Codificación

Botón

```
 #region Analizador Semantico
public void IntercambiarIdentificadoresParaSemantica()
{
    rtxTokenSemantica.Text = "";
    string[] ArregloLineasFuente = new string[rtxFuente.Lines.Length];
    string[] ArregloLineasLexico = new string[rtxToken.Lines.Length];

    for (int i = 0; i <= rtxFuente.Lines.Length - 1; i++)
    {
        ArregloLineasFuente[i] = rtxFuente.Lines[i].Trim(' '); //Guarda cada linea
en una celda del arreglo - Tambien le quita los espacios al final de la cadena
        int numeroTokens = ArregloLineasFuente[i].Split(' ').Length; //Obtiene el
numero de tokens en la linea - i -
        string[] AuxiliarTokensFuente = new string[numeroTokens]; //Inicializa el
arreglo auxiliar con el numero de tokens
        ArregloLineasFuente[i].Split(' ').CopyTo(AuxiliarTokensFuente, 0);
//Almacenar la linea - i - en un arreglo auxiliar (cada token en una celda)

        ArregloLineasLexico[i] = rtxToken.Lines[i].Trim(' '); //Guarda cada linea
en una celda del arreglo - Tambien le quita los espacios al final de la cadena
        int numeroTokensLexico = ArregloLineasLexico[i].Split(' ').Length;
//Obtiene el numero de tokens en la linea - i -
        string[] AuxiliarTokensLexico = new string[numeroTokensLexico];
//Inicializa el arreglo auxiliar con el numero de tokens
        ArregloLineasLexico[i].Split(' ').CopyTo(AuxiliarTokensLexico, 0);
//Almacenar la linea - i - en un arreglo auxiliar (cada token en una celda)

        if (Array.Exists(AuxiliarTokensFuente, element =>
element.StartsWith("@")))
        {
            for (int j = 0; j <= AuxiliarTokensFuente.Length - 1; j++)
            {
                if (AuxiliarTokensLexico[j] == "IDEN" &&
AuxiliarTokensFuente[j].Contains('@'))
                {
                    AuxiliarTokensLexico[j] = AuxiliarTokensFuente[j];
                }
            }
        }
        string strLinea = string.Join(" ", AuxiliarTokensLexico);
        rtxTokenSemantica.AppendText(strLinea+"\n");
    }

    string[] ArregloLineasSemantica = new string[rtxTokenSemantica.Lines.Length];
    string strTextoRtxSemanticaNuevo = "";
```

```
    for (int i = 0; i <= rtxTokenSemantica.Lines.Length - 1; i++)
    {
        ArregloLineasSemantica[i] = rtxTokenSemantica.Lines[i].Trim(' '); //Guarda
cada linea en una celda del arreglo - Tambien le quita los espacios al final de la
cadena
        int numeroTokens = ArregloLineasSemantica[i].Split(' ').Length; //Obtiene
el numero de tokens en la linea - i -
        string[] AuxiliarTokensSemantica = new string[numeroTokens]; //Inicializa
el arreglo auxiliar con el numero de tokens
        ArregloLineasSemantica[i].Split(' ').CopyTo(AuxiliarTokensSemantica, 0);
//Almacenar la linea - i - en un arreglo auxiliar (cada token en una celda)

        for (int j = 0; j <= AuxiliarTokensSemantica.Length - 1; j++)
        {
            foreach (Simbolo simbolo in ListaSimbolos)
            {
                if (AuxiliarTokensSemantica[j] == simbolo.Lexema)
                {
                    AuxiliarTokensSemantica[j] = simbolo.Tipo;
                }
            }

        }
        string strLinea = string.Join(" ", AuxiliarTokensSemantica);
        strTextoRtxSemanticaNuevo += strLinea + "\n";
    }
    rtxTokenSemantica.Text = "";
    rtxTokenSemantica.Text = strTextoRtxSemanticaNuevo;
}

public string AjustarTokensParaSemantica (string strCadena)
{
    var reemplazos = new Dictionary<string, string>();
    reemplazos.Add("CORE", "REAL");
    reemplazos.Add("COEN", "ENTR");
    reemplazos.Add("LETR", "CADE");

   foreach (var reemplazo in reemplazos) {
      strCadena = strCadena.Replace(reemplazo.Key, reemplazo.Value);
   }

   return strCadena;
}

private void btnSemantica_Click(object sender, EventArgs e)
{
    ListaErroresSemantica.Clear();
    dgvErroresSemantica.Rows.Clear();
    rtxDerivacionesJELU.Text = "";
    rtxDerivacionesSemantica.Text = "";
    rtxJELUVertical.Text = "";
    lstSalidaJELU.Clear();
    IntercambiarIdentificadoresParaSemantica(); //Cambiar los identificadores por
su tipo
```

```
    rtxTokenSemantica.Text =
AjustarTokensParaSemantica(rtxTokenSemantica.Text.TrimEnd(char.Parse("\n")));
//Ajuste de tokens restantes: CORE, COEN, LETR

    string[] ArregloLineasSemantica = new string[rtxTokenSemantica.Lines.Length];
    bool enviarABottomUp = true;

    for (int i = 0; i <= rtxTokenSemantica.Lines.Length - 1; i++)
    {
        intLineaErrorSemantica = i + 1;
        ArregloLineasSemantica[i] = rtxTokenSemantica.Lines[i].Trim(' '); //Guarda
cada linea en una celda del arreglo - Tambien le quita los espacios al final de la
cadena
        int numeroTokens = ArregloLineasSemantica[i].Split(' ').Length; //Obtiene
el numero de tokens en la linea - i -
        string[] AuxiliarTokensSemantica = new string[numeroTokens]; //Inicializa
el arreglo auxiliar con el numero de tokens
        ArregloLineasSemantica[i].Split(' ').CopyTo(AuxiliarTokensSemantica, 0);
//Almacenar la linea - i - en un arreglo auxiliar (cada token en una celda)

        //Valida si hay un ERROR en la salida de sintaxis
        enviarABottomUp = !rtxDerivaciones.Text.Contains("ERROR DE SINTAXIS");

        if (enviarABottomUp)
        { //Si no hay error envía los tokens de la linea actual a
BottomUpSemantico
            BottomUpSemantico(AuxiliarTokensSemantica);
        }
        else
        { //Si encuentra un error (de sintaxis) avisa para corregir desde código
fuente
            rtxTokenSemantica.Text = "";
            rtxDerivacionesSemantica.Text = "";
            MessageBox.Show("Existe un error en sintáxis", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            break;
        }
    }

    //Arreglo de lineas de sintaxis para metodo JELU
    string[] ArregloTokensSintaxis = new string[rtxTokenSintaxis.Lines.Length];

    if (enviarABottomUp) {
        //Para metodo JELU
        for (int i = 0; i <= rtxTokenSintaxis.Lines.Length - 1; i++)
        {
            ArregloTokensSintaxis[i] = rtxTokenSintaxis.Lines[i].Trim(' ');
//Guarda cada linea en una celda del arreglo - Tambien le quita los espacios al
final de la cadena
            int numeroTokens = ArregloTokensSintaxis[i].Split(' ').Length;
//Obtiene el numero de tokens en la linea - i -
            string[] AuxiliarTokens = new string[numeroTokens]; //Inicializa el
arreglo auxiliar con el numero de tokens
            ArregloTokensSintaxis[i].Split(' ').CopyTo(AuxiliarTokens, 0);
```

```
//Almacenar la linea - i - en un arreglo auxiliar (cada token en una celda)
            BottomUpJELU(AuxiliarTokens);
        }
        PintarErroresSemantica("ERROR DE SEMANTICA", Color.Red, 0);
        tabPestania.SelectedIndex = 1;
        BottomUpJELUVertical(lstSalidaJELU.ToArray());
        PintarErroresJELU("ERROR DE SEMANTICA", Color.Red, 0);
    }
    if(dgvErroresSemantica.ColumnCount == 1)
    {
        CadenaInfija();
    }
}

public void BottomUpSemantico(string[] arrTokens)
{
    int cantidadTokens = arrTokens.Length; //Obtiene el tamaño de la cadena de
tokens
    string cadenaTokens = string.Join(" ", arrTokens); //Convertir el arreglo de
la cadena de tokens a un string

    rtxDerivacionesSemantica.Text += cadenaTokens + "\n"; //Agrega la cadena de
tokens al textbox de derivaciones

    if (dictReglasSemanticas.ContainsKey(cadenaTokens) &&
dictReglasSemanticas[cadenaTokens] == "S" ||
dictReglasSemanticas.ContainsKey(cadenaTokens) &&
dictReglasSemanticas[cadenaTokens] == "NOAP")
    { //Si encuentra una S
        rtxDerivacionesSemantica.Text += dictReglasSemanticas[cadenaTokens] +
"\n";
        rtxDerivacionesSemantica.Text += dictReglasSemanticas[cadenaTokens] == "S"
? "\n" : "";
        rtxDerivacionesSemantica.Text += dictReglasSemanticas[cadenaTokens] ==
"NOAP" ? "\n" : "";
    }
    else //Caso: No encuentra una S directa, necesita derivarse o validar que
realmente no existe
    {
        string cadenaTokensModificada = cadenaTokens, strBloque;
        int aux = 0, restador = 1;

        do
        {
            int cantidadAuxiliar = cadenaTokensModificada.Split(' ').Length;
//toma la cantidad de tokens en la cadena modificada
            string[] ArrTokensModificados = new string[cantidadAuxiliar];
            cadenaTokensModificada.Split(' ').CopyTo(ArrTokensModificados, 0); //
Convierto en arreglo la cadena modificada

            strBloque = string.Join(" ",
ArrTokensModificados.Skip(aux).Take(cantidadAuxiliar - restador));    //Se obtiene
el bloque actual del recorrido
```

```
            if (dictReglasSemanticas.ContainsKey(strBloque)) //Pregunta si el
diccionario contiene el bloque
            {
                cadenaTokensModificada = cadenaTokensModificada.Replace(strBloque,
dictReglasSemanticas[strBloque]); //Si encuentra el bloque lo reemplaza con la
gramatica
                rtxDerivacionesSemantica.Text += cadenaTokensModificada + "\n";
//Visuales
                rtxDerivacionesSemantica.Text += dictReglasSemanticas[strBloque]
== "S" & cadenaTokensModificada.Split(' ').Length == 1 ? "\n" : "";//Visuales

                if ((dictReglasSemanticas[strBloque] != "S" &&
dictReglasSemanticas[strBloque] != "NOAP") & cadenaTokensModificada.Split('
').Length == 1)
                {
                    rtxDerivacionesSemantica.Text += "ERROR DE SEMANTICA\n\n";
                    AgregarErroresSemantica();
                }
                else
                {
                    rtxDerivacionesSemantica.Text += "";
                }
                aux = 0;
                restador = 0;
            }
            else //Si no lo contiene lleva a cabo el sig. proceso para saber si ya
recorrió toda la cadena antes de decrementar el numero de tkns
            {
                if ((aux + (cantidadAuxiliar - restador)) == cantidadAuxiliar)
                {
                    aux = 0;
                    restador++;
                }
                else { aux++; }

                if (strBloque == "" | (cadenaTokensModificada.Split('
').Contains("S") & cadenaTokensModificada.Split(' ').Length > 1))
                {
                    rtxDerivacionesSemantica.Text += "ERROR DE SEMANTICA\n\n";
//Visuales
                    AgregarErroresSemantica();
                    break; //Controlar si encuentra un error de sintaxis romper
para que no se cicle
                }
            }
        } while (cadenaTokensModificada.Split(' ').Length > 1);
    }
}

public void BottomUpJELU(string[] arrTokens)
{
    string cadenaTokens = string.Join(" ", arrTokens);
    rtxDerivacionesJELU.Text += cadenaTokens + "\n"; //Agrega la cadena de tokens
al richtextbox de derivaciones
```

```
    string valorEncontrado; //Para saber si el diccionario contiene el valor
buscado, si no lo encuentra esta variable será null
    if (dictGramaticasJELU.ContainsKey(cadenaTokens) &&
dictGramaticasJELU.TryGetValue(cadenaTokens, out valorEncontrado))
    {
        rtxDerivacionesJELU.Text += string.Concat(valorEncontrado, "\n\n");
        lstSalidaJELU.Add(valorEncontrado);
    }
    else
    { //BottomUp Core
        string cadenaTokensModificada = cadenaTokens, strBloque;
        int aux = 0, restador = 1;

        do
        {
            int cantidadAuxiliar = cadenaTokensModificada.Split(' ').Length;
//toma la cantidad de tokens en la cadena modificada
            string[] ArrTokensModificados = new string[cantidadAuxiliar];
            cadenaTokensModificada.Split(' ').CopyTo(ArrTokensModificados, 0); //
Convierto en arreglo la cadena modificada

            strBloque = string.Join(" ",
ArrTokensModificados.Skip(aux).Take(cantidadAuxiliar - restador)); //Se obtiene el
bloque actual del recorrido

            if (dictGramaticasJELU.ContainsKey(strBloque))
            {
                cadenaTokensModificada = cadenaTokensModificada.Replace(strBloque,
dictGramaticasJELU[strBloque]);

                rtxDerivacionesJELU.Text += string.Concat(cadenaTokensModificada,
"\n"); //VISUALES
                //rtxDerivacionesJELU.Text += cadenaTokensModificada.Split('
').Length == 1 ? "\n" : ""; //VISUALES
                if (cadenaTokensModificada.Split(' ').Length == 1)
                {
                    rtxDerivacionesJELU.Text += "\n";
                    lstSalidaJELU.Add(cadenaTokensModificada);
                }
                //lstSalidaJELU.Add(valorEncontrado);

                dictGramaticasJELU.TryGetValue(cadenaTokens, out valorEncontrado);

                if (valorEncontrado != null && cadenaTokensModificada.Split('
').Length != 1) {
                    rtxDerivacionesJELU.Text += "";
                }

                aux = 0;
                restador = 0;
            }
            else
            {
```

```
                if ((aux + (cantidadAuxiliar - restador)) == cantidadAuxiliar) {
                    aux = 0;
                    restador++;
                }
                else {
                  aux++;
                }

                if ((strBloque == "" || ((cadenaTokensModificada.Split('
').Contains("S") || cadenaTokensModificada.Split(' ').Contains("CON")))) &&
cadenaTokensModificada.Split(' ').Length > 1)
                {
                    rtxDerivacionesJELU.Text += "ERROR DE SEMANTICA\n\n";
//Visuales
                    break; //Controlar si encuentra un error de sintaxis romper
para que no se cicle
                }
            }

        } while (cadenaTokensModificada.Split(' ').Length > 1);
    }
}

public void BottomUpJELUVertical(string[] arrTokens)
{
    string cadenaTokens = string.Join(" ", arrTokens);
    rtxJELUVertical.Text += cadenaTokens + "\n"; //Agrega la cadena de tokens al
richtextbox de derivaciones

    string valorEncontrado; //Para saber si el diccionario contiene el valor
buscado, si no lo encuentra esta variable será null
    if (dictGramaticasJELU.ContainsKey(cadenaTokens) &&
dictGramaticasJELU.TryGetValue(cadenaTokens, out valorEncontrado))
    {
        rtxJELUVertical.Text += string.Concat(valorEncontrado, "\n\n");
    }
    else
    { //BottomUp Core
        string cadenaTokensModificada = cadenaTokens, strBloque;
        int aux = 0, restador = 1;

        do
        {
            int cantidadAuxiliar = cadenaTokensModificada.Split(' ').Length;
//toma la cantidad de tokens en la cadena modificada
            string[] ArrTokensModificados = new string[cantidadAuxiliar];
            cadenaTokensModificada.Split(' ').CopyTo(ArrTokensModificados, 0); //
Convierto en arreglo la cadena modificada

            strBloque = string.Join(" ",
ArrTokensModificados.Skip(aux).Take(cantidadAuxiliar - restador)); //Se obtiene el
bloque actual del recorrido

            if (dictGramaticasJELU.ContainsKey(strBloque))
```

```
            {
                cadenaTokensModificada = cadenaTokensModificada.Replace(strBloque,
dictGramaticasJELU[strBloque]);

                rtxJELUVertical.Text += string.Concat(cadenaTokensModificada,
"\n"); //VISUALES
                if (cadenaTokensModificada.Split(' ').Length == 1 &&
cadenaTokensModificada.Split(' ').Contains("CON"))
                {
                    rtxJELUVertical.Text += "ERROR DE SEMANTICA\n\n";
                    ListaErroresSemantica.Add("ERROR: FALTA ABRIR O CERRAR UNA
INSTRUCCIÓN");
                    ActualizarErroresSemantica();
                }

                dictGramaticasJELU.TryGetValue(cadenaTokens, out valorEncontrado);

                if (valorEncontrado != null && cadenaTokensModificada.Split('
').Length != 1)
                {
                    rtxJELUVertical.Text += "";
                }

                aux = 0;
                restador = 0;
            }
            else
            {
                if ((aux + (cantidadAuxiliar - restador)) == cantidadAuxiliar)
                {
                    aux = 0;
                    restador++;
                }
                else
                {
                    aux++;
                }

                if ((strBloque == "" || cadenaTokensModificada.Split('
').Contains("S")) && cadenaTokensModificada.Split(' ').Length > 1)
                {
                    rtxJELUVertical.Text += "ERROR DE SEMANTICA\n\n"; //Visuales
                    ListaErroresSemantica.Add("ERROR: FALTA ABRIR O CERRAR UNA
INSTRUCCIÓN");
                    ActualizarErroresSemantica();
                    break; //Controlar si encuentra un error de sintaxis romper
para que no se cicle
                }
            }

        } while (cadenaTokensModificada.Split(' ').Length > 1);
    }
}
```

```
#endregion









#region Analizador Léxico
private void btnAnalizador_Click(object sender, EventArgs e)
{
    TotalErrores = 0;
    rtxToken.Text = "";
    dgvErrores.Rows.Clear();
    dgvTablaSimbolos.Rows.Clear();
    //arreglo obtiene el largo de las lineas y su longitud
    string[] ArregloFuente = new string[rtxFuente.Lines.Length];
    //guarda cada caracter del texto
    for (int i = 0; i <= rtxFuente.Lines.Length - 1; i++) {
        ArregloFuente[i] = Blanco(rtxFuente.Lines[i]);
    }
    // * Inicia recorrido de las líneas del código
    for (int j = 0; j <= ArregloFuente.Length - 1; ++j) {
        int intEstado = 0;
        string Cadena = "";
        // * Foreach que permite hacer un recorrido para analizar la cadena
        foreach (char chrCaracter in ArregloFuente[j]) {
            if (chrCaracter.ToString() != " ") {
                // * DataSet --> Huecos en memoria que almacenan base de datos,
similar a access
                // * DataTable --> Es un elemento de DataSet.La info que llega
aqui, se manda en automatico a DataSet
                // * Un DataTable contiene un DataColumn y DataRows
                // * DataColumn --> Columnas de DataTable
                // * DataRows --> Filas de DataTable

                // * El DataAdapter conduce datos de la base de datos al DataSet y
viceversa. Además se puede abrir y cerrar
                // * una conexión por sí solo
                Cadena += chrCaracter.ToString();
                foreach (DataColumn Columna in dtMatriz.Columns) {
                    //*Compara el caracter de arreglo con el caracter de la
columna de la matriz
                    if (chrCaracter.ToString() == Columna.ColumnName) {
```

```
                        if (dtMatriz.Rows[intEstado][Columna].ToString() != "FDC")
{
                            String encabezado = Columna.ColumnName;
                            intEstado = int.Parse(dtMatriz.Rows[intEstado]
[encabezado].ToString());
                            break;
                        }
                    }
                }
            }
            else
            {
                if(intEstado == 0) {
                    break;
                }
                //Se obtiene el estado final
                intEstado = int.Parse(dtMatriz.Rows[intEstado]
[dtMatriz.Columns.Count - 2].ToString());
                string strToken = dtMatriz.Rows[intEstado][dtMatriz.Columns.Count
- 1].ToString();
                if(strToken=="ERROR") {
                    dgvErrores.Rows.Add(j + 1, strToken, "ERROR EN LA LÍNEA " +
Lineas);
                    rtxToken.Text += strToken + " ";
                    TotalErrores++;
                    lblNumErrores.Text = TotalErrores.ToString();
                    lstErrores.Add(strToken);
                }
                else if(strToken=="PR02") {
                    strTipo = "ENTERO";
                    rtxToken.Text += strToken + " ";
                }
                else if(strToken=="PR08") {
                    strTipo = "REAL";
                    rtxToken.Text += strToken + " ";
                }
                else {
                    rtxToken.Text += strToken + " ";
                }
                /*LLENADO TABLA DE SÍMBOLOS
                AQUI ENTRARA EN CASO DE SER UN IDENTIFICADOR, SE ASEGURARA
MEDIANTE EL USO DE LA CADENAAUX QUE CONTENDRA EL
                TIPO DE DATO
                ENTERO VAR1 O REAL VAR2
                */
                if (strToken == "IDEN" && CadenaAux == "ENTERO" || CadenaAux ==
"REAL" || CadenaAux == "CADENA") {

                    switch (CadenaAux)
                    {
                        case "ENTERO":
                            CadenaAux = "ENTR";
                            dgvTablaSimbolos.Rows.Add(Cadena, strToken,
CadenaAux);
```

```
                            break;
                        case "REAL":
                            dgvTablaSimbolos.Rows.Add(Cadena, strToken,
CadenaAux);
                            break;
                        case "CADENA":
                            CadenaAux = "CADE";
                            dgvTablaSimbolos.Rows.Add(Cadena, strToken,
CadenaAux);
                            break;
                    }
                }
                /*SINO EN CASO DE QUE EL TOKEN SEA COEN...*/
                else if (strToken == "COEN") {
                    //LA COLUMNA VALOR SE LLENA CON UNA CADENA
                    dgvTablaSimbolos.Rows.Add(Cadena, strToken, "ENTR");
                }
                /*SINO EN CASO DE QUE EL TOKEN SEA CORE...*/
                else if (strToken == "CORE") {
                    dgvTablaSimbolos.Rows.Add(Cadena, strToken, "REAL");
                }
                CadenaAux = Cadena;
                Cadena = "";
                intEstado = 0;
            }
        }
        rtxToken.Text += "\n";
    }
    rtxToken.SelectionStart = rtxToken.Text.Length;
    posicion = rtxToken.SelectionStart;
    //Borra duplicados de dgvTablaSimbolos
    RemoverDuplicados(dgvTablaSimbolos);
    //Guarda la tabla de simbolos a una lista de objetos
    GuardarSimbolos();
}
#endregion
```