```
using System;
using System.Collections
.Generic;
using System.Text.RegularExpressions;

namespace WindowsFormsApplication2
{
    internal class Analizador
    {
        private string text;
        private List<string> mylist = null;
        private string errores;
        private bool hayErrores;
        public Stack<string> pila = null;
        public int LARGO = 18;
        public int ANCHO = 30;
        public int estadoInicial;
        public int entradaInicial;
        public string[,] tablaLR = new string[,]
         //(      )     ,    c_cuant  c_log  c_palab c_comp  c_var  c_neg    $
           Atom Expr Func Comp  LTerm  Term
        { { "s1", "-1", "-1", "s7" ,  "-1",    "s8",  "-1", "s9",  "s10",   "-1",
            "2", "3", "4", "5", "-1",   "6"},//0
          { "s1", "-1", "-1", "s7" ,  "-1",    "s8",  "-1", "s9",  "s10",   "-1",
             "2", "11", "4", "5", "-1",   "6"},//1
          { "-1", "r5", "-1", "-1" ,  "r5",    "-1",  "-1", "-1",  "-1",   "r5",
            "-1", "-1", "-1", "-1", "-1", "-1"},//2
          { "-1", "-1", "-1", "-1" ,  "s12",    "-1",  "-1", "-1",  "-1",   "AC",
             "-1", "-1", "-1", "-1", "-1", "-1"  },//3
          { "-1", "r11", "r11", "-1" , "r11",  "-1",  "r11", "-1",  "-1",
            "r11", "-1", "-1", "-1", "-1", "-1", "-1" },//4
          { "-1", "r7", "-1", "-1" ,  "r7",    "-1",  "-1", "-1",  "-1",   "r7",
            "-1", "-1", "-1", "-1", "-1", "-1" },//5
          { "-1", "-1", "-1", "-1" ,  "-1",    "-1",  "s13", "-1",  "-1",   "-1",
             "-1", "-1", "-1", "-1", "-1", "-1" },//6
          { "-1", "-1", "-1", "-1" ,  "-1",    "-1",  "-1", "s14",  "-1",   "-1",
             "-1", "-1", "-1", "-1", "-1", "-1" },//7
          { "s15", "r14", "r14", "-1" ,"r14",  "-1",  "r14", "-1",  "-1",
            "r14", "-1", "-1", "-1", "-1", "-1", "-1"},//8
          { "-1", "r13", "r13", "-1" ,  "r13", "-1",  "r13", "-1",  "-1",
            "r13", "-1", "-1", "-1", "-1", "-1", "-1" },//9
          { "s1", "-1", "-1",  "s7" ,  "-1",    "s8",  "-1", "s9",  "s10",  "-1",
             "2", "16", "4", "5", "-1", "6" },//10
          { "-1", "s17", "-1", "-1" ,  "s12",    "-1",  "-1", "-1",  "-1",
            "-1", "-1", "-1", "-1", "-1", "-1", "-1"},//11
          { "s1", "-1", "-1", "s7" ,  "-1",    "s8",  "-1", "s9",  "s10",   "-1",
             "2", "18", "4", "5", "-1", "6" },//12
          { "-1", "-1", "-1", "-1" ,  "-1",    "s20",  "-1", "s9",  "-1",   "-1",
             "-1", "-1", "4", "-1", "-1", "19" },//13
          { "s1", "-1", "-1", "s7" ,  "-1",    "s8",  "-1", "s9",  "s10",   "-1",
             "2", "21", "4", "5", "-1", "6"   },//14
          { "-1", "-1", "-1", "-1" ,  "-1",    "s20",  "-1", "s9",  "-1",   "-1",
```

```
                "-1", "-1", "4", "-1", "22", "23"  },//15
37            { "-1", "r1", "-1", "-1" ,  "r1",    "-1",  "-1", "-1",  "-1",   "r1",
                "-1", "-1", "-1", "-1", "-1", "-1"  },//16
38            { "-1", "r2", "-1", "-1" ,  "r2",    "-1",  "-1", "-1",  "-1",   "r2",
                "-1", "-1", "-1", "-1", "-1", "-1"  },//17
39            { "-1", "r3", "-1", "-1" ,  "s12",    "-1",  "-1", "-1",  "-1",   "r3",
                "-1", "-1", "-1", "-1", "-1", "-1"   },//18
40            { "-1", "r8", "-1", "-1" ,  "r8",    "-1",  "-1", "-1",  "-1",   "r8",
                "-1", "-1", "-1", "-1", "-1", "-1" },//19
41            { "s24", "r14", "r14", "-1" ,"r14",  "-1",  "r14", "-1",  "-1",  "r14",
                "-1", "-1", "-1", "-1", "-1", "-1"   },//20
42            { "-1", "r4", "-1", "-1" ,  "r4",    "-1",  "-1", "-1",  "-1",   "r4",
                "-1", "-1", "-1", "-1", "-1", "-1"  },//21
43            { "-1", "s25", "-1", "-1" ,  "-1",    "-1",  "-1", "-1",  "-1",   "-1",
                "-1", "-1", "-1", "-1", "-1", "-1" },//22
44            { "-1", "r9", "s26", "-1" ,  "-1",    "-1",  "-1", "-1",  "-1",   "-1",
                "-1", "-1", "-1", "-1", "-1", "-1"  },//23
45            { "-1", "-1", "-1", "-1" ,  "-1",    "s20",  "-1", "s9",  "-1",   "-1",
                "-1", "-1", "4", "-1", "27", "23"  },//24
46            { "-1", "r6", "r12", "-1" , "r6",    "-1",  "r12", "-1",  "-1", "r6",
                "-1", "-1", "-1", "-1", "-1", "-1"  },//25
47            { "-1", "-1", "-1", "-1" ,  "-1",    "s20",  "-1", "s9",  "-1",   "-1",
                "-1", "-1", "4", "-1", "28", "23"   },//26
48            { "-1", "s29", "-1", "-1" ,  "-1",    "-1",  "-1", "-1",  "-1",   "-1",
                "-1", "-1", "-1", "-1", "-1", "-1" },//27
49            { "-1", "r10", "-1", "-1" ,  "-1",    "-1",  "-1", "-1",  "-1",   "-1",
                "-1", "-1", "-1", "-1", "-1", "-1"  },//28
50            { "-1", "r12", "r12", "-1" ,  "r12",  "-1",  "r12", "-1",  "-1",
                "r12", "-1", "-1", "-1", "-1", "-1", "-1"   }//29
51        };
52
53
54        public Analizador(string text)
55        {
56            this.text = text;
57            errores = "Errores:\n";
58            hayErrores = false;
59            mylist = new List<string>();
60            pila = new Stack<string>();
61            pila.Push("$");
62            pila.Push("0");
63            estadoInicial = entradaInicial = 0;
64        }
65
66        /**
67        * Elimina N elementos de la pila dependiendo de la regla
68        **/
69
70        public int eliminaElementos(string regla) {
71            if (regla.Equals("r12"))
72                return 8; // F->p(S)
73            if (regla.Equals("r9"))
```

```
74                  return 2; // S->T
75              if (regla.Equals("r5"))
76                  return 2; // E -> A
77              if (regla.Equals("r11"))
78                  return 2; //T->F
79              if (regla.Equals("r7"))
80                  return 2; //A->N
81              if (regla.Equals("r14"))
82                  return 2; //T->p
83              if (regla.Equals("r13"))
84                  return 2; //T->v
85              if (regla.Equals("r1"))
86                  return 4; //E->~E
87              if (regla.Equals("r2"))
88                  return 6; //E->(E)
89              if (regla.Equals("r3"))
90                  return 6; //E-> ElE
91              if (regla.Equals("r8"))
92                  return 6; //N->TrT
93              if (regla.Equals("r4"))
94                  return 6; //E->cvE
95              if (regla.Equals("r10"))
96                  return 6; //S->T,S
97              if (regla.Equals("r6"))
98                  return 8; //A->p(S)
99              return 0;
100         }
101
102         //Cuando usas una regla es necesario hacer una transicion
103         //En este caso sera necesario verificar los tipos de regla para
104         //ofrecer una nueva entrada en la tabla
105         public int aplicaRegla(string regla) {
106             if (regla.Equals("r12"))
107                 return 12; // F->p(S)
108             if (regla.Equals("r9") || regla.Equals("r10"))
109                 return 14; // S->T //S->T,S
110             if (regla.Equals("r5") || regla.Equals("r1") ||
111                 regla.Equals("r2") || regla.Equals("r3") ||
112                 regla.Equals("r4"))
113                 return 11; // E -> A
114             if (regla.Equals("r11") || regla.Equals("r14") ||
115                 regla.Equals("r13"))
116                 return 15; //T->F
117             if (regla.Equals("r7") || regla.Equals("r6"))
118                 return 10; //A->N
119             if (regla.Equals("r8"))
120                 return 13; //N->TrT
121             return -1;
122         }
123
124         /**
125         *Regresa el valor de la entrada en forma numerica
```

```
126         **/
127         public int dameEntrada(string entrada) {
128             string patronVariable = "(^[A-Z_\\d]+)";
129             string patronPalabra = "(^[a-z_\\d]+)";
130 
131             MatchCollection matchesVariable = Regex.Matches(entrada,
                  patronVariable);
132             MatchCollection matchesPalabra = Regex.Matches(entrada,
                  patronPalabra);
133 
134             if (entrada.Equals("("))
135                 return 0;
136             else if (entrada.Equals(")"))
137                 return 1;
138             else if (entrada.Equals(","))
139                 return 2;
140             else if (entrada.Equals("#") || entrada.Equals("@"))
141                 return 3;
142             else if (entrada.Equals("&") || entrada.Equals("|") ||
143                      entrada.Equals("->") || entrada.Equals("<-") ||
144                      entrada.Equals("<->"))
145                 return 4;
146             else if (matchesPalabra.Count > 0)
147                 return 5;
148             else if (entrada.Equals("=") || entrada.Equals("!=") ||
149                      entrada.Equals("<") || entrada.Equals(">") ||
150                      entrada.Equals("<=") || entrada.Equals(">="))
151                 return 6;
152             else if (matchesVariable.Count > 0)
153                 return 7;
154             else if (entrada.Equals("~"))
155                 return 8;
156             else if (entrada.Equals("$"))
157                 return 9;
158             return -1;
159         }
160 
161         /***
162         *
163         * 200 lineas de un analizador lexico resumidas en solo 5
164         ***/
165         public void analizaLexico()
166         {
167             string patron = "(@|#|[a-z_\\d]+|[A-Z_\\d]+|\\(|\\)|\\->|<\\-|<\\->|
                  &|\\||\\,|~|=|!=|<|>|<=|>=)";
168             MatchCollection matches = Regex.Matches(this.text, patron);
169             for (int i = 0, j = matches.Count; i < j; i++)
170             {
171                 mylist.Add(matches[i].ToString());
172             }
173 
174             validaCoincidencias();
```

```
175         }
176
177         public void validaCoincidencias()
178         {
179             string tmpTexto = this.text;
180
181
182             foreach (string element in mylist)
183             {
184                 if (tmpTexto.StartsWith(element)){
185                     int posicion = 1;
186                     if (element.Length > 1)//Usa dos o mas caracteres
187                         posicion = element.Length;
188                     tmpTexto = tmpTexto.Substring(posicion);
189                     continue;
190                 }
191                 else // Hay un error se debe reportar
192                 {
193                     int posicion = text.IndexOf(tmpTexto[0]);
194                     string error = string.Format("\nError!! -> Caracter invalido
                         '{0}' en la posicion '{1}' \n",
195                                                                                 tmpTexto
                         [0], posicion);
196                     errores += error;
197                     hayErrores = true;
198                     break;
199                 }
200             }
201         }
202
203         //Inicia el nucleo del programa cumpliendo las reglas propuestas
204         public void analizaSintactico()
205         {
206             mylist.Add("$");
207             bool aceptacion = false;
208             int entrada = 0;
209             string accion = "";
210             int cantEliminar = 0;
211             int numElemento = 0;
212
213             while (!aceptacion) {
214                 numElemento++;
215                 entrada = dameEntrada(mylist[0]);
216                 if (entrada == -1)
217                 {
218                     hayErrores = true;
219                     agregaErrores(string.Format("\nEste elemento hace no cumplir
                         la regla {0}:{1}\n", numElemento, mylist[0]));
220                     break;
221                 }
222                 accion = tablaLR[estadoInicial, entrada];
223                 if (accion.StartsWith("s")) {//Desplazamiento
```

```
224                     pila.Push(mylist[0]);
225                     mylist.RemoveAt(0);
226                     accion = accion.Substring(1);
227                     pila.Push(accion);
228                     estadoInicial = int.Parse(pila.Peek());
229                 } else if (accion.StartsWith("r"))//Es una regla
230                 {
231                     cantEliminar = eliminaElementos(accion);
232                     while (cantEliminar-- > 0) {
233                         pila.Pop();
234                     }
235                     estadoInicial = int.Parse(pila.Peek());
236                     entrada = aplicaRegla(accion);
237                     if (estadoInicial.Equals("-1") || entrada.Equals("-1")) {
238                         hayErrores = true;
239                         agregaErrores(string.Format("\nNo se cumple la regla {0}
                          \n{1}", accion, tipoRegla(accion)));
240                         break;
241                     }
242 
243 
244                     accion = tablaLR[estadoInicial, entrada];
245                     pila.Push("soybasura"); //Literalmente hechamos basura
246                     pila.Push(accion);
247                     estadoInicial = int.Parse(accion);
248 
249                 }
250                 else if (accion.Equals("AC"))//Cadena de aceptacion
251                 {
252                     aceptacion = true;
253                     break;
254                 }else if (accion.Equals("-1"))
255                 {
256                     agregaErrores(string.Format("\nElemento donde fallo la cadena
                       {0}:{1}\n", numElemento, mylist[0]));
257                     hayErrores = true;
258                     break;
259                 }
260             }
261 
262         }
263 
264         private object tipoRegla(string regla)
265         {
266             if (regla.Equals("r12"))
267                 return "F->p(S)";
268             if (regla.Equals("r9"))
269                 return "LT->T";
270             if (regla.Equals("r5"))
271                 return "Exp -> Atom";
272             if (regla.Equals("r11"))
273                 return "Term->Func";
```

```
274             if (regla.Equals("r7"))
275                 return "Atom->Compar";
276             if (regla.Equals("r14"))
277                 return "Termino->Objeto";
278             if (regla.Equals("r13"))
279                 return "Termino->Variable";
280             if (regla.Equals("r1"))
281                 return "Exp->~Exp";
282             if (regla.Equals("r2"))
283                 return "Exp->(Exp)";
284             if (regla.Equals("r3"))
285                 return "Exp-> Exp log Exp";
286             if (regla.Equals("r8"))
287                 return "Comp->Term com Term";
288             if (regla.Equals("r4"))
289                 return "Exp->cuant variable Exp";
290             if (regla.Equals("r10"))
291                 return "LT->Term,LT";
292             if (regla.Equals("r6"))
293                 return "Atom->pred(ListERM)";
294             return "Regla desconocida para el sistema";
295         }
296 
297         public bool falloPrograma()
298         {
299             return hayErrores;
300         }
301 
302         public string dameErrores()
303         {
304             return errores;
305         }
306 
307         public void agregaErrores(string error)
308         {
309             errores += error;
310         }
311 
312         public string elementosEncontrados()
313         {
314             string nvoCadena = string.Format("Elementos para la cadena {0} \n",
                  this.text);
315             int numero = 0;
316             foreach(string elemento in this.mylist)
317             {
318                 nvoCadena+= string.Format("\n {0}: {1} \n", numero++, elemento);
319             }
320 
321             return nvoCadena;
322         }
323     }
324 }
```