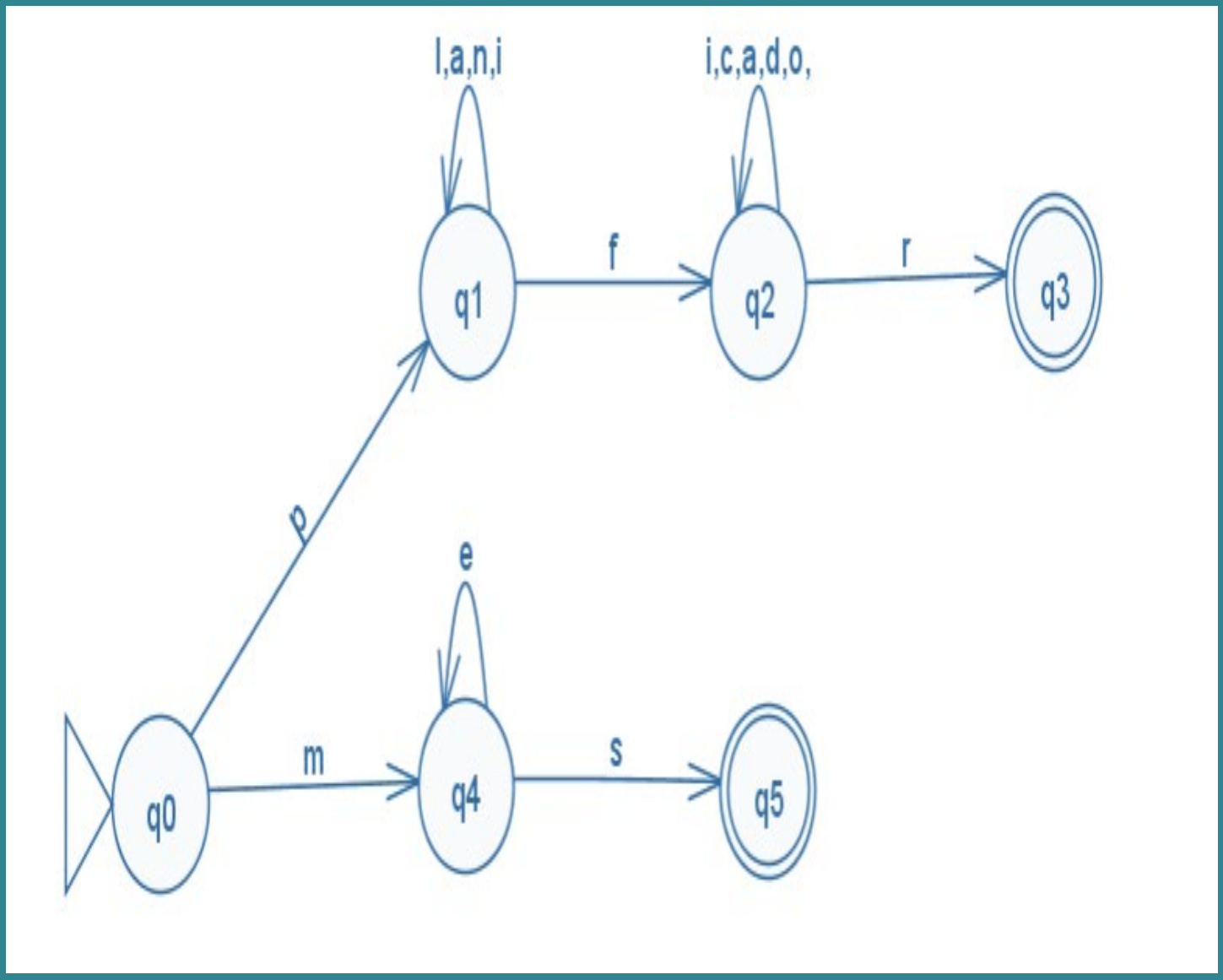
MANUAL TECNICO SISTEMA DE PLANIFICACION

INDICE

ANALIZADOR:

Para la creación de este analizador se cuenta con palabras reservados y signos, a continuación se da un ejemplo de como se planifico la solución a través de un diagrama de transiciones:

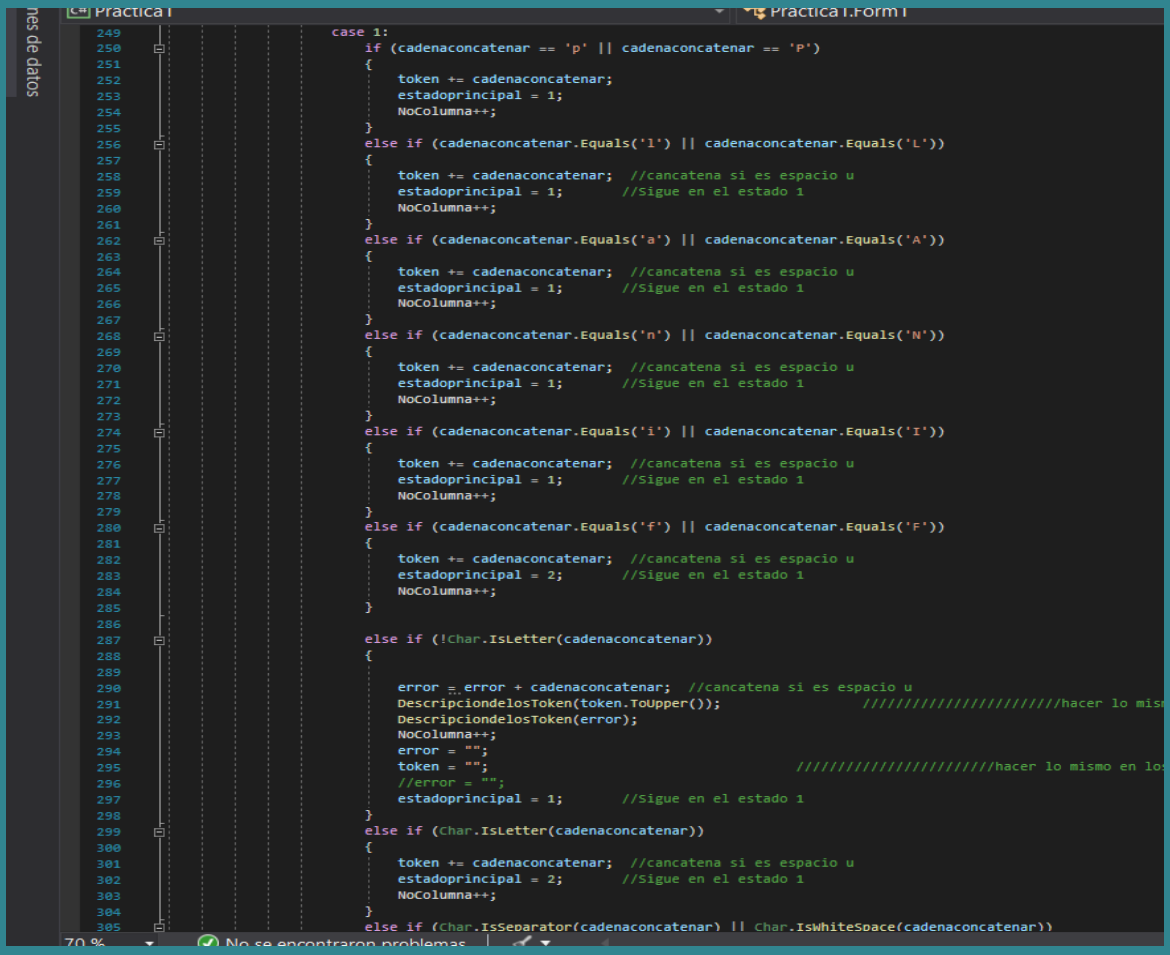


Como se puede observar en el diagrama para pasar del estado q0 al estado q1, se puede únicamente que en la cadena venga una letra p.

Al trabajarlo en nivel de codificación se utilizo un ciclo y dos switch en los cuales se lee letra por letra, haciendo las validaciones respectivas para la aceptación de estados y leer la siguiente letra al completar una palabra reservado, signo o cadena. A continuación, se muestra parte del código a manera de explicar de forma general.

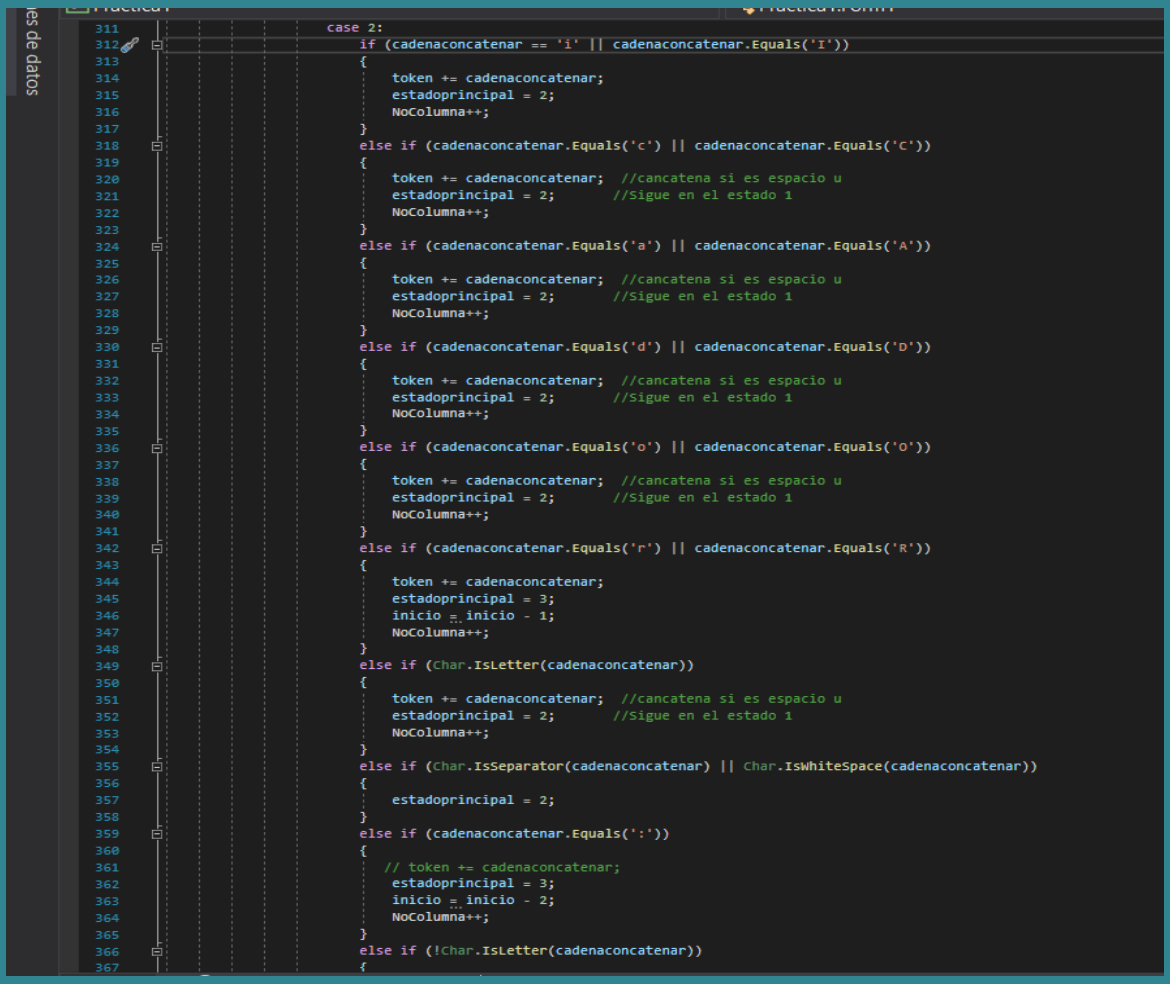
```
1 reference
public void Analizador(string cadena)
{
    int inicio = 0;
    int estadoprincipal = 0;
    char cadenaconcatenar;
    string token = "";
    string error = "";



    for (inicio = 0; inicio < cadena.Length; inicio++)
    {
        cadenaconcatenar = cadena[inicio];
        switch (estadoprincipal)
        {
            case 0:
                switch (cadenaconcatenar)
                {
                    case '\n':
                        NoFila++;                                   ////nuevo
                        NoColumna = 0;                             ////////////nuevo
                        estadoprincipal = 0;
                        break;
                    case ' ':
                        NoColumna++;
                        estadoprincipal = 0;
                        break;
                    case '\t':
                        NoColumna++;
                        estadoprincipal = 0;
                        break;
                    case '\r':
                    case '\b':
                    case '\f':
                        estadoprincipal = 0; //si es espacio o salto de linea o tab sigue en el estado 0
                        break;

                    case 'p':
                        token += cadenaconcatenar;
                        estadoprincipal = 1;
                        break;

                    case 'P':
                        token += cadenaconcatenar;
                        estadoprincipal = 1;
```

Como se puede observar la estructura empieza con un ciclo for, el cual recorrerá, letra por letra la entrada de texto que el usuario seleccione para analizar, luego se tiene un switch, el cual define el estado en que se encuentra esta ira cambiando dependiendo la letra que encuentre, así serán sus validaciones. Por ejemplo, en el caso de la letra p, esa letra la concatena y se va al estado 1, en el cual tenemos lo siguiente:

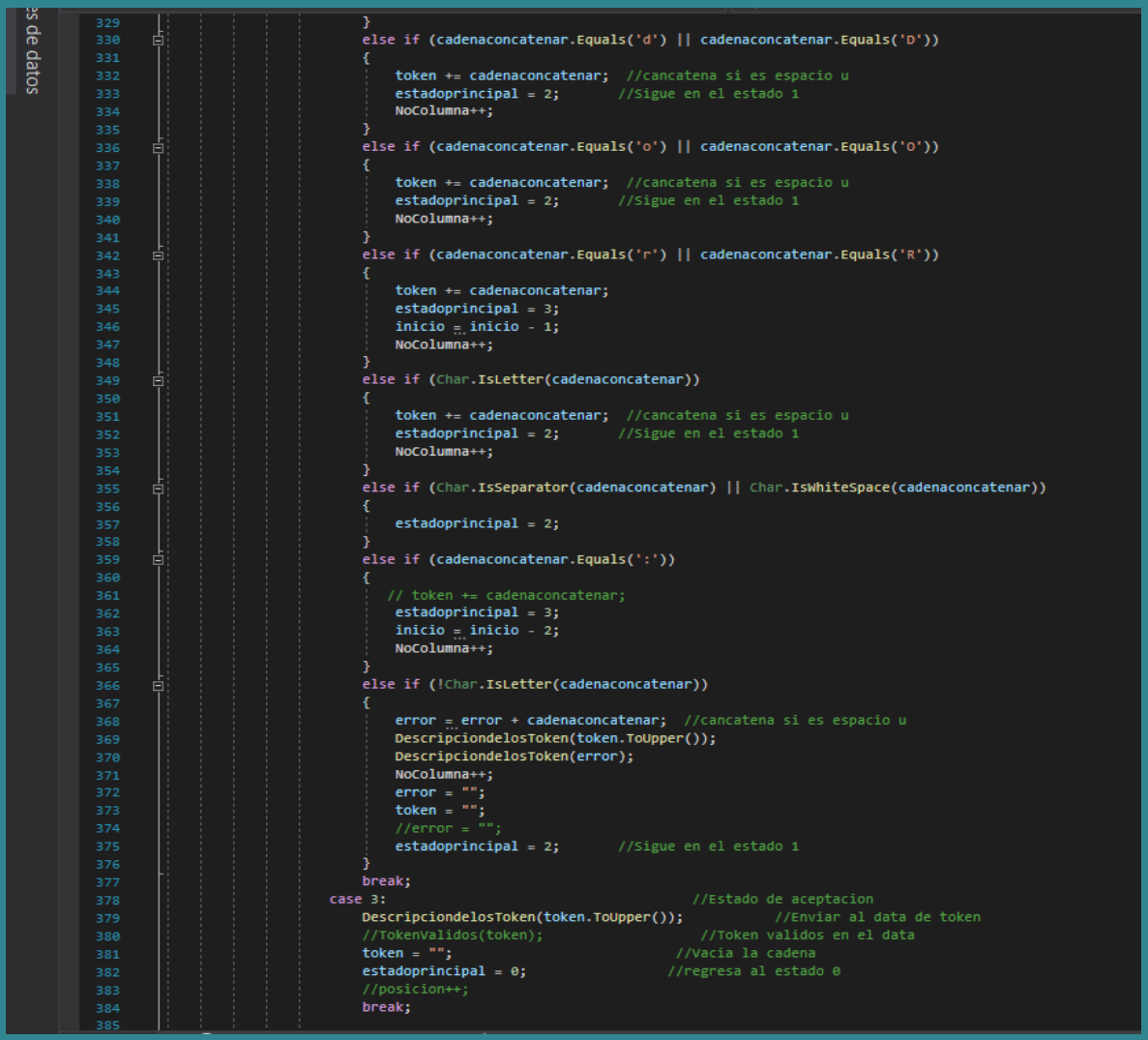


```
249                 case 1:
250                     if (cadenaconcatenar == 'p' || cadenaconcatenar == 'P')
251                     {
252                         token += cadenaconcatenar;
253                         estadoprincipal = 1;
254                         NoColumna++;
255                     }
256                     else if (cadenaconcatenar.Equals('l') || cadenaconcatenar.Equals('L'))
257                     {
258                         token += cadenaconcatenar;  //cancatena si es espacio u
259                         estadoprincipal = 1;       //Sigue en el estado 1
260                         NoColumna++;
261                     }
262                     else if (cadenaconcatenar.Equals('a') || cadenaconcatenar.Equals('A'))
263                     {
264                         token += cadenaconcatenar;  //cancatena si es espacio u
265                         estadoprincipal = 1;       //Sigue en el estado 1
266                         NoColumna++;
267                     }
268                     else if (cadenaconcatenar.Equals('n') || cadenaconcatenar.Equals('N'))
269                     {
270                         token += cadenaconcatenar;  //cancatena si es espacio u
271                         estadoprincipal = 1;       //Sigue en el estado 1
272                         NoColumna++;
273                     }
274                     else if (cadenaconcatenar.Equals('i') || cadenaconcatenar.Equals('I'))
275                     {
276                         token += cadenaconcatenar;  //cancatena si es espacio u
277                         estadoprincipal = 1;       //Sigue en el estado 1
278                         NoColumna++;
279                     }
280                     else if (cadenaconcatenar.Equals('f') || cadenaconcatenar.Equals('F'))
281                     {
282                         token += cadenaconcatenar;  //cancatena si es espacio u
283                         estadoprincipal = 2;       //Sigue en el estado 1
284                         NoColumna++;
285                     }
286
287                     else if (!Char.IsLetter(cadenaconcatenar))
288                     {
289
290                         error = error + cadenaconcatenar;  //cancatena si es espacio u
291                         DescripciondelosToken(token.ToUpper());                ////////////////////////hacer lo mis
292                         DescripciondelosToken(error);
293                         NoColumna++;
294                         error = "";
295                         token = "";                                        ////////////////////////hacer lo mismo en los
296                         //error = "";
297                         estadoprincipal = 1;       //Sigue en el estado 1
298                     }
299                     else if (Char.IsLetter(cadenaconcatenar))
300                     {
301                         token += cadenaconcatenar;  //cancatena si es espacio u
302                         estadoprincipal = 2;       //Sigue en el estado 1
303                         NoColumna++;
304                     }
305                     else if (Char.IsSeparator(cadenaconcatenar) || Char.IsWhiteSpace(cadenaconcatenar))
```

70 % No se encontraron problemas



```
311                 case 2:
312                     if (cadenaconcatenar == 'i' || cadenaconcatenar.Equals('I'))
313                     {
314                         token += cadenaconcatenar;
315                         estadoprincipal = 2;
316                         NoColumna++;
317                     }
318                     else if (cadenaconcatenar.Equals('c') || cadenaconcatenar.Equals('C'))
319                     {
320                         token += cadenaconcatenar;  //cancatena si es espacio u
321                         estadoprincipal = 2;       //Sigue en el estado 1
322                         NoColumna++;
323                     }
324                     else if (cadenaconcatenar.Equals('a') || cadenaconcatenar.Equals('A'))
325                     {
326                         token += cadenaconcatenar;  //cancatena si es espacio u
327                         estadoprincipal = 2;       //Sigue en el estado 1
328                         NoColumna++;
329                     }
330                     else if (cadenaconcatenar.Equals('d') || cadenaconcatenar.Equals('D'))
331                     {
332                         token += cadenaconcatenar;  //cancatena si es espacio u
333                         estadoprincipal = 2;       //Sigue en el estado 1
334                         NoColumna++;
335                     }
336                     else if (cadenaconcatenar.Equals('o') || cadenaconcatenar.Equals('O'))
337                     {
338                         token += cadenaconcatenar;  //cancatena si es espacio u
339                         estadoprincipal = 2;       //Sigue en el estado 1
340                         NoColumna++;
341                     }
342                     else if (cadenaconcatenar.Equals('r') || cadenaconcatenar.Equals('R'))
343                     {
344                         token += cadenaconcatenar;
345                         estadoprincipal = 3;
346                         inicio = inicio - 1;
347                         NoColumna++;
348                     }
349                     else if (Char.IsLetter(cadenaconcatenar))
350                     {
351                         token += cadenaconcatenar;  //cancatena si es espacio u
352                         estadoprincipal = 2;       //Sigue en el estado 1
353                         NoColumna++;
354                     }
355                     else if (Char.IsSeparator(cadenaconcatenar) || Char.IsWhiteSpace(cadenaconcatenar))
356                     {
357                         estadoprincipal = 2;
358                     }
359                     else if (cadenaconcatenar.Equals(':'))
360                     {
361                         // token += cadenaconcatenar;
362                         estadoprincipal = 3;
363                         inicio = inicio - 2;
364                         NoColumna++;
365                     }
366                     else if (!Char.IsLetter(cadenaconcatenar))
367                     {
```



```
                }
                else if (cadenaconcatenar.Equals('d') || cadenaconcatenar.Equals('D'))
                {
                    token += cadenaconcatenar;  //cancatena si es espacio u
                    estadoprincipal = 2;        //Sigue en el estado 1
                    NoColumna++;
                }
                else if (cadenaconcatenar.Equals('o') || cadenaconcatenar.Equals('O'))
                {
                    token += cadenaconcatenar;  //cancatena si es espacio u
                    estadoprincipal = 2;        //Sigue en el estado 1
                    NoColumna++;
                }
                else if (cadenaconcatenar.Equals('r') || cadenaconcatenar.Equals('R'))
                {
                    token += cadenaconcatenar;
                    estadoprincipal = 3;
                    inicio = inicio - 1;
                    NoColumna++;
                }
                else if (Char.IsLetter(cadenaconcatenar))
                {
                    token += cadenaconcatenar;  //cancatena si es espacio u
                    estadoprincipal = 2;        //Sigue en el estado 1
                    NoColumna++;
                }
                else if (Char.IsSeparator(cadenaconcatenar) || Char.IsWhiteSpace(cadenaconcatenar))
                {
                    estadoprincipal = 2;
                }
                else if (cadenaconcatenar.Equals(':'))
                {
                   // token += cadenaconcatenar;
                    estadoprincipal = 3;
                    inicio = inicio - 2;
                    NoColumna++;
                }
                else if (!Char.IsLetter(cadenaconcatenar))
                {
                    error = error + cadenaconcatenar;  //cancatena si es espacio u
                    DescripciondelosToken(token.ToUpper());
                    DescripciondelosToken(error);
                    NoColumna++;
                    error = "";
                    token = "";
                    //error = "";
                    estadoprincipal = 2;        //Sigue en el estado 1
                }
                break;
            case 3:                                     //Estado de aceptacion
                DescripciondelosToken(token.ToUpper());          //Enviar al data de token
                //TokenValidos(token);                  //Token validos en el data
                token = "";                             //Vacia la cadena
                estadoprincipal = 0;                    //regresa al estado 0
                //posicion++;
                break;
```

En el estado 1 tiene diferentes validaciones para concatenar cada letra que se le presente y así pasar al estado dos el cual concatena otro conjunto de letras y pasa al estado 3 con la letra “ r ” o con el signo “ : “, cuando pasa el estado 3, que es un estado de aceptación, este manda la palabra concatenada a un método llamado “ descripción de los token”, este método verifica si la palabra enviada existe en la lista de tokens válidos, sino es el caso entonces manda la palabra a otro método llamado “lista de errores”.

Boton Analizar:

Este boton redirige a varios metodos para poder realizar el analisis lexico requerido por la entrada de texto y asi poder formar las diversas planificaciones que tiene la entrada.

Si la entrada de informacion no contiener errores se genera la planificacion satisfactoriamente, como podemos obseravar en la siguiente imagen la codificacion para realizar la planificacion, se explicara mas adelante la funcionalidad general de cada metodo.

```
        foreach (TextBox t in tab1.SelectedTab.Controls.OfType<TextBox>())
        {
            t.Refresh();
            //PSubnodo(t.Text);
            ReporteLexema ="";
            ReporteError ="";
            Analizador(t.Text);
            GenerarHtml();
            GenerarHtmlErrores();
            fechas = new int[fil, 3];
            descripcion = new string[fil, 5];                                   ////////////////////**********
            if (noerror == 0)
            {
                NombreTreeview(t.Text);
                Asubnodo(t.Text);
                Msubnodo(t.Text);
                Dsubnodo(t.Text);
                Calendario();
                ConcatenarDias();
            }
            else
            {
                MessageBox.Show("No se pudo generar la planificacion, \r\n"+
                    "debido a que hay palabras/signos no reconocidos, \r\n"+
                    "verifique en el reporte de errores, para corregir."
                    , "Informacion Sistema de Planificacion", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }

    }
```

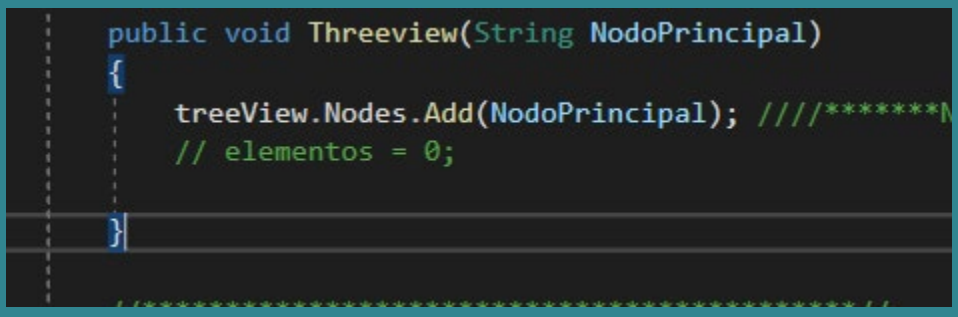
TreeView:

Para poder visualizar la planificación en el treeview se tiene tres métodos, los cuales son:

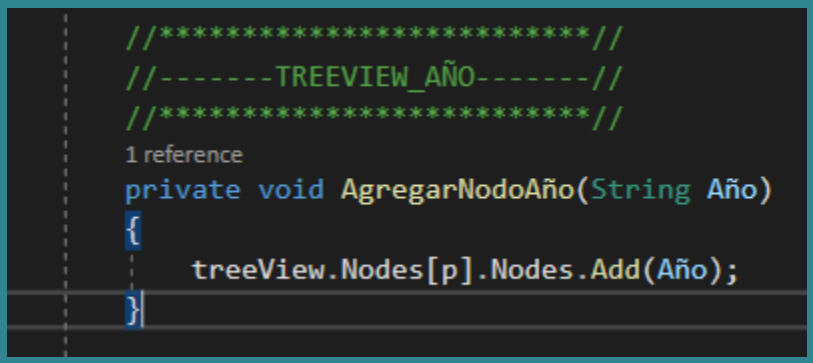
- NombreTreeview.
- Asubnodo.
- Msubnodo.
- Dsubnodo.

El método **"NombreTreeview"** nos sirve para crear el Nodo padre, el cual contendrá el nombre de la planificación y los años, meses y días en los cuales hay determinada actividad.

El método funciona de la siguiente manera, cada vez que en el análisis léxico encuentra la palabra planificador se genera un nuevo nodo padre y se le concatena el nombre de dicha planificación como se puede observar en la siguiente imagen:



El método **"Asubnodo"** nos sirve para agregar el año a la planificación que esta en curso y lo agrega mandando como parámetro el año, como se puede visualizar en la siguiente imagen.



De igual manera es la funcionalidad en los métodos para meses y días, a continuación se presenta una imagen de dichos métodos:

```
//***************************//
//-------TREEVIEW_MES-------//
//***************************//
1 reference
private void AgregarNodoMes(String Mes)
{
    treeView.Nodes[q].Nodes[r].Nodes.Add(Mes);
}
//***************************//
//-------TREEVIEW_DIA-------//
//***************************//
1 reference
private void AgregarNodoDia(String Dia)
{
    treeView.Nodes[s].Nodes[t].Nodes[u].Nodes.Add(Dia);
}
```

Como se puede observar para agregar hijos simplemente se agregan sub-nodos, dependiendo el nivel jerárquico en que los queramos.