Compiladores

Analisador léxico

Eric Emanuel
Vinícius Costa

Expressões regulares

Números

dígito: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Os números em Lua Simplificada são inteiros e seus literais são escritos em notação decimal, ou seja:

número: dígito+

Identificadores

letra: [a-z A-Z]

Identificadores em Lua Simplificada podem ser qualquer sequência de letras, dígitos e sublinhados, não começando com um dígito, ou seja:

identificador: [_ letra][_ letra dígito]*

Operadores

operador: + | - | * | / | ^ | = | ~= | <= | >= | < | > | == | (|) | { | } | [|] | ; | : | , | . | ..

Strings

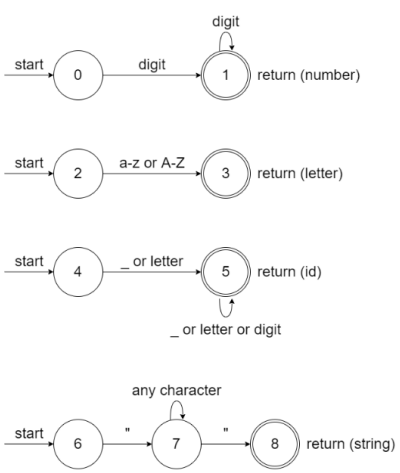
espaçamento: [\a | \b | \f | \n | \r | \t | \v | \\ | \”]

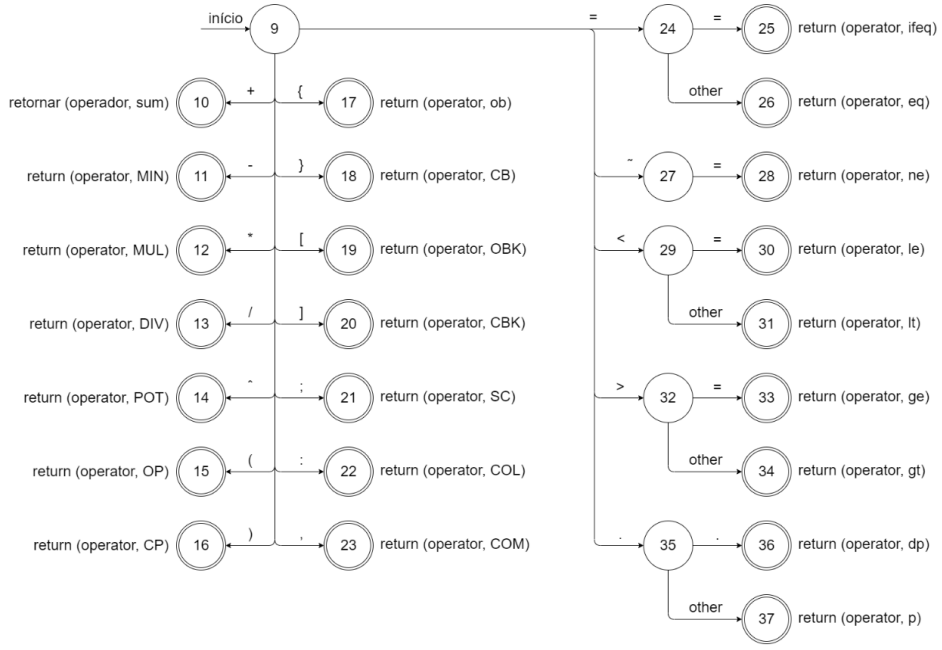
Os literais de string são delimitados por aspas duplas e podem conter sequências de espaçamentos, ou seja:

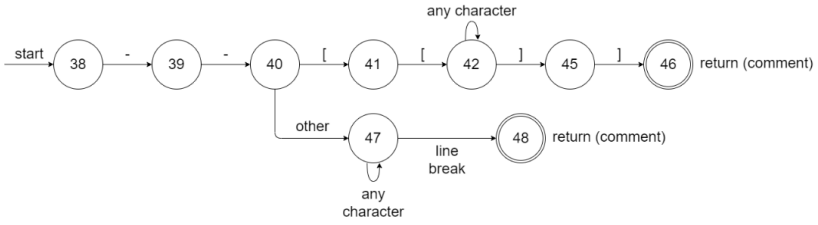
string: “ [.]* ”

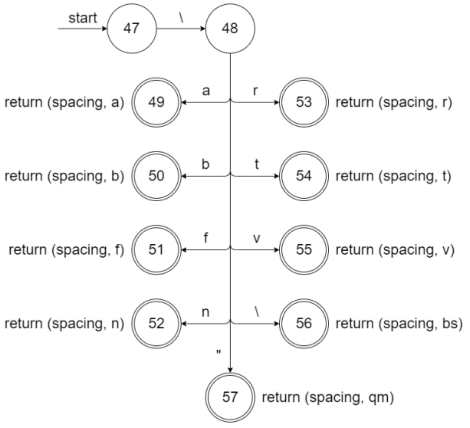
comentário: -- ([.]* \n | [[[.]* --]])

Autômatos


digit
start
0
digit
1
return (number)
start
2
a-z or A-Z
3
return (letter)
start
4
_ or letter
5
return (id)
_ or letter or digit
any character
start
6
"
7
"
8
return (string)


início
9
retornar (operador, sum)
10
+
{
17
return (operator, ob)
return (operator, MIN)
11
-
}
18
return (operator, CB)
return (operator, MUL)
12
*
[
19
return (operator, OBK)
return (operator, DIV)
13
/
]
20
return (operator, CBK)
return (operator, POT)
14
^
;
21
return (operator, SC)
return (operator, OP)
15
(
:
22
return (operator, COL)
return (operator, CP)
16
)
,
23
return (operator, COM)
=
24
=
25
return (operator, ifeq)
other
26
return (operator, eq)
~
27
=
28
return (operator, ne)
<
29
=
30
return (operator, le)
other
31
return (operator, lt)
>
32
=
33
return (operator, ge)
other
34
return (operator, gt)
.
35
.
36
return (operator, dp)
other
37
return (operator, p)


any character
start
38
-
39
-
40
[
41
[
42
]
45
]
46
return (comment)
other
47
line
break
48
return (comment)
any
character


start
47
\
48
return (spacing, a)
49
a
r
53
return (spacing, r)
return (spacing, b)
50
b
t
54
return (spacing, t)
return (spacing, f)
51
f
v
55
return (spacing, v)
return (spacing, n)
52
n
\
56
return (spacing, bs)
"
57
return (spacing, qm)

Funcionamento

O Programa começa com a função main chamando a função readfile, que lê o arquivo onde se encontra o código e irá destrinchar esse conteúdo para que possa ser lido pelo analisador léxico.

O próximo passo na função main é, por meio da condição de parada “@”, usar um loop para chamar função nextToken, que é responsável por ler o conteúdo salvo, relacionar cada termo desse conteúdo a uma “classe” e a um atributo e receber como resultado o token no formato <classe, Atributo>, que foi definido em uma struct.

Como cada classe tem uma característica específica que as define, usamos dessas características para identificá-las por meio das seguintes funções:

IsDigit: função booleana que retorna True se identificar um número e False caso não identifique.
IsLetter: função booleana que retorna True se identificar uma letra e False caso não identifique.

Dentro da função nextToken foi definido um loop que irá perdurar até o fim do conteúdo salvo. Dentro desse loop, fazemos o tratamento dos símbolos lidos por meio de um switch. Dentro desse switch foram estabelecidos os cases necessários para cada situação, sendo:

Case 0: identificação do próximo estado
Case 1: retorno de um número
Case 5: retorno de um id
Case 7: retorno de uma string
Cases 10 ao 37: retorno de operadores
Case 999: indica o fim do conteúdo lido.

Vale ressaltar que esse código lê caractere a caractere o conteúdo salvo. Para os casos que contém padrões com mais de um caractere, foi implementado um esquema de concatenação de caracteres para se ter o termo desejado, por exemplo: W + H + I + L + E = WHILE.

Como o exemplo acima indica, foram definidas também palavras reservadas e atributos com números para que a Struct Token pudesse retornar o número da linha da tabela referente ao termo desejado, podendo futuramente passar essa informação ao analisador sintático, que é a próxima etapa desse trabalho.

Exemplo

Entrada:

```
check = “lexico”;
cont= 20;
while! (cont <30){
    ?cont = cont +1;
}
@
```

Saída:

```
<check,>
<operator, EQ>
<Literal, "lexico">
<operator, SC>
<cont,>
<operator, EQ>
<number, 20>
<operator, SC>
<while,>
[error: unrecognized pattern]
<operator, OP>
<cont,>
<operator, LT>
<number, 30>
<operator, CP>
<operator, OB>
[error: unrecognized pattern]
<cont,>
<operator, EQ>
<cont,>
<operator, SUM>
<number, 1>
<operator, SC>
<operator, CB>
```