

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de La Laguna



**TECNOLÓGICO
NACIONAL DE MÉXICO®**

Ingeniería en Sistemas Computacionales
Taller de Sistemas Operativos

Manual Técnico del Compilador 7Tokens

#21130598 Viramontes Gutiérrez Edmundo

#22130628 Maldonado Paz Neo Brandon

Mictlantecuhtli

#21130596 Hernández Aguilar Diego Jesús

Docente - Pablo Saucedo Martinez

Torreón, Coahuila, México

30 de Mayo del 2024

Control de Versiones

Versión		Fecha		Descripción		Autores	
2.0		Mayo 30 de 2024		Versión Inicial del Documento		<div>- Edmundo Viramontes Gutiérrez</div> <div>- Neo Brandon Maldonado Paz</div> <div>- Diego Jesús Hernández Aguilar</div>	

Índice:

Introducción

Objetivo

Requerimientos de Software

Requerimientos Mínimos de Hardware

Explicacion del codigo

Modificaciones

1. Introducción

En este manual se presentan las modificaciones y avances hechos para el compilador 7Tokens, el compilador utiliza varias clases con el motivo de ser capaz de crear símbolos gramaticales los cuales pueden ser analizados según automatas definidos para un analisis lexico y popular pilas las cuales serán expuestas a producciones gramaticales con el motivo de un analisi sintactico.

2. Objetivo

El objetivo de este manual es el entender y explicar de una manera comprensiva el funcionamiento del compilador 7tokens el cual fue proporcionado por el docente de la materia.

Específicos:

- La agregación al análisis léxico de ser capaz de reconocer operadores dobles (++ , − , **).
- La reparación del análisis sintáctico en función de establecer los mensajes de errores descriptivos y en qué línea se encuentra.
- Explicación del código del programa.

3. Requerimientos de Software

En esta sección se detalla los requisitos mínimos del sistema para poder ejecutar los aplicativos usados para modificar el software compilación de código Java:

Java: 21.0.1; Java HotSpot(TM) 64-Bit Server VM 21.0.1+12-LTS-29

Esta es la versión de Java utilizada para el desarrollo del compilador y por ende la versión recomendada para su utilización.

Apache NetBeans IDE 19

Si el usuario administrador / programador desea realizar alguna modificación sobre la base del código entonces deberá instalar la versión 19 de Apache NetBeans para comenzar con el reconocimiento del código por parte de un IDE

confiable, este IDE puede ser encontrado en el enlace

<https://netbeans.apache.org/front/main/index.html> la cual es la página oficial del IDE de desarrollo.

4. Requerimientos Mínimos de Hardware

De los requisitos de hardware para la ejecución del compilador de software se necesitará de un equipo con al menos estos requisitos mínimos

Como parte de sistema operativo probado se requiere al menos:

- Sistema Operativo: Windows 7, Windows 8.1

Un procesador de al menos:

- Procesador: Intel Core Celeron , Intel Core Duo

Memoria RAM:

- Memoria RAM: 4 GB

Almacenamiento mínimo:

- Disco Duro: 20 Mb

Resolución de pantalla depende mucho de su gusto, pero mínimo correspondiente es:

- Resolución de pantalla: 1280 x 720 píxeles

Periféricos para poder interactuar con el software:

- Periféricos: Teclado, ratón

5. Explicacion del codigo

Clase SimbGram.java

```
public class SimbGram {  
  
    String _elem;  
  
    public SimbGram(String sValor) {  
        _elem = sValor;  
    }  
}
```

```

        public String Elem() {
            return _elem;
        }
    }
}

```

Esta clase es utilizada para poder crear símbolos gramaticales los cuales van a ser analizados posteriormente por las otras clases.

Clase Pila.java

```

public class Pila {

    final int MAX = 5000;
    SimbGram[] _elems;
    int _tope;

    public Pila() {
        _elems = new SimbGram[MAX];
        for (int i = 0; i < _elems.length; i++) {
            _elems[i] = new SimbGram("");
        }
        _tope = 0;
    }

    public boolean Empty() {
        return _tope == 0 ? true : false;
    }

    public boolean Full() {
        return _tope == _elems.length ? true : false;
    }

    public void Push(SimbGram oElem) {
        _elems[_tope++] = oElem;
    }

    public int Length() {
        return _tope;
    }

    public SimbGram Pop() {
        return _elems[--_tope];
    }

    public void Inicia() {
        _tope = 0;
    }
}

```

```
public SimbGram Tope() {
    return _elems[_tope - 1];
}
}
```

Esta clase tiene como función la generación de una pila [], la cual se va a utilizar para poder comprobar la existencia de las producciones de gramática. La clase cuenta con los métodos:

- Empty() = Este método lo que hace es vaciar el tope de la pila.
- Full() = Este método nos indica si la pila se encuentra llena.
- Push(SimbGram oElem) = Este método nos permite insertar a la pila un elemento SimboloGramatical.
- Length() = Este método nos retorna la longitud de la pila.
- Pop() = Este método saca de la pila un elemento.
- Inicia() = Este método establece el tope como 0.

Clase Automata.java

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Automata {

    String textoDel;

    public boolean Reconoce(String texto, int iniToken, int[] i, int noAuto) {
        textoDel = texto;
        switch (noAuto) {
            //----- Automata delin-----
            case 0:
                return ReconoceDelin(texto, i, iniToken);
            //break;
            //----- Automata id-----
            case 1:
                return ReconoceID(texto, i, iniToken);
            //break;
            //----- Automata opRelacional-----
            case 2:
                return ReconoceOpRelacional(texto, i, iniToken);
            //----- Automata OpAsig-----
            case 3:
                return ReconoceOpAsig(texto, i, iniToken);
            //break;
            //----- Automata Incremento-----
            case 4:
                return ReconoceIncDec(texto, i, iniToken);
            //----- Automata oparit-----
            case 5:
                return ReconoceOparit(texto, i, iniToken);
            //break;
            //----- Automata opLogico-----
            case 6:
                return ReconoceOpLogico(texto, i, iniToken);
            //----- Automata num-----
            case 7:
                return ReconoceNum(texto, i, iniToken);
            //break;
            //----- Automata sep-----
            case 8:
                return ReconoceSep(texto, i, iniToken);
            //----- Automata cadena-----
            case 9:
                return ReconoceCad(texto, i, iniToken);
            //break;
            //----- Automata character-----
            case 10:
                return ReconoceCar(texto, i, iniToken);
            //break;
            //----- Automata termInst-----
            case 11:
                return ReconoceTer(texto, i, iniToken);
        }
    }

    return false;
}
```

```
//private boolean RemoveDelim(String text, int[] _a, int initToken) {
//    //Si hay un espacio
//    //Expresión regular superada en la documentación para espacios
//    Pattern pa = Pattern.compile("\\p{Space}*");
//    Matcher ma = pa.matcher(text.toCharArray().toCharArray());
//    String tex = "";
//    //Ceroes hasta que deje de haber espacios
//    while ((ma.matches()) & {
//        try {
//            tex = text.substring(inicio: initToken, ++_i[0]);
//            ma = pa.matcher(tex);
//            //Excepción por si llegamos a la frontera leyendo espacios
//        } catch (Exception e) {
//            ma = pa.matcher(tex + " ");
//        }
//    }
//    //Si ya no sigue y ya no es nula, se que llegamos al final
//    if ((ma.matches()) && !tex.equals("\\p{Space}*)) {
//        _i[0]--;
//        return true;
//    } //Reoperación de error lógico
//    else {
//        _i[0] = initToken;
//        return false;
//    }
//}

private boolean RemoveID(String text, int[] _a, int initToken) {
//    //Expresión regular
//    Pattern pa = Pattern.compile("^([a-zA-z]_|[a-zA-z]{0-9})_*");
//    String tex = "";
//    Matcher ma = pa.matcher(text.toCharArray().toCharArray());
//    //Hacemos coincide aumentamos la longitud de la cadena, coincidencia del
//    //texto cuando ya no coincide, o sea cuando acaba de leer todo el
//    //cadena
//    while ((ma.matches()) & {
//        try {
//            tex = text.substring(inicio: initToken, ++_i[0]);
//            ma = pa.matcher(tex);
//            //Excepción por si llegamos a la frontera
//        } catch (Exception e) {
//            ma = pa.matcher(tex + " ");
//            //System.out.println(e.getMessage());
//        }
//    }
//    //Hacia que ya no lee y si la cadena coincide no sea nula, se que llegamos al final
//    //Si tiene espacio? No estoy seguro
//    if ((ma.matches()) && !tex.equals("\\p{Space}*)) {
//        _i[0]--;
//        return true;
//    } //Reoperación de error lógico
//    else {
//        _i[0] = initToken;
//        return false;
//    }
//}

private boolean RemoveOp(String text, int[] _a, int initToken) {
//    //Expresión regular para los operadores, puede ser = o la combinación de
//    //algos con igual
//    Pattern pa = Pattern.compile("^[=|\\+|-|\\*|\\/|<|>]*");
//    String txt;
//    Matcher ma;
//    //Si la longitud es mayor a dos, incrementare tomar todo el lexema
//    //y compararlo
//    if (text.length() - _i[0] == 2) {
//        txt = text.substring(inicio: initToken, _i[0] + 2);
//        ma = pa.matcher(txt);
//    }
```

[illegible]

```

10 def isalnum(s):
11     """Return True if the string s is alphanumeric.
12     Return False otherwise.
13     """
14     return s.isalnum()
15
16 def isalpha(s):
17     """Return True if the string s is alphabetic.
18     Return False otherwise.
19     """
20     return s.isalpha()
21
22 def isdigit(s):
23     """Return True if the string s is a digit.
24     Return False otherwise.
25     """
26     return s.isdigit()
27
28 def islower(s):
29     """Return True if the string s is all lowercase.
30     Return False otherwise.
31     """
32     return s.islower()
33
34 def isupper(s):
35     """Return True if the string s is all uppercase.
36     Return False otherwise.
37     """
38     return s.isupper()
39
40 def isnumeric(s):
41     """Return True if the string s is numeric.
42     Return False otherwise.
43     """
44     return s.isnumeric()
45
46 def isprintable(s):
47     """Return True if the string s is printable.
48     Return False otherwise.
49     """
50     return s.isprintable()
51
52 def isspace(s):
53     """Return True if the string s is whitespace.
54     Return False otherwise.
55     """
56     return s.isspace()
57
58 def startswith(s, prefix):
59     """Return True if the string s starts with the prefix.
60     Return False otherwise.
61     """
62     return s.startswith(prefix)
63
64 def endswith(s, suffix):
65     """Return True if the string s ends with the suffix.
66     Return False otherwise.
67     """
68     return s.endswith(suffix)
69
70 def find(s, substring):
71     """Return the index of the first occurrence of the substring.
72     Return -1 if the substring is not found.
73     """
74     return s.find(substring)
75
76 def rfind(s, substring):
77     """Return the index of the last occurrence of the substring.
78     Return -1 if the substring is not found.
79     """
80     return s.rfind(substring)
81
82 def index(s, substring):
83     """Return the index of the first occurrence of the substring.
84     Raise ValueError if the substring is not found.
85     """
86     return s.index(substring)
87
88 def rindex(s, substring):
89     """Return the index of the last occurrence of the substring.
90     Raise ValueError if the substring is not found.
91     """
92     return s.rindex(substring)
93
94 def count(s, substring):
95     """Return the number of occurrences of the substring.
96     """
97     return s.count(substring)
98
99 def replace(s, old, new):
100     """Return a new string where the old substring is replaced by the new substring.
101     """
102     return s.replace(old, new)
103
104 def split(s, separator):
105     """Return a list of the words in the string s, separated by the separator.
106     """
107     return s.split(separator)
108
109 def rsplit(s, separator):
110     """Return a list of the words in the string s, separated by the separator.
111     """
112     return s.rsplit(separator)
113
114 def splitlines(s):
115     """Return a list of the lines in the string s, separated by the line separator.
116     """
117     return s.splitlines()
118
119 def join(s, separator):
120     """Return a string where the elements of the list s are joined by the separator.
121     """
122     return separator.join(s)
123
124 def format(s, *args, **kwargs):
125     """Return a string where the placeholders in the string s are replaced by the arguments.
126     """
127     return s.format(*args, **kwargs)
128
129 def fformat(s, *args, **kwargs):
130     """Return a string where the placeholders in the string s are replaced by the arguments.
131     """
132     return s.format(*args, **kwargs)
133
134 def asctime(t):
135     """Return a string representing the time t in a human-readable format.
136     """
137     return t.strftime('%A %B %d %H:%M:%S %Y')
138
139 def localtime(t):
140     """Return a time struct representing the local time t.
141     """
142     return time.localtime(t)
143
144 def gmtime(t):
145     """Return a time struct representing the UTC time t.
146     """
147     return time.gmtime(t)
148
149 def strftime(s, t):
150     """Return a string representing the time t in a human-readable format.
151     """
152     return t.strftime(s)
153
154 def strptime(s, format):
155     """Return a time struct representing the time t in a human-readable format.
156     """
157     return time.strptime(s, format)
158
159 def timegm(t):
160     """Return a time struct representing the UTC time t.
161     """
162     return time.gmtime(t)
163
164 def mktime(t):
165     """Return a time struct representing the local time t.
166     """
167     return time.localtime(t)
168
169 def timegm(t):
170     """Return a time struct representing the UTC time t.
171     """
172     return time.gmtime(t)
173
174 def mktime(t):
175     """Return a time struct representing the local time t.
176     """
177     return time.localtime(t)
178
179 def timegm(t):
180     """Return a time struct representing the UTC time t.
181     """
182     return time.gmtime(t)
183
184 def mktime(t):
185     """Return a time struct representing the local time t.
186     """
187     return time.localtime(t)
188
189 def timegm(t):
190     """Return a time struct representing the UTC time t.
191     """
192     return time.gmtime(t)
193
194 def mktime(t):
195     """Return a time struct representing the local time t.
196     """
197     return time.localtime(t)
198
199 def timegm(t):
200     """Return a time struct representing the UTC time t.
201     """
202     return time.gmtime(t)
203
204 def mktime(t):
205     """Return a time struct representing the local time t.
206     """
207     return time.localtime(t)
208
209 def timegm(t):
210     """Return a time struct representing the UTC time t.
211     """
212     return time.gmtime(t)
213
214 def mktime(t):
215     """Return a time struct representing the local time t.
216     """
217     return time.localtime(t)
218
219 def timegm(t):
220     """Return a time struct representing the UTC time t.
221     """
222     return time.gmtime(t)
223
224 def mktime(t):
225     """Return a time struct representing the local time t.
226     """
227     return time.localtime(t)
228
229 def timegm(t):
230     """Return a time struct representing the UTC time t.
231     """
232     return time.gmtime(t)
233
234 def mktime(t):
235     """Return a time struct representing the local time t.
236     """
237     return time.localtime(t)
238
239 def timegm(t):
240     """Return a time struct representing the UTC time t.
241     """
242     return time.gmtime(t)
243
244 def mktime(t):
245     """Return a time struct representing the local time t.
246     """
247     return time.localtime(t)
248
249 def timegm(t):
250     """Return a time struct representing the UTC time t.
251     """
252     return time.gmtime(t)
253
254 def mktime(t):
255     """Return a time struct representing the local time t.
256     """
257     return time.localtime(t)
258
259 def timegm(t):
260     """Return a time struct representing the UTC time t.
261     """
262     return time.gmtime(t)
263
264 def mktime(t):
265     """Return a time struct representing the local time t.
266     """
267     return time.localtime(t)
268
269 def timegm(t):
270     """Return a time struct representing the UTC time t.
271     """
272     return time.gmtime(t)
273
274 def mktime(t):
275     """Return a time struct representing the local time t.
276     """
277     return time.localtime(t)
278
279 def timegm(t):
280     """Return a time struct representing the UTC time t.
281     """
282     return time.gmtime(t)
283
284 def mktime(t):
285     """Return a time struct representing the local time t.
286     """
287     return time.localtime(t)
288
289 def timegm(t):
290     """Return a time struct representing the UTC time t.
291     """
292     return time.gmtime(t)
293
294 def mktime(t):
295     """Return a time struct representing the local time t.
296     """
297     return time.localtime(t)
298
299 def timegm(t):
300     """Return a time struct representing the UTC time t.
301     """
302     return time.gmtime(t)
303
304 def mktime(t):
305     """Return a time struct representing the local time t.
306     """
307     return time.localtime(t)
308
309 def timegm(t):
310     """Return a time struct representing the UTC time t.
311     """
312     return time.gmtime(t)
313
314 def mktime(t):
315     """Return a time struct representing the local time t.
316     """
317     return time.localtime(t)
318
319 def timegm(t):
320     """Return a time struct representing the UTC time t.
321     """
322     return time.gmtime(t)
323
324 def mktime(t):
325     """Return a time struct representing the local time t.
326     """
327     return time.localtime(t)
328
329 def timegm(t):
330     """Return a time struct representing the UTC time t.
331     """
332     return time.gmtime(t)
333
334 def mktime(t):
335     """Return a time struct representing the local time t.
336     """
337     return time.localtime(t)
338
339 def timegm(t):
340     """Return a time struct representing the UTC time t.
341     """
342     return time.gmtime(t)
343
344 def mktime(t):
345     """Return a time struct representing the local time t.
346     """
347     return time.localtime(t)
348
349 def timegm(t):
350     """Return a time struct representing the UTC time t.
351     """
352     return time.gmtime(t)
353
354 def mktime(t):
355     """Return a time struct representing the local time t.
356     """
357     return time.localtime(t)
358
359 def timegm(t):
360     """Return a time struct representing the UTC time t.
361     """
362     return time.gmtime(t)
363
364 def mktime(t):
365     """Return a time struct representing the local time t.
366     """
367     return time.localtime(t)
368
369 def timegm(t):
370     """Return a time struct representing the UTC time t.
371     """
372     return time.gmtime(t)
373
374 def mktime(t):
375     """Return a time struct representing the local time t.
376     """
377     return time.localtime(t)
378
379 def timegm(t):
380     """Return a time struct representing the UTC time t.
381     """
382     return time.gmtime(t)
383
384 def mktime(t):
385     """Return a time struct representing the local time t.
386     """
387     return time.localtime(t)
388
389 def timegm(t):
390     """Return a time struct representing the UTC time t.
391     """
392     return time.gmtime(t)
393
394 def mktime(t):
395     """Return a time struct representing the local time t.
396     """
397     return time.localtime(t)
398
399 def timegm(t):
400     """Return a time struct representing the UTC time t.
401     """
402     return time.gmtime(t)
403
404 def mktime(t):
405     """Return a time struct representing the local time t.
406     """
407     return time.localtime(t)
408
409 def timegm(t):
410     """Return a time struct representing the UTC time t.
411     """
412     return time.gmtime(t)
413
414 def mktime(t):
415     """Return a time struct representing the local time t.
416     """
417     return time.localtime(t)
418
419 def timegm(t):
420     """Return a time struct representing the UTC time t.
421     """
422     return time.gmtime(t)
423
424 def mktime(t):
425     """Return a time struct representing the local time t.
426     """
427     return time.localtime(t)
428
429 def timegm(t):
430     """Return a time struct representing the UTC time t.
431     """
432     return time.gmtime(t)
433
434 def mktime(t):
435     """Return a time struct representing the local time t.
436     """
437     return time.localtime(t)
438
439 def timegm(t):
440     """Return a time struct representing the UTC time t.
441     """
442     return time.gmtime(t)
443
444 def mktime(t):
445     """Return a time struct representing the local time t.
446     """
447     return time.localtime(t)
448
449 def timegm(t):
450     """Return a time struct representing the UTC time t.
451     """
452     return time.gmtime(t)
453
454 def mktime(t):
455     """Return a time struct representing the local time t.
456     """
457     return time.localtime(t)
458
4
```

[illegible]

```

private boolean ReconocerExponencial(String texto, int[] a, int infToken) {
    Pattern pa = Pattern.compile(regex("(0-9)+"));
    String tex = "";

    boolean no = pa.matcher(texto).substring(infToken, infToken + 1)[0];
    //Se va al comienzo. Esta bandera almacena algunos resultados (según dependa)
    HashMap band = False;
    //Bandera coincide con el patrón, o sea 0-9
    while (pa.matcher(tex)) {
        try {
            //Se va al inicio, zero token
            tex = texto.substring(infToken, infToken + 1)[0];
            no = pa.matcher(tex).text();
            //En caso de que sea en la primera posición el carácter zero
        } catch (Exception e) {
            tex = texto.substring(infToken, infToken + 1);
            no = pa.matcher(tex).text() + "0";
        }
    }

    //En la coincidencia pa, y el texto no es igual a zero, quiere ser que
    //sepa terminado de reconocer, o también que haya chocado con un
    //patron o sea 0 en alguna parte.
    if (pa.matcher(tex).text().equals(infToken)) {
        //En primer punto al final, llamamos al ReconocerExponencial
        if (tex.charAt(tex.length() - 1) == ',') {
            band = ReconocerExponencial(texto, infToken);
        }
        //En bandera se escribe si no genera un punto al inicio. ¿Por qué?
        //Porque puede darse el caso de que sea exponencial.
        while {
            a[0]++;
            //Si no viene punto, se va a ir por ahí y al final va a regresar
            //true, que viene exponencial, para los números enteros (0-9)
            band = True;
        }

        //En también puede darse el caso de que sea un punto al inicio,
        //entonces se va a ReconocerExponencial
    }
    while {
        if (texto.substring(infToken, infToken + 1)[0].equals(infToken)) {
            //En primer punto, reconocemos a zero 0. Aquí almacenamos
            //el reconocimiento en bandera, que regresamos después.
            a[0]++;
            band = ReconocerExponencial(texto, infToken);
        }
        while {
            a[0] = infToken;
            band = False;
        }
    }

    //En hemos hecho aquí al la bandera se escribe. Esta bandera nos informa
    //de que reconocemos reconocemos exponencial.
    if (band) {
        try {
            char g = texto.charAt(a[0]);
            //En el al final hay una E, en cuyo caso se almacenará como float
            //y no como double, aunque todos pertenecen a uno
            if (texto.charAt(a[0]) == 'E') {
                texto.charAt(a[0]) == 'E' {
                    a[0]++;
                    return true;
                }
                ReconocerExponencial(texto, infToken);
            }
        } catch (Exception e) {
        }
    }
}

//Aquí regresamos la bandera con el resultado de ReconocerExponencial que devuelve
//una llamada a ReconocerExponencial o la llamada a ReconocerExponencial.
return band;
}

```



```
private boolean RecorreCadena(String texto, int[] _i, int initToken) {
    //Aquí ya llega siempre a cero, o sea ya tenemos una parte del
    //cadena
    //Regresamos verdad para los que tenemos con un -
    Pattern pa = Pattern.compile("[0-9]+|[1-9]+");
    Matcher ma = ma.matcher(texto.substring(initToken, _i[0]));
    //Mientras lo
    if (ma.matches()) {
        //El resultado con un punto, pero que sea un 00012 (E) 0.0)
        //Pero que termine a 00012
        RecorreCadena(texto, _i, initToken);
    }
    return ma.matches();
}
```

```
private boolean RecorreCadena(String texto, int[] _i, int initToken) {
    Pattern pa = Pattern.compile("[0-9]+|[1-9]+");
    String txt = "";
    Matcher ma = ma();
    try {
        ma = ma.matcher(texto.substring(initToken, _i[0]));
        _i[0] = texto.substring(initToken, _i[0]);
        return false;
    } catch (Exception e) {
        _i[0] = texto.substring(initToken, _i[0]) + 1;
        ma = ma.matcher(txt + "0");
    }
}

while (ma.matches()) {
    try {
        txt = texto.substring(initToken, _i[0]);
        ma = ma.matcher(txt);
        _i[0] = texto.substring(initToken, _i[0]) + 1;
        ma = ma.matcher(txt + "0");
    }
}

if (ma.matches()) {
    _i[0] = texto.substring(initToken, _i[0]);
    return true;
} else {
    _i[0] = initToken;
    return false;
}
```

```
private boolean RecorreCadena(String texto, int[] _i, int initToken) {
    //Aquí ya llega siempre a cero, o sea ya tenemos una parte del
    //cadena
    Pattern pa = Pattern.compile("[0-9]+|[1-9]+");
    String txt = "";
    Matcher ma = ma();
    try {
        //Mientras que la expresión hasta que sea y compare
        ma = ma.matcher(texto.substring(initToken, _i[0]));
        _i[0] = texto.substring(initToken, _i[0]);
        return false;
    } catch (Exception e) {
        //El fallo, no sabemos que tiempo aquí desde ahí, por lo que
        //se termina todo después
        _i[0] = _i[0] + 1;
        _i[0] = 1;
        return false;
    }
}

//Mientras que se ejecutamos, avanzamos el token, aquí ya viene
//Por cuánto después del punto i, ya que pasamos la última posición.
while (ma.matches()) {
    try {
        //Este es más bien para tener número después del punto.
        ma = ma.matcher(texto.substring(initToken, _i[0]));
        _i[0] = texto.substring(initToken, _i[0]);
        _i[0] = texto.substring(initToken, _i[0]) + "0";
    }
}
```

```

}
//En la primera, todo bien.
if (ma.matches()) {
    _i[0] = texto.substring(initToken, _i[0]);
    return true;
} else {
    _i[0] = initToken;
    return false;
}
}
```

//Este método se ejecutará para obtener la i y tener el token completo.
//Se ejecutará que termine de RecorreCadena, así que ya tenemos el token.

```
private void RecorreCadena(String texto, int[] _i, int initToken) {
    //Expresión regular de números, con punto y sin punto
    Pattern pa = "[0-9]+|[0-9]+";
    String txt = "[0-9]+|[0-9]+";
    String ma = "[0-9]+|[0-9]+";
    String _i = "[0-9]+|[0-9]+";
    //Se ejecutará que aquí llega siempre a cero, o sea que tenemos
    //En la i el número
    Pattern pa = Pattern.compile("[0-9]+|[0-9]+");
    Matcher ma = ma();
    try {
        //Aquí ya llega siempre a cero, o sea ya tenemos una parte del
        //cadena
        _i[0] = texto.substring(initToken, _i[0]);
        return true;
    } catch (Exception e) {
        _i[0] = texto.substring(initToken, _i[0]) + 1;
        ma = ma.matcher(txt + "0");
    }
}
```

```

if (ma.matches()) {
    _i[0] = texto.substring(initToken, _i[0]);
    return true;
} else {
    _i[0] = initToken;
    return false;
}
}
```

```

}
//Mientras que la expresión hasta que sea y compare
ma = ma.matcher(texto.substring(initToken, _i[0]));
_i[0] = texto.substring(initToken, _i[0]);
return false;
} catch (Exception e) {
    //El fallo, no sabemos que tiempo aquí desde ahí, por lo que
    //se termina todo después
    _i[0] = _i[0] + 1;
    _i[0] = 1;
    return false;
}
}
```

```
private boolean RecorreCadena(String texto, int[] _i, int initToken) {
    //Expresión regular de números, con punto y sin punto
    Pattern pa = Pattern.compile("[0-9]+|[0-9]+");
    String txt = "[0-9]+|[0-9]+";
    String ma = "[0-9]+|[0-9]+";
    String _i = "[0-9]+|[0-9]+";
    //Se ejecutará que aquí llega siempre a cero, o sea que tenemos
    //En la i el número
    Pattern pa = Pattern.compile("[0-9]+|[0-9]+");
    Matcher ma = ma();
    try {
        //Aquí ya llega siempre a cero, o sea ya tenemos una parte del
        //cadena
        _i[0] = texto.substring(initToken, _i[0]);
        return true;
    } catch (Exception e) {
        _i[0] = texto.substring(initToken, _i[0]) + 1;
        ma = ma.matcher(txt + "0");
    }
}
```

```
private boolean RecorreCadena(String texto, int[] _i, int initToken) {
    //Expresión regular de números, con punto y sin punto
    Pattern pa = Pattern.compile("[0-9]+|[0-9]+");
    String txt = "[0-9]+|[0-9]+";
    String ma = "[0-9]+|[0-9]+";
    String _i = "[0-9]+|[0-9]+";
    //Se ejecutará que aquí llega siempre a cero, o sea que tenemos
    //En la i el número
    Pattern pa = Pattern.compile("[0-9]+|[0-9]+");
    Matcher ma = ma();
    try {
        //Aquí ya llega siempre a cero, o sea ya tenemos una parte del
        //cadena
        _i[0] = texto.substring(initToken, _i[0]);
        return true;
    } catch (Exception e) {
        _i[0] = texto.substring(initToken, _i[0]) + 1;
        ma = ma.matcher(txt + "0");
    }
}

//Mientras que la expresión hasta que sea y compare
ma = ma.matcher(texto.substring(initToken, _i[0]));
_i[0] = texto.substring(initToken, _i[0]);
return false;
} catch (Exception e) {
    //El fallo, no sabemos que tiempo aquí desde ahí, por lo que
    //se termina todo después
    _i[0] = _i[0] + 1;
    _i[0] = 1;
    return false;
}
}
```

```
private boolean RecorreCadena(String texto, int[] _i, int initToken) {
    //Expresión regular de números, con punto y sin punto
    Pattern pa = Pattern.compile("[0-9]+|[0-9]+");
    String txt = "[0-9]+|[0-9]+";
    String ma = "[0-9]+|[0-9]+";
    String _i = "[0-9]+|[0-9]+";
    //Se ejecutará que aquí llega siempre a cero, o sea que tenemos
    //En la i el número
    Pattern pa = Pattern.compile("[0-9]+|[0-9]+");
    Matcher ma = ma();
    try {
        //Aquí ya llega siempre a cero, o sea ya tenemos una parte del
        //cadena
        _i[0] = texto.substring(initToken, _i[0]);
        return true;
    } catch (Exception e) {
        _i[0] = texto.substring(initToken, _i[0]) + 1;
        ma = ma.matcher(txt + "0");
    }
}

//Mientras que la expresión hasta que sea y compare
ma = ma.matcher(texto.substring(initToken, _i[0]));
_i[0] = texto.substring(initToken, _i[0]);
return false;
} catch (Exception e) {
    //El fallo, no sabemos que tiempo aquí desde ahí, por lo que
    //se termina todo después
    _i[0] = _i[0] + 1;
    _i[0] = 1;
    return false;
}
}
```

```
private boolean RecorreCadena(String texto, int[] _i, int initToken) {
    //Expresión regular de números, con punto y sin punto
    Pattern pa = Pattern.compile("[0-9]+|[0-9]+");
    String txt = "[0-9]+|[0-9]+";
    String ma = "[0-9]+|[0-9]+";
    String _i = "[0-9]+|[0-9]+";
    //Se ejecutará que aquí llega siempre a cero, o sea que tenemos
    //En la i el número
    Pattern pa = Pattern.compile("[0-9]+|[0-9]+");
    Matcher ma = ma();
    try {
        //Aquí ya llega siempre a cero, o sea ya tenemos una parte del
        //cadena
        _i[0] = texto.substring(initToken, _i[0]);
        return true;
    } catch (Exception e) {
        _i[0] = texto.substring(initToken, _i[0]) + 1;
        ma = ma.matcher(txt + "0");
    }
}
```

Esta clase tiene la función de identificar el tipo de autónoma que es el texto, existen bastantes métodos en este clase (ver imágenes) pero estos se manejan de una manera muy similar, mediante el uso de java.util.regex.Matcher y java.util.regex.Pattern establecemos parámetros para poder identificar patrones específicos con el fin de identificar el tipo, los métodos identifican doce tipos de autómatas: Espacios, Identificador, Operador Relacional, Operador Asignacion, Operadores Dobles, Operador Aritmetico, Operador Logico, Numero, Separador, Cadena, Caracter y Punto y Coma.

Clase Lexico.java

```
public class Lexico {

    //Número máximo de autómatas
    final int TOTALES = 12;
    final int MAXTOKENS = 500;
    String[] _lexemas;
    String[] _tokens;
    String _lexema;
    int _noTokens;
    int[] _i = {0};
    int _iniToken;
    Automata _aUTO;

    //Revista un arreglo de palabras reservadas y si la encuentra retorna false,
    //si no, retorna verdadero, ya que si no se
    private boolean ResId() {
        String[] palabras = {"abstract", "assert", "boolean", "break", "byte", "case", "catch", "class", "continue", "default", "do", "double", "else", "enum", "extends", "final", "finally", "float", "for", "if", "implements", "import", "int", "interface", "long", "native", "new", "package", "private", "protected", "public", "return", "short", "static", "super", "switch", "synchronized", "this", "throw", "throws", "transient", "try", "void", "while"};
        for (int i = 0; i < palabras.length; i++) {
            if (_lexema.equals(palabras[i])) {
                return false;
            }
        }
        return true;
    }

    public int NoTokens() {
        return _noTokens;
    }

    public String[] Tokens() {
        return _tokens;
    }

    public String[] Lexemas() {
        return _lexemas;
    }

    public Lexico() // constructor por defecto
    {
        _lexemas = new String[MAXTOKENS];
        _tokens = new String[MAXTOKENS];
        _aUTO = new Automata();
        _i[0] = 0;
        _iniToken = 0;
        _noTokens = 0;
    }

    //Toma todo en cuenta
    public void Inicia() {
        _i[0] = 0;
        _iniToken = 0;
        _noTokens = 0;
    }

    public boolean Analiza(String texto) {
        boolean reconocido;
        int noAuto;

        while (_i[0] < texto.length()) {
            reconocido = false;
            noAuto = 0;

            // noAuto inicializa en cero, el ciclo for lo que hace es: si noAuto
            for (; noAuto < TOTALES && !reconocido;) {
                //Revisa si se reconoce el token utilizando metodo Reconoce()
                if (noAuto.Reconoce(texto, _iniToken, _i[0], noAuto)) {
                    reconocido = true;
                } else {
                    // Si el if no reconoce el token con ese noAuto, el noAuto sum
                    noAuto++;
                }
            }

            //Si lo reconoció, tomamos el número del autómata, con el motivo de
            if (reconocido) {
                _lexema = texto.substring(_iniToken, _i[0]);
                switch (noAuto) {

                    case 0:
                        _tokens[_noTokens] = "Espacio";
                        break;

                    //..... Automata id.....
                    //Si es ID, guarda el token como id, si no como palabra
                    case 1:
                        if (ResId()) {
                            _tokens[_noTokens] = "Identificador";
                        } else {
                            _tokens[_noTokens] = "Palabra Reservada";
                        }
                        break;

                    case 2:
                        _tokens[_noTokens] = "Operador Relacional";
                        break;

                    case 3:
                        _tokens[_noTokens] = "Operador Asignacion";
                        break;

                    case 4:
                        _tokens[_noTokens] = "Operadores Dobles";
                        break;

                    case 5:
                        _tokens[_noTokens] = "Operador Aritmetico";
                        break;

                    case 6:
                        _tokens[_noTokens] = "Operador Logico";
                        break;

                    case 7:
                        _tokens[_noTokens] = "Numero";
                        break;

                    case 8:
                        _tokens[_noTokens] = "Separador";
                        break;

                    case 9:
                        _tokens[_noTokens] = "Cadena";
                        break;

                    case 10:
                        _tokens[_noTokens] = "Caracter";
                        break;

                    case 11:
                        _tokens[_noTokens] = "Punto y Coma";
                        break;
                }

                if (noAuto != 0) {
                    _lexemas[_noTokens++] = _lexema;
                } else {
                    return false; //La parte de recuperación del error léxico se reemplaza por
                } //este return false para indicar que existe un error léxico

                //Movemos iniToken, estamos listos para leer el siguiente token
                _iniToken = _i[0];
            }
        }
        return true; //El análisis léxico ha sido exitoso cuando acaba el while
    }
}
```

Clase SintDescNRP.java

Esta clase tiene como objetivo identificar error en la sintaxis según la gramática establecida, siendo los errores que es capaz de analizar:

- Error (1) hace referencia a que no se reconoce el token en las variables terminales establecidas.
- Error (2) hace referencia a que no es capaz de asignar una producción,

AnaLex7TokensApp.frame

PROGRAMA FUENTE

PAREJAS TOKENS-LEXEMAS

TOKENS	LEXEMAS
--------	---------

Análisis Lexico

Errores Encontrados

Error en la línea

Resultado del Análisis

PROGRAMA FUENTE

3232

PAREJAS TOKENS-LEXEMAS

TOKENS	LEXEMAS
Numero	3232

Análisis Lexico

Errors found: 2

Error en la línea 1: No se encontró ';'.

Syntax Error (2), No se encontro una produccion adecuada

Frame en el cual nos permite insertar secuencias de texto y nos muestra su análisis léxico en el cual nos identifica la pareja token-lexema, además de si nuestro texto tiene algún error de sintaxis o léxico.

5. Modificaciones:

Cambios realizados por #21130598

AnaLex7TokensApp.java:

1. los imports:

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.HashMap;
import java.util.Map;
```

//Declaración

```
private Map<String, Number> numericVariables;
```

//Inicialización

```
numericVariables = new HashMap<>();
```

Con el motivo de mandarlos llamar en los nuevos métodos que agregue

2. modificaciones a métodos ya establecidos:

```
en: private void btnAnaSinActionPerformed(java.awt.event.ActionEvent evt)
```

```
findVariables(lineas[i]);
```

```
    jLabel_FormGen.setText("Variables numericas y sus valores: " +
    numericVariables);
```

Con el motivo de buscar las variables numéricas (int, double, etc, checar el matcher del método findVariables)

3. nuevos métodos:

Estos buttons utilizan métodos nuevos (Ver más adelante que establecen estos nuevos métodos)

```
private void formGeneral_btnActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String[] lineas = txtProgFuente.getText().split("\n");
    for (int i = 0; i < lineas.length; i++) {
        try {
            evaluateFormulaGeneral(lineas[i]);
        }
    }
}
```

```
    } catch (Exception ex) {
```

```
        Logger.getLogger(AnaLex7TokensApp.class.getName()).log(Level.SEVERE, null,
        ex);
    }
}

}
```

Este btn le das el click al botón y busca en las líneas del JTextArea "formulaGeneral" mediante el método evaluateFormulaGeneral().

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    jLabel_FormGen.setText("");
    DresultadosFG_lbl.setText("");
    resultadosFG_lbl1.setText("");
    numericVariables.clear();
    errorFG_lbl.setText("");
}
```

Este btn limpia parámetros y labels

```
private void jButton_EJEMPLOBIENActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    txtaProgFuente.setText("int a = 2;\nint b = 3;\nint c = 1;\nformulaGeneral(a, b,
c)");
}
```

Pobla el JTextField con un ejemplo de fórmula general correcto.

```
private void jButton_EJEMPLOMALActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    txtaProgFuente.setText("int a = 2;\nint b = 3;\nint c = 99;\nformulaGeneral(a, b,
c)");
}
```

Pobla el JTextField con un ejemplo de fórmula general incorrecto.

```

private void findVariables(String text) {
    //PATRON DE IDENTIFICACION DE VARIABLES

    String variablePattern =
"\b(int|double|float|long|short)\s+(\w+)\s*=\s*([-+]?\d*\.\d+);";
    Pattern pattern = Pattern.compile(variablePattern);
    Matcher matcher = pattern.matcher(text);

    while (matcher.find()) {
        String type = matcher.group(1);
        String name = matcher.group(2);
        String value = matcher.group(3);

        switch (type) {
            case "int":
                //LO AGREGAMOS AL AL HASHMAP SI ES NUMERICO PA
                numericVariables.put(name, Integer.parseInt(value));
                break;
            case "double":
                numericVariables.put(name, Double.parseDouble(value));
                break;
            case "float":
                numericVariables.put(name, Float.parseFloat(value));
                break;
            case "long":
                numericVariables.put(name, Long.parseLong(value));
                break;
            case "short":
                numericVariables.put(name, Short.parseShort(value));
                break;
            default:
                break;
        }
    }
}

```

Este metodo busca el patron (variables numericas declaradas e inicializadas de manera correcta) dentro del JTextArea y si matchea las clasifica y las agrega a nuestro HashMap(numericVariables) con a su tipo y valor.

```

private void evaluateFormulaGeneral(String expression) throws Exception {
    // Busca formulaGeneral() y saca el valor de las variables

```

```

Pattern pattern =
Pattern.compile("formulaGeneral\\((\\w+),\\s*(\\w+),\\s*(\\w+)\\)");
Matcher matcher = pattern.matcher(expression);
if (matcher.find()) {
    String aVar = matcher.group(1);
    String bVar = matcher.group(2);
    String cVar = matcher.group(3);

    // Get the values of the variables
    double a = numericVariables.get(aVar).doubleValue();
    double b = numericVariables.get(bVar).doubleValue();
    double c = numericVariables.get(cVar).doubleValue();

    // Evaluate the quadratic formula
    double temp1 = Math.pow(b, 2); // b^2
    double temp2 = 4 * a;          // 4a
    double temp3 = temp2 * c;      // 4ac
    double temp4 = temp1 - temp3; // b^2 - 4ac

    double d = temp4;
    DresultadosFG_lbl.setText("<html>Valor de d: <b>" + d + "</b></html>");
    //System.out.println("Value of d: " + d);

    if (d < 0) {
        errorFG_lbl.setText("<html>La solucion no es real. <b>Numeros
imaginarios</b></html>");
        System.out.println("NO, FALLO!!! CHeCA el plantamiento del ejercicio
manco");
    } else {
        double temp5 = -b;
        double temp6 = 2 * a;
        double temp7 = Math.sqrt(d);
        double temp8 = temp5 + temp7;
        double temp9 = temp5 - temp7; //alterno
        double temp10 = temp8 / temp6;
        double temp11 = temp9 / temp6; //alterno

        double x1 = (-b + Math.sqrt(d)) / (2 * a);
        double x2 = (-b - Math.sqrt(d)) / (2 * a);

        numericVariables.put("x1", x1);
        numericVariables.put("x2", x2);

        resultadosFG_lbl1.setText("<html>x1: <b>" + x1 + "</b> x2: <b>" + x2
+ "</b></html>");
        temps.setText("<html>temp1: "+temp1+ " -> <em>b^2</em><br>" +
            "temp2: "+temp2+ " -> <em>4a</em><br>" +

```



```

        "temp3: "+temp3 + " -> <em>4ac</em><br>" +
        "temp4: "+temp4 + " -> <em>b^2 - 4ac</em><br>" +
        "temp5: "+temp5 + " -> <em>- b</em><br>" +
        "temp6: "+temp6 + " -> <em>2 * a</em><br>" +
        "temp7: "+temp7 + " -> <em>sqrt(b^2 - 4ac)</em><br>" +
        "temp8: "+temp8 + " -> <em>- b <b>+</b> sqrt(b^2 -
4ac)</em><br>" +
        "temp9: "+temp9 + " -> <em>- b <b>-</b> sqrt(b^2 -
4ac)</em><br>" +
        "temp10: "+temp10 + " -> <em><b>+</b> /(2a)</em><br>" +
        "temp11: "+temp11 + " -> <em><b>-</b> /(2a)</em></html>");
//System.out.println("x1: " + x1);
//System.out.println("x2: " + x2);
    }
    } else {
        errorFG_lbl.setText("No se encontro error en el calculo");
    }
}

```

Este método al igual que el anterior busca dentro del JTextField un patrón (formulaGeneral(a, b, c)) y al encontrarlo se declaran las variables aVar, bVar, cVar con el motivo de poblarlas con las variables ya establecidas dentro de nuestro HashMap (numericVariables), se calcula los temporales y si d es menor que 0 esto significa que la solución NO es real, es decir sería imaginaria, en cambio si d es mayor que 0 se calcula los temporales de x1 y x2.

```

// ----- Prueba variable numerica =
letra-----
    String ultimaLinea = lineas[lineas.length - 1];

    String regex = "^(int|double|short|float|long)\\b.*";

    int prueba = 0;

    for (int io = 0; io < lineas.length; io++) {
        String linea = lineas[i].trim();
        boolean apertura2 = linea.matches(regex);

        if (apertura2) {
            prueba = 1;
        } else {
            prueba = 2;
        }

        // Para probar, podemos imprimir el valor de 'prueba' y la línea
        correspondiente
        System.out.println("Linea: " + linea + " -> Prueba: " + prueba);
    }
}

```

```

    }

    if (ultimaLinea.length() >= 2) {
        char penultimoCaracter = ultimaLinea.charAt(ultimaLinea.length() - 2);

        // Verificar si el penúltimo carácter es una letra o un dígito
        if (Character.isLetter(penultimoCaracter) && prueba == 1) {
            System.out.println("El penúltimo carácter es una letra.");
            lblResult1.setText("Error en la línea " + (i + 1) + ": No se puede asignar una
letra a una variable numerica");
        } else if (Character.isDigit(penultimoCaracter)) {
            System.out.println("El penúltimo carácter es un dígito.");
        }
    }
}

```

```

//-----
-----

```

Este cambio detecta un error si se le asigna a una variable numérica
(int,double,short,float,long) una letra.

PROGRAMA FUENTE

PAREJAS TOKENS-LEXEMAS

TOKENS	LEXEMAS

Análisis Lexico

Errores Encontrados

Error en la línea

Resultado del Análisis

Resultados:

Error Fomula General:

Fomula General

Limpiar

Ejemplo con plantamiento correcto

Ejemplo con plantamiento incorrecto

Temporales:

Se le dio una actualización al UI para un mayor tamaño y se le agregó el display de temporales.

6. Bibliografía

NetBeans IDE. (n.d.). Oracle. Retrieved Mayo 1, 2024, from

<https://www.oracle.com/mx/tools/technologies/netbeans-ide.html>

¿Qué es Java y por qué lo necesito? (n.d.). Java. Retrieved Mayo 1, 2024, from

https://www.java.com/es/download/help/whatis_java.html