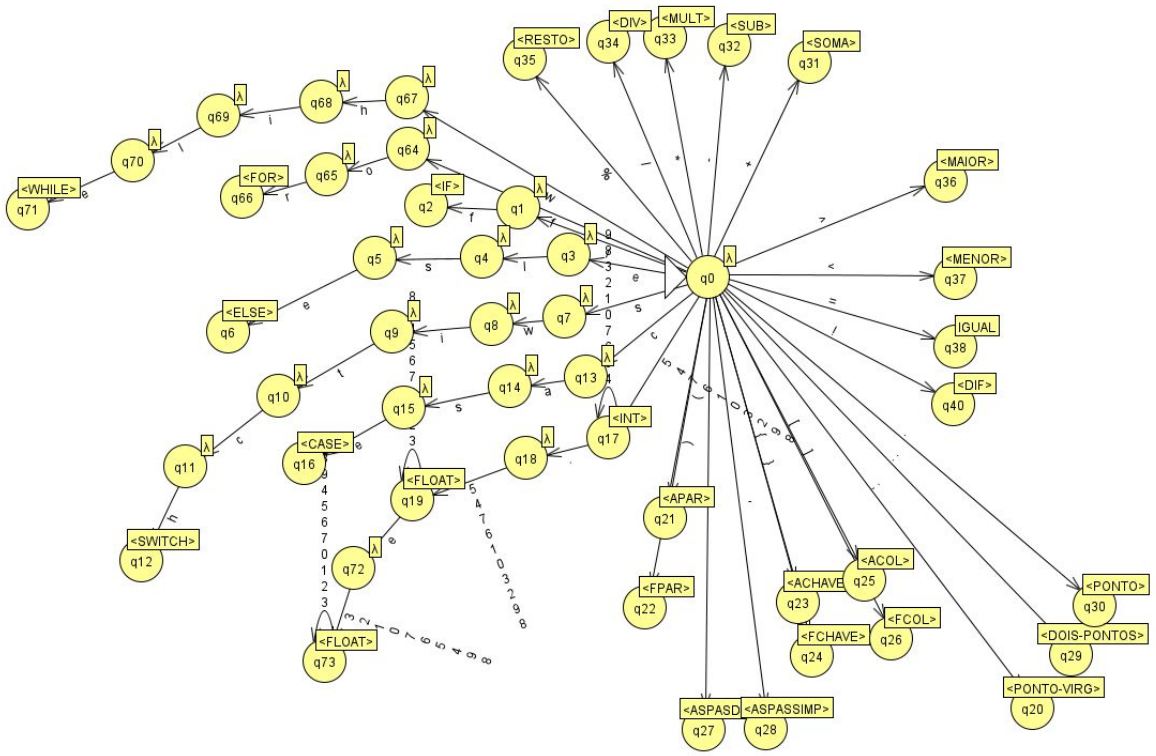
Exemplo de um Analisador Léxico construído através da Análise Preditiva

Expressões regulares para os tokens:

Token	ER	Exemplos de lexemas
<IF>	(i)(f)	if
<ELSE>	(e)(l)(s)(e)	else
<SWITCH>	(s)(w)(i)(t)(c)(h)	switch
<CASE>	(c)(a)(s)(e)	case
<WHILE>	(w)(h)(i)(l)(e)	while
<FOR>	(f)(o)(r)	for
<INT>	(0|...|9)+	0, 25, 5555, 64
<FLOAT>	.|e	. , e
<APAR>	(	(
<FPAR>	)	)
<ASPASDUP>	"	"
<ASPASSIMP>	‘	‘
<ACHAVE>	{	{
<FCHAVE>	}	}
<ACOL>	[	[
<FCOL>	]	]
<PONTO-VIRG>	;	;
<DOIS-PONTOS>	:	:
<PONTO>	.	.
<SOMA>	+	+
<SUB>	-	-
<MULT>	*	*
<DIV>	/	/
<RESTO>	&	&
<DIF>	!	!
<IGUAL>		
<MAIOR>	>	>
<MENOR>	<	<
<VAZIO>	\n | \t	
<EOF>	Representação do fim	

Máquina de Moore:



Teste 1:

```
if(1+1 == 2)
```

Console de saída:

<IF><APAR><INT><SOMA><INT><IGUAL><INT><FPAR>

Teste 2:

```
while(1*2 != 3)
```

Console de saída:

<WHILE><APAR><INT><MULT><INT><DIF><IGUAL><INT><FPAR>

Teste 3:

```
if(1+1 < 4){
}

while(1+1 >= 2){
}
```

Console de saída:

<IF><APAR><INT><SOMA><INT><MENOR><INT><FPAR><ACHAVE><FCHAVE><WHIL
E><APAR><INT><SOMA><INT><MAIOR><IGUAL><INT><FPAR><ACHAVE><FCHAVE>

Teste 4:

```
1$1
```

Console de saída:

<INT>
Erro léxico: caractere encontrado:$
era(m) esperados(s):
{'IF': ['i', 'f', '<IF>'], 'ELSE': ['e', 'l', 's', 'e', '<ELSE>'], 'SWITCH': ['s', 'w', 'i', 't', 'c', 'h', '<SWITCH>'], 'CASE': ['c', 'a', 's', 'e', 'CASE'], 'WHILE': ['w', 'h', 'i', 'l', 'e', '<WHILE>'], 'FOR': ['f', 'o', 'r', '<FOR>'], 'INT': ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '<INT>'], 'FLOAT': ['.', 'e', '<FLOAT>'], 'APAR': ['(', '<APAR>'], 'FPAR': [')', '<FPAR>'], 'ASPASDUP': ['"', '<ASPASDUP>'], 'ASPASSIMP': ["'", '<ASPASSIMP>'], 'ACHAVE': ['{', '<ACHAVE>'], 'FCHAVE': ['}', '<FCHAVE>'], 'ACOL': ['[', '<ACOL>'], 'FCOL': [']', '<FCOL>'], 'PONTO-VIRG': [';', '<PONTO-VIRG>'], 'DOIS-PONTOS': [':', '<DOIS-PONTOS>'], 'PONTO': ['.', '<PONTO>'], 'SOMA': ['+', '<SOMA>'], 'SUB': ['-', '<SUB>'], 'MULT': ['*', '<MULT>'], 'DIV': ['/', '<DIV>'], 'RESTO': ['%', '<RESTO>'], 'DIF': ['!', '<DIF>'], 'IGUAL': ['=', '<IGUAL>'], 'MAIOR': '>', 'MENOR': '<', 'VAZIO': [' ', '\n', '\t'], 'EOF': ''}
<INT>

Teste 5:

```
else_
```

Console de saída:

<ELSE>
Erro léxico: caractere encontrado:_
era(m) esperados(s):
{'IF': ['i', 'f', '<IF>'], 'ELSE': ['e', 'l', 's', 'e', '<ELSE>'], 'SWITCH': ['s', 'w', 'i', 't', 'c', 'h', '<SWITCH>'], 'CASE': ['c', 'a', 's', 'e', 'CASE'], 'WHILE': ['w', 'h', 'i', 'l', 'e', '<WHILE>'], 'FOR': ['f', 'o', 'r', '<FOR>'], 'INT': ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '<INT>'], 'FLOAT': ['.', 'e', '<FLOAT>'], 'APAR': ['(', '<APAR>'], 'FPAR': [')', '<FPAR>'], 'ASPASDUP': ['"', '<ASPASDUP>'], 'ASPASSIMP': ["'", '<ASPASSIMP>'], 'ACHAVE': ['{', '<ACHAVE>'], 'FCHAVE': ['}', '<FCHAVE>'], 'ACOL': ['[', '<ACOL>'], 'FCOL': [']', '<FCOL>'], 'PONTO-VIRG': [';', '<PONTO-VIRG>'], 'DOIS-PONTOS': [':', '<DOIS-PONTOS>'], 'PONTO': ['.', '<PONTO>'], 'SOMA': ['+', '<SOMA>'], 'SUB': ['-', '<SUB>'], 'MULT': ['*', '<MULT>'], 'DIV': ['/', '<DIV>'], 'RESTO': ['%', '<RESTO>'], 'DIF': ['!', '<DIF>'], 'IGUAL': ['=', '<IGUAL>'], 'MAIOR': '>', 'MENOR': '<', 'VAZIO': [' ', '\n', '\t'], 'EOF': ''}

Implementação do analisador léxico por intermédio de análise preditiva:

```
8   import sys
9   from abc import ABC, abstractmethod
10  class Analisador(ABC):
11
12      def __init__(self):
13
14          self.NOME_DEFAULT_ARQUIVO_ENTRADA = 'entrada.txt'
15          self.tokens = {
16
17
18              'IF': ['i','f', '<IF>'],
19              'ELSE': ['e','l','s','e', '<ELSE>'],
20              'SWITCH': ['s','w','i','t','c','h', '<SWITCH>'],
21              'CASE': ['c','a','s','e', 'CASE'],
22              'WHILE': ['w','h','i','l','e', '<WHILE>'],
23              'FOR': ['f','o','r', '<FOR>'],
24              'INT': ['0','1','2','3','4','5','6','7','8','9','<INT>'],
25              'FLOAT': ['.','e', '<FLOAT>'],
26              'APAR': ['(', '<APAR>'],
27              'FPAR': [')','<FPAR>'],
28              'ASPASDUP': ['"', '<ASPASDUP>'],
29              'ASPASSIMP': ["'", '<ASPASSIMP>'],
30              'ACHAVE': ['{', '<ACHAVE>'],
31              'FCHAVE': ['}', '<FCHAVE>'],
32              'ACOL': ['[', '<ACOL>'],
33              'FCOL': [']', '<FCOL>'],
34              'PONTO-VIRG': [';','<PONTO-VIRG>'],
35              'DOIS-PONTOS': [':', '<DOIS-PONTOS>'],
36              'PONTO': ['.', '<PONTO>'],
37              'SOMA': ['+','<SOMA>' ],
38              'SUB': ['-', '<SUB>'],
39              'MULT': ['*', '<MULT>'],
40              'DIV': ['/', '<DIV>'],
41              'RESTO': ['%', '<RESTO>'],
42              'DIF': ['!', '<DIF>'],
43              'IGUAL': ['=', '<IGUAL>'],
44              'MAIOR': '>',
45              'MENOR': '<',
46              'VAZIO': [' ', '\n', '\t'],
47              'EOF': ''
48          }
49
50      @abstractmethod
51      def Analisador(self, _nomeArquivoEntrada):
52          self.nomeArquivoEntrada = self._nomeArquivoEntrada
53
54      def Analisador(self):
55          self.nomeArquivoEntrada = self.NOME_DEFAULT_ARQUIVO_ENTRADA
56
57  class AnalisadorLexico(Analisador):
58
59      def __init__(self,_nomeArquivoEntrada):
60          super().__init__()
61          self.proxCaractere = ''
62          self.linha = 1
63          self.posicao = -1
64          self.tokenReconhecido = ''
65          self.s = []
```

```
66          try:
67              self.file = open(_nomeArquivoEntrada, "r")
68              self.entrada = self.file.read()
69              self.file.close()
70              self.leProxCaractere()
71
72          except:
73              print("Erro na leitura do arquivo" + _nomeArquivoEntrada)
74
75
76      def leProxCaractere(self):
77
78          try:
79
80              self.posicao += 1
81              self.proxCaractere = self.entrada[self.posicao]
82
83
84          except IndexError:
85
86              self.proxCaractere ='<EXIT>'
87
88      def proxCaractereIs(self, s):
89
90          if self.proxCaractere in s:
91              return True
92          else:
93              return False
94
95  class MyAnalisadorLexico(AnalisadorLexico):
96
97      def __init__(self,_nomeArquivoEntrada):
98           super().__init__(_nomeArquivoEntrada)
99
100     def s0(self):
101
102         if(self.proxCaractereIs(self.tokens['INT'])):
103
104             self.leProxCaractere()
105             self.s1()
106
107         elif(self.proxCaractereIs(self.tokens['IF'][0])):
108             self.leProxCaractere()
109             self.s2()
110
111         elif(self.proxCaractereIs(self.tokens['ELSE'][0])):
112
113             self.leProxCaractere()
114             self.s3()
115
116         elif(self.proxCaractereIs(self.tokens['WHILE'][0])):
117
118
119             self.leProxCaractere()
120             self.s4()
121
122         elif(self.proxCaractereIs(self.tokens['SWITCH'][0])):
123             self.leProxCaractere()
```

```
124             self.s5()
125
126         elif(self.proxCaractereIs(self.tokens['CASE'][0])):
127             self.leProxCaractere()
128             self.s6()
129
130         elif(self.proxCaractereIs(self.tokens['FLOAT'])):
131             self.leProxCaractere()
132             self.s7()
133
134         elif(self.proxCaractereIs(self.tokens['FOR'][0])):
135             self.leProxCaractere()
136             self.s8()
137
138         elif(self.proxCaractereIs(self.tokens['APAR'])):
139             self.leProxCaractere()
140             self.s9()
141
142         elif(self.proxCaractereIs(self.tokens['FPAR'])):
143             self.leProxCaractere()
144             self.s10()
145
146         elif(self.proxCaractereIs(self.tokens['ASPASDUP'])):
147             self.leProxCaractere()
148             self.s11()
149
150         elif(self.proxCaractereIs(self.tokens['ASPASSIMP'])):
151             self.leProxCaractere()
152             self.s12()
153
154         elif(self.proxCaractereIs(self.tokens['ACHAVE'])):
155             self.leProxCaractere()
156             self.s13()
157
158         elif(self.proxCaractereIs(self.tokens['FCHAVE'])):
159             self.leProxCaractere()
160             self.s14()
161
162         elif(self.proxCaractereIs(self.tokens['ACOL'])):
163             self.leProxCaractere()
164             self.s15()
165
166         elif(self.proxCaractereIs(self.tokens['FCOL'])):
167             self.leProxCaractere()
168             self.s16()
169
170         elif(self.proxCaractereIs(self.tokens['PONTO-VIRG'])):
171             self.leProxCaractere()
172             self.s17()
173
174         elif(self.proxCaractereIs(self.tokens['DOIS-PONTOS'])):
175             self.leProxCaractere()
176             self.s18()
177
178         elif(self.proxCaractereIs(self.tokens['PONTO'])):
179             self.leProxCaractere()
180             self.s19()
181
```

```
182         elif(self.proxCaractereIs(self.tokens['SOMA'])):
183             self.leProxCaractere()
184             self.s20()
185 
186         elif(self.proxCaractereIs(self.tokens['SUB'])):
187             self.leProxCaractere()
188             self.s21()
189 
190         elif(self.proxCaractereIs(self.tokens['MULT'])):
191             self.leProxCaractere()
192             self.s22()
193 
194         elif(self.proxCaractereIs(self.tokens['DIV'])):
195             self.leProxCaractere()
196             self.s23()
197 
198         elif(self.proxCaractereIs(self.tokens['RESTO'])):
199             self.leProxCaractere()
200             self.s24()
201 
202         elif(self.proxCaractereIs(self.tokens['DIF'])):
203             self.leProxCaractere()
204             self.s25()
205 
206         elif(self.proxCaractereIs(self.tokens['IGUAL'])):
207             self.leProxCaractere()
208             self.s26()
209 
210         elif(self.proxCaractereIs(self.tokens['MAIOR'])):
211             self.leProxCaractere()
212             self.s27()
213 
214         elif(self.proxCaractereIs(self.tokens['MENOR'])):
215             self.leProxCaractere()
216             self.s28()
217 
218         elif(self.proxCaractereIs(self.tokens['VAZIO'])):
219             self.leProxCaractere()
220             self.s0()
221 
222         else:
223 
224             if(self.proxCaractere == '<EXIT>'):
225                sys.exit()
226 
227             print('\nErro léxico: caractere encontrado:'+self.proxCaractere)
228             print('era(m) esperados(s):')
229             print(self.tokens)
230 
231             self.leProxCaractere()
232 
233             self.s0()
234 
235     def s1(self):
236 
237         self.tokenReconhecido = '<INT>'
238 
239         if(self.proxCaractereIs(self.tokens['INT'])):
```

```
240                 self.leProxCaractere()
241                 self.s1()
242 
243         def s2(self):
244 
245             self.tokenReconhecido = '<IF>'
246 
247             if(self.proxCaractereIs(self.tokens['IF'])):
248 
249                 self.leProxCaractere()
250                 self.s2()
251 
252         def s3(self):
253 
254             self.tokenReconhecido = '<ELSE>'
255 
256             if(self.proxCaractereIs(self.tokens['ELSE'])):
257 
258                 self.leProxCaractere()
259                 self.s3()
260 
261         def s4(self):
262 
263             self.tokenReconhecido = '<WHILE>'
264 
265             if(self.proxCaractereIs(self.tokens['WHILE'])):
266 
267                 self.leProxCaractere()
268                 self.s4()
269 
270         def s5(self):
271 
272             self.tokenReconhecido = '<SWITCH>'
273 
274             if(self.proxCaractereIs(self.tokens['SWITCH'])):
275 
276                 self.leProxCaractere()
277                 self.s5()
278 
279         def s6(self):
280 
281             self.tokenReconhecido = '<CASE>'
282 
283             if(self.proxCaractereIs(self.tokens['CASE'])):
284 
285                 self.leProxCaractere()
286                 self.s6()
287 
288         def s7(self):
289 
290             self.tokenReconhecido = '<FLOAT>'
291 
292             if(self.proxCaractereIs(self.tokens['FLOAT'])):
293 
294                 self.leProxCaractere()
295                 self.s7()
296 
297         def s8(self):
```

```
299             self.tokenReconhecido = '<FOR>'
300
301             if(self.proxCaractereIs(self.tokens['FOR'])):
302
303                 self.leProxCaractere()
304                 self.s8()
305
306         def s9(self):
307
308             self.tokenReconhecido = '<APAR>'
309
310             if(self.proxCaractereIs(self.tokens['APAR'])):
311
312                 self.leProxCaractere()
313                 self.s9()
314
315         def s10(self):
316
317             self.tokenReconhecido = '<FPAR>'
318
319             if(self.proxCaractereIs(self.tokens['FPAR'])):
320
321                 self.leProxCaractere()
322                 self.s10()
323
324         def s11(self):
325
326             self.tokenReconhecido = '<ASPASDUP>'
327
328             if(self.proxCaractereIs(self.tokens['ASPASDUP'])):
329
330                 self.leProxCaractere()
331                 self.s11()
332
333         def s12(self):
334
335             self.tokenReconhecido = '<ASPASSIMP>'
336
337             if(self.proxCaractereIs(self.tokens['ASPASSIMP'])):
338
339                 self.leProxCaractere()
340                 self.s12()
341
342         def s13(self):
343
344             self.tokenReconhecido = '<ACHAVE>'
345
346             if(self.proxCaractereIs(self.tokens['ACHAVE'])):
347
348                 self.leProxCaractere()
349                 self.s13()
350
351         def s14(self):
352
353             self.tokenReconhecido = '<FCHAVE>'
354
355             if(self.proxCaractereIs(self.tokens['FCHAVE'])):
356
```

```
357             self.leProxCaractere()
358             self.s14()
359
360     def s15(self):
361
362         self.tokenReconhecido = '<ACOL>'
363
364         if(self.proxCaractereIs(self.tokens['ACOL'])):
365
366             self.leProxCaractere()
367             self.s15()
368
369     def s16(self):
370
371         self.tokenReconhecido = '<FCOL>'
372
373         if(self.proxCaractereIs(self.tokens['FCOL'])):
374
375             self.leProxCaractere()
376             self.s16()
377
378     def s17(self):
379
380         self.tokenReconhecido = '<PONTO-VIRG>'
381
382         if(self.proxCaractereIs(self.tokens['PONTO-VIRG'])):
383
384             self.leProxCaractere()
385             self.s17()
386
387     def s18(self):
388
389         self.tokenReconhecido = '<DOIS-PONTOS>'
390
391         if(self.proxCaractereIs(self.tokens['DOIS-PONTOS'])):
392
393             self.leProxCaractere()
394             self.s18()
395
396     def s19(self):
397
398         self.tokenReconhecido = '<PONTO>'
399
400         if(self.proxCaractereIs(self.tokens['PONTO'])):
401
402             self.leProxCaractere()
403             self.s19()
404
405     def s20(self):
406
407         self.tokenReconhecido = '<SOMA>'
408
409         if(self.proxCaractereIs(self.tokens['SOMA'])):
410             self.leProxCaractere()
411             self.s20()
412
413     def s21(self):
414
```

```
415             self.tokenReconhecido = '<SUB>'
416
417             if(self.proxCaractereIs(self.tokens['SUB'])):
418                 self.leProxCaractere()
419                 self.s21()
420
421         def s22(self):
422
423             self.tokenReconhecido = '<MULT>'
424
425             if(self.proxCaractereIs(self.tokens['MULT'])):
426                 self.leProxCaractere()
427                 self.s22()
428
429         def s23(self):
430
431             self.tokenReconhecido = '<DIV>'
432
433             if(self.proxCaractereIs(self.tokens['DIV'])):
434                 self.leProxCaractere()
435                 self.s23()
436
437         def s24(self):
438
439             self.tokenReconhecido = '<RESTO>'
440
441             if(self.proxCaractereIs(self.tokens['RESTO'])):
442                 self.leProxCaractere()
443                 self.s24()
444
445         def s25(self):
446
447             self.tokenReconhecido = '<DIF>'
448
449             if(self.proxCaractereIs(self.tokens['DIF'])):
450                 self.leProxCaractere()
451                 self.s25()
452
453         def s26(self):
454
455             self.tokenReconhecido = '<IGUAL>'
456
457             if(self.proxCaractereIs(self.tokens['IGUAL'])):
458                 self.leProxCaractere()
459                 self.s26()
460
461         def s27(self):
462
463             self.tokenReconhecido = '<MAIOR>'
464
465             if(self.proxCaractereIs(self.tokens['MAIOR'])):
466                 self.leProxCaractere()
467                 self.s27()
468
469
470         def s28(self):
471
472             self.tokenReconhecido = '<MENOR>'
```

```
474            if(self.proxCaractereIs(self.tokens['MENOR'])):
475                self.leProxCaractere()
476                self.s28()
477
478
479
480
481
482
483
484
485    #Main
486    A = MyAnalisadorLexico('file.txt')
487
488    while True:
489      A.s0()
490      print(A.tokenReconhecido, end='')
491
492      if A.proxCaractere =='<EXIT>':
493        break
494
495
496
497
498
499
500
501
502
503
504    #file.close()
505
```

Exemplo de Analisador Sintático construído através de Análise Preditiva

Gramática Livre de contexto

```
 1 S -> inicio .
 2 inicio -> corpo .
 3 corpo -> cmdIf | cmdWhile | cmdFor | cmdSwitch | cmdSwitchCase .
 4 exp -> <NUM> expB .
 5 expB -> <SOMA> <NUM> | <SUB> <NUM> | <MULT> <NUM> | <DIV> <NUM> .
 6 cmdIf -> <IF> <APAR> exp <FPAR> <ACHA> exp <FCHA> .
 7 cmdWhile -> <WHILE> <APAR> exp <FPAR> <ACHA>  exp <FCHA> .
 8 cmdFor -> <FOR> <APAR> exp <FPAR> <ACHA>  exp <FCHA> .
 9 cmdSwitch -> <SWITCH> <APAR> exp <FPAR> <ACHA> listCmdSwitchCase <FCHA> .
10 listCmdSwitchCase -> cmdSwitchCase listCmdSwitchCase .
11 cmdSwitchCase -> <CASE> <DOIS-PONTOS> exp .
12 <SOMA> -> + .
13 <SUB> -> - .
14 <MULT> -> * .
15 <DIV> -> / .
16 <NUM> -> 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 .
```

Teste do analisador sintático:

Teste 1:

```
if(1+1 == 2)
{
}
```

Console de saída:
Análise realizada com sucesso no arquivo file.txt

Teste 2:

```
while(1*2 != 3)
{
}
```

Console de saída:
Análise realizada com sucesso no arquivo file.txt

Teste 3:

```
if*1+1 < 4)
{
}

while(1+1 >= 2)
{
}
```

Console de saída:
Erro sintático: token encontrado: <MULT>
era(m) esperados(s): <APAR>

Teste 4:

```
else=
```

Console de saída:
Erro sintático: token encontrado: <IGUAL>
era(m) esperados(s): <ACHAR>

Implementação por intermédio da análise preditiva:

É importante salientar, que o exemplo abaixo é um complemento da implementação acima, ou seja, para executar o código abaixo, você só precisa juntá-lo ao código acima.

```
class AnalisadorSintatico(Analisador):

    def __init__(self, _nomeArquivoEntrada):
        self.t = []
        self.A = MyAnalisadorLexico(_nomeArquivoEntrada)
        super().__init__()
        self.leProxToken()

    def leProxToken(self):
        self.A.s0()

    def reconhece(self, t):

        if  self.A.tokenReconhecido in t:
            self.leProxToken()
        else:
            print('\nErro Sintático: token encontrado:' +  self.A.tokenReconhecido)
            print('era(m) esperados(s):' , end = '')
            print(t)

    def proxTokenIs(self, t):

        if self.A.tokenReconhecido in t:
            return True
        else:
            return False

class MyAnalisadorSintatico(AnalisadorSintatico):
    def __init__(self,_nomeArquivoEntrada):

         self.A = AnalisadorSintatico(_nomeArquivoEntrada)

         super().__init__(_nomeArquivoEntrada)
    def inicio(self):
        self.corpo()

    def corpo(self):

        if (self.proxTokenIs(self.tokens['IF'])):
            self.leProxToken()
            self.cmdIf()
            self.corpo()
        elif (self.proxTokenIs(self.tokens['WHILE'])):
            self.leProxToken()
            self.cmdWhile()
            self.corpo()
        elif (self.proxTokenIs(self.tokens['FOR'])):
            self.leProxToken()
            self.cmdFor()
            self.corpo()
        elif (self.proxTokenIs(self.tokens['SWITCH'])):
            self.leProxToken()
            self.cmdSwitch()
            self.corpo()
        elif (self.proxTokenIs(self.A.tokens['EOF'])):
            print('\nAnalise Sintática: Concluída')
        else:
            print('\nErro Sintático: token encontrado:'+  self.A.tokenReconhecido)
```

```
546
547
548
549     def exp(self):
550
551         if (self.proxTokenIs(self.tokens['INT'])):
552             self.leProxToken()
553             if (self.proxTokenIs(self.tokens['SOMA'])):
554                 self.leProxToken()
555                 self.exp()
556             elif (self.proxTokenIs(self.tokens['SUB'])):
557                 self.leProxToken()
558                 self.exp()
559             elif (self.proxTokenIs(self.tokens['MULT'])):
560                 self.leProxToken()
561                 self.exp()
562             elif (self.proxTokenIs(self.tokens['DIV'])):
563                 self.leProxToken()
564                 self.exp()
565             elif (self.proxTokenIs(self.tokens['RESTO'])):
566                 self.leProxToken()
567                 self.exp()
568             elif (self.proxTokenIs(self.tokens['DIF'])):
569                 self.leProxToken()
570                 self.exp()
571             elif (self.proxTokenIs(self.tokens['IGUAL'])):
572                 self.leProxToken()
573                 self.exp()
574             elif (self.proxTokenIs(self.tokens['MAIOR'])):
575                 self.leProxToken()
576                 self.exp()
577             elif (self.proxTokenIs(self.tokens['MENOR'])):
578                 self.leProxToken()
579                 self.exp()
580         elif (self.proxTokenIs(self.tokens['INT'])):
581             self.leProxToken()
582         elif (self.proxTokenIs(self.tokens['VAZIO'])):
583             self.leProxToken()
584
585         else:
586             print('\nErro Sintático: token encontrado:'+  self.A.tokenReconhecido)
587             print('era(m) esperados(s):', end = '')
588             print( self.t)
589
590     def bloco(self):
591         self.exp()
592
593     def cmdIf(self):
594         self.reconhece(self.tokens['APAR'])
595         self.exp()
596         self.reconhece(self.tokens['FPAR'])
597         self.reconhece(self.tokens['ACHAVE'])
598         self.bloco()
599         self.reconhece(self.tokens['FCHAVE'])
600
601     def cmdWhile(self):
602         self.reconhece(self.tokens['APAR'])
603         self.exp()
```

```
594             self.reconhece(self.tokens['APAR'])
595             self.exp()
596             self.reconhece(self.tokens['FPAR'])
597             self.reconhece(self.tokens['ACHAVE'])
598             self.bloco()
599             self.reconhece(self.tokens['FCHAVE'])
600
601         def cmdWhile(self):
602             self.reconhece(self.tokens['APAR'])
603             self.exp()
604             self.reconhece(self.tokens['FPAR'])
605             self.reconhece(self.tokens['ACHAVE'])
606             self.bloco()
607             self.reconhece(self.tokens['FCHAVE'])
608
609         def cmdFor(self):
610             self.reconhece(self.tokens['APAR'])
611             self.exp()
612             self.reconhece(self.tokens['FPAR'])
613             self.reconhece(self.tokens['ACHAVE'])
614             self.bloco()
615             self.reconhece(self.tokens['FCHAVE'])
616
617         def cmdSwitch(self):
618             self.reconhece(self.tokens['APAR'])
619             self.exp()
620             self.reconhece(self.tokens['FPAR'])
621             self.reconhece(self.tokens['ACHAVE'])
622             self.listCmdSwitchCase()
623             self.reconhece(self.tokens['FCHAVE'])
624
625         def listCmdSwitchCase(self):
626             self.cmdSwitchCase()
627             self.listCmdSwitchCase()
628
629         def cmdSwitchCase(self):
630             self.reconhece(self.tokens['CASE'])
631             self.reconhece(self.tokens['INT'])
632             self.reconhece(self.tokens['DOIS-PONTOS'])
633             self.bloco()
634
635
636     #Main
637     #A = MyAnalisadorLexico('file.txt')
638
639     #while True:
640     #     A.s0()
641     #     print(A.tokenReconhecido, end='')
642
643     #     if A.proxCaractere == '<EXIT>':
644     #         break
645
646
647
648     tst = MyAnalisadorSintatico('file.txt')
649     tst.inicio()
650     print('Analise conclída com sucesso')
```