CEFET-MG COMPILADORES ENTREGA:30/04/2019

KELANG

ANALISADOR LÉXICO

Integrantes do grupo: Nathã Carvalhais, Matheus Santos, Túlio Mitre

FORMA DE USO DO COMPILADOR

O compilador está em produção na linguagem JAVA, arquivo top-level de nome kelang.java no diretório /Code. É necessário que esse arquivo esteja no mesmo diretório da pasta /lexical, que contém os arquivos necessários para importação do analisador léxico.

Os arquivos foram compilados usando o Java Compiler (javac) em comando por terminal Linux, no diretório /Code

$ javac kelang.java

gerando os arquivos executáveis pelo comando no terminal

$ *java kelang arquivo_texto*

IMPLEMENTAÇÃO DO ANALISADOR LÉXICO

Sobre as classes implementadas

A primeira classe a ser definida é um token para armazenar os dados obtidos do texto. Isso foi feito com a classe *public class Lexeme*, que contém dados públicos:

- *token*, tipo *String* (nativa Java);

- *type*, tipo *TokenType*.

O tipo TokenType foi definido em tokens especiais, palavras-chave, símbolos, operadores e outros tipos necessários para operar a linguagem no arquivo "TokenType.java", contendo uma estrutura *public enum TokenType*. Um lexema possui apenas um construtor, também público, que recebe um token e um tipo e atribui os dados recebidos às variáveis locais.

A classe *class SymbolTable* é a forma de ligar o texto obtido com os tipos possíveis de token definidos na enumeração TokenType utilizando uma estrutura de mapa privada *HashMap<String, TokenType>.* Essa tabela de símbolos possui duas funções internas:

- *public boolean contains(String token),* que verifica se a tabela contém determinado token recebido e retorna verdadeiro caso afirmativo ou falso caso negativo;

- *public TokenType find(String token),* que recebe um token e retorna seu tipo de acordo com os dados na tabela. Se esse dado não estiver presente, retorna tipo inválido.

A classe *public class Kelang* possui o modulo *public static void main(String[] args),* classe top-level, usada para receber os arquivos a serem analisados e tratamento de erros e exceções. Na fase de análise léxica, cria-se um objeto da classe Lexema e executa a análise.

O arquivo LexicalAnalysis.java contém a classe e funções necessárias para a análise do texto, identificar e tipificar os tokens pela função *public Lexeme nextToken(),* que retorna um token especificado utilizando a tabela de símbolos e a máquina de estados para a gramática da linguagem em questão.

Sobre o funcionamento da máquina de estados

Estado 0: Inicial. Espaços, tabulações, retornos e quebras de linha são lidos, mas não registrados como tokens ou como parte da tabela de símbolos. Além disso, direciona a entrada para diferentes estados para processar cada tipo de token e já retorna tokens de símbolos simples.

Estado 1: Comentário. Aguarda a leitura de uma quebra de linha e não armazena nada;

Estado 2: String. O estado 0 identifica um caractere "{" e o estado 2 armazena todos os caracteres da entrada até encontrar o símbolo "}";

Estado 3: Identificadores iniciados com letras ou palavras reservadas. Se o token possuir apenas uma letra, registra-se um ID. Se possuir mais de uma letra, utiliza-se as funções de busca na tabela de símbolos para verificar a existência de palavras-chave;

Estado 4: Identificadores iniciados com o caractere "_", possível ler caracteres, dígitos e repetições do caractere "_";

Estado 5: Operadores;

Estado 6: Operadores "|";

Estado 7: Operadores "&";

Estado 8: Constantes numéricas. Direciona para outro estado se encontrar um ".";

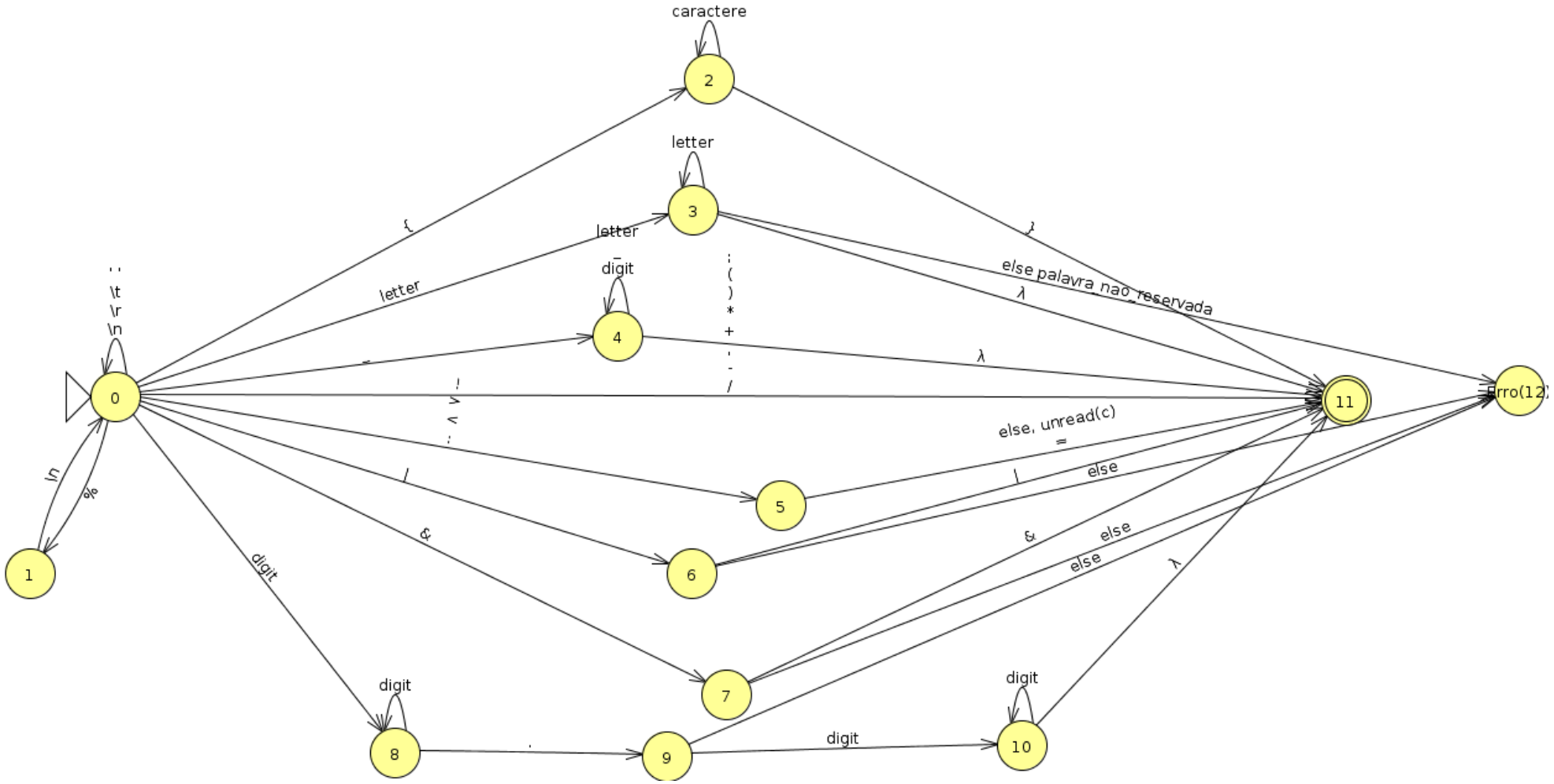
Estado 9: Números reais;

Estado 10: Números reais;

Estado 11: Estado final da máquina, formando um token.

Estado 12: Estado de erro, normalmente TokenType inválido.

Diagrama da máquina de estados:


caractere
2
letter
3
letter
digit
4
0
1
5
6
7
8
9
10
11
Erro(12)
{
}
else palavra_nao_reservada
λ
\t
\r
\n
%
digit
&
else, unread(c)
=
else
.
digit
digit
digit
λ

REALIZAÇÃO DE TESTES

Todos os testes foram realizados em duas diferentes máquinas do DGO/NTIC no CEFET-MG e um laptop Dell Inspiron 15, no Linux e Windows, respectivamente. Em todas as máquinas o programa foi compilado com o javac por terminal.

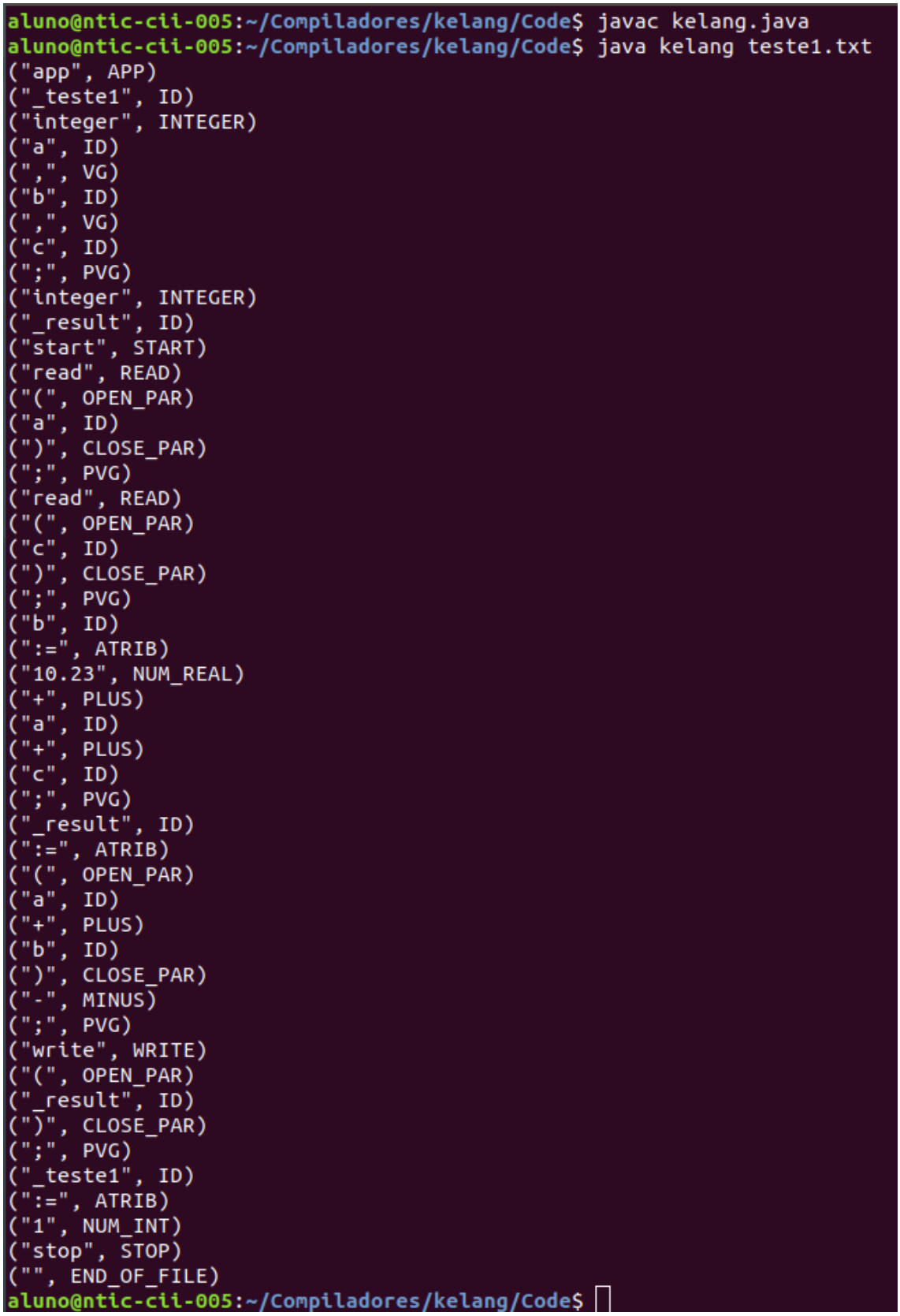
TESTE 1

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste1.txt
("app", APP)
01: Lexema inválido [teste]
```

A gramática não aceita identificadores com mais de uma letra, colocamos um caractere "_" para validar a identificação de "teste1". O mesmo foi feito para o identificador "result", transformando a entrada para _result.

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste1.txt
("app", APP)
("_teste1", ID)
02: Lexema inválido [int]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

"int" não é uma palavra reservada da gramática, mas sim "integer".



Sucesso no teste 1.

TESTE 2

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste2.txt
("app", APP)
02: Lexema inválido [int]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

"int" não é uma palavra reservada da gramática, mas sim "integer".

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste2.txt
("app", APP)
("integer", INTEGER)
("a", ID)
(";", PVG)
("integer", INTEGER)
("b", ID)
(";", PVG)
("real", REAL)
("f", ID)
(",", VG)
("_f", ID)
("start", START)
("read", READ)
("(", OPEN_PAR)
("a", ID)
(")", CLOSE_PAR)
(";", PVG)
("read", READ)
("(", OPEN_PAR)
("f", ID)
(")", CLOSE_PAR)
(";", PVG)
("f", ID)
(":=", ATRIB)
("a", ID)
("*", MUL)
("a", ID)
(";", PVG)
("b", ID)
(":=", ATRIB)
("b", ID)
("+", PLUS)
("a", ID)
("/", DIV)
("2", NUM_INT)
("*", MUL)
("(", OPEN_PAR)
("3", NUM_INT)
("+", PLUS)
("5", NUM_INT)
(")", CLOSE_PAR)
(";", PVG)
("write", WRITE)
("(", OPEN_PAR)
("f", ID)
(")", CLOSE_PAR)
(";", PVG)
("write", WRITE)
("(", OPEN_PAR)
("b", ID)
(")", CLOSE_PAR)
(";", PVG)
("", END_OF_FILE)
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

Sucesso no teste 2.

TESTE 3

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste3.txt
("app", APP)
01: Lexema inválido [pessoa]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

A gramática não aceita identificadores com mais de uma letra, colocamos um caractere "_" para validar a identificação de "pessoa".

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste3.txt
("app", APP)
("_pessoa", ID)
02: Lexema inválido [int]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

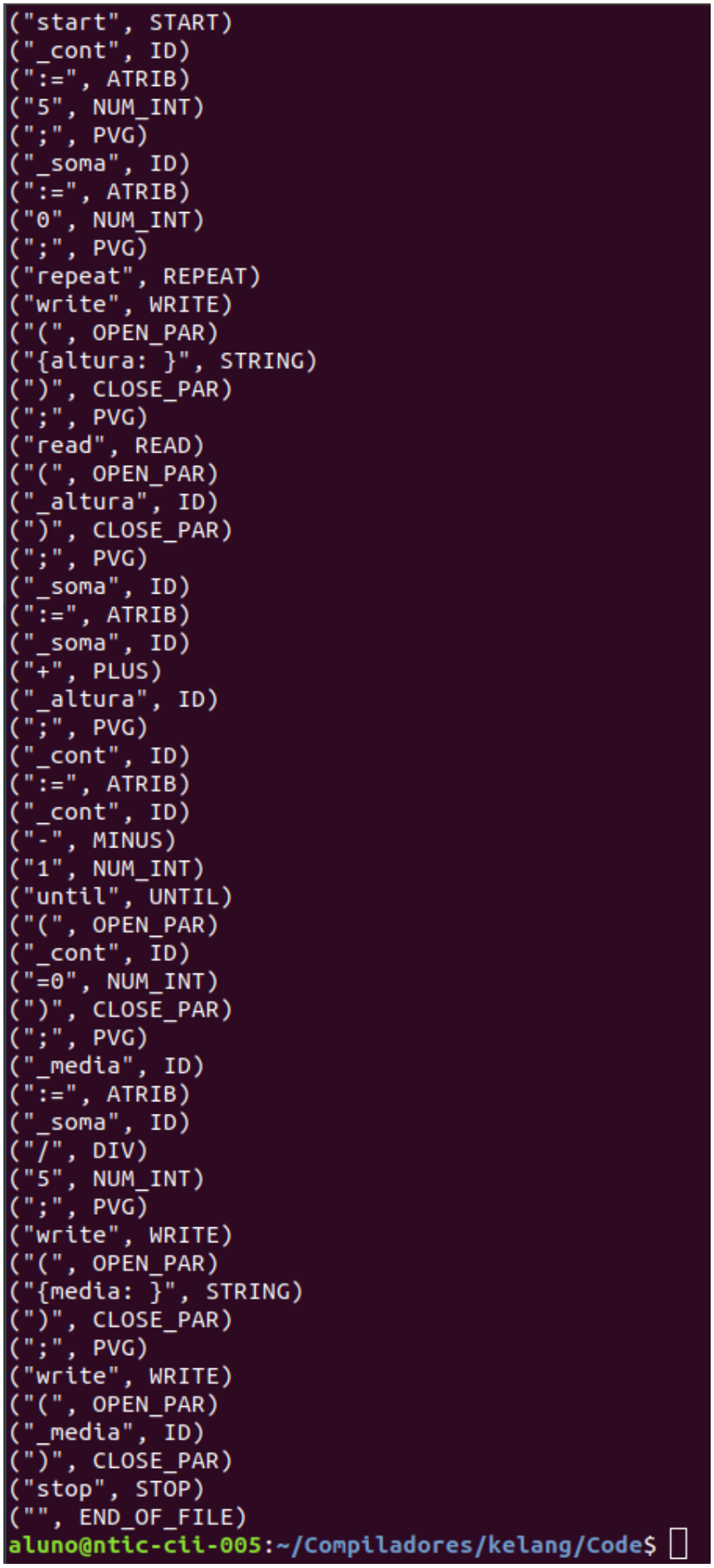
"int" não é uma palavra reservada da gramática, mas sim "integer".

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste3.txt
("app", APP)
("_pessoa", ID)
("integer", INTEGER)
02: Lexema inválido [:]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

":" não é uma palavra reservada/símbolo da gramática.

"#" não é uma palavra reservada/símbolo da gramática.

```
("start", START)
("_cont", ID)
(":=", ATRIB)
("5", NUM_INT)
(";", PVG)
("_soma", ID)
(":=", ATRIB)
("0", NUM_INT)
(";", PVG)
("repeat", REPEAT)
("write", WRITE)
("(", OPEN_PAR)
("{altura: }", STRING)
(")", CLOSE_PAR)
(";", PVG)
("read", READ)
("(", OPEN_PAR)
("_altura", ID)
(")", CLOSE_PAR)
(";", PVG)
("_soma", ID)
(":=", ATRIB)
("_soma", ID)
("+", PLUS)
("_altura", ID)
(";", PVG)
("_cont", ID)
(":=", ATRIB)
("_cont", ID)
("-", MINUS)
("1", NUM_INT)
("until", UNTIL)
("(", OPEN_PAR)
("_cont", ID)
("=0", NUM_INT)
(")", CLOSE_PAR)
(";", PVG)
("_media", ID)
(":=", ATRIB)
("_soma", ID)
("/", DIV)
("5", NUM_INT)
(";", PVG)
("write", WRITE)
("(", OPEN_PAR)
("{media: }", STRING)
(")", CLOSE_PAR)
(";", PVG)
("write", WRITE)
("(", OPEN_PAR)
("_media", ID)
(")", CLOSE_PAR)
("stop", STOP)
("", END_OF_FILE)
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```



Sucesso no teste 3.

TESTE 4

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste4.txt
("app", APP)
01: Lexema inválido [teste]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

A gramática não aceita identificadores com mais de uma letra, colocamos um caractere "_" para validar a identificação de "teste4".

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste4.txt
("app", APP)
("_teste4", ID)
02: Lexema inválido [int]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

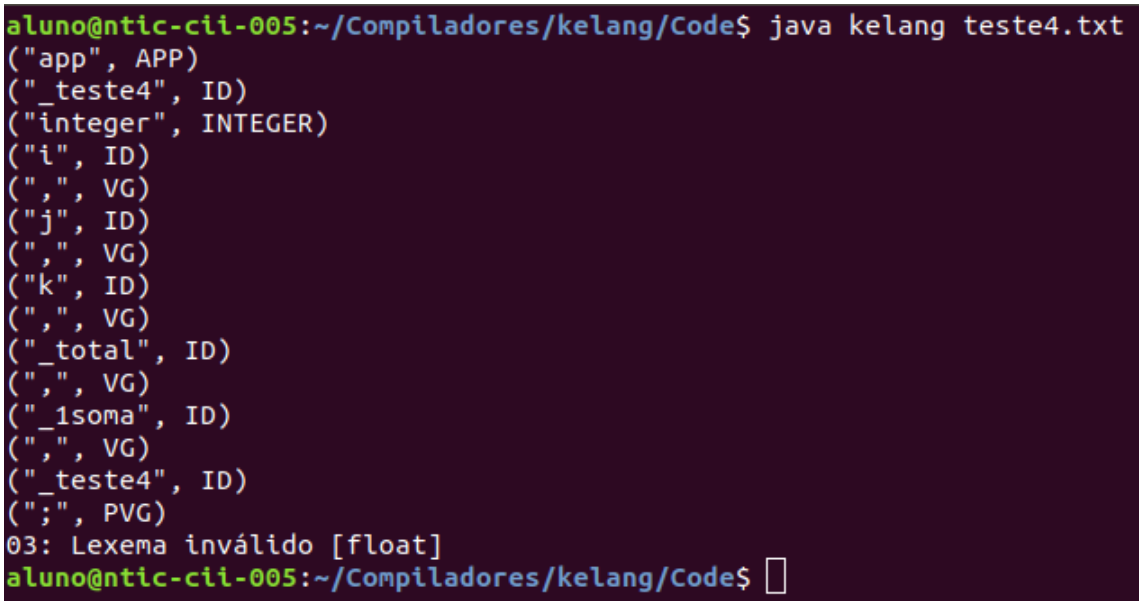
"int" não é uma palavra reservada da gramática, mas sim "integer".

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste4.txt
("app", APP)
("_teste4", ID)
("integer", INTEGER)
("i", ID)
(",", VG)
("j", ID)
(",", VG)
("k", ID)
(",", VG)
02: Lexema inválido [@]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

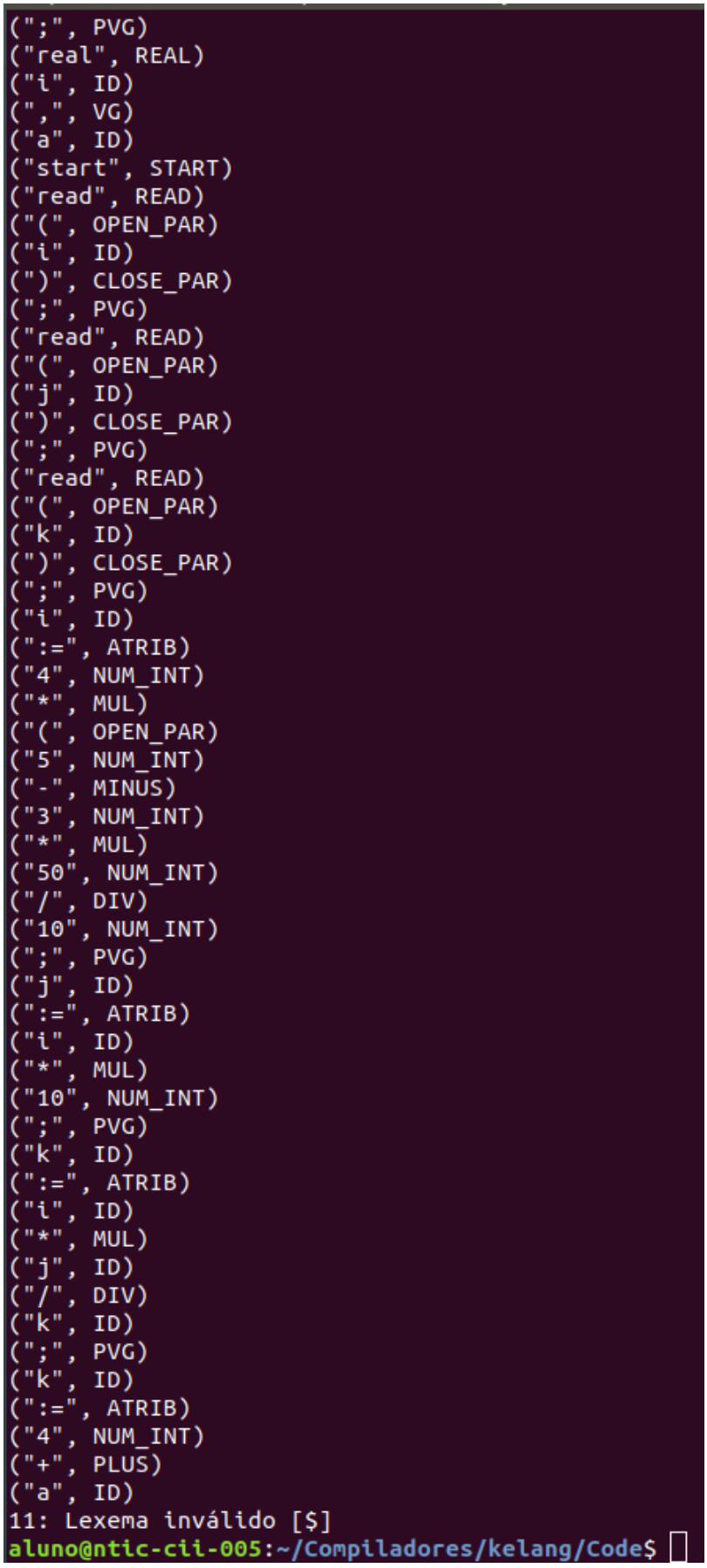
"@" não é uma palavra reservada/símbolo da gramática.

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste4.txt
("app", APP)
("_teste4", ID)
("integer", INTEGER)
("i", ID)
(",", VG)
("j", ID)
(",", VG)
("k", ID)
(",", VG)
("_total", ID)
(",", VG)
("1", NUM_INT)
02: Lexema inválido [soma]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

A gramática não aceita identificadores com mais de uma letra, colocamos um caractere "_" para validar a identificação de "1soma".



"float" não é uma palavra reservada da gramática, mas sim "real".



"@" não é uma palavra reservada/símbolo da gramática.

```
("read", READ)
("(", OPEN_PAR)
("k", ID)
(")", CLOSE_PAR)
(";", PVG)
("i", ID)
(":=", ATRIB)
("4", NUM_INT)
("*", MUL)
("(", OPEN_PAR)
("5", NUM_INT)
("-", MINUS)
("3", NUM_INT)
("*", MUL)
("50", NUM_INT)
("/", DIV)
("10", NUM_INT)
(";", PVG)
("j", ID)
(":=", ATRIB)
("i", ID)
("*", MUL)
("10", NUM_INT)
(";", PVG)
("k", ID)
(":=", ATRIB)
("i", ID)
("*", MUL)
("j", ID)
("/", DIV)
("k", ID)
(";", PVG)
("k", ID)
(":=", ATRIB)
("4", NUM_INT)
("+", PLUS)
("a", ID)
(";", PVG)
("write", WRITE)
("(", OPEN_PAR)
("i", ID)
(")", CLOSE_PAR)
(";", PVG)
("write", WRITE)
("(", OPEN_PAR)
("j", ID)
(")", CLOSE_PAR)
(";", PVG)
("write", WRITE)
("(", OPEN_PAR)
("k", ID)
(")", CLOSE_PAR)
("stop", STOP)
("", END_OF_FILE)
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

Sucesso no teste 4.

TESTE 5

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste5.txt
("app", APP)
01: Lexema inválido [teste]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

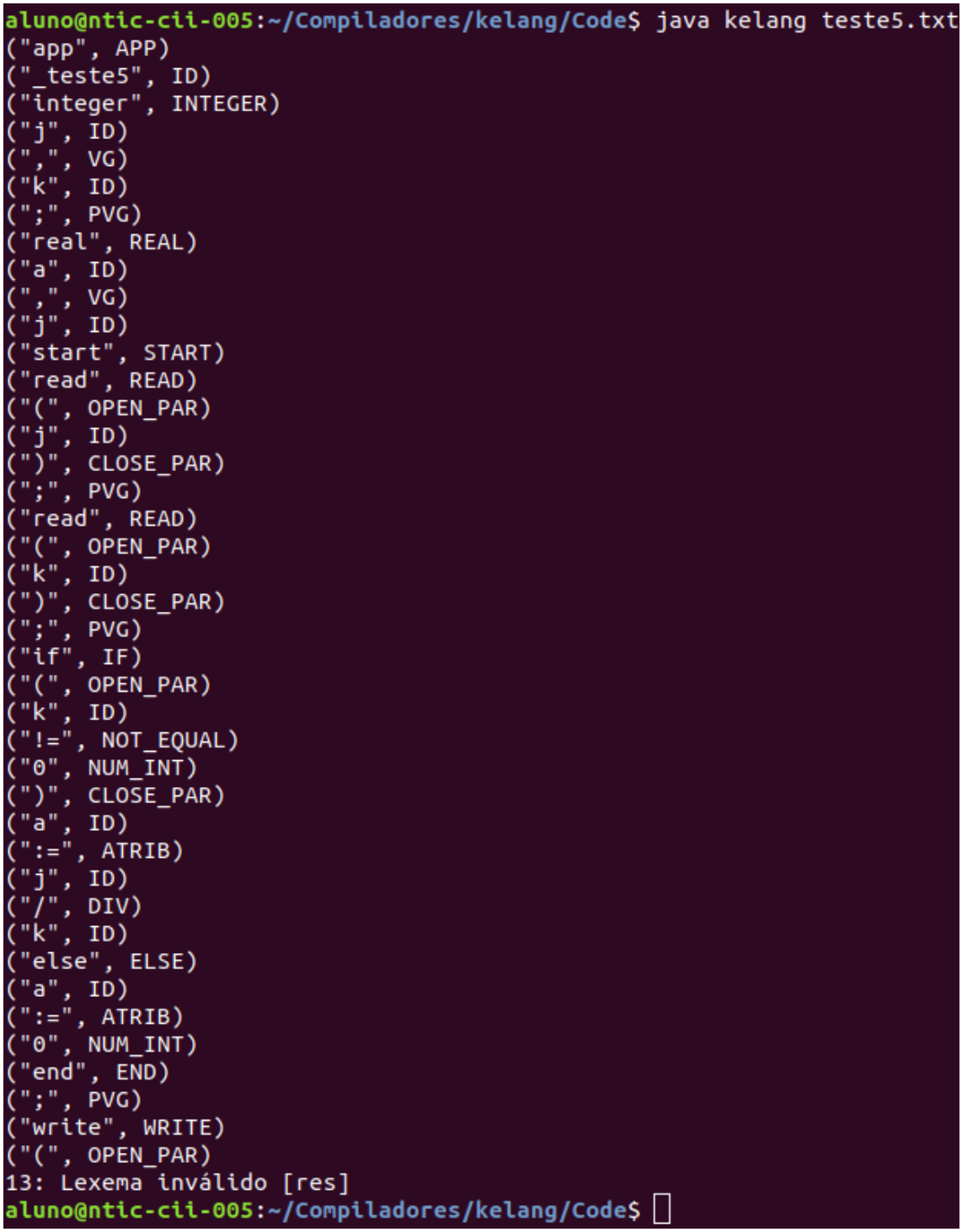
A gramática não aceita identificadores com mais de uma letra, colocamos um caractere "_" para validar a identificação de "teste5".

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste5.txt
("app", APP)
("_teste5", ID)
02: Lexema inválido [int]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

"int" não é uma palavra reservada da gramática, mas sim "integer".

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste5.txt
("app", APP)
("_teste5", ID)
("integer", INTEGER)
("j", ID)
(",", VG)
("k", ID)
(";", PVG)
("real", REAL)
("a", ID)
(",", VG)
("j", ID)
("start", START)
("read", READ)
("(", OPEN_PAR)
("j", ID)
(")", CLOSE_PAR)
(";", PVG)
("read", READ)
("(", OPEN_PAR)
("k", ID)
(")", CLOSE_PAR)
(";", PVG)
("if", IF)
("(", OPEN_PAR)
("k", ID)
("!=", NOT_EQUAL)
("0", NUM_INT)
(")", CLOSE_PAR)
08: Lexema inválido [result]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

A gramática não aceita identificadores com mais de uma letra, como "a" não era usado, assumimos que era o resultado.



A gramática não aceita identificadores com mais de uma letra, como “a” não era usado, assumimos que era o resultado.

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste5.txt
("app", APP)
("_teste5", ID)
("integer", INTEGER)
("j", ID)
(",", VG)
("k", ID)
(";", PVG)
("real", REAL)
("a", ID)
(",", VG)
("j", ID)
("start", START)
("read", READ)
("(", OPEN_PAR)
("j", ID)
(")", CLOSE_PAR)
(";", PVG)
("read", READ)
("(", OPEN_PAR)
("k", ID)
(")", CLOSE_PAR)
(";", PVG)
("if", IF)
("(", OPEN_PAR)
("k", ID)
("!=", NOT_EQUAL)
("0", NUM_INT)
(")", CLOSE_PAR)
("a", ID)
(":=", ATRIB)
("j", ID)
("/", DIV)
("k", ID)
("else", ELSE)
("a", ID)
(":=", ATRIB)
("0", NUM_INT)
("end", END)
(";", PVG)
("write", WRITE)
("(", OPEN_PAR)
("a", ID)
(")", CLOSE_PAR)
("stop", STOP)
("", END_OF_FILE)
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

Sucesso no teste 5.

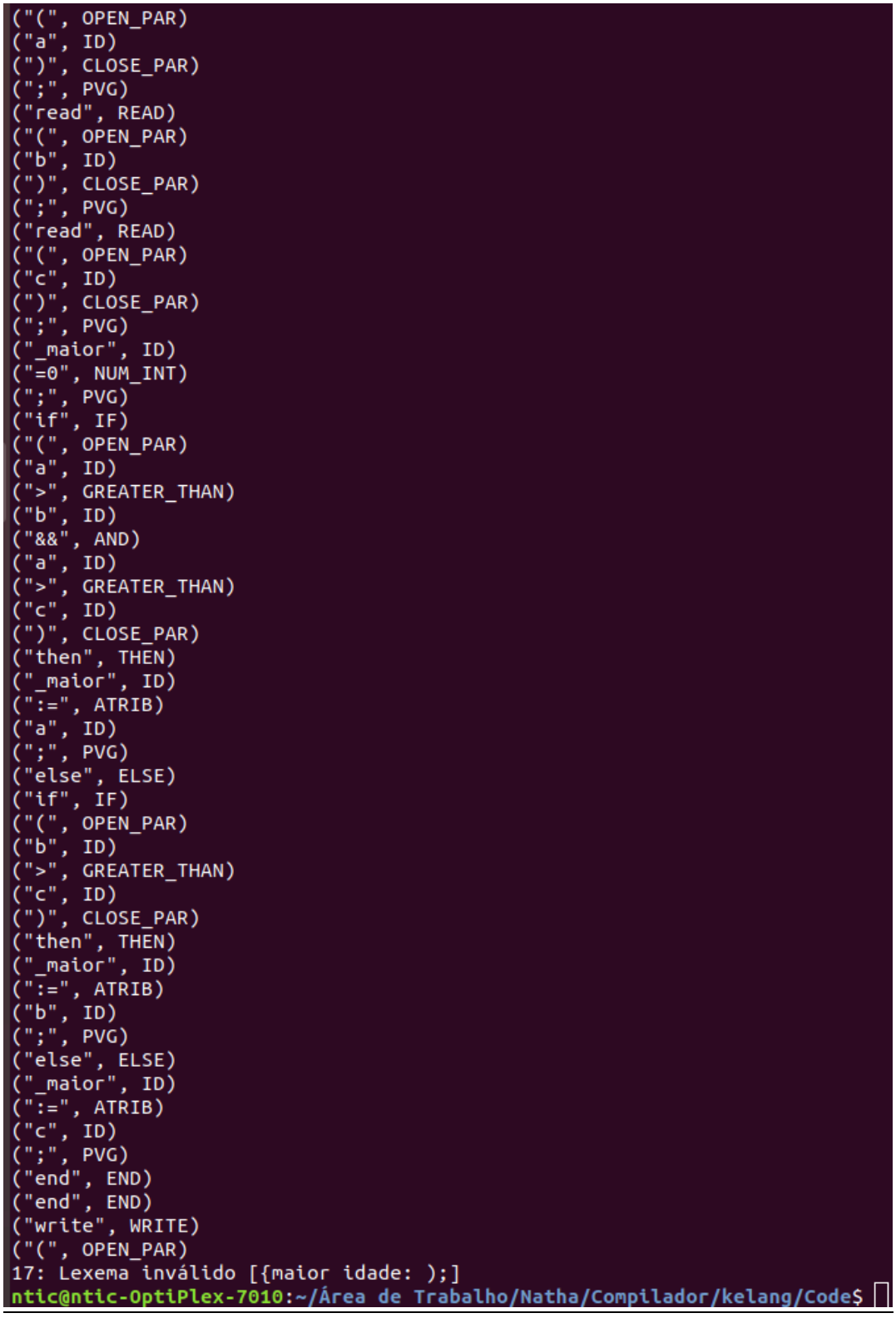
TESTE 6

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste6.txt
02: Lexema inválido [int]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

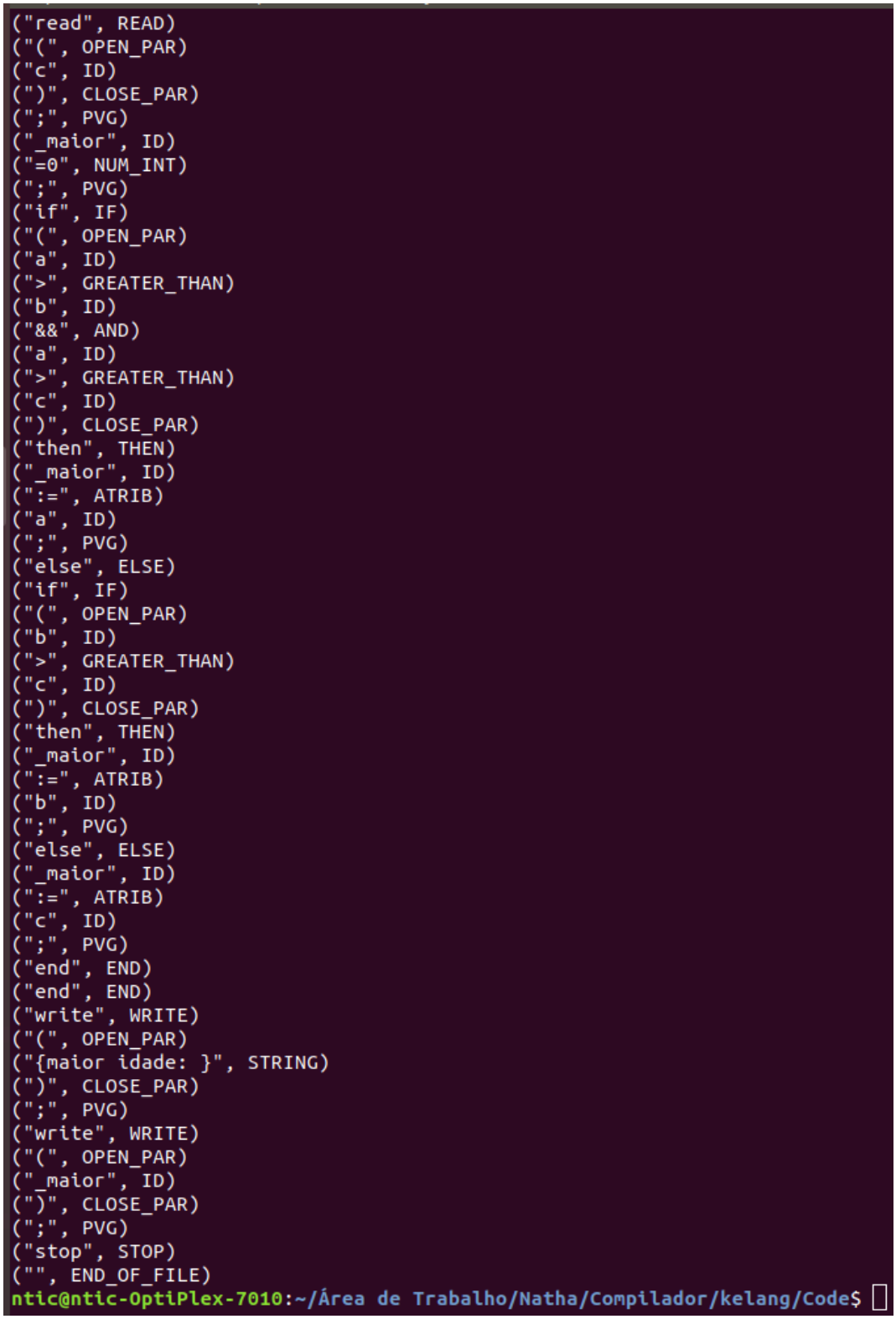
"int" não é uma palavra reservada da gramática, mas sim "integer".

```
aluno@ntic-cii-005:~/Compiladores/kelang/Code$ java kelang teste6.txt
("integer", INTEGER)
("a", ID)
(",", VG)
("b", ID)
(",", VG)
("c", ID)
(",", VG)
02: Lexema inválido [maior]
aluno@ntic-cii-005:~/Compiladores/kelang/Code$
```

A gramática não aceita identificadores com mais de uma letra, colocamos um caractere "_" para validar a identificação de "maior".



String sem fecha de caractere chave “}” inválida.



Sucesso no teste 6.

TESTE 7

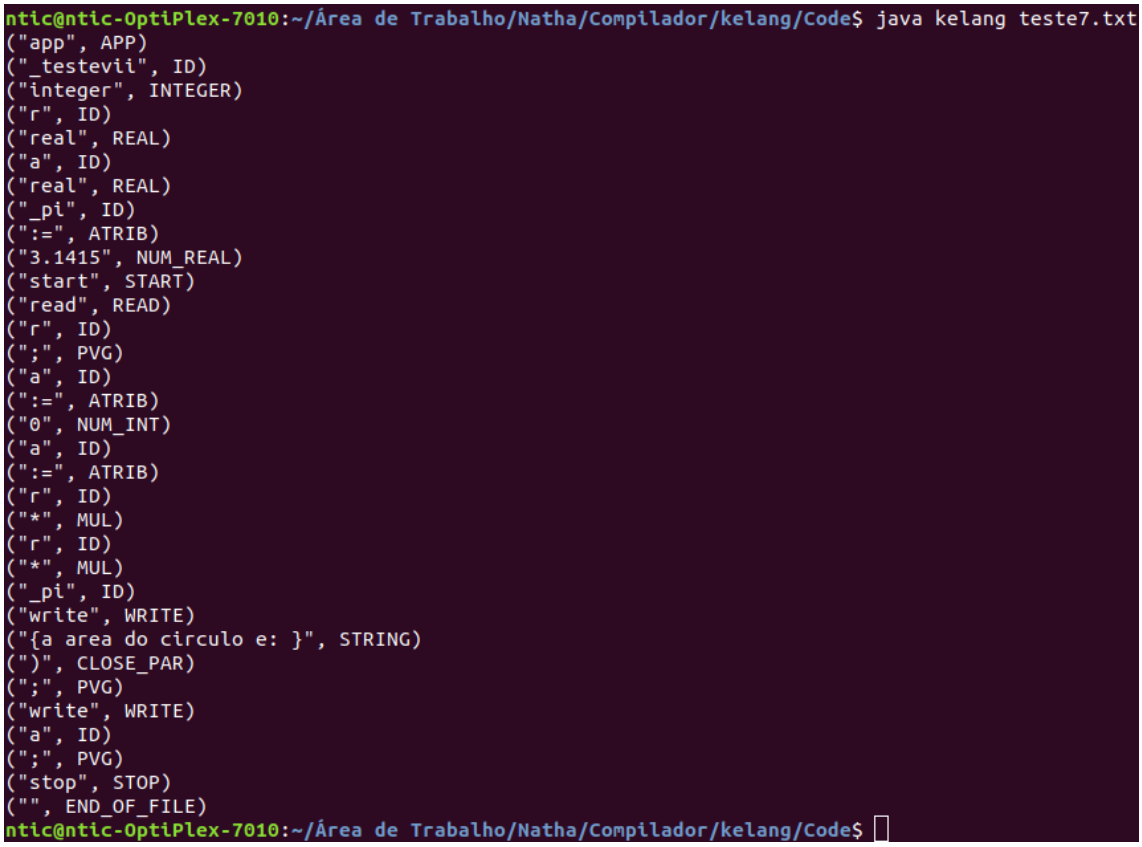
```
1   %Calculo aproximado da area de um circulo
2   app testeVII
3       int r;
4       real A;
5       real pi := 3,1415;
6       start
7           read(r);
8
9           A := 0;
10          A := r * r * pi;
11
12          write({A area do circulo e: });
13          write(A);
14      stop
```

```
ntic@ntic-OptiPlex-7010:~/Área de Trabalho/Natha/Compilador/kelang/Code$ java kelang teste7.txt
("app", APP)
02: Lexema inválido [testevii]
ntic@ntic-OptiPlex-7010:~/Área de Trabalho/Natha/Compilador/kelang/Code$
```

A gramática não aceita identificadores com mais de uma letra, colocamos um caractere "_" para validar a identificação de "testevii".

```
ntic@ntic-OptiPlex-7010:~/Área de Trabalho/Natha/Compilador/kelang/Code$ java kelang teste7.txt
("app", APP)
("_testevii", ID)
03: Lexema inválido [int]
ntic@ntic-OptiPlex-7010:~/Área de Trabalho/Natha/Compilador/kelang/Code$
```

"int" não é uma palavra reservada da gramática, mas sim "integer".



Sucesso no teste 7.

TESTE 8

```
1   app teste8
2       start
3           %Programa Comentario, vamos ver se dá Comentario
4           %stop
5       stop
6       _xd
7       6.1
8       7
9       {arroz}
```

```
ntic@ntic-OptiPlex-7010:~/Área de Trabalho/Natha/Compilador/kelang/Code$ java kelang teste8.txt
("app", APP)
01: Lexema inválido [teste]
ntic@ntic-OptiPlex-7010:~/Área de Trabalho/Natha/Compilador/kelang/Code$
```

gramática não aceita identificadores com mais de uma letra, colocamos um caractere "_" para validar a identificação de "teste8".

```
ntic@ntic-OptiPlex-7010:~/Área de Trabalho/Natha/Compilador/kelang/Code$ java kelang teste8.txt
("app", APP)
("_teste8", ID)
("start", START)
("stop", STOP)
("_xd", ID)
("6.1", NUM_REAL)
("7", NUM_INT)
("{arroz}", STRING)
("", END_OF_FILE)
ntic@ntic-OptiPlex-7010:~/Área de Trabalho/Natha/Compilador/kelang/Code$
```

Sucesso no teste 8.