

MANUAL TÉCNICO

Expresiones regulares

C= Cualquier cosa

D= [0-9] L=[a-z]

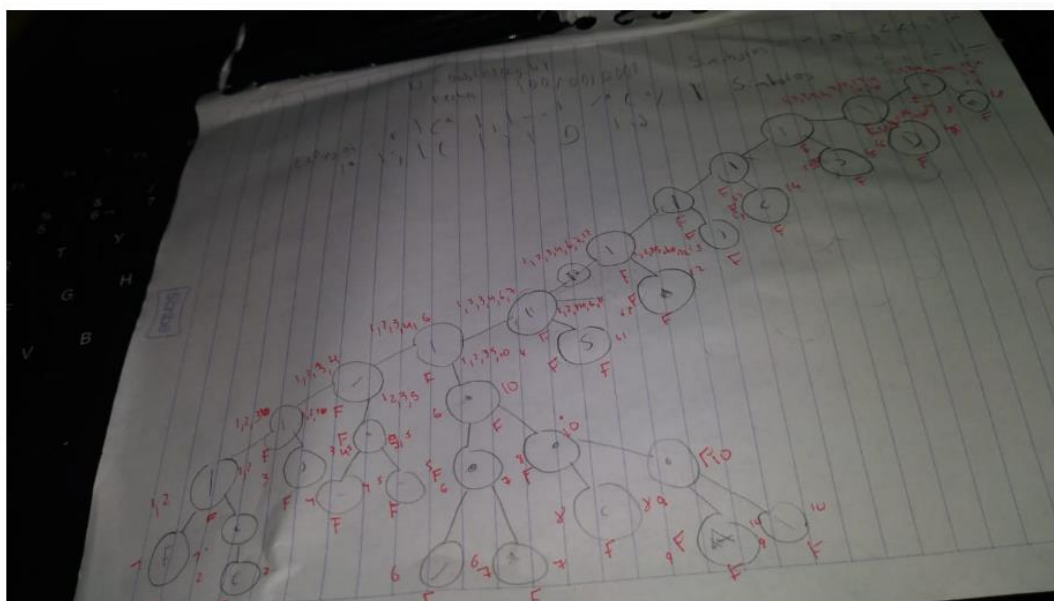
Fecha = { DD/DD/DDDD }

$$\text{Id} = \{ L(L|D|_)^* \}$$
$$\text{Double} = \{D+.D+)\}$$

Expresión Regular Proyecto 1

```
'Fecha' | " C*" , | --C| /* C */| < | > | <= | =>| !=| *  
|; | ( | )|Double|Digito+
```

Árbol Sintáctico



Lista de Tokens

1	Token igual	=
2	Token coma	,
3	Token pyc	;
4	Token Pa	(
5	Token Pc)
6	Token CorA	[
7	Token CorC]
8	Token punto	.
9	Resv Tabla	Tabla
10	Resv Insertar	Insertar
11	Resv Eliminar	Eliminar
12	Resv Modificar	Modificar
13	Resv Seleccionar	Seleccionar
14	Resv Actualizar	Actualizar
15	Resv Crear	Crear
16	Resv entero	entero
17	Resv fecha	fecha

18	Resv flotante	flotante
19	Resv cadena	cadena
20	Resv En	En
21	Resv De	De
22	Resv Donde	Donde
23	Resv Como	Como
24	Resv Y	Y
25	Resv O	O
26	Resv Establecer	Establecer
27	Resv valores	Valores
28	ID	soyId_5
29	Digitos	45
30	Fechas	'10/10/2000'
31	Menor igual	<=
32	Menor	<
33	Mayor	>
34	Mayor igual	>=
35	Comentario Mult	/* */
36	Comentario Lin	-- com
37	Asterisco	*

GRAMÁTICA

Gramatica Proyecto

<Inicio> ::= Tk-Crear <CREAR TABLA> //ya

|Tk-Insertar <INSERTAR> //ya

|Tk-Seleccionar <SELECCIONAR> //ya

|Tk-Eliminar <Eliminar> //

|Tk-Actualizar <ACTUALIZAR>

<CREAR TABLA > ::= Tk TABLA TK ID TK (<Contenido> Tk) Tk;

<Contenido> ::= TK ID <TIPO CONTENIDO> <OTRO CONTENIDO>

<TIPO CONTENIDO> ::= ENTERO

| STRING

| FLOTANTE

| FECHA

<OTRO CONTENIDO> ::= TK COMA <Contenido>

| Epsilon

<INSERTAR> ::= TK EN TK ID TK Valores Tk (<Valorl> Tk) Tk;

<Valorl> ::= <TipoValor> <OtroValor>

<TipoValor> ::= Digito

| CADENA

| DECIMAL

| FECHA

<OtroValor> TK , <Valorl>

| Epsilon

**<SELECCIONAR> ::= <SELECCION> TK ID TK_DE <SELECT TABLE> <DONDE>
TK;**

<SELECCION> ::= TK*

| TK ID TK . TK ID <Al Seleccionar>

| <OTRA SELECCION>

<OTRA SELECCION> ::= TK , <Select>

<Select> TK ID TK . <TIPO SELECT> //FACTORIZAR

<TIPO SELECT> TK ID <Alias> <OTRA SELECCION>

| TK* <OTRA SELECCION>

<Alias> Tk como TK ID

| Epsilon

< SELECT TABLE> TK ID <OTRA TABLA>

<OTRA TABLA> ::= TK , <SELECT TABLE>

| Epsilon

<DONDE> TK DONDE <CONDITION>

| Epsilon

**<CONDITION>::= <DATOS> <SYMBOL> <DATOS> <O CONDITION>
TABLA1.COLUMNNA = TABLA2.COLUMNNA 6 = TABLA2.COLUMNNA**

<SYMBOL> TK = | TK != | TK < | TK <= | TK > | TK >=;

<DATO> DIGITO | DECIMAL | CADENA | FECHA | TK ID //fALTA ID.ID

< O CONDITION > TK Y <CONDITION>

| TK O <CONDITION>

| Epsilon

/* ACTUALIZAR <tabla>

ESTABLECER (<campo1>=<valor1> [, <campo2>=<valor2>,...])

[DONDE <Condicion1>

(Y|O) <Condicion2>,

....]; */

<Eliminar> TK DE TK ID <T DONDE > TK PUNTO Y COMA

<ACTUALIZAR> TK ID TK ESTA BLECER tk(<ESTABLECER> TK)

<T DONDE>

<ESTABLECER > TK ID TK = <DATO2.0> <OTRO ESTABLECER>

<DATO 2.0> DIGITO | DECIMAL | CADENA | FECHA

<OTRO ESTABLECER> TK, <ESTABLECER>

/Epsilon

```
ireferencia
public void GrafoPrueba()
{
    Tablas alv = Table.Inicio;
    while (alv != null)
    {
        Console.WriteLine("Nombre Tabla:" + " " + alv.GetNombre());
        foreach (Atributo aux in alv.A)
        {
            Console.WriteLine("Columna " + " " + aux.GetNombre());

            foreach (Object item in aux.Objetos)
            {
                Console.WriteLine(item);
            }
        }
        alv = alv.Sig;
    }
}
```

Para en análisis descendente recursivo se hacen uso de llamadas a métodos y al método emparejar para los no terminales que pide tokens para seguir con el análisis.

```

1 referencia
void INSERTAR()
{
    emparejar(Tokens.TipoToken.EN);

    emparejar(Tokens.TipoToken.ID);

    emparejar(Tokens.TipoToken.VALORES);
    emparejar(Tokens.TipoToken.PARENTESIS_ABIERTO);

    ValueInsertar();
    emparejar(Tokens.TipoToken.PARENTESIS_CERRADO);
    emparejar(Tokens.TipoToken.PUNTO_y_COMA);
}

2 referencias
void ValueInsertar()
{
    TipoValue();
    TipoValue2();
}

```

Se tienen valores según el tipo de Token que permiten conocer el tipo de token reconocido.

```

2 referencias
void Ayuda()
{
    ayuda = true;
    while (alvAct.getTipo_Token() != Tokens.TipoToken.PUNTO_y_COMA && cont < ListaTok.Count - 1)
    {
        cont++;
        alvAct = ListaTok.ElementAt(cont);
    }
}

1 referencia
public string GetTipString(Tokens.TipoToken tip)
{
    switch (tip)
    {
        case Tokens.TipoToken.ACTUALIZAR:
            return "Actualizar";
        case Tokens.TipoToken.Asterisco:
            return "**";

        case Tokens.TipoToken.CADENA:
            return "Cadena";

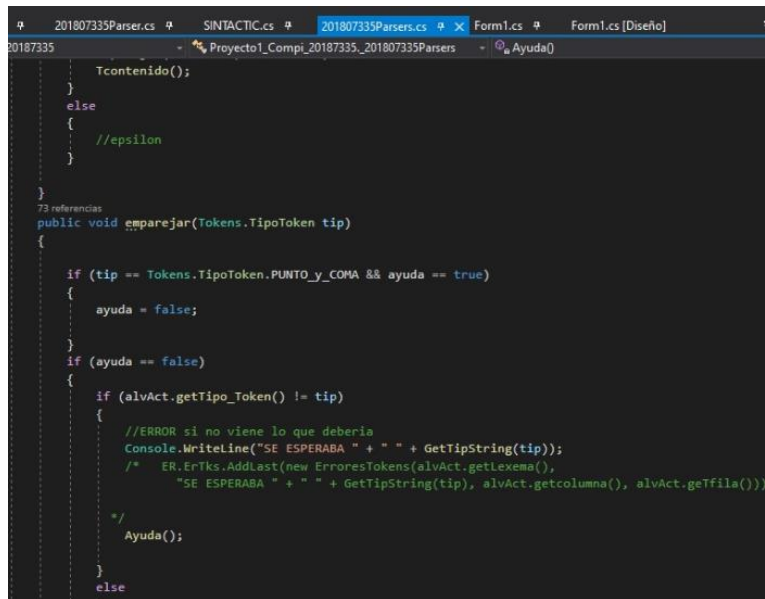
        case Tokens.TipoToken.COMILLA:
            return "COMILLA";

        case Tokens.TipoToken.LAST:
            return "LAST";

        case Tokens.TipoToken.COMA:
            return "COMA";
    }
}

```


El método emparejar es fundamental en el análisis descendente recursivo ya que ve si existe un error o no comparándolo con los tokens y si se rigen a lo esperado según la gramática, caso contrario debería recuperarme.



```
201807335Parsers.cs  SINTACTIC.cs  201807335Parsers.cs  Form1.cs  Form1.cs [Diseño]
201807335  Proyecto1_Compi_201807335_201807335Parsers  Ayuda()

    Tcontenido();
    }
    else
    {
        //epsilon
    }
}
73 referencias
public void emparejar(Tokens.TipoToken tip)
{
    if (tip == Tokens.TipoToken.PUNTO_y_COMA && ayuda == true)
    {
        ayuda = false;
    }
    if (ayuda == false)
    {
        if (alvAct.getTipo_Token() != tip)
        {
            //ERROR si no viene lo que deberia
            Console.WriteLine("SE ESPERABA " + " " + GetTipString(tip));
            /*  ER.ErTks.AddLast(new ErroresTokens(alvAct.getLexema(),
                "SE ESPERABA " + " " + GetTipString(tip), alvAct.getcolumna(), alvAct.getfila()));
            */
            Ayuda();
        }
    }
    else
```

Para realizar el grafo se hicieron uso de listas y objetos que fueron retornados gracias al análisis sintáctico, al ser recursivo se tenían todos los nodos, usando la posición de memoria para no repetir.

```

99+ referencias
public Arbol(String Nombre )
{
    this.Nombre=Nombre;
    A = new LinkedList<Arbol>();
}

2 referencias
public int getLongitud() {
    return A.Count;
}

3 referencias
public string getDot()
{
    int cont = 0;
    string Dot = this.GetHashCode().ToString() + "[label=\"" + this.Nombre + "\"]\n";

    if (getLongitud() > 0)
    {
        while (cont < getLongitud())
        {
            Dot += this.GetHashCode().ToString() + "->" + A.ElementAt(cont).GetHashCode() + "\n";
            Dot += A.ElementAt(cont).getDot() + "\n";
            cont++;
        }
    }

    return Dot;
}

```

The screenshot shows the Visual Studio IDE with the following details:

- Menu Bar:** Archivo, Editar, Ver, Proyecto, Compilar, Depurar, Prueba, Analizar, Herramientas, Extensiones, Ventana, Ayuda, Buscar (Ctrl+G).
- Toolbox:** Debug, Any CPU, Iniciar, and various icons for solution explorer, search, and other IDE functions.
- Output Window:** Shows the command prompt output: "C:\> Proyecto1_Compil_20187335.graficador - %generaDot(string rdoto, string rpng)".
- Code Editor:**
 - Line 11-13:**

```

String ruta;
StringBuilder grafica;

```
 - Line 14:**

```

Inferencia
public Graficador()

```
 - Line 15-17:**

```

{
    ruta = Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory);
}

```
 - Line 18:**

```

Inferencia
private void generarDot(String rdoto, String rpng)

```
 - Line 19-29:**

```

{
    File.WriteAllText(rdoto, grafica.ToString());
    String comandoDot = "dot -Tpng \"\" + rdoto + "\" -o \"" + rpng + "\"";
    var comando = String.Format(comandoDot);
    var procesoStart = new System.Diagnostics.ProcessStartInfo("cmd", "/C" + comando);
    var procedimiento = new System.Diagnostics.Process();
    procedimiento.StartInfo = procesoStart;
    procedimiento.Start();
    procedimiento.WaitForExit();
}

```
 - Line 30:**

```

}

```
 - Line 31:**

```

Inferencia
public void graficar(String texto)

```
 - Line 32-39:**

```

{
    grafica = new StringBuilder();
    String rdoto = ruta + "\\Arbol.dot";
    String rpng = ruta + "\\Arbol.png ";
    grafica.Append(texto);
    this.generarDot(rdoto, rpng);
}

```
 - Line 40:**

```

}

```
- Status Bar:** 100%, No se encontraron problemas, Línea 20, Carácter 20, SPC, CRLF.