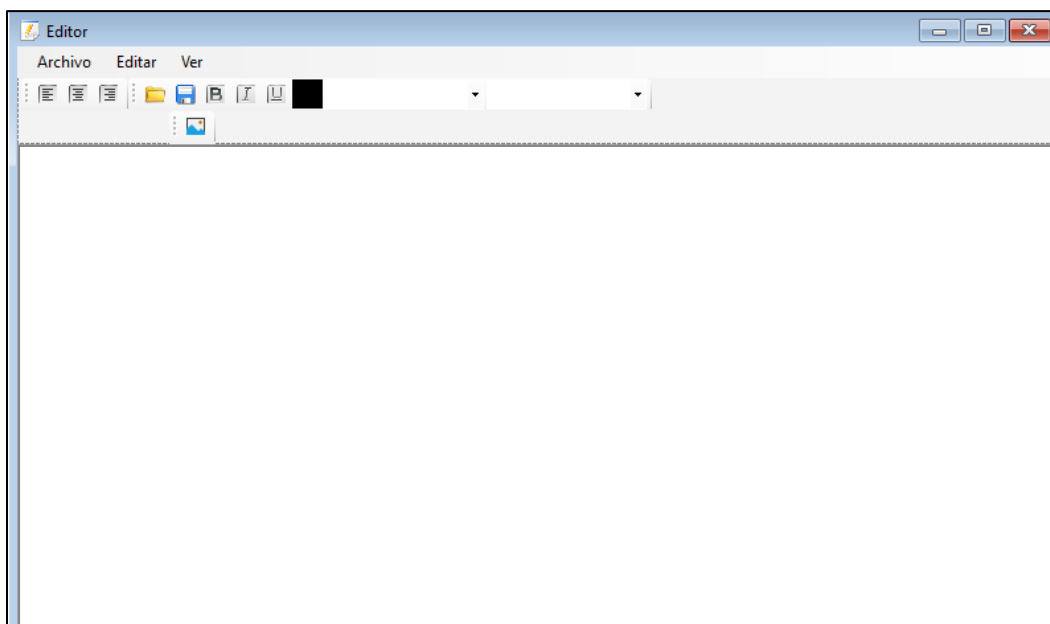
Librerías usadas

Para empezar, daremos una breve descripción de lo que hacen las librerías empleadas en ambos scripts del proyecto (`Form1.cs` y `Form2.cs`) (Los `designer.cs` no usan librerías)

- ❖ `System`; Proporciona clases y funciones fundamentales del lenguaje, incluyendo tipos de datos básicos, excepciones y funciones de conversión.
- ❖ `System.Collections.Generic`; Ofrece colecciones genéricas, como listas, diccionarios y conjuntos, que permiten almacenar y manipular grupos de objetos de forma eficiente y segura.
- ❖ `System.ComponentModel`; Contiene clases que permiten implementar el comportamiento de componentes y controles, incluyendo la gestión de eventos y la serialización.
- ❖ `System.Data`; Proporciona clases para la gestión de datos en aplicaciones, incluyendo estructuras para trabajar con bases de datos, como `DataTable` y `DataSet`.
- ❖ `System.Drawing`; Incluye clases para trabajar con gráficos y manipular imágenes, permitiendo la creación y el dibujo de elementos gráficos en aplicaciones.
- ❖ `System.Linq`; Proporciona funcionalidades para consultas integradas en C#, permitiendo trabajar con colecciones de datos utilizando sintaxis declarativa.
- ❖ `System.Text`; Contiene clases para la manipulación de cadenas de texto y la codificación, incluyendo la clase `StringBuilder` para el manejo eficiente de cadenas mutables.
- ❖ `System.Threading.Tasks`; Ofrece clases para la programación asíncrona y paralela, facilitando la ejecución de tareas en segundo plano y la gestión de la concurrencia.
- ❖ `System.Windows.Forms`; Proporciona clases para crear aplicaciones de escritorio con interfaz gráfica en Windows, incluyendo controles de usuario, formularios y manejo de eventos.
- ❖ `System.Drawing.Text`; Proporciona clases para trabajar con tipos de fuentes y la gestión de texto en gráficos, permitiendo la manipulación de fuentes instaladas y personalizadas en aplicaciones.
- ❖ `System.Text.RegularExpressions`; Ofrece clases para trabajar con expresiones regulares, permitiendo la búsqueda, coincidencia y manipulación de cadenas de texto basadas en patrones definidos.

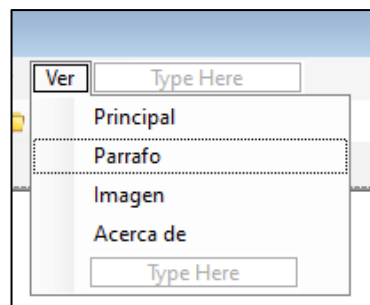
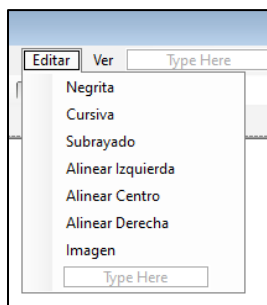
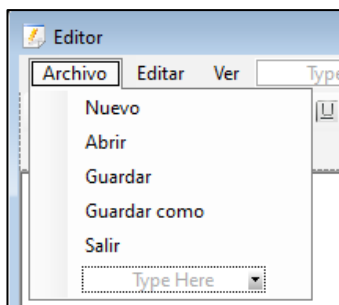
Ahora, los archivos .cs (C#) de este proyecto corresponden a 2 diseños y 2 scripts. Comencemos por los diseños...

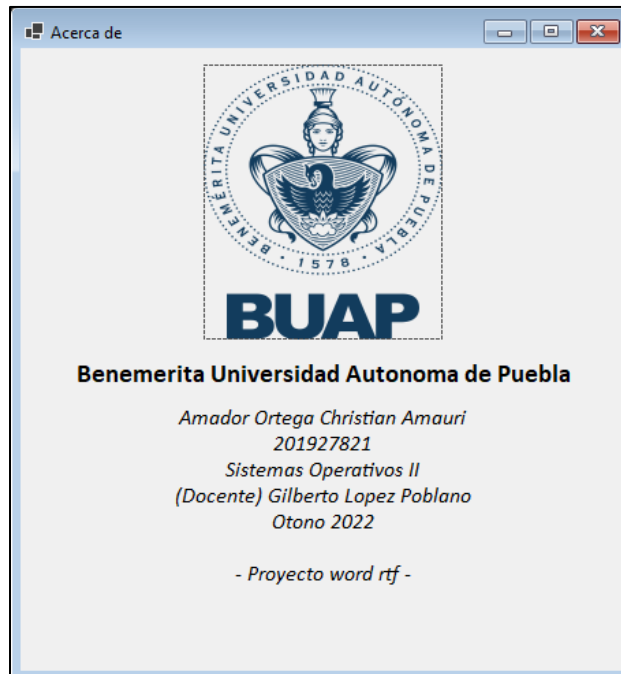
Diseños



#1 **Form1** (Form1.Designer.cs) (Form main)

Contiene los botones y las opciones con las que editaremos nuestros textos, tales como; Alinear a la izquierda, centro o derecha. Abrir archivo, Guardar archivo. establecer letra como: negrita, cursiva, subrayada. Cambiar el color y tamaño de la fuente. E insertar imágenes. Todas ellas corresponden a las funciones básicas de todo editor de textos / documentos.





#2 **Form2** (*Form2.Designer.cs*) (*Acerca de*)

Contiene información sobre el autor del código (no posee funcionalidad extra). Es invocada por la opción “Acerca de” en la opción “Ver” del menú strip de la ventana principal.

Scripts

#1 *Form1.cs*

La estructura inicial de este script consta de:

- 2 librerías
- namespace definido como: `wordNuevo`
- `public partial class Form1 : Form`

```
1 using System.Drawing.Text;
2 using System.Text.RegularExpressions;
3
4 namespace wordNuevo
5 {
6     4 referencias
7     public partial class Form1 : Form
8     {
```

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

```
4 referencias
public partial class Form1 : Form
{
    String archivo;
    String notificacion;
    Form2 acerca;

    1 referencia
    public Form1() { ... }

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

    1 referencia
    private void principalToolStripMenuItem_Click(object sender, EventArgs e)

    1 referencia
    private void parrafoToolStripMenuItem_Click(object sender, EventArgs e)

    1 referencia
    private void ...
```

(Todos los métodos de Form1 : Form, comprimidos para poder apreciarlos) (son 44)

- 3 variables (tipo `String` y `Form2`)
- un constructor
- 43 métodos

Comencemos por las opciones de la opción “Archivo” del menu strip de la ventana principal:

El método `nuevoToolStripMenuItem_Click()` corresponde a la opción “Nuevo”

```
1 referencia
284 private void nuevoToolStripMenuItem_Click(object sender, EventArgs e)
285 {
286     richTextBox1.Clear();
287     archivo = "";
288     toolStripStatusLabel1.Text = "Creando nuevo archivo...";
289     notificacion = "El archivo se ha creado correctamente";
290     toolStripProgressBar1.Visible = true;
291     timer1.Start();
292 }
293
```

El método `abrirToolStripMenuItem_Click()` corresponde a la opción “abrir”. Manda a llamar al método `abrir()`, en donde se usa un `OpenFileDialog` para leer archivos con filtro .BUAP files:

```
2 referencias
132 public void abrir()
133 {
134     OpenFileDialog aArch = new OpenFileDialog();
135     aArch.DefaultExt = "*.buap";
136     aArch.Filter = "BUAP Files|*.buap";
137
138     if (aArch.ShowDialog() == DialogResult.OK)
139     {
140         archivo = aArch.FileName;
141         richTextBox1.LoadFile(aArch.FileName);
142         toolStripStatusLabel1.Text = "Abriendo...";
143         notificacion = "El archivo se ha abierto correctamente";
144         toolStripProgressBar1.Visible = true;
145         timer1.Start();
146     }
147 }
148
```

El método `guardarToolStripMenuItem_Click()` corresponde a la opción “Guardar”. está hecho para sobrescribir el archivo en el que se está trabajando, pero si se está trabajando sobre un documento nuevo, se manda a llamar al método `guardar()`.

```
1 referencia
170 private void guardarToolStripMenuItem_Click(object sender, EventArgs e)
171 {
172     if (archivo == "")
173     {
174         guardar();
175     }
176     else
177     {
178         richTextBox1.SaveFile(archivo, RichTextBoxStreamType.RichText);
179         toolStripStatusLabel1.Text = "Guardando...";
180         notificacion = "El archivo se ha guardado correctamente";
181         toolStripProgressBar1.Visible = true;
182         timer1.Start();
183     }
184 }
185
```

El método `guardar()` funciona como cada método de guardado de archivos que hemos trabajado antes, usa un filtro de formato de fichero para verificar que se guarden adecuadamente y mediante un `SaveFileDialog()` respaldado por un pequeño algoritmo de confirmación, realiza el proceso correspondiente:

```
113 3 referencias
114 public void guardar()
115 {
116     SaveFileDialog gArch = new SaveFileDialog();
117     gArch.DefaultExt = "*.buap";
118     gArch.Filter = "BUAP Files|*.buap";
119
120     if (gArch.ShowDialog() == DialogResult.OK)
121     {
122         archivo = gArch.FileName;
123         richTextBox1.SaveFile(gArch.FileName, RichTextBoxStreamType.RichText);
124         toolStripStatusLabel1.Text = "Guardando...";
125         notificacion = "El archivo se ha guardado correctamente";
126         toolStripProgressBar1.Visible = true;
127         timer1.Start();
128     }
129 }
130 }
```

El método `guardarComoToolStripMenuItem_Click()` corresponde a la opción “Guardar como”. Manda a llamar al método `guardar()` :

```
155 1 referencia
156 private void guardarComoToolStripMenuItem_Click(object sender, EventArgs e)
157 {
158     guardar();
}
```

El método `salirToolStripMenuItem_Click()` corresponde a la opción “Salir”. Este termina la ejecución de la aplicación:

```
165 1 referencia
166 private void salirToolStripMenuItem_Click(object sender, EventArgs e)
167 {
168     Application.Exit();
}
```

Sigamos con las opciones de la opción “Editar” del menú strip de la ventana principal. Estas son relativamente sencillas y parecidas, en algunos casos cada una manda a llamar a un sub-método que cumple su propósito (prácticamente como paso extra), de modo que por cada opción en este menú, tenemos dos métodos (el que llama al método real, y el que es llamado en primer lugar):

El método `boldToolStripMenuItem_Click()` manda a llamar al método `negrita()`, en donde se realiza el proceso real, el cual consiste simplemente en usar funciones predefinidas del lenguaje para darle a la fuente el efecto que deseamos:

```
342 1 referencia
343 private void boldToolStripMenuItem_Click(object sender, EventArgs e)
344 {
345     negrita();
}

218 2 referencias
219 public void negrita()
220 {
221     if (richTextBox1.SelectionFont != null)
222     {
223         if (richTextBox1.SelectionFont.Bold == true)
224         {
225             normal();
226         }
227         else
228         {
229             richTextBox1.SelectionFont = new Font(richTextBox1.SelectionFont.FontName,
230             richTextBox1.SelectionFont.Size, FontStyle.Bold);
231         }
232     }
233     richTextBox1.Focus();
}
```

Como dijimos, este concepto se repite para las opciones de cursiva, subrayada y agregar imagen:

```
347 1 referencia
348 private void italicToolStripMenuItem_Click(object sender, EventArgs e)
349 {
350     cursiva();
351 }

352 1 referencia
353 private void subrayadoToolStripMenuItem_Click(object sender, EventArgs e)
354 {
355     subrayado();
356 }

213 1 referencia
214 private void imagenToolStripMenuItem1_Click(object sender, EventArgs e)
215 {
216     agregarImagen();
217 }

192 2 referencias
193 public void agregarImagen()
194 {
195     OpenFileDialog iArch = new OpenFileDialog();
196     iArch.DefaultExt = "*.jpg";
197     iArch.Filter = "Image Files|*.jpg";
198
199     if (iArch.ShowDialog() == DialogResult.OK)
200     {
201         Bitmap imagen = new Bitmap(iArch.FileName);
202         Clipboard.SetDataObject(imagen);
203         DataFormats.Format formato = DataFormats.GetFormat(DataFormats.Bitmap);
204         richTextBox1.Paste(formato);
205         toolStripStatusLabel1.Text = "Cargando...";
206         notificacion = "La imagen se ha cargado correctamente";
207         toolStripProgressBar1.Visible = true;
208         timer1.Start();
209     }
210 }
211 }
```

Para las opciones de alinear a la derecha, centro o izquierda; se ejecuta el cambio en una simple línea de código, en un solo cambio en el atributo mostrado a continuación:

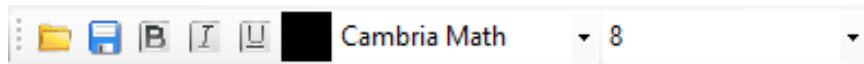
```
357 1 referencia
358 private void alinearIzqToolStripMenuItem_Click(object sender, EventArgs e)
359 {
360     richTextBox1.SelectionAlignment = HorizontalAlignment.Left;
361 }

362 1 referencia
363 private void alinearCentroToolStripMenuItem_Click(object sender, EventArgs e)
364 {
365     richTextBox1.SelectionAlignment = HorizontalAlignment.Center;
366 }

367 1 referencia
368 private void alinearDerechaToolStripMenuItem_Click(object sender, EventArgs e)
369 {
370     richTextBox1.SelectionAlignment = HorizontalAlignment.Right;
371 }
372 }
```

Finalmente, revisemos las opciones de la opción “Ver”; Las primeras tres opciones de esta opción corresponden a los métodos que establecen como visibles o NO visibles los tres conjuntos de íconos mostrados en la ventana principal. Esta acción se ejecuta en una sola línea, negando una vez el estado actual de visibilidad del conjunto de íconos:

- “Principal” corresponde a los íconos de color, tamaño y tipo de fuente:



- “Párrafo” corresponde a los íconos de alineado del texto:



- “Imagen” corresponde al ícono de imagen:



Métodos respectivos:

```

52  private void principalToolStripMenuItem_Click(object sender, EventArgs e)
53  {
54      toolStrip1.Visible = !toolStrip1.Visible;
55  }
56
57  1 referencia
58  private void parrafoToolStripMenuItem_Click(object sender, EventArgs e)
59  {
60      toolStrip2.Visible = !toolStrip2.Visible;
61  }
62
63  1 referencia
64  private void imagenToolStripMenuItem_Click(object sender, EventArgs e)
65  {
66      toolStrip3.Visible = !toolStrip3.Visible;
67  }

```

La opción `acercaDeToolStripMenuItem_Click_1()` “Acerca de” manda a llamar al método el cual crea un objeto de tipo `Form2`, el cual existe única y exclusivamente con el propósito de cubrir esta opción (mostrar información):

```

413  1 referencia
414  private void acercaDeToolStripMenuItem_Click_1(object sender, EventArgs e)
415  {
416      Form2 acerca = new Form2();
417      acerca.Show();
418  }
419

```

Ahora, para cada método de cada ícono en la ventana main, simplemente se manda a llamar el método correspondiente (otra vez), de ahí que el código parezca tan largo. En realidad, no son tantos métodos como parecen, simplemente hay muchas formas de mandarlos a llamar...

```
1 referencia
329 private void toolStripButton5_Click(object sender, EventArgs e)
330 {
331     subrayado();
332 }
```

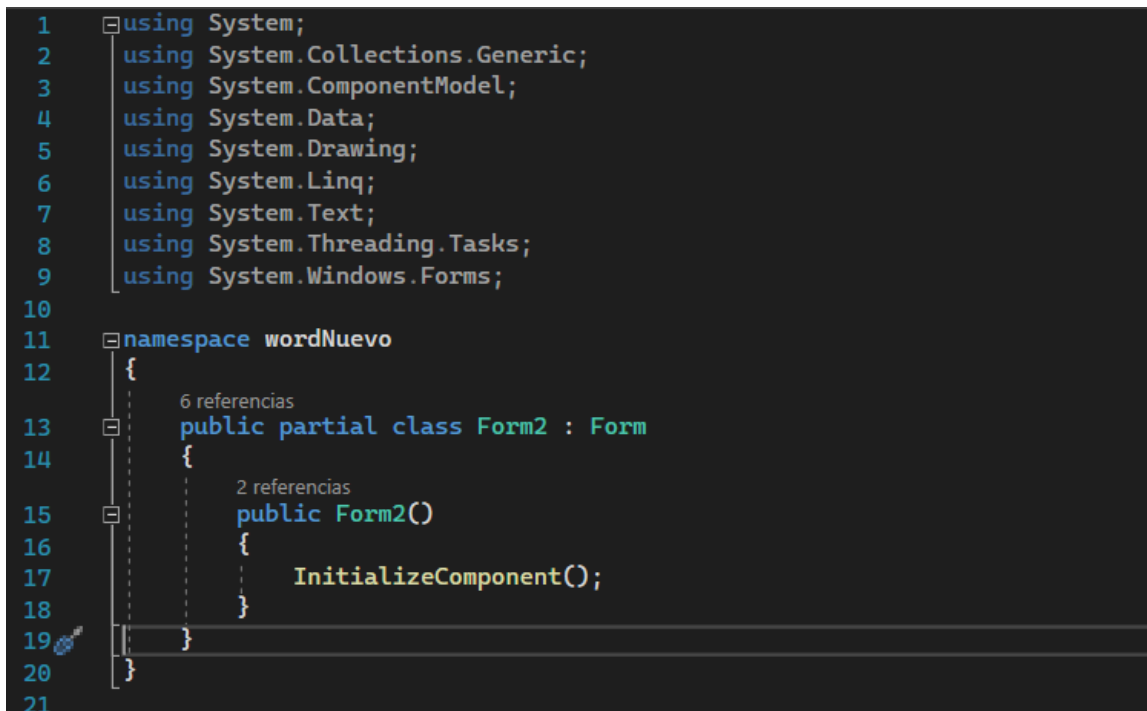
El método `Form1_Load()` es un método que se manda a llamar en el script `Form1.Designer.cs`. Este sirve para agregar a la ventana principal los Items y características que esta usa (no hay mucho más en lo que ahondar, esto es más técnico realmente):

```
21 1 referencia
22 private void Form1_Load(object sender, EventArgs e)
23 {
24     toolStripComboBox2.Items.Add(8);
25     toolStripComboBox2.Items.Add(9);
26     toolStripComboBox2.Items.Add(10);
27     toolStripComboBox2.Items.Add(11);
28     toolStripComboBox2.Items.Add(12);
29     toolStripComboBox2.Items.Add(14);
30     toolStripComboBox2.Items.Add(16);
31     toolStripComboBox2.Items.Add(18);
32     toolStripComboBox2.Items.Add(20);
33     toolStripComboBox2.Items.Add(22);
34     toolStripComboBox2.Items.Add(24);
35     toolStripComboBox2.Items.Add(26);
36     toolStripComboBox2.Items.Add(28);
37     toolStripComboBox2.Items.Add(36);
38     toolStripComboBox2.Items.Add(48);
39     toolStripComboBox2.Items.Add(72);
40
41     InstalledFontCollection ifc = new InstalledFontCollection();
42     FontFamily[] familia = ifc.Families;
43     for (int i = 0; i < ifc.Families.Length; i++)
44     {
45         toolStripComboBox1.Items.Add(ifc.Families[i].Name);
46     }
47
48     toolStripComboBox1.SelectedIndex = 41;
49     toolStripComboBox2.SelectedIndex = 0;
50 }
51
```

#2 *Form2.cs*

La estructura inicial de este script consta de:

- 9 librerías
- `namespace` definido como: `wordNuevo`
- `public partial class Form2 : 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
11 namespace wordNuevo
12 {
13     public partial class Form2 : Form
14     {
15         public Form2()
16         {
17             InitializeComponent();
18         }
19     }
20 }
21
```

Dentro de `Form2 : Form` simplemente se encuentra el constructor de la clase `Form2` (ventana “Acerca de”) la cual es llamada cuando se presiona la opción “Acerca de” en la opción “Ver” del menú strip de la ventana principal.

Conclusión



Como conclusión y comentario final; Este proyecto es más didáctico que funcional. El resultado sigue estando por debajo de muchos editores de texto o documentos. Con este producto podríamos hacer notas personales simples o algo por el estilo. Aunque si quisiéramos, podríamos tomar esta misma versión y trabajar sobre ella para añadirle tantos parches y modificaciones como queramos, para darle el nivel, calidad y compatibilidad deseadas. Y también debemos recordar que, al estar construido enteramente con funciones definidas en el lenguaje, hacer una documentación detallada de cada parte del código se convertiría más bien en una guía del lenguaje o de las librerías. Esta documentación se reserva el derecho de mostrar únicamente detalles exteriores sobre la estructura de nuestro producto.

Referencias

[1] Amador, C. (2022). [Repositorio en Google Drive del proyecto, y de otros proyectos del curso de sistemas operativos II, BUAP FCC, PERIODO OTOÑO 2022], disponible en: <https://drive.google.com/drive/folders/1yZujMI51XAnEZBMj-4ykUWMfHxBC9uYp?usp=sharing>

[2] Visual Studio 2022 Community Edition: descargar la versión gratuita más reciente. (2016, Septiembre 13). Recuperado de: <https://visualstudio.microsoft.com/es/vs/community/>

[3] dotnet-bot. (s.f.). .NET API browser. Microsoft.com. (librerías de C#) Recuperado de: <https://learn.microsoft.com/en-us/dotnet/api/>