Instituto tecnológico de Culiacán

Nombre:

Carrasco Medina Carlos Ivan

Proyecto:

Compilador

Materia:

Lenguajes y autómatas 2

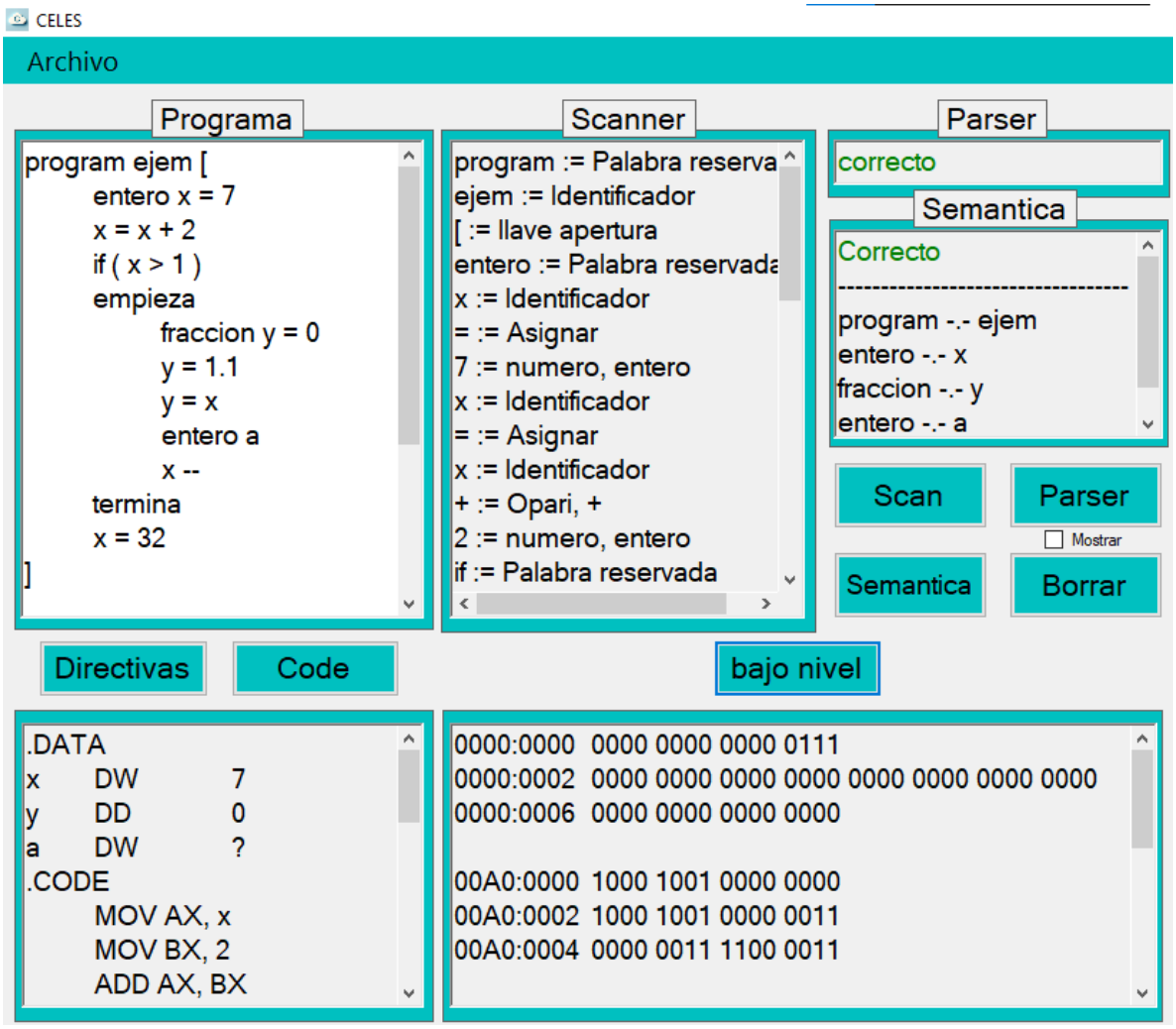
Maestro:

Rosalío Zatarain Cabada

Fecha:

19/11/2024

Ejemplo de ejecución:



Lenguaje:

CELES-> program IDENTIFICADOR [FORMAS]

FORMAS-> FORMA+

FORMA-> COMENTARIO | ENTRADA | SALIDA | CONDICIONAL | DECLARAR | ASIGNAR

COMENTARIO-> ## PALABRAS

ENTRADA -> VARIABLE . leer

SALIDA -> escribir (VARIABLE) | escribir (PALABRAS)

CONDICIONAL -> if empieza (CONDICION) FORMAS termina | ELSE empieza FORMAS termina

CONDICION -> VARIABLE COMPARADOR VARIABLE | VARIABLE COMPARADOR NUMERO | VARIABLE COMPARADOR PALABRA |

COMPARADOR -> == | < | > | <= | >= | < >

DECLARAR -> TIPO VARIABLE = NUMERO | TIPO VARIABLE = FRACCION | TIPO VARIABLE

TIPO -> ENTERO | FRACCION

VARIABLE -> PALABRA NUMERO*

PALABRAS -> PALABRAS+ | VARIABLE | PALABRA | NUMERO | FRACCION

PALABRA -> LETRA+

LETRA -> A|B|C...|Z | a | b | c |...z

FRACCION -> NUMERO . NUMERO

NUMERO -> DIJITO+

DIJITO -> 0 | 1 | 2 | ...|9

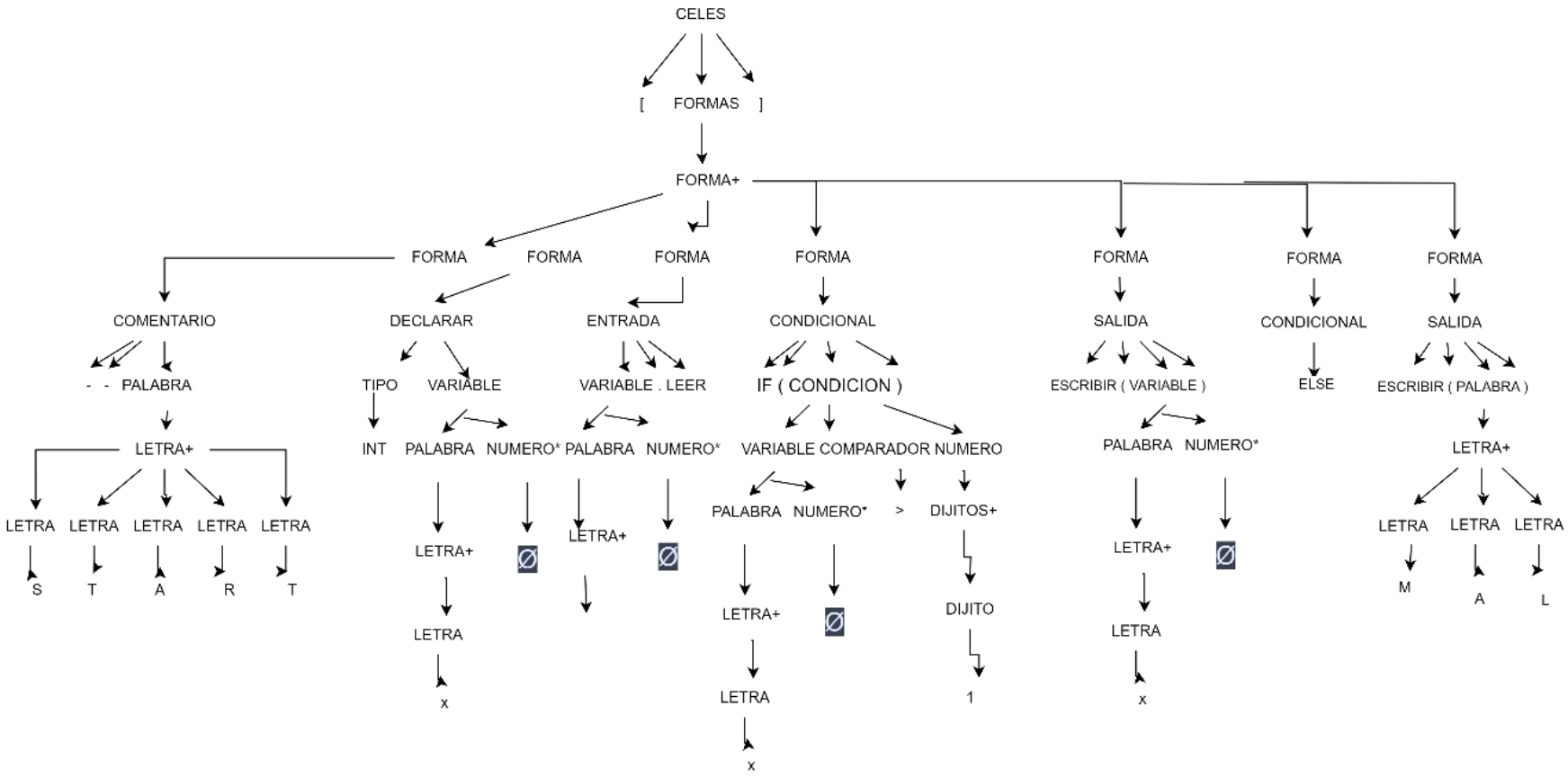
ASIGNAR -> VARIABLE ASIGNATIVO { * OPERADOR {VARIABLE | ENTERO | DECIMAL } | VARIABLE INCREMENTAR

ASIGNATIVO -> = | += | -= | *= | /=

OPERADOR -> + | - | * | / | % |

INCREMENTAR -> ++ | --

Árbol sintáctico:



Código (se hizo en solo un documento por temas de presentación y limitaciones con mi equipo en ese momento):

```
namespace opeScanner
{
   public partial class rtbLectura : Form
   {
      //variables estaticas
      static char[] separadores = { ' ', '\n', '\r', '\t' };
      static string NombreArchivo = "";
      static string[,] tabla;
      static int conteo;
      static bool errorScan, parserMostrar = true;

      public rtbLectura()
      {
         InitializeComponent();
      }

      //metodo principal que se activa al dar click al boton scan
      private void btnScan_Click(object sender, EventArgs e)
      {
         //lectura del contenido del RichTextBox
         string lectura = rtbLeer.Text;
         errorScan = false;

         //separa los tokens con split y quita los vacios
         string[] tokenes = lectura.Split(separadores, StringSplitOptions.RemoveEmptyEntries);

         tabla = new string[6, tokenes.Length];
```

```
        //reinicio de variable
        rtbDevolver.Text = "";
        conteo = 0;//variable para manejar el conteo de los tokens


        //imprime cada token con su identificador
        if (!parserMostrar)
        {
          foreach (var token in tokenes)
          {
            comprobador(token, conteo);//manda a buscar su identificador
            tabla[1, conteo] = token;
            rtbDevolver.Text += token + " := " + tabla[2, conteo] + "," + tabla[3, conteo] + "\n";//lo
coloca en el RichTextBox
            conteo++;
          }
        }
        else
        {
          foreach (var token in tokenes)
          {
            comprobador(token, conteo);//manda a buscar su identificador
            tabla[1, conteo] = token;
            rtbDevolver.Text += token + " := " + tabla[2, conteo] + "\n";//lo coloca en el
RichTextBox
            conteo++;
          }
        }
      }
```

```
public void comprobador(string lectura, int n)
{
   //comprueba que lectura tenga al menos un simbolo
   if (1 == lectura.Length || 2 == lectura.Length)
   {
      //Dependiendo que dato tenga lectura
      switch (lectura.Substring(0, 1))
      {
         case "<":
            //comprueba la cantidad de simbolos
            if (lectura.Length >= 2)
            {
               switch (lectura.Substring(1, 1))
               {
                  case "=":
                     tabla[2, n] = "Oprel, Mayor igual";
                     tabla[3, n] = "34";
                     break;
                  case ">":
                     tabla[2, n] = "Oprel, Diferente";
                     tabla[3, n] = "32";
                     break;
                  default:
                     tabla[2, n] = "ERROR SE ESPERA '<', '<=' o '<>', no: " + lectura.Substring(1, 1);
                     errorScan = true;
                     break;
               }
            }
            else
```

```
    {
      tabla[2, n] = "OPREL, Mayor que";
      tabla[3, n] = "33";
    }
    break;
  case "=":
    //comprueba la cantidad de simbolos
    if (lectura.Length >= 2)
    {
      if (lectura.Substring(1, 1) == "=")
      {
        tabla[2, n] = "Oprel, Igual que";
        tabla[3, n] = "31";
      }
      else
      {
        tabla[2, n] = "ERROR, se espera '=' o '==', no: " + lectura.Substring(1, 1);
        errorScan = true;
      }
    }
    else
    {
      tabla[2, n] = "Asignar";
      tabla[3, n] = "25";
    }
    break;
  case ">":
    //comprueba la cantidad de simbolos
    if (lectura.Length >= 2)
```

```
    {
      switch (lectura.Substring(1, 1))
      {
        case "=":
          tabla[2, n] = "Oprel, Mayor que";
          tabla[3, n] = "36";
          break;
        default:
          tabla[2, n] = "ERROR, se espera '>' o '>=', no: " + lectura.Substring(1, 1);
          errorScan = true;
          break;
      }
    }
    else
    {
      tabla[2, n] = "Oprel, Menor que";
      tabla[3, n] = "35";
    }
    break;
  case "+":
    //comprueba la cantidad de simbolos
    if (lectura.Length >= 2)
    {
      switch (lectura.Substring(1, 1))
      {
        case "+":
          tabla[2, n] = "Opari, ++";
          tabla[3, n] = "19";
          break;
```

```
            case "=":
              tabla[2, n] = "Opari, +=";
              tabla[3, n] = "21";
              break;
            default:
              tabla[2, n] = "ERROR, se espera '+' , '++' o '+=', no: " + lectura.Substring(1, 1);
              errorScan = true;
              break;
          }
        }
        else
        {
          tabla[2, n] = "Opari, +";
          tabla[3, n] = "15";
        }
        break;
      case "-":
        //comprueba la cantidad de simbolos
        if (lectura.Length >= 2)
        {
          switch (lectura.Substring(1, 1))
          {
            case "-":
              tabla[2, n] = "Opari, --";
              tabla[3, n] = "20";
              break;
            case "=":
              tabla[2, n] = "Opari, -=";
              tabla[3, n] = "22";
```

```
            break;
          default:
            tabla[2, n] = "ERROR, se espera '-' , '--' o '-=', no: " + lectura.Substring(1, 1);
            errorScan = true;
            break;
        }
      }
      else
      {
        tabla[2, n] = "Opari, -";
        tabla[3, n] = "16";
      }
      break;
    case "/":
      //comprueba la cantidad de simbolos
      if (lectura.Length >= 2)
      {
        if (lectura.Substring(1, 1) == "=")
        {
          tabla[2, n] = "Oparl, /=";
          tabla[3, n] = "24";
        }
        else
        {
          tabla[2, n] = "Error, se espera '/' o '/=' ,no: " + lectura.Substring(1, 1);
          errorScan = true;
        }
      }
      else
```

```
    {
        tabla[2, n] = "Oparl, /";
        tabla[3, n] = "18";
    }
    break;
case "*":
    //comprueba la cantidad de simbolos
    if (lectura.Length == 2)
    {
        if (lectura.Substring(1, 1) == "=")
        {
            tabla[2, n] = "Oparl, *=";
            tabla[3, n] = "23";
        }
        else
        {
            tabla[2, n] = "Error, se espera '*' o '*=' ,no: " + lectura.Substring(1, 1);
            errorScan = true;
        }
    }
    else
    {
        tabla[2, n] = "Oparl, *";
        tabla[3, n] = "17";
    }
    break;
case "%":
    //comprueba la cantidad de simbolos
    if (lectura.Length >= 2)
```

```
    {
        tabla[2, n] = "Error, caracter inesperado: " + lectura.Substring(1, 1);
        errorScan = true;
    }
    else
    {
        tabla[2, n] = "Oparl, %";
        tabla[3, n] = "14";
    }
    break;
case "[":
    //comprueba la cantidad de simbolos
    if (lectura.Length >= 2)
    {
        tabla[2, n] = "Error, caracter inesperado: " + lectura.Substring(1, 1);
        errorScan = true;
    }
    else
    {
        tabla[2, n] = "llave apertura";
        tabla[3, n] = "7";
    }
    break;
case "]":
    //comprueba la cantidad de simbolos
    if (lectura.Length >= 2)
    {
        tabla[2, n] = "Error, caracter inesperado: " + lectura.Substring(1, 1);
        errorScan = true;
```

```
    }
    else
    {
        tabla[2, n] = "llave cerradura";
        tabla[3, n] = "8";
    }
    break;
case ".":
    //comprueba la cantidad de simbolos
    if (lectura.Length >= 2)
    {
        tabla[2, n] = "Error, caracter inesperado: " + lectura.Substring(1, 1);
        errorScan = true;
    }
    else
    {
        tabla[2, n] = "punto";
        tabla[3, n] = "27";
    }
    break;
case "(":
    //comprueba la cantidad de simbolos
    if (lectura.Length >= 2)
    {
        tabla[2, n] = "Error, caracter inesperado: " + lectura.Substring(1, 1);
        errorScan = true;
    }
    else
    {
```

```
            tabla[2, n] = "Parentesis apertura";
            tabla[3, n] = "29";
        }
        break;
    case ")":
        //comprueba la cantidad de simbolos
        if (lectura.Length >= 2)
        {
            tabla[2, n] = "Error, caracter inesperado: " + lectura.Substring(1, 1);
            errorScan = true;
        }
        else
        {
            tabla[2, n] = "Parentesis cerradura";
            tabla[3, n] = "30";
        }
        break;
    case "#":
        //comprueba la cantidad de simbolos
        if (lectura.Length >= 2)
        {
            if (lectura.Substring(1, 1) == "#")
            {
                tabla[2, n] = "Comentario";
                tabla[3, n] = "37";
            }
            else
            {
                tabla[2, n] = "Error, se espera ## ,no: " + lectura.Substring(1, 1);
```

```
                errorScan = true;
            }
          }
          else
          {
            tabla[2, n] = "Error, se espera ## ,no: " + lectura.Substring(1, 1);
            errorScan = true;
          }
          break;
        default:
          comprobarNoSimb(lectura, n);
          break;
      }
    }
    else
    {
      comprobarNoSimb(lectura, n);
    }
}

private void comprobarNoSimb(string lectura, int n)
{
    int numero;//solo para comprobaciones
    double numero2;

    //si el primer caracter es una letra empieza la comprobacion de ser una palabra
    if (char.IsLetter(lectura.Substring(0, 1).ToCharArray()[0]))
    {
      if (lectura.Length > 1)
```

```
    {
        validarPalabra(lectura.Substring(1), lectura, n);
    }
    else
    {
        tabla[2, n] = "Identificador";
        tabla[3, n] = "6";
    }
}
//si es numero entero
else if (int.TryParse(lectura, out numero) && (numero != 0))
{
    tabla[2, n] = "numero, entero";
    tabla[3, n] = "3";
}
//si es numero fraccionario
else if (double.TryParse(lectura, out numero2))
{
    if (numero2 == 0)
    {
        tabla[2, n] = "numero, entero";
        tabla[3, n] = "3";
    }
    else
    {
        tabla[2, n] = "numero, fraccion";
        tabla[3, n] = "4";
    }
}
```

```
        else
        {
            tabla[2, n] = "Error de sintaxis";
            errorScan = true;
        }
    }

    //metodo para validar que la palabra sea reservada o indentificador
    //recursivo hasta que no queden simbolos en el string
    public void validarPalabra(string entrada, string completa, int n)
    {
        if (char.IsLetter(entrada.Substring(0, 1).ToCharArray()[0]) ||
char.IsDigit(entrada.Substring(0, 1).ToCharArray()[0]) || (entrada.Substring(0,
1).ToCharArray()[0]) == '_')
        {
            if (entrada.Length > 1)
            {
                validarPalabra(entrada.Substring(1), completa, n);
            }
            else
            {
                //comprueba si es una palabra reservada o identificador
                if (validarPReservada(completa))
                {
                    tabla[2, n] = "Palabra reservada";
                    traductor(completa, n);
                }
                else
                {
                    tabla[2, n] = "Identificador";
```

```
            tabla[3, n] = "6";
          }
        }
      }
      //en caso de no cumplir con las condiciones
      else
      {
        tabla[2, n] = "Caracter inesperado: ' " + entrada.Substring(0, 1) + " '";
        errorScan = true;
      }
    }

    //metodo que valida si la entrada es una de las palabras reservadas
    public bool validarPReservada(string entrada)
    {
      string[] palabrasReservadas = { "entero", "fraccion","cadena", "private", "static",
                                      "program", "if", "else", "empieza", "termina", "celes",
                                      "leer", "escribir" };
      for (int i = 0; i < palabrasReservadas.Length; i++)
      {
        if (string.Compare(entrada.ToLower(), palabrasReservadas[i].ToLower()) == 0)
        {
          return true;//true si si es una palabra reservada
        }
      }
      return false;//si no es una palabra reservada
    }
```

```
//============================================================================
================================================

    //Movimientos de archivo

//============================================================================
================================================

    private void button1_Click_1(object sender, EventArgs e)

    {

      rtbDevolver.Text = "";

      rtbParser.Text = "";

      rtbSemantica.Text = "";

    }


    private void abrirToolStripMenuItem_Click(object sender, EventArgs e)

    {

      var respuesta = MessageBox.Show("¿Abrir nuevo?", "Celes", MessageBoxButtons.YesNo,
MessageBoxIcon.Question);


      if (respuesta == DialogResult.Yes)

      {

        NombreArchivo = "";

        rtbLeer.Text = "";

        rtbDevolver.Text = "";

        rtbParser.Text = "";

      }

    }


    private void guardarToolStripMenuItem_Click(object sender, EventArgs e)

    {

      ofdArchivos.Filter = "Archivos de texto *(.txt)|*.txt";
```

```
        if (ofdArchivos.ShowDialog() == DialogResult.OK)
        {
           //comprueba que no halla un archivo abierto
           if (NombreArchivo != "")
           {
              //si hay un archivo abierto pregunta al usuario si quiere abrir el otro de todas formas
              var respuesta = MessageBox.Show("Ya hay un archivo abierto\n¿abrir otro?", "Celes",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);

              if (respuesta == DialogResult.Yes)
              {
                 NombreArchivo = ofdArchivos.FileName;
                 try
                 {
                    var sr = new StreamReader(NombreArchivo);
                    rtbLeer.Text = sr.ReadToEnd();
                 }
                 catch (SecurityException ex)
                 {
                    MessageBox.Show("Error al seleccionar archivo");
                 }
              }
           }
           else//si no hay un archivo seleccionado sinplemente lo habre
           {
              NombreArchivo = ofdArchivos.FileName;
              try
              {
                 var sr = new StreamReader(NombreArchivo);
```

```
                rtbLeer.Text = sr.ReadToEnd();
                sr.Close();
            }
            catch (SecurityException ex)
            {
                MessageBox.Show("Error al seleccionar archivo");
            }
        }
    }

}

private void borrarToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (NombreArchivo == "")
    {
        var respuesta = MessageBox.Show("¿Crear archivo?", "Celes",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);

        if (respuesta == DialogResult.Yes)
        {
            SaveFileDialog saveFileDialog1 = new SaveFileDialog();
            saveFileDialog1.Filter = "Archivos de texto *(.txt)|*.txt";
            saveFileDialog1.Title = "Guarda este archivo";
            saveFileDialog1.ShowDialog();

            if (saveFileDialog1.FileName != "")
            {
                // Saves the Image via a FileStream created by the OpenFile method.
```

```
            System.IO.FileStream fs =
              (System.IO.FileStream)saveFileDialog1.OpenFile();

            fs.Close();

            NombreArchivo = saveFileDialog1.FileName;

            //guarda el archivo
            using (StreamWriter escribir = new
StreamWriter(Path.Combine(Application.StartupPath, NombreArchivo)))
            {
              string cadena = rtbLeer.Text;
              escribir.WriteLine(String.Format("{0}", cadena));
              escribir.Flush();
              escribir.Close();
            }
          }
        }
        else
        {
          using (StreamWriter escribir = new
StreamWriter(Path.Combine(Application.StartupPath, NombreArchivo)))
          {
            string cadena = rtbLeer.Text;
            escribir.WriteLine(String.Format("{0}", cadena));
            escribir.Flush();
            escribir.Close();
          }
        }
```

```
}

private void cerrarToolStripMenuItem_Click(object sender, EventArgs e)
{
   rtbLeer.Text = "";
   rtbDevolver.Text = "";
   rtbParser.Text = "";
   NombreArchivo = "";
}



//Parser
private void btnParser_Click(object sender, EventArgs e)
{
   if (!errorScan)
   {
      //arreglo con la columna de la tabla
      int[] enteros = new int[conteo];

      // Convertir cada cadena en un entero
      for (int i = 0; i < enteros.Length; i++)
      {
         enteros[i] = int.Parse(tabla[3, i]);
      }
      //si devuelve true es que esta bien y lo marca en verde
      if (comprobarParser(enteros))
      {
         rtbParser.Text = "correcto";
         rtbParser.SelectAll();
```

```
            rtbParser.SelectionColor = Color.Green;
            rtbParser.DeselectAll();
        }
        //de lo contrario error y rojo
        else
        {
            rtbParser.Text = "error";
            rtbParser.SelectAll();
            rtbParser.SelectionColor = Color.Red;
            rtbParser.DeselectAll();
        }
    }
    else
    {
        MessageBox.Show("Error en el scan");
    }
}

private void traductor(string reservado, int n)
{
    switch (reservado)
    {
        case "program":
            tabla[3, n] = "9";
            break;
        case "if":
            tabla[3, n] = "10";
            break;
        case "else":
```

```
            tabla[3, n] = "11";
            break;
        case "empieza":
            tabla[3, n] = "12";
            break;
        case "termina":
            tabla[3, n] = "13";
            break;
        case "leer":
            tabla[3, n] = "26";
            break;
        case "escribir":
            tabla[3, n] = "28";
            break;
        case "entero":
            tabla[3, n] = "38";
            break;
        case "fraccion":
            tabla[3, n] = "39";
            break;
    }
}

public bool comprobarParser(int[] secuencia)
{
    bool correcto = false;
    //como debemos tener al menos 5 minimo
    if (secuencia.Length >= 5)
    {
```

```
        //comprueba :   " program identificador [ "    , y que termine con " ] "
        if (secuencia[0] == 9 && secuencia[1] == 6 && secuencia[2] == 7 &&
secuencia[secuencia.Length - 1] == 8)
        {
          apuntador = 3;
          correcto = forma(secuencia);
        }
      }
      return correcto;
    }

    //comprueba que forma es
    static int apuntador, copiaApuntador;
    public bool forma(int[] secuencia) //el apuntador donde va la secuencia
    {
      bool correcto = false;
      switch (secuencia[apuntador])
      {
        //comienza comentario
        case 37:
          apuntador++;
          if (secuencia[apuntador] == 6)
          {
            apuntador++;
            correcto = true;
          }
          break;
        //Entrada
        case 6:
```

```
            apuntador++;
            if (secuencia[apuntador] == 27)
            {
              apuntador++;
              if (secuencia[apuntador] == 26)
              {
                apuntador++;
                correcto = true;
              }
            }
            //asignar con += , -=, *=, /= , =
            else if (secuencia[apuntador] <= 25 && secuencia[apuntador] >= 21)
            {
              apuntador++;
              if (secuencia[apuntador] == 3 || secuencia[apuntador] == 4 ||
secuencia[apuntador] == 6)
              {
                apuntador++;
                copiaApuntador = apuntador;
                correcto = aritmetrico(secuencia);
              }
            }
            //asignar con ++ y --
            else if (secuencia[apuntador] == 20 || secuencia[apuntador] == 19)
            {
              apuntador++;
              correcto = true;
            }
            break;
```

```
//Salida
case 28:
  apuntador++;
  if (secuencia[apuntador] == 29)
  {
    apuntador++;
    if (secuencia[apuntador] == 6)
    {
      apuntador++;
      if (secuencia[apuntador] == 30)
      {
        apuntador++;
        correcto = true;
      }
    }
  }
  break;
//condicional
case 10:
  apuntador++;
  if (secuencia[apuntador] == 29)
  {
    apuntador++;
    if (condicion(secuencia))
    {
      if (secuencia[apuntador] == 30)
      {
        apuntador++;
        if (cuerpo(secuencia))
```

```
          {
            apuntador++;
            if (secuencia[apuntador] == 11)
            {
              apuntador++;
              if (cuerpo(secuencia))
              {
                apuntador++;
                correcto = true;
              }
            }
            else
            {
              correcto = true;
            }
          }
        }
      }
    }
    break;
  //declarar variable entero
  case 38:
    apuntador++;
    if (secuencia[apuntador] == 6)
    {
      apuntador++;
      if (secuencia[apuntador] == 25)
      {
        apuntador++;
```

```
            if (secuencia[apuntador] == 3 || secuencia[apuntador] == 6)
            {
              apuntador++;
              correcto = true;
            }
          }
          else
          {
            correcto = true;
          }
        }
        break;
      //declarar variable fraccion
      case 39:
        apuntador++;
        if (secuencia[apuntador] == 6)
        {
          apuntador++;
          if (secuencia[apuntador] == 25)
          {
            apuntador++;
            if (secuencia[apuntador] == 4 || secuencia[apuntador] == 3 ||
secuencia[apuntador] == 6)
            {
              apuntador++;
              correcto = true;
            }
          }
          else
```

```
            {
                correcto = true;
            }
        }
        break;
    }

    //si todavia hay otra forma
    if (correcto && apuntador != conteo - 1 && secuencia[apuntador] != 13)
    {
        correcto = forma(secuencia);
    }

    return correcto;
}

//metodo que comprueba que es una operacion aritmetrica
public bool aritmetrico(int[] secuencia)
{
    if (secuencia[apuntador] <= 18 && secuencia[apuntador] >= 14)
    {
        apuntador++;
        if (secuencia[apuntador] == 3 || secuencia[apuntador] == 4 || secuencia[apuntador] ==
6)
        {
            apuntador++;
            return aritmetrico(secuencia);
        }
        else
```

```
      {
         apuntador = copiaApuntador;
         return false;
      }
   }
   return true;
}


//metodo que comprueba si el cuerpo de una condicion coincide
public bool cuerpo(int[] secuencia)
{
   if (secuencia[apuntador] == 12)
   {
      apuntador++;
      if (forma(secuencia))
      {
         if (secuencia[apuntador] == 13)
         {
            return true;
         }
      }
   }
   return false;
}

//metodo que comprueba la estructura de una condicion
public bool condicion(int[] secuencia)
{
```

```
        if (secuencia[apuntador] == 6)
        {
            apuntador++;
            if (secuencia[apuntador] <= 36 && secuencia[apuntador] >= 31)
            {
                apuntador++;
                if (secuencia[apuntador] == 6 || secuencia[apuntador] == 3 || secuencia[apuntador]
== 4)
                {
                    apuntador++;
                    return true;
                }
            }
        }
        return false;
    }


//=========================================================================
==============================================

    // Boton de semantica

//=========================================================================
==============================================

    static string[,] tablaVariable;
    int conteoVariable = 0;
    string comentarioSemantico = "";
    private void btnSemantica_Click(object sender, EventArgs e)
    {
        rtbSemantica.Text = "";
        tablaVariable = new string[4, 100];
```

```
conteoVariable = 1;
comentarioSemantico = "";
bool semanticaTrue = true;

//guarda el program como identificador
tablaVariable[0, 0] = "program";
tablaVariable[1, 0] = tabla[1, 1];

for (int i = 3; i < conteo; i++)
{
   if (int.Parse(tabla[3, i]) == 38 || int.Parse(tabla[3, i]) == 39)
   {
      if (existeVar(i + 1) == "")
      {
         tablaVariable[0, conteoVariable] = tabla[1, i];
         tablaVariable[1, conteoVariable] = tabla[1, i + 1];
         //verifica si tiene valor
         if (int.Parse(tabla[3, i + 2]) <= 25 && int.Parse(tabla[3, i + 2]) >= 21)
         {
            tablaVariable[2, conteoVariable] = "si";
            tablaVariable[3, conteoVariable] = tabla[1, i + 3];
         }
         else
         {
            tablaVariable[2, conteoVariable] = "no";
            tablaVariable[3, conteoVariable] = "?";
         }
         conteoVariable++;
         i++;
```

```
    }
    else
    {
        comentarioSemantico = ", variable ' " + tabla[1, i + 1] + " ' ya esta declarada";
        semanticaTrue = false;
        break;
    }
}
//comprueba si es una variable
else if (int.Parse(tabla[3, i]) == 6)
{
    //comprueba que este en la tabla de variables
    if (existeVar(i) == "")
    {
        comentarioSemantico = ", variable ' " + tabla[1, i] + " ' no esta declarada";
        semanticaTrue = false;
        break;
    }
    //le sigue un asignativo
    else if (int.Parse(tabla[3, i + 1]) <= 25 && int.Parse(tabla[3, i + 1]) >= 21)
    {
        for (int j = 0; j < conteoVariable; j++)
        {
            if (tablaVariable[1, j] == tabla[1, i])
            {
                if (tablaVariable[0, j] == "entero")
                {
                    if (!tipoDato(i + 2))
                    {
```

```
                        comentarioSemantico = ", " + tabla[1, i] + " " + tabla[1, i + 1] + " " + tabla[1, i
+ 2] + " tipo de dato invalido";
                        semanticaTrue = false;
                        break;
                    }
                    else
                    {
                        tablaVariable[2, numVar(i)] = "si";
                    }
                }
            }
        }
        else if (tablaVariable[2, numVar(i)] == "no")
        {
            comentarioSemantico = ", la variable '" + tablaVariable[1, numVar(i)] + "' no
inicializada";
            semanticaTrue = false;
            break;
        }
    }
}

if (semanticaTrue)
{
    rtbSemantica.Text = "Correcto" + comentarioSemantico;
    rtbSemantica.Text += "\n---------------------------------\n";

    for (int i = 0; i < conteoVariable; i++)
    {
```

```
            rtbSemantica.Text += tablaVariable[0, i] + " -.- " + tablaVariable[1, i] + "\n";
        }

        //Cambiar de color
        rtbSemantica.SelectionStart = 0; // posición inicial
        rtbSemantica.SelectionLength = 8; // longitud del texto a cambia
        rtbSemantica.SelectionColor = Color.Green;
    }
    else
    {
        rtbSemantica.Text = "Incorrecto" + comentarioSemantico;
        rtbSemantica.Text += "\n---------------------------------\n";

        for (int i = 0; i < conteoVariable; i++)
        {
            rtbSemantica.Text += tablaVariable[0, i] + " -.- " + tablaVariable[1, i] + "\n";
        }
        //Cambiar de color
        rtbSemantica.SelectionStart = 0; // posición inicial
        rtbSemantica.SelectionLength = 10; // longitud del texto a cambia
        rtbSemantica.SelectionColor = Color.Red;
    }
    rtbSemantica.SelectionLength = 0;
    rtbSemantica.SelectionColor = Color.Black;
}

private bool tipoDato(int n)
{
    int dato = n - 2;
```

```
        while ((int.Parse(tabla[3, n]) <= 18 && 15 <= int.Parse(tabla[3, n])) || (int.Parse(tabla[3, n])
== 3 || 4 == int.Parse(tabla[3, n]) || 6 == int.Parse(tabla[3, n])))

        {

          if (int.Parse(tabla[3, n]) == 4)

          {

            return false;

          }

          else if (int.Parse(tabla[3, n]) == 6)

          {

            if (!(existeVar(n) == existeVar(dato)))

            {

              comentarioSemantico = ", [Advertencia] Posible perdida de datos entre: '" +
tabla[1, dato] + "' & '" + tabla[1, n] + "'";

            }

            return true;

          }

          n++;

        }

        return true;

      }


      private string existeVar(int n)

      {

        for (int j = 0; j < conteoVariable; j++)

        {

          if (tablaVariable[1, j] == tabla[1, n])

          {

            return tablaVariable[0, j];

          }

        }
```

```
    return "";
}

private int numVar(int n)
{
    for (int j = 0; j < conteoVariable; j++)
    {
        if (tablaVariable[1, j] == tabla[1, n])
        {
            return j;
        }
    }
    return 0;
}

private void cbxComprobar_CheckedChanged(object sender, EventArgs e)
{
    if (parserMostrar)
    {
        parserMostrar = false;
    }
    else
    {
        parserMostrar = true;
    }
}

private void btnAumentar_Click(object sender, EventArgs e)
{
```

```
    string selectedOption = cbbAgrandar.SelectedItem?.ToString();

    // Asignar el texto al RichTextBox según la opción seleccionada
    switch (selectedOption)
    {
        case "Programa":
            rtbAumentar.Text = rtbLeer.Text;
            break;
        case "Scanner":
            rtbAumentar.Text = rtbDevolver.Text;
            break;
        case "Parser":
            rtbAumentar.Text = rtbParser.Text;
            break;
        case "Semantica":
            rtbAumentar.Text = rtbSemantica.Text;
            break;
        case "Directivas":
            rtbAumentar.Text = rtbDirectivas.Text;
            break;
        default:
            rtbAumentar.Text = "Por favor selecciona una opción válida.";
            break;
    }
}

//Boton de directivas
private void btnDirectivas_Click(object sender, EventArgs e)
{
```

```
    rtbDirectivas.Text = ".DATA"; //coloca .data al inicio

    for (int i = 1; i < conteoVariable; i++)
    {
        rtbDirectivas.Text += "\n" + lineaCode(i); //agrega cada linea del .code
    }

    rtbDirectivas.Text += "\n.CODE"; //coloca .code al final del .data
}

private string lineaCode(int index)
{
    string tipo;
    if (tablaVariable[0, index] == "entero") //si es entero asigna DW por el contrario DD
    {
        tipo = "DW";
    }
    else
    {
        tipo = "DD";
    }
    //regresa la variable con el tipo y su valor
    return tablaVariable[1, index] + "\t" + tipo + "\t\t" + tablaVariable[3, index];
}

int saltos = 0;
string tipoRegistro = "";
bool eselse = false;
private void btnCode2_Click(object sender, EventArgs e)
```

```
{
   saltos = 1;
   btnDirectivas_Click(sender, e);
   for (int i = 3; i < conteo; i++)
   {
      tipoRegistro = "";
      i = traduccionesIndice(i);
   }
}

private int traduccionesIndice(int i)
{
   //comprueba que sea un = , += , *=, -=, /=
   if (int.Parse(tabla[3, i]) <= 25 && int.Parse(tabla[3, i]) >= 21)
   {
      //comprueba que la primera sea * / + -
      if (int.Parse(tabla[3, i + 2]) <= 18 && int.Parse(tabla[3, i + 2]) >= 15)
      {
         //selecciona el tipo de registro
         if (int.Parse(tabla[3, i - 1]) == 39)
         {
            tipoRegistro = "E";
         }
         //separar la asignacion
         for (int j = i + 4; j < conteo; j = j + 2)
         {
            if (!(int.Parse(tabla[3, j]) <= 18 && int.Parse(tabla[3, j]) >= 15))
            {
               rtbDirectivas.Text += traduccionOperaciones(GetSubArray(tabla, i + 1, j - 1));
```

```
            break;
          }
        }
        rtbDirectivas.Text += "\n\tMOV " + tabla[1, i - 1] + ", " +tipoRegistro +"AX";
    }
    //si es una declaracion simplemente se lo salta
    else if (int.Parse(tabla[3, i-2]) == 38 || int.Parse(tabla[3, i-2]) == 39)
    {
        i = i + 2;
    }
    //si es una asignacion
    else
    {
        if (existeVar(i-1) != "entero")
        {
            tipoRegistro = "E";
        }
        else
        {
            tipoRegistro = "";
        }
        rtbDirectivas.Text += "\n\n\tMOV " + tipoRegistro + "AX, " + tabla[1, i+1];
        rtbDirectivas.Text += "\n\tMOV " + tabla[1, i-1] + ", " + tipoRegistro + "AX";
    }
}
//si es un if
else if (int.Parse(tabla[3, i]) == 10)
{
    if(existeVar(i + 2) != "entero")
```

```
    {
      tipoRegistro = "E";
    }
    rtbDirectivas.Text += "\n\n\tMOV " + tipoRegistro + "AX, " + tabla[1, i + 2];
    rtbDirectivas.Text += "\n\tCMP " + tipoRegistro + "AX, " + tabla[1, i + 4];

    switch (tabla[1, i + 3])
    {
      case "<":
        rtbDirectivas.Text += "\n\tJGE ELSE" + saltos;
        break;
      case "<=":
        rtbDirectivas.Text += "\n\tJG ELSE" + saltos;
        break;
      case ">":
        rtbDirectivas.Text += "\n\tJLE ELSE" + saltos;
        break;
      case ">=":
        rtbDirectivas.Text += "\n\tJL ELSE" + saltos;
        break;
      case "<>":
        rtbDirectivas.Text += "\n\tJE ELSE" + saltos;
        break;
      case "==":
        rtbDirectivas.Text += "\n\tJNE ELSE" + saltos;
        break;
    }
    i = i + 6;
  }
```

```
//si es un termina
else if (int.Parse(tabla[3, i]) == 13)
{
   if (int.Parse(tabla[3, i+1]) == 11)
   {
      eselse = true;
      rtbDirectivas.Text += "\n\tJMP FIN" + saltos;
      rtbDirectivas.Text += "\nELSE" + saltos + ":";
   }
   else if (eselse)
   {
      eselse = false;
      rtbDirectivas.Text += "\nFIN" + saltos + ":";
      saltos++;
   }
   else
   {
      rtbDirectivas.Text += "\nELSE" + saltos + ":";
      saltos++;
   }
}
//si es un ++
else if(int.Parse(tabla[3, i]) == 19)
{
   rtbDirectivas.Text += "\n\n\tINC " + tabla[1, i - 1];
}
//si es un --
else if(int.Parse(tabla[3, i]) == 20)
{
```

```
        rtbDirectivas.Text += "\n\n\tDEC " + tabla[1, i-1];
    }
    //si se encuentra un punto acompañado de una variable
    else if(int.Parse(tabla[3, i]) == 27)
    {

    }
    return i;
}

private string traduccionOperaciones(string[] tokens)
{
    string devolver = "";
    //primera vuelta
    devolver += "\n\tMOV " + tipoRegistro + "AX, " + tokens[0];
    devolver += "\n\tMOV " + tipoRegistro + "BX, " + tokens[2];
    switch (tokens[1])
    {
        case "+":
            devolver += "\n\tADD " + tipoRegistro + "AX, " + tipoRegistro + "BX";
            break;
        case "-":
            devolver += "\n\tSUB " + tipoRegistro + "AX, " + tipoRegistro + "BX";
            break;
        case "*":
            devolver += "\n\tIMUL " + tipoRegistro + "AX, " + tipoRegistro + "BX";
            break;
        case "/":
            devolver += "\n\tIDIV " + tipoRegistro + "AX, " + tipoRegistro + "BX";
```

```
            break;
        }
        devolver += "\n";
        //segunda vuelta en adelante
        for (int i = 4; i < tokens.Length; i = i + 2)
        {
            devolver += "\n\tMOV " + tipoRegistro + "BX, " + tokens[i];
            switch (tokens[i-1])
            {
                case "+":
                    devolver += "\n\tADD " + tipoRegistro + "AX, " + tipoRegistro + "BX";
                    break;
                case "-":
                    devolver += "\n\tSUB " + tipoRegistro + "AX, " + tipoRegistro + "BX";
                    break;
                case "*":
                    devolver += "\n\tIMUL " + tipoRegistro + "AX, " + tipoRegistro + "BX";
                    break;
                case "/":
                    devolver += "\n\tIDIV " + tipoRegistro + "AX, " + tipoRegistro + "BX";
                    break;
            }
            devolver += "\n";
        }
        return devolver;
    }


    static int conteoData = 0, conteoCode = 0;
```

```
    private void button2_Click(object sender, EventArgs e)
    {
        conteoData = 0;
        conteoCode = 0;
        rtbBajoNivel.Text = "";
        char[] separadores2 = { ' ', '\n', '\r', '\t', ',' };
        string[] tokenesDirectivas = rtbDirectivas.Text.Split(separadores2,
StringSplitOptions.RemoveEmptyEntries);

        foreach (var token in tokenesDirectivas)
        {
            if(token == ".DATA")
            {
                puntoData(tokenesDirectivas);
            }
            else if(token == ".CODE")
            {
                puntoCode(tokenesDirectivas);
            }
        }
    }

    private void puntoCode(string[] tokenesDirectivas)
    {
        for (int i = 0; i < tokenesDirectivas.Length; i++)
        {
            if (tokenesDirectivas[i] != ".CODE")
            {
                switch (tokenesDirectivas[i])
```

```
            {
                case "MOV":
                    i = i + 2;
                    movData(tokenesDirectivas[i - 1], tokenesDirectivas[i]);
                    rtbBajoNivel.Text += "\n";
                    conteoCode += 2;
                    break;
                case "ADD":
                    i = i + 2;
                    rtbBajoNivel.Text += "00A0:" + ponerFomato(4, conteoCode.ToString("X")) +
"\t0000 0011 1100 0011\n";
                    conteoCode += 2;
                    rtbBajoNivel.Text += "\n";
                    break;
                case "CMP":
                    i = i + 2;
                    rtbBajoNivel.Text += "00A0:" + ponerFomato(4, conteoCode.ToString("X")) +
"0001 1110 0000 0000";
                    rtbBajoNivel.Text += "\n";
                    break;
                case "DEC":
                    i++;
                    rtbBajoNivel.Text += "00A0:" + ponerFomato(4, conteoCode.ToString("X")) +
"1111 1111 " + direccionVar(tokenesDirectivas[i]);
                    rtbBajoNivel.Text += "\n";
                    break;
            }
        }
        else if (tokenesDirectivas[i] == ".CODE")
        {
```

```
            rtbBajoNivel.Text += "\n";
          }
        }
      }

      private void movData(string registro, string variable)
      {
        string resultado = "00A0:" + ponerFomato(4, conteoCode.ToString("X")) + "\t1000 100"
+sacarW(registro) +" ";
        resultado += "0000 0" + regresarR(registro) +" ";
        resultado += direccionVar(variable);
        rtbBajoNivel.Text += resultado;
      }

      private string direccionVar(string var)
      {
        string resultado23 = "";
        for (int i = 0; i < conteoTI; i++)
        {
          if (tablaIndicadores[0, i] == var)
          {
            return "0000 0000 0000 0000" + dividirCada4(ponerFomato(16
,Convert.ToString(Convert.ToInt32(tablaIndicadores[1, i], 16), 2)));
          }
        }
        return resultado23;
      }

      private string regresarR(string r)
      {
```

```
    switch (r)
    {
        case "AX":
        case "EAX":
            return "000";
        case "BX":
        case "EBX":
            return "011";

    }
    return "000";
}

private int sacarW(string registro)
{
    return registro.Substring(registro.Length - 1, 1) == "X" ? 1 : 0;
}

string[,] tablaIndicadores;
int conteoTI = 0;
private void puntoData(string[] tokenesDirectivas)
{
    tablaIndicadores = new string[2, 100];
    conteoTI = 0;
    for (int i = 0;i < tokenesDirectivas.Length; i++)
    {
        if (tokenesDirectivas[i] != ".DATA")
        {
            switch (tokenesDirectivas[i])
```

```
            {
                case "DW":
                    i++;
                    string direccion = ponerFomato(4, conteoData.ToString("X"));
                    if (tokenesDirectivas[i] == "?")
                    {
                        rtbBajoNivel.Text += "0000:" + ponerFomato(4, conteoData.ToString("X")) +
"\t0000 0000 0000 0000 \n";
                    }
                    else
                    {
                        rtbBajoNivel.Text += "0000:" + ponerFomato(4, conteoData.ToString("X")) + "\t"
+ dividirCada4(ponerFomato(16, Convert.ToString((int.Parse(tokenesDirectivas[i])), 2))) +"\n";
                    }
                    conteoData = conteoData + 2;
                    tablaIndicadores[0, conteoTI] = tokenesDirectivas[i-1];
                    tablaIndicadores[1, conteoTI] = direccion;
                    conteoTI++;
                    break;
                case "DD":
                    i++;
                    string direccion2 = ponerFomato(4, conteoData.ToString("X"));
                    if (tokenesDirectivas[i] == "?")
                    {
                        rtbBajoNivel.Text += "0000:" + ponerFomato(4, conteoData.ToString("X")) +
"\t0000 0000 0000 0000 0000 0000 0000 0000 \n";
                    }
                    else
                    {
```

```
                    rtbBajoNivel.Text += "0000:" + ponerFomato(4,conteoData.ToString("X")) + "\t"
+ dividirCada4(ponerFomato(32, Convert.ToString((int.Parse(tokenesDirectivas[i])), 2))) + "\n";
                }
                conteoData = conteoData + 4;
                tablaIndicadores[0, conteoTI] = tokenesDirectivas[i - 1];
                tablaIndicadores[1, conteoTI] = direccion2;
                conteoTI++;
                break;
            }
        }
        else if(tokenesDirectivas[i] == ".CODE")
        {
            rtbBajoNivel.Text += "\n";
            break;
        }
    }
}

private string ponerFomato(int numCeros, string binario)
{
    return binario.Length == numCeros ? binario : ponerFomato(numCeros, "0" +binario);
}

private string dividirCada4(string binario)
{
    string devolver = "";
    for (int i = 0; i < binario.Length; i = i + 4)
    {
        devolver += binario.Substring(i, 4) +" ";
```

```
        }
        return devolver;
    }

    private string[] jerarquia(string[] tokens)
    {
        //Si no se extiende mas de 4 no es posible que se cambie el orden
        if (tokens.Length < 4)
        {
            return tokens;
        }
        //mide mas de 4 entonces se pueden aplicar que reorganice
        List<string> resultado = new List<string>(tokens);

        if((resultado[1] != "*" && resultado[1] != "/") && (resultado[resultado.Count-2] == "*" ||
resultado[resultado.Count - 2] == "/"))
        {
            //voltear lista
            resultado.Reverse();
        }

        //Si tiene un + num + pasar al ultimo +num;
        for (int i = 1; i < resultado.Count-2; i++)
        {
            if ((resultado[i] == "+" || resultado[i] == "-") && (resultado[i+2] == "+" || resultado[i+2] ==
"-"))
            {
                List<string> temporal = resultado.GetRange(i, i + 1); ;
                resultado.RemoveRange(i, i + 1);
                resultado.AddRange(temporal);
```

```
            }
        }

        string tmeporaofjaof = "";
        for (int i = 0; i < resultado.Count; i++)
        {
            tmeporaofjaof += "[ " +resultado[i] +" ]";
        }
        MessageBox.Show(tmeporaofjaof);

        return resultado.ToArray();
    }

    static string[] GetSubArray(string[,] array, int start, int end)
    {
        // Longitud del subarreglo
        int length = end - start + 1;
        string[] result = new string[length];

        // Recorrer desde el índice de inicio hasta el índice final
        for (int i = 0; i < length; i++)
        {
            result[i] = array[1, start + i];
        }
        return result;
    }
  }
}
```