RELATÓRIO DE COMPILADORES

Professor: Ricardo Terra Nunes Bueno Villela
Alunos: Felipe Crisóstomo Silva Oliveira
Henrique Assis Moreira
Julia Aparecida de Faria Morais

SUMÁRIO

1. INTRODUÇÃO

O trabalho prático da disciplina de **Compiladores (GCC 130)** do ***2° Semestre de 2025***, tem como objetivo apresentar o desenvolvimento na construção de um **analisador léxico** utilizando a ferramenta **Flex**. Essa etapa é fundamental, pois o analisador léxico corresponde ao primeiro módulo de um compilador, sendo responsável por transformar a sequência de caracteres do código-fonte em uma sequência de tokens significativos para as etapas posteriores da compilação.

A motivação se encontra na importância de compreender de forma prática como se dá o reconhecimento de padrões léxicos em linguagens de programação, além de exercitar a implementação de técnicas de análise que desconsidere espaços em branco e comentários, identifique corretamente lexemas e reporte erros com suas respectivas posições no código. Dessa forma, a construção do analisador léxico proporciona uma visão concreta do funcionamento inicial de um compilador, servindo como base para as etapas seguintes de análise sintática e semântica.

Neste relatório, o leitor encontrará uma descrição das principais **decisões de projeto** tomadas durante a implementação, a exposição das **dificuldades encontradas**, bem como a apresentação de **diagramas de transição (DFAs)** para duas classes de tokens. Além disso, são incluídos **arquivos de teste** utilizados para validar a implementação, juntamente com suas respectivas saídas.

2. DESCRIÇÕES DO PROJETO

2.1. Decisões

2.1.1. Desmembramento do conjunto de palavras reservadas

Para as próximas fases do trabalho, consideramos mais adequado que cada palavra reservada possua o seu próprio token, o que facilitará o tratamento e a análise durante o processo de compilação.

2.1.2. Regras para os identificadores

Mantivemos o padrão adotado pela maioria das linguagens de programação, no qual os identificadores podem iniciar com letras ou com o caractere “_”, e, a partir daí, podem conter números, letras e “_”. Não há restrições quanto ao caractere final do identificador.

2.1.3. Tratamento de erros

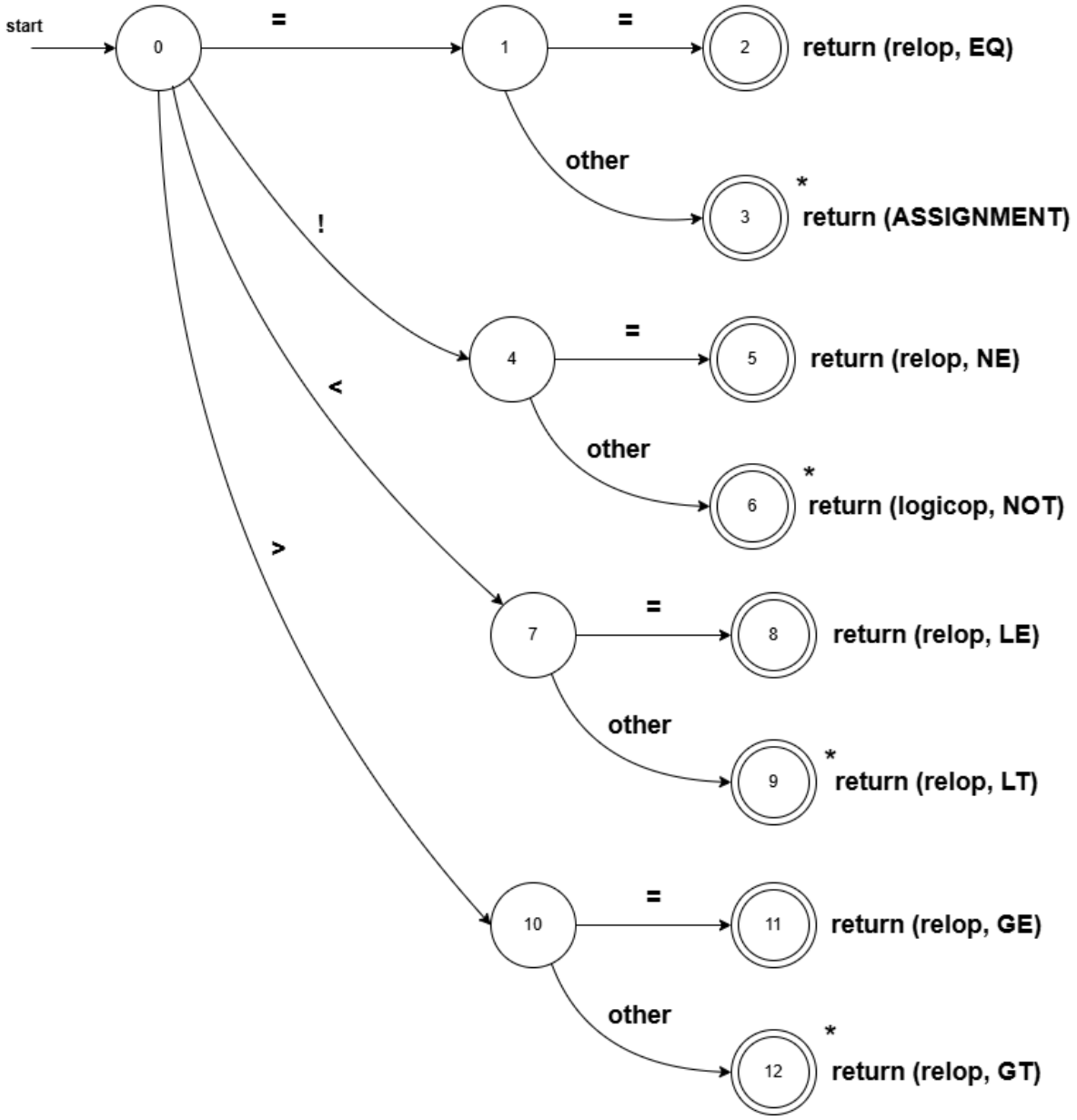
Definimos o tratamento de dois tipos de erros léxicos, cada um com mensagens específicas:

- **Identificadores iniciados por números**, que são inválidos;
- **Caracteres não reconhecidos pela linguagem**.

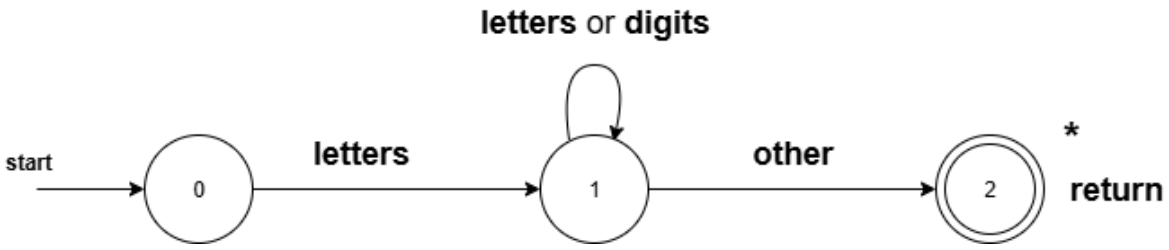
Em ambos os casos, o compilador gera uma mensagem de erro indicando a natureza do problema e o caracteriza como **erro léxico**, informando também a linha e a coluna em que ocorreu.

3. DIAGRAMAS DE TRANSIÇÃO

3.1. Diagrama de *RELOP*



3.2. Diagrama de *ID*



4. TESTES EXECUTADOS

4.1. Arquivo de Teste

teste.txt

```
int x = 10;
bool flag = false;
if (x >= 5 && flag) {
    print(x);
} else {
    read(y);
    z1 = x + 20;
}
while (flag || x < 0) {
    x = x - 1;
}

// Teste de identificadores válidos e inválidos
_x1
abc_123
_ // identificador válido
1abc // erro léxico: identificador começando com número
10abc // erro léxico: identificador começando com número

// Teste de números inteiros
123
-456
0

```

```
// Teste de operadores
a = b + c * d / e % f - g;
h == i;
j != k;
l <= m;
n >= o;
p < q;
r > s;
t && u;
v || w;
!x;

// Teste de símbolos especiais
( ) { } ; ,

// Teste de comentários
// Comentário de linha
/* Comentário de bloco */
abc /* Comentário no meio */ def

// Teste de espaços, tabs e novas linhas
a      b    c
d
e

// Teste de erro léxico: token inesperado
@ # $ ~

10
```

4.1. Arquivo de Saída

4.1.1. Lexema, Token, Linha e Coluna

```
None
Type: int - Line: 1 | Column: 1
ID: x - Line: 1 | Column: 5
Assignment: = - Line: 1 | Column: 7
Number: 10 - Line: 1 | Column: 9
Symbol: ; - Line: 1 | Column: 11
Type: bool - Line: 2 | Column: 1
ID: flag - Line: 2 | Column: 6
Assignment: = - Line: 2 | Column: 11
Keyword FALSE: false - Line: 2 | Column: 13
Symbol: ; - Line: 2 | Column: 18
Keyword IF: if - Line: 3 | Column: 1
Symbol: ( - Line: 3 | Column: 4
ID: x - Line: 3 | Column: 5
Relational Operator: >= - Line: 3 | Column: 7
Number: 5 - Line: 3 | Column: 10
Logical Operator: && - Line: 3 | Column: 12
ID: flag - Line: 3 | Column: 15
Symbol: ) - Line: 3 | Column: 19
Symbol: { - Line: 3 | Column: 21
Keyword PRINT: print - Line: 4 | Column: 5
Symbol: ( - Line: 4 | Column: 10
ID: x - Line: 4 | Column: 11
Symbol: ) - Line: 4 | Column: 12
Symbol: ; - Line: 4 | Column: 13
Symbol: } - Line: 5 | Column: 1
Keyword ELSE: else - Line: 5 | Column: 3
Symbol: { - Line: 5 | Column: 8
Keyword READ: read - Line: 6 | Column: 5
Symbol: ( - Line: 6 | Column: 9
```

```
ID: y - Line: 6 | Column: 10
Symbol: ) - Line: 6 | Column: 11
Symbol: ; - Line: 6 | Column: 12
ID: z1 - Line: 7 | Column: 5
Assignment: = - Line: 7 | Column: 8
ID: x - Line: 7 | Column: 10
Arithmetic Operator: + - Line: 7 | Column: 12
Number: 20 - Line: 7 | Column: 14
Symbol: ; - Line: 7 | Column: 16
Symbol: } - Line: 8 | Column: 1
Keyword WHILE: while - Line: 9 | Column: 1
Symbol: ( - Line: 9 | Column: 7
ID: flag - Line: 9 | Column: 8
Logical Operator: || - Line: 9 | Column: 13
ID: x - Line: 9 | Column: 16
Relational Operator: < - Line: 9 | Column: 18
Number: 0 - Line: 9 | Column: 20
Symbol: ) - Line: 9 | Column: 21
Symbol: { - Line: 9 | Column: 23
ID: x - Line: 10 | Column: 5
Assignment: = - Line: 10 | Column: 7
ID: x - Line: 10 | Column: 9
Arithmetic Operator: - - Line: 10 | Column: 11
Number: 1 - Line: 10 | Column: 13
Symbol: ; - Line: 10 | Column: 14
Symbol: } - Line: 11 | Column: 1
ID: _x1 - Line: 14 | Column: 1
ID: abc_123 - Line: 15 | Column: 1
ID: _ - Line: 16 | Column: 1
ERRO: 1abc - Line: 17 | Column: 1 - Message: ID starting
with a number: "1abc". Lexical Error.
ERRO: 10abc - Line: 18 | Column: 1 - Message: ID starting
with a number: "10abc". Lexical Error.
Number: 123 - Line: 21 | Column: 1
Number: -456 - Line: 22 | Column: 1
Number: 0 - Line: 23 | Column: 1
```

```
ID: a - Line: 26 | Column: 1
Assignment: = - Line: 26 | Column: 3
ID: b - Line: 26 | Column: 5
Arithmetic Operator: + - Line: 26 | Column: 7
ID: c - Line: 26 | Column: 9
Arithmetic Operator: * - Line: 26 | Column: 11
ID: d - Line: 26 | Column: 13
Arithmetic Operator: / - Line: 26 | Column: 15
ID: e - Line: 26 | Column: 17
Arithmetic Operator: % - Line: 26 | Column: 19
ID: f - Line: 26 | Column: 21
Arithmetic Operator: - - Line: 26 | Column: 23
ID: g - Line: 26 | Column: 25
Symbol: ; - Line: 26 | Column: 26
ID: h - Line: 27 | Column: 1
Relational Operator: == - Line: 27 | Column: 3
ID: i - Line: 27 | Column: 6
Symbol: ; - Line: 27 | Column: 7
ID: j - Line: 28 | Column: 1
Relational Operator: != - Line: 28 | Column: 3
ID: k - Line: 28 | Column: 6
Symbol: ; - Line: 28 | Column: 7
ID: l - Line: 29 | Column: 1
Relational Operator: <= - Line: 29 | Column: 3
ID: m - Line: 29 | Column: 6
Symbol: ; - Line: 29 | Column: 7
ID: n - Line: 30 | Column: 1
Relational Operator: >= - Line: 30 | Column: 3
ID: o - Line: 30 | Column: 6
Symbol: ; - Line: 30 | Column: 7
ID: p - Line: 31 | Column: 1
Relational Operator: < - Line: 31 | Column: 3
ID: q - Line: 31 | Column: 5
Symbol: ; - Line: 31 | Column: 6
ID: r - Line: 32 | Column: 1
Relational Operator: > - Line: 32 | Column: 3
```

```
ID: s - Line: 32 | Column: 5
Symbol: ; - Line: 32 | Column: 6
ID: t - Line: 33 | Column: 1
Logical Operator: && - Line: 33 | Column: 3
ID: u - Line: 33 | Column: 6
Symbol: ; - Line: 33 | Column: 7
ID: v - Line: 34 | Column: 1
Logical Operator: || - Line: 34 | Column: 3
ID: w - Line: 34 | Column: 6
Symbol: ; - Line: 34 | Column: 7
Logical Operator: ! - Line: 35 | Column: 1
ID: x - Line: 35 | Column: 2
Symbol: ; - Line: 35 | Column: 3
Symbol: ( - Line: 38 | Column: 1
Symbol: ) - Line: 38 | Column: 3
Symbol: { - Line: 38 | Column: 5
Symbol: } - Line: 38 | Column: 7
Symbol: ; - Line: 38 | Column: 9
Symbol: , - Line: 38 | Column: 11
ID: abc - Line: 43 | Column: 1
ID: def - Line: 43 | Column: 31
ID: a - Line: 46 | Column: 1
ID: b - Line: 46 | Column: 8
ID: c - Line: 46 | Column: 13
ID: d - Line: 47 | Column: 1
ID: e - Line: 48 | Column: 1
ERRO: @ - Line: 51 | Column: 1 - Message: Unexpected
token: "@". Lexical Error.
ERRO: # - Line: 51 | Column: 3 - Message: Unexpected
token: "#". Lexical Error.
ERRO: $ - Line: 51 | Column: 5 - Message: Unexpected
token: "$". Lexical Error.
ERRO: ~ - Line: 51 | Column: 7 - Message: Unexpected
token: "~". Lexical Error.
Number: 10 - Line: 53 | Column: 1
Fim de arquivo - Line: 53 | Column: 3
```

4.1.2. Tabela de Símbolos

None

```
---- Tabela de Simbolos ----
1 Token: <TYPE> | Value: int | Line: 1 | Column: 1
2 Token: <ID> | Value: x | Line: 1 | Column: 5
3 Token: <ASSIGNMENT> | Value: = | Line: 1 | Column: 7
4 Token: <NUMBER> | Value: 10 | Line: 1 | Column: 9
5 Token: <SYMBOL> | Value: ; | Line: 1 | Column: 11
6 Token: <TYPE> | Value: bool | Line: 2 | Column: 1
7 Token: <ID> | Value: flag | Line: 2 | Column: 6
8 Token: <ASSIGNMENT> | Value: = | Line: 2 | Column: 11
9 Token: <FALSE> | Value: false | Line: 2 | Column: 13
10 Token: <SYMBOL> | Value: ; | Line: 2 | Column: 18
11 Token: <IF> | Value: if | Line: 3 | Column: 1
12 Token: <SYMBOL> | Value: ( | Line: 3 | Column: 4
13 Token: <ID> | Value: x | Line: 3 | Column: 5
14 Token: <RELOP> | Value: >= | Line: 3 | Column: 7
15 Token: <NUMBER> | Value: 5 | Line: 3 | Column: 10
16 Token: <LOGICOP> | Value: && | Line: 3 | Column: 12
17 Token: <ID> | Value: flag | Line: 3 | Column: 15
18 Token: <SYMBOL> | Value: ) | Line: 3 | Column: 19
19 Token: <SYMBOL> | Value: { | Line: 3 | Column: 21
20 Token: <PRINT> | Value: print | Line: 4 | Column: 5
21 Token: <SYMBOL> | Value: ( | Line: 4 | Column: 10
22 Token: <ID> | Value: x | Line: 4 | Column: 11
23 Token: <SYMBOL> | Value: ) | Line: 4 | Column: 12
24 Token: <SYMBOL> | Value: ; | Line: 4 | Column: 13
25 Token: <SYMBOL> | Value: } | Line: 5 | Column: 1
26 Token: <ELSE> | Value: else | Line: 5 | Column: 3
27 Token: <SYMBOL> | Value: { | Line: 5 | Column: 8
28 Token: <READ> | Value: read | Line: 6 | Column: 5
29 Token: <SYMBOL> | Value: ( | Line: 6 | Column: 9
30 Token: <ID> | Value: y | Line: 6 | Column: 10
31 Token: <SYMBOL> | Value: ) | Line: 6 | Column: 11
```

```
32 Token: <SYMBOL> | Value: ; | Line: 6 | Column: 12
33 Token: <ID> | Value: z1 | Line: 7 | Column: 5
34 Token: <ASSIGNMENT> | Value: = | Line: 7 | Column: 8
35 Token: <ID> | Value: x | Line: 7 | Column: 10
36 Token: <ARITHOP> | Value: + | Line: 7 | Column: 12
37 Token: <NUMBER> | Value: 20 | Line: 7 | Column: 14
38 Token: <SYMBOL> | Value: ; | Line: 7 | Column: 16
39 Token: <SYMBOL> | Value: } | Line: 8 | Column: 1
40 Token: <WHILE> | Value: while | Line: 9 | Column: 1
41 Token: <SYMBOL> | Value: ( | Line: 9 | Column: 7
42 Token: <ID> | Value: flag | Line: 9 | Column: 8
43 Token: <LOGICOP> | Value: || | Line: 9 | Column: 13
44 Token: <ID> | Value: x | Line: 9 | Column: 16
45 Token: <RELOP> | Value: < | Line: 9 | Column: 18
46 Token: <NUMBER> | Value: 0 | Line: 9 | Column: 20
47 Token: <SYMBOL> | Value: ) | Line: 9 | Column: 21
48 Token: <SYMBOL> | Value: { | Line: 9 | Column: 23
49 Token: <ID> | Value: x | Line: 10 | Column: 5
50 Token: <ASSIGNMENT> | Value: = | Line: 10 | Column: 7
51 Token: <ID> | Value: x | Line: 10 | Column: 9
52 Token: <ARITHOP> | Value: - | Line: 10 | Column: 11
53 Token: <NUMBER> | Value: 1 | Line: 10 | Column: 13
54 Token: <SYMBOL> | Value: ; | Line: 10 | Column: 14
55 Token: <SYMBOL> | Value: } | Line: 11 | Column: 1
56 Token: <ID> | Value: _x1 | Line: 14 | Column: 1
57 Token: <ID> | Value: abc_123 | Line: 15 | Column: 1
58 Token: <ID> | Value: _ | Line: 16 | Column: 1
59 Token: <NUMBER> | Value: 123 | Line: 21 | Column: 1
60 Token: <NUMBER> | Value: -456 | Line: 22 | Column: 1
61 Token: <NUMBER> | Value: 0 | Line: 23 | Column: 1
62 Token: <ID> | Value: a | Line: 26 | Column: 1
63 Token: <ASSIGNMENT> | Value: = | Line: 26 | Column: 3
64 Token: <ID> | Value: b | Line: 26 | Column: 5
65 Token: <ARITHOP> | Value: + | Line: 26 | Column: 7
66 Token: <ID> | Value: c | Line: 26 | Column: 9
67 Token: <ARITHOP> | Value: * | Line: 26 | Column: 11
```

```
68 Token: <ID> | Value: d | Line: 26 | Column: 13
69 Token: <ARITHOP> | Value: / | Line: 26 | Column: 15
70 Token: <ID> | Value: e | Line: 26 | Column: 17
71 Token: <ARITHOP> | Value: % | Line: 26 | Column: 19
72 Token: <ID> | Value: f | Line: 26 | Column: 21
73 Token: <ARITHOP> | Value: - | Line: 26 | Column: 23
74 Token: <ID> | Value: g | Line: 26 | Column: 25
75 Token: <SYMBOL> | Value: ; | Line: 26 | Column: 26
76 Token: <ID> | Value: h | Line: 27 | Column: 1
77 Token: <RELOP> | Value: == | Line: 27 | Column: 3
78 Token: <ID> | Value: i | Line: 27 | Column: 6
79 Token: <SYMBOL> | Value: ; | Line: 27 | Column: 7
80 Token: <ID> | Value: j | Line: 28 | Column: 1
81 Token: <RELOP> | Value: != | Line: 28 | Column: 3
82 Token: <ID> | Value: k | Line: 28 | Column: 6
83 Token: <SYMBOL> | Value: ; | Line: 28 | Column: 7
84 Token: <ID> | Value: l | Line: 29 | Column: 1
85 Token: <RELOP> | Value: <= | Line: 29 | Column: 3
86 Token: <ID> | Value: m | Line: 29 | Column: 6
87 Token: <SYMBOL> | Value: ; | Line: 29 | Column: 7
88 Token: <ID> | Value: n | Line: 30 | Column: 1
89 Token: <RELOP> | Value: >= | Line: 30 | Column: 3
90 Token: <ID> | Value: o | Line: 30 | Column: 6
91 Token: <SYMBOL> | Value: ; | Line: 30 | Column: 7
92 Token: <ID> | Value: p | Line: 31 | Column: 1
93 Token: <RELOP> | Value: < | Line: 31 | Column: 3
94 Token: <ID> | Value: q | Line: 31 | Column: 5
95 Token: <SYMBOL> | Value: ; | Line: 31 | Column: 6
96 Token: <ID> | Value: r | Line: 32 | Column: 1
97 Token: <RELOP> | Value: > | Line: 32 | Column: 3
98 Token: <ID> | Value: s | Line: 32 | Column: 5
99 Token: <SYMBOL> | Value: ; | Line: 32 | Column: 6
100 Token: <ID> | Value: t | Line: 33 | Column: 1
101 Token: <LOGICOP> | Value: && | Line: 33 | Column: 3
102 Token: <ID> | Value: u | Line: 33 | Column: 6
103 Token: <SYMBOL> | Value: ; | Line: 33 | Column: 7
```

```
104 Token: <ID> | Value: v | Line: 34 | Column: 1
105 Token: <LOGICOP> | Value: || | Line: 34 | Column: 3
106 Token: <ID> | Value: w | Line: 34 | Column: 6
107 Token: <SYMBOL> | Value: ; | Line: 34 | Column: 7
108 Token: <LOGICOP> | Value: ! | Line: 35 | Column: 1
109 Token: <ID> | Value: x | Line: 35 | Column: 2
110 Token: <SYMBOL> | Value: ; | Line: 35 | Column: 3
111 Token: <SYMBOL> | Value: ( | Line: 38 | Column: 1
112 Token: <SYMBOL> | Value: ) | Line: 38 | Column: 3
113 Token: <SYMBOL> | Value: { | Line: 38 | Column: 5
114 Token: <SYMBOL> | Value: } | Line: 38 | Column: 7
115 Token: <SYMBOL> | Value: ; | Line: 38 | Column: 9
116 Token: <SYMBOL> | Value: , | Line: 38 | Column: 11
117 Token: <ID> | Value: abc | Line: 43 | Column: 1
118 Token: <ID> | Value: def | Line: 43 | Column: 31
119 Token: <ID> | Value: a | Line: 46 | Column: 1
120 Token: <ID> | Value: b | Line: 46 | Column: 8
121 Token: <ID> | Value: c | Line: 46 | Column: 13
122 Token: <ID> | Value: d | Line: 47 | Column: 1
123 Token: <ID> | Value: e | Line: 48 | Column: 1
124 Token: <NUMBER> | Value: 10 | Line: 53 | Column: 1
----------------------------
```

5. DIFICULDADES ENCONTRADOS

5.1. Analisador Léxico

5.1.1. Problema com Contagem de Colunas

Foi identificado um problema relacionado ao cálculo da posição da coluna no código-fonte. Inicialmente, o incremento da coluna era realizado de forma unitária, independentemente do tamanho do lexema reconhecido. Dessa forma, palavras compostas por múltiplos caracteres eram contabilizadas como se possuíssem apenas uma coluna, gerando inconsistências na indicação de posição dos tokens.

A solução adotada consistiu em ajustar o contador de colunas para considerar o comprimento efetivo do lexema reconhecido, de modo que a posição final fosse corretamente incrementada pelo número de caracteres do token, e não apenas por uma unidade. Essa abordagem garante maior precisão na localização dos tokens e, consequentemente, na geração de mensagens de erro e relatórios de análise.

5.1.2. Problema Com Sequência de Regras

Durante os testes, foi observado um problema relacionado à ordem de aplicação das regras no analisador léxico. Um lexema composto por múltiplos caracteres alfabéticos era reconhecido prematuramente pela regra destinada a um único caractere (`letters`), impedindo que fosse corretamente classificado como identificador (`id`).

A solução implementada consistiu em reordenar as regras de modo que identificadores fossem reconhecidos prioritariamente e, posteriormente, remover a regra específica de `letters`, visto que esta se tornou redundante e inacessível no fluxo de análise. Essa modificação assegurou o reconhecimento adequado de identificadores e eliminou conflitos de precedência entre as expressões regulares.