Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação (ICMC)
SCC0605 - Teoria da Computação e Compiladores

Trabalho 1: Análise Léxica

Lucas Furco Granela - 11299978
Matheus Yasuo Ribeiro Utino - 11233689
Pedro Ribas Serras - 11234328
Vinícius Silva Montanari - 11233709

Docente: Dr. Thiago A. S. Pardo

São Carlos
05 de junho de 2022

Sumário

1 Introdução

Esse projeto se destina a construção de um analisador léxico, a primeira etapa de um compilador, responsável por ler o arquivo como código-fonte realizado na linguagem P– e identificar os tokens/símbolos correspondentes, além de poder tratar alguns erros de forma básica.

Este documento exibe as decisões de projeto e os procedimentos adotados pelo grupo para o desenvolvimento do analisador, onde sua construção pode ser dividida em três etapas: decisões de projeto, construção dos autômatos e codificação.

2 Construção dos autômatos

Após ter decisões de projeto bem definidas, é necessário construir os autômatos que vão reconhecer as cadeias, para que assim seja possível codifica-las.

2.1 Autômato para reconhecimento de Identificadores

Para um correto reconhecimento dos identificadores pelo autômato é necessário que o primeiro caractere sempre seja uma letra. Caso contrário, o autômato entrará em um estrado de erro, acusando que o identificador não começa com uma letra.

Em seguida, o autômato aceita tanto letras quando dígitos, transicionando para o estado de aceitação quando algum outro um símbolo conhecido pela linguagem (exceto letras e números) é inserido, ou caso contrário, para um estado de erro que acusa que o identificador é mal formado.

Caso nenhum erro ocorrer, é feito uma busca na tabela de palavras reservadas, para conferir se a palavra reconhecida não é reservada pelo compilador. Caso for encontrada a palavra na tabela, ela será considerada como uma palavra reservada, do contrário, um identificador.

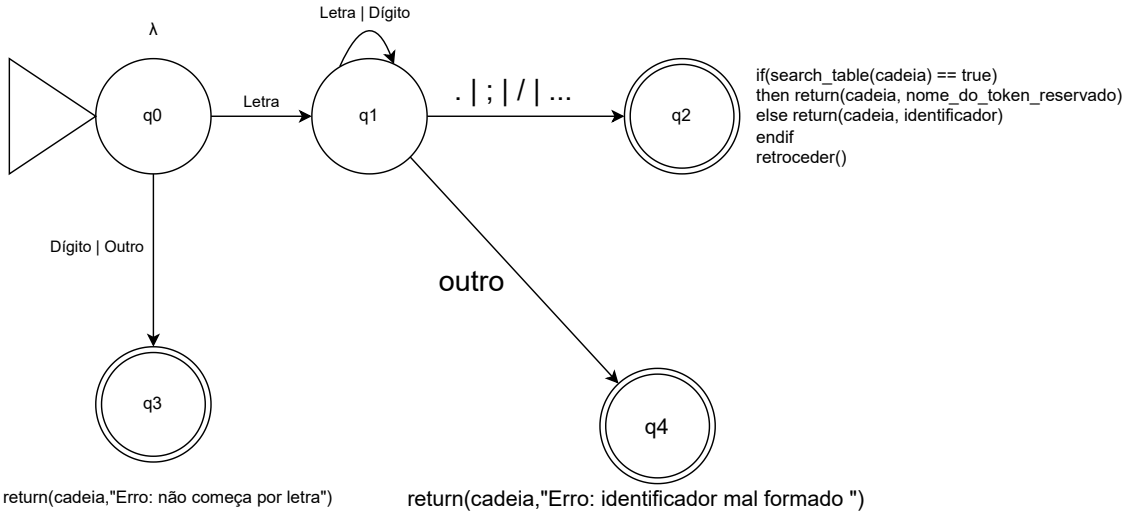

Figura 1: Autômato para reconhecimento de identificadores.

2.2 Autômato para reconhecimento de números inteiros e reais

Esse autômato define regras de formatação e classificação dos números inseridos, onde duas classificações são possíveis: números inteiros e números reais. Sendo que ambos os números podem aceitar símbolos negativos e positivos (+ e -).

O autômato inicialmente em q_0 transiciona para o próximo estado caso um digito for consumido, ou para erro no caso de um outro símbolo consumido. Em seguida, em q_1 o autômato passa a consumir qualquer dígito, transicionando para um estado de aceitação q_8, que reconhece a cadeia como um número inteiro quando um símbolo reconhecido pela linguagem (exceto letras), para q_2 quando um ponto for consumido, passando a reconhecer a cadeia apenas como real, ou para um estado de erro q_5 quando um outro símbolo é inserido. Em q_2, após o ponto, o autômato passa a esperar a inserção de um dígito para passar ao estado q_6, transicionando para erro no caso de não existir um número após o ponto. Por fim, em q_6, o autômato consome qualquer dígito e transiciona para q_7 quando um símbolo reconhecido pela linguagem for reconhecido (exceto letras novamente), ou para um estado de erro quando algum outro símbolo for inserido.

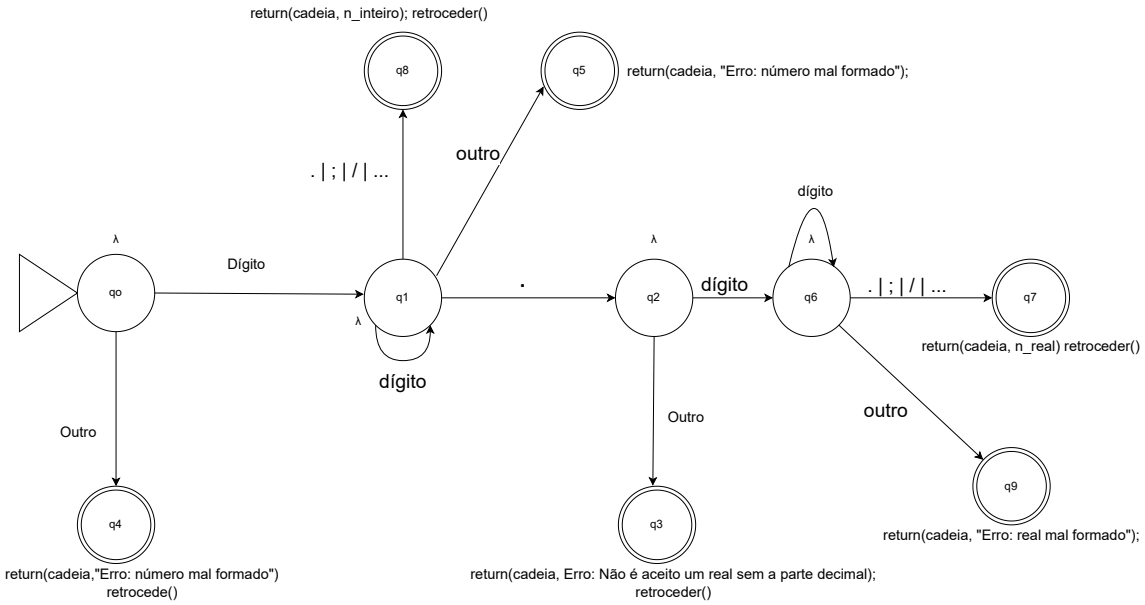

Figura 2: Autômato para reconhecimento de números reais e inteiros.

2.3 Autômato para reconhecimento de comentários

Esse autômato reconhece e estabelece a formatação para o uso do comentário. Inicialmente o autômato aguarda a abertura do comentário, dada pelo símbolo de chave a direita '{', caso o contrário é emitada uma mensagem de erro indicando que o comentário não foi aberto. Em seguida, o autômato começa a consumir todos os símbolos diferentes de '{' e ' n'. Após isso, caso seja encontrado um ' n' indica que o comentário não foi corretamente fechado e então é exibido uma mensagem de erro. Por fim, caso seja identificado o símbolo '}' significa que o comentário foi fechado e portanto o comentário reconhecido sem erros.

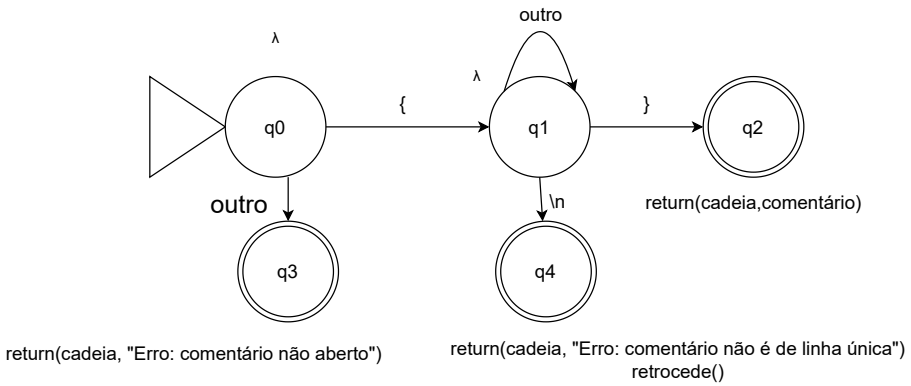

Figura 3: Autômato para reconhecimento de comentários.

2.4 Autômato para reconhecimento de operadores

O autômato para reconhecimento de operadores tem um funcionamento simples e uma grande quantidade de estados.

Inicialmente, caso um identificador único seja reconhecido, ou seja, qualquer identificador que não tenha composição com outro símbolo (como '+', '-', ',' entre outros), a cadeia já é aceita. Enquanto em casos que um símbolo que possua uma combinação (como '>', '<', ':', entre outros) o autômato transiciona para um estado em que, ou é consumido um símbolo que exista combinação com o consumido anteriormente (no caso do ' ' o '=' por exemplo), ou se consome qualquer outro símbolo, indicando que o identificador é único.

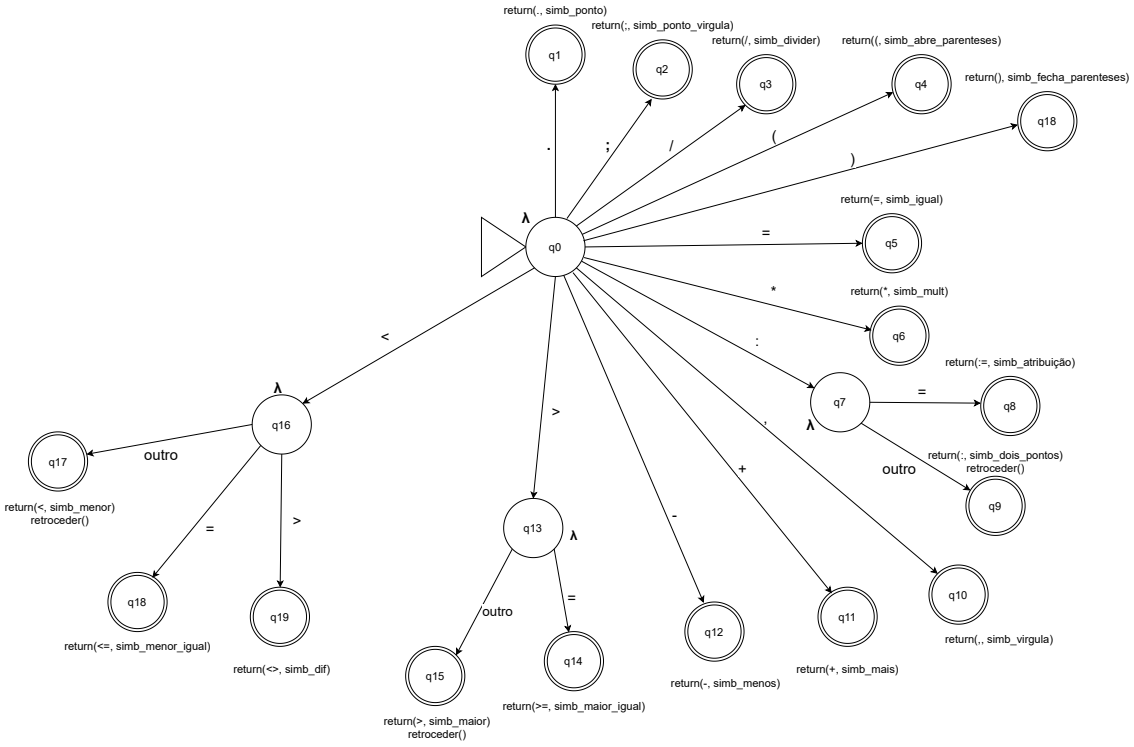

Figura 4: Autômato para reconhecimento de operadores

2.5 Autômato final

O autômato final se trata de uma junção de todos os autômatos descritos anteriormente em um único autômato, com algumas adaptações.

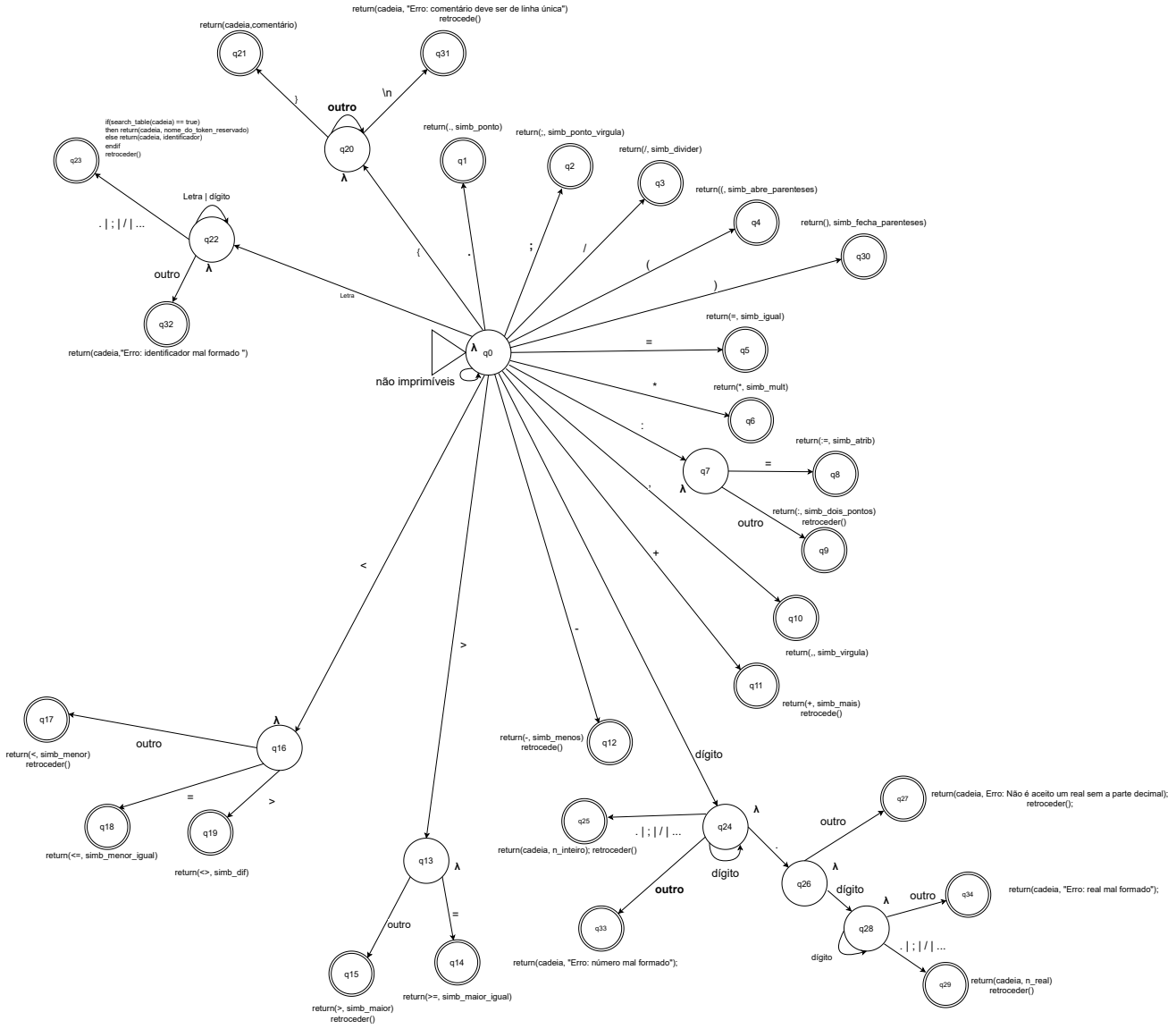

Figura 5: Autômato final

3 Decisões de projeto

Algumas decisões foram tomadas para que fosse possível fazer o analisador sintático. Inicialmente, juntamos todos os autômatos em um grande autômato para facilitar o desenvolvimento. A partir desse grande automato foi feita a tabela de transições. Essa tabela abrange todas as transições e apenas as transições. Ela foi feita usando uma matriz de inteiros sendo que o índice da linha refere-se

ao estado e o índice da coluna a todos os símbolos da tabela ASCII. Para saber se os estados eram finais, se tinha que retroceder e a mensagem que eles retornavam, foi feito um *array* de estados, onde o índice é relativo ao estado, sendo que foi criado uma *structure* para guardar as informações de um estado. Também foi feito um *array* de strings para armazenar as palavras reservadas da linguagem, já que são apenas 17 para o p– (contando com o for).

Para tratar os erros com mais precisão e menos generalidade, foi criado um array com os caracteres que são reconhecidos pelo estado inicial do automato. Tendo esses caracteres, em alumas transições de "outros", foi trocado para por duas transições, sendo considerado um estado final sem erro quando o elemento de parada era um dos elementos reconhecidos pelo automato principal e reconhecido como erro quando não reconhecido pelo estado inicial. Por exemplo, para reconhecer o identificador, é necessário uma transição de parada "outro", então foram criados dois estados finais, uma de erro e outro não. O que não é de erro é reconhecido quando o simbolo de parada também é reconhecido pelo estado inicial do automato, como "+", -", ",", ".", etc. E o de erro é acontece com o novo "outro", desconsiderando os elementos reconhecidos pelo automato inicial.

Os caracteres não reconhecidos pelo automato principal não geram um estado no automato, mas no código eles são considerados como estado -1, sendo que eles retornam "Erro: carácter não reconhecido". Por facilidade, os símbolos não imprimíveis ('$\backslash t$', '$\backslash n$' e ' '), também são considerados estados -1, mas eles em vez retornar esse erro, eles são simplesmente ignorados (isso no estado inicial 0).

4 Compilação do código

É necessário alguns requisitos para a compilação do código:

- gcc
- make

Com os requisitos satisfeitos, para compilar o código fonte a partir do Windows, Linux e Mac, basta caminhar até a pasta raiz do projeto pelo terminal e executar o seguinte comando:

Em seguida para executar o código, basta usar o seguinte comando:

```
make
```

Código 1: Compilação do código

```
make run
```

Código 2: Execução do código

5 Exemplos de uso

Com o código compilado, podemos realizar alguns testes para testar o comportamento do analisador léxico construído. Abaixo são exibidos dois exemplos. O exemplo 1 consiste de uma entrada que não possuí erros, enquanto a segunda contém erros léxicos.

5.1 Exemplo 1 - Sem erros

```
program leimprime;
{exemplo 1}
var a: real;
var b: integer;
procedure nomep(x: real);
var a, c: integer;
begin
read(c, a);
if a<x+c then
begin
a:= c+x;
write(a);
end
else c:= a+x;
end;
begin {programa principal}
read(b);
nomep(b);
end.
```

Código 3: Entrada - Exemplo 1

```
program, program
leimprime, identificador
;, simb_ponto_virgula
{exemplo 1}, comentario
var, var
a, identificador
:, simb_dois_pontos
real, real
;, simb_ponto_virgula
var, var
b, identificador
:, simb_dois_pontos
integer, integer
;, simb_ponto_virgula
procedure, procedure
nomep, identificador
(, simb_abre_parenteses
x, identificador
:, simb_dois_pontos
real, real
), simb_fecha_parenteses
;, simb_ponto_virgula
var, var
a, identificador
,, simb_virgula
c, identificador
:, simb_dois_pontos
integer, integer
;, simb_ponto_virgula
begin, begin
read, read
(, simb_abre_parenteses
c, identificador
,, simb_virgula
a, identificador
), simb_fecha_parenteses
;, simb_ponto_virgula
if, if
a, identificador
<, simb_menor
x, identificador
+, simb_mais
c, identificador
then, then
begin, begin
a, identificador
:=, simb_atrib
```

```
c, identificador
+, simb_mais
x, identificador
;, simb_ponto_virgula
write, write
(, simb_abre_parenteses
a, identificador
), simb_fecha_parenteses
;, simb_ponto_virgula
end, end
else, else
c, identificador
:=, simb_atrib
a, identificador
+, simb_mais
x, identificador
;, simb_ponto_virgula
end, end
;, simb_ponto_virgula
begin, begin
{programa principal}, comentario
read, read
(, simb_abre_parenteses
b, identificador
), simb_fecha_parenteses
;, simb_ponto_virgula
nomep, identificador
(, simb_abre_parenteses
b, identificador
), simb_fecha_parenteses
;, simb_ponto_virgula
end, end
., simb_ponto
```

Código 4: Saída - Exemplo 1

5.2 Exemplo 2 - Com erros

```
prog@ram leimprime;
{comentario em uma linha}
{comentario com quebra(invalido) :)
}
@var a: real;
var b: integer;
procedure nomep(x: real);
```

```
vasdfar a, c: integer;
begin
read(c);
a:=23;
a:=23#;
a:=23.;
a:=23.32;
a:=23.32%;
a:=23.3%2
if a<x+c then
begin
a:= c + x;
write(a);
end
else c:= a+x;
end;
begin {programa principal}
read(b);
nomep(b);
end.
```

Código 5: Entrada - Exemplo 2

```
prog@, Erro: identificador mal formado
ram, identificador
leimprime, identificador
;, simb_ponto_virgula
{comentario em uma linha}, comentario
{comentario com quebra(invalido) :), Erro: comentario deve ser de linha unica
}, Caractere invalido
@, Caractere invalido
var, var
a, identificador
:, simb_dois_pontos
real, real
;, simb_ponto_virgula
var, var
b, identificador
:, simb_dois_pontos
integer, integer
;, simb_ponto_virgula
procedure, procedure
nomep, identificador
(, simb_abre_parenteses
x, identificador
:, simb_dois_pontos
```

```
real, real
), simb_fecha_parenteses
;, simb_ponto_virgula
vasdfar, identificador
a, identificador
,, simb_virgula
c, identificador
:, simb_dois_pontos
integer, integer
;, simb_ponto_virgula
begin, begin
read, read
(, simb_abre_parenteses
c, identificador
), simb_fecha_parenteses
;, simb_ponto_virgula
a, identificador
:=, simb_atrib
23, n_inteiro
;, simb_ponto_virgula
a, identificador
:=, simb_atrib
23#, Erro: numero mal formado
;, simb_ponto_virgula
a, identificador
:=, simb_atrib
23., Erro: Nao e aceito um real sem a parte decimal
;, simb_ponto_virgula
a, identificador
:=, simb_atrib
23.32, n_real
;, simb_ponto_virgula
a, identificador
:=, simb_atrib
23.32%, Erro: real mal formado
;, simb_ponto_virgula
a, identificador
:=, simb_atrib
23.3%, Erro: real mal formado
2, n_inteiro
if, if
a, identificador
<, simb_menor
x, identificador
+, simb_mais
c, identificador
then, then
```

```
begin, begin
a, identificador
:=, simb_atrib
c, identificador
+, simb_mais
x, identificador
;, simb_ponto_virgula
write, write
(, simb_abre_parenteses
a, identificador
), simb_fecha_parenteses
;, simb_ponto_virgula
end, end
else, else
c, identificador
:=, simb_atrib
a, identificador
+, simb_mais
x, identificador
;, simb_ponto_virgula
end, end
;, simb_ponto_virgula
begin, begin
{programa principal}, comentario
read, read
(, simb_abre_parenteses
b, identificador
), simb_fecha_parenteses
;, simb_ponto_virgula
nomep, identificador
(, simb_abre_parenteses
b, identificador
), simb_fecha_parenteses
;, simb_ponto_virgula
end, end
., simb_ponto
```

Código 6: Saída - Exemplo 2