

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

**DANIEL DE PAULA BRAGA LOPES
GUILHERME FERNANDES MARCHEZINI
LAURO CÉSAR JACQUES SANTOS**

Relatório Analisador Sintático

Belo Horizonte
2017

1. Forma de Utilização

A execução do compilador é feita via terminal do Linux utilizando o .jar que está localizado na pasta dist passando o nome do .jar e o arquivo de entrada que será compilado, da seguinte forma:

```
java -jar Compilador.jar <nome_arquivo>
```

Abaixo um exemplo de como compilar o primeiro código de teste disponibilizado na especificação do trabalho:

```
java -jar Compilador.jar ../test/teste1.tst
```

2. Modificações

A gramática original da linguagem apresenta prefixos comuns e recursão à esquerda, elementos que impossibilitam a implementação de um parser LL(1) para a mesma. A seguir a gramática modificada, solucionando tais problemas e reescrita no formato BNF:

```
program'          ::= program "$"
program           ::= init decl-stmt-list stop
decl-stmt-list    ::= "id" assign-or-decl | stmt-no-assign ";" stmt-list-tail
assign-or-decl    ::= "!=" simple-expr ";" stmt-list-tail
                  | ident-list-tail is type ";" decl-stmt-list-tail
stmt-no-assign    ::= if-stmt | do-stmt | read-stmt | write-stmt
decl-stmt-list-tail ::= decl-stmt-list | λ
ident-list-tail   ::= "," "id" ident-list-tail | λ
type              ::= integer | string
stmt-list         ::= stmt ";" stmt-list-tail
stmt-list-tail    ::= stmt ";" stmt-list-tail | λ
stmt              ::= assign-stmt | if-stmt | do-stmt
                  | read-stmt | write-stmt
assign-stmt       ::= "id" "!=" simple_expr
if-stmt           ::= if "(" condition ")" begin stmt-list end if-suffix
if-suffix         ::= else begin stmt-list end | λ
condition         ::= expression
do-stmt           ::= do stmt-list do-suffix
do-suffix         ::= while "(" condition ")"
read-stmt         ::= read "(" "id" ")"
write-stmt        ::= write "(" writable ")"
writable          ::= simple-expr
expression        ::= simple-expr expression-suffix
expression-suffix ::= relop simple-expr | λ
simple-expr        ::= term simple-expr-tail
simple-expr-tail   ::= addop term simple-expr-tail | λ
term              ::= factor-a term-tail
term-tail         ::= mulop factor-a term-tail | λ
factor-a          ::= factor | not factor | "-" factor
factor            ::= "id" | constant | "(" expression ")"
relop             ::= "=" | ">" | ">=" | "<" | "<=" | "<>"
addop             ::= "+" | "-" | or
mulop             ::= "*" | "/" | and
constant          ::= "num" | "literal"
```

Obs.: Algumas produções da gramática original foram substituídas por tokens.

Para a implementação do parser LL(1), foram encontrados os conjuntos FIRST e FOLLOW de todos os símbolos não terminais da gramática:

Símbolo	First	Follow
program	init	\$
decl-stmt-list	id, if, do, read, write	stop
z1	:=, ",", is	stop
z2	if, do, read, write	;
decl-stmt-list-tail	id, if, do, read, write, λ	stop
ident-list-tail	",", λ	is
type	integer, string	;
stmt-list	id, if, do, read, write	end, while
stmt-list-tail	id, if, do, read, write, λ	stop, end, while
stmt	id, if, do, read, write	;
assign-stmt	id	;
if-stmt	if	;
if-suffix	else, λ	;
condition	id, num, literal, (, not, -)
do-stmt	do	;
do-suffix	while	;
read-stmt	read	;
write-stmt	write	;
writable	id, num, literal, (, not, -)
expression	id, num, literal, (, not, -)
expression-suffix	>, =, >=, <, <=, <>, λ)
simple-expr	id, num, literal, (, not, -	;,), >, =, >=, <, <=, <>
simple-expr-tail	or, +, -, λ	;,), >, =, >=, <, <=, <>
term	id, num, literal, (, not, -	or, +, -, ;,), >, =, >=, <, <=, <>
term-tail	*, /, and, λ	or, +, -, ;,), >, =, >=, <, <=, <>
factor-a	id, num, literal, (, not, -	*, /, and, or, +, -, ;,), >, =, >=, <, <=, <>
factor	id, num, literal, (*, /, and, or, +, -, ;,), >, =, >=, <, <=, <>
relop	>, =, >=, <, <=, <>	id, num, (, not, -
addop	or, +, -	id, num, (, not, -
mulop	*, /, and	id, num, (, not, -
constant	num, literal	*, /, and, or, +, -, ;,), >, =, >=, <, <=, <>

3. Implementação

Abaixo uma breve explicação das classes existentes no compilador:

3.1 Lexer

Classe que implementa o analisador léxico. Seu construtor insere as palavras reservadas na tabela de símbolos. Possui um método **scan** que devolve um Token.

3.2 LexicalException

Classe para imprimir na tela o motivo de ocorrer uma determinada exceção. Existem três casos:

- Token inválido: É passado um token inválido ou não esperado;
- Fim de arquivo inesperado: O arquivo termina quando ainda deveria possuir alguma informação;
- Default: Ocorre quando é um erro diferente dos dois anteriores.

3.3 Num

Classe para representar um Token número.

3.4 Tag

Classe que define as constantes para os tokens.

3.5 Token

Representa um Token genérico. Contém a constante que representa o Token.

3.6 Word

Representa um token de palavras reservadas, identificadores e tokens compostos, tais como != e &&.

3.7 Syntaxer

Essa classe implementa completamente o parser LL(1) com todas os módulos necessários, como por exemplo o 'eat' e 'advance', sendo responsável também pela recuperação de erros, que foi implementada usando a heurística dos follows.

3.8 SyntaticException

Classe para imprimir na tela os erros de sintaxe encontrados no programa fonte.

4. Testes

A seguir os testes e seus respectivos resultados:

4.1 Teste 1

Código:

```
init
  a, b, c, result is integer;

  read(a);
  read(c);
  b := 10;
  result := (a * c) / (b + 5 - 345);
  write(result);
stop
```

Resultado:

Análise sintática terminada com sucesso.

4.2 Teste 2

4.2.1 Primeira execução

Código:

```
  a, _valor, b is integer;

init
  read(a);
  b := a * a;
  write(b);
  b = b + a / 2 * (a + 5);
  Write(b);
Stop
```

Resultado:

Erro sintático na linha 1: Token 'a' inesperado. Esperando 'init'.
Análise sintática terminada com erro(s).

4.2.2 Segunda execução

Código:

```
init
  a, _valor, b is integer;

  read(a);
  b := a * a;
  write(b);
  b = b + a / 2 * (a + 5);
  Write(b);
stop
```

Resultado:

Erro sintático na linha 2: Token '_' inesperado. Esperando 'ID'.
Erro sintático na linha 7: Token '=' inesperado. Esperando ':='.
Erro sintático na linha 10: Fim de arquivo inesperado. Esperando 'ID', 'NUM', 'STRING', '(', 'not', '-'.
Análise sintática terminada com erro(s).

4.2.3 Terceira execução

Código:

```
init
  a, valor, b is integer;

  read(a);
  b := a * a;
  write(b);
  b := b + a / 2 * (a + 5);
  Write(b);
stop
```

Resultado:

Análise sintática terminada com sucesso.

4.3 Teste 3

4.3.1 Primeira execução

Código:

```
{ Programa de Teste
Calculo de idade }
init
  cont_ is int;
  media, idade, soma_ is integer;
begin
  cont_ = 5;
  soma = 0;
do
  write("Altura:" );
  read (altura);
  soma := soma altura;
  cont_ := cont_ - 1;
while(cont_ > 0)
write("Media: ");
write (soma / qtd);
stop
```

Resultado:

Erro sintático na linha 4: Token 'int' inesperado. Esperando 'integer', 'string'.
Erro sintático na linha 6: Token 'begin' inesperado. Esperando 'ID', 'if', 'do', 'read', 'write'.
Análise sintática terminada com erro(s).

4.3.2 Segunda execução

Código:

```
{ Programa de Teste
Calculo de idade }
init
  cont_ is integer;
  media, idade, soma_ is integer;

  cont_ = 5;
  soma = 0;
do
  write("Altura:" );
  read (altura);
  soma := soma altura;
  cont_ := cont_ - 1;
while(cont_ > 0)
write("Media: ");
write (soma / qtd);
stop
```

Resultado:

Erro sintático na linha 7: Token '=' inesperado. Esperando ':=', ',', 'is'.
Análise sintática terminada com erro(s).

4.3.3 Terceira execução

Código:

```
{ Programa de Teste
Calculo de idade }
init
    cont_ is integer;
    media, idade, soma_ is integer;

    cont_ := 5;
    soma = 0;
    do
        write("Altura:" );
        read (altura);
        soma := soma altura;
        cont_ := cont_ - 1;
    while(cont_ > 0)
    write("Media: ");
    write (soma / qtd);
stop
```

Resultado:

Erro sintático na linha 8: Token '=' inesperado. Esperando ':='.
Erro sintático na linha 12: Token 'altura' inesperado. Esperando '*', '/', 'and', 'or', '+', '-', ';', ')', '>', '=', '>=', '<', '<=', '<>'.
Erro sintático na linha 14: Token 'while' inesperado. Esperando 'stop'.
Análise sintática terminada com erro(s).

4.3.4 Quarta execução

Código:

```
{ Programa de Teste
Calculo de idade }
init
    cont_ is integer;
    media, idade, soma_ is integer;

    cont_ := 5;
    soma := 0;
    do
        write("Altura:");
        read(altura);
        soma := soma + altura;
        cont_ := cont_ - 1;
    while(cont_ > 0)
    write("Media: ");
    write (soma / qtd);
stop
```

Resultado:

Erro sintático na linha 15: Token 'write' inesperado. Esperando ';'.
Análise sintática terminada com erro(s).

4.3.4 Quinta execução

Código:

```
{ Programa de Teste
Calculo de idade }
init
    cont_ is integer;
    media, idade, soma_ is integer;

    cont_ := 5;
    soma := 0;
do
    write("Altura:");
    read(altura);
    soma := soma + altura;
    cont_ := cont_ - 1;
while(cont_ > 0);
write("Media: ");
write (soma / qtd);
stop
```

Resultado:

Análise sintática terminada com sucesso.

4.4 Teste 4

4.4.1 Primeira execução

Código:

```
init
    i, j, k, total, lsoma is integer

    read(I);
    k := i * (5 - i * 50 / 10);
    j := i * 10;
    k := i * j / k;
    k := 4 + a;
    write(i);
    write(j);
    write(k);
```

Resultado:

Erro sintático na linha 2: Token '1' inesperado. Esperando 'ID'.
Erro sintático na linha 4: Token 'read' inesperado. Esperando ';'.'
Erro sintático na linha 5: Token ';' inesperado. Esperando '>', '=', '>=', '<', '<=', '<>', ')'.
Erro sintático na linha 12: Fim de arquivo inesperado. Esperando 'ID', 'if', 'do', 'read', 'write', 'end', 'while', 'stop'.

4.4.2 Segunda execução

Código:

```
init
  i, j, k, total, soma is integer;

  read(I);
  k := i * (5 - i * 50 / 10);
  j := i * 10;
  k := i * j / k;
  k := 4 + a;
  write(i);
  write(j);
  write(k);
stop
```

Resultado:

Análise sintática terminada com sucesso.

4.5 Teste 5

4.5.1 Primeira execução

Código:

```
init
// Programa com if

  j, k, m is integer;
  a, j is string;

  read(j);
  read(k);

  if (j == "ok")
  begin
    result = k/m
  end
  else
  begin
    result := 0;
    write ("Invalid entry");
  end

  write(result);
```

Resultado:

Erro sintático na linha 10: Token '=' inesperado. Esperando 'ID', 'NUM', 'STRING', '(', 'not', '-'.
Erro sintático na linha 12: Token '=' inesperado. Esperando ':='.
Erro sintático na linha 20: Token 'write' inesperado. Esperando 'else', ';'.
Erro sintático na linha 21: Fim de arquivo inesperado. Esperando 'ID', 'if', 'do', 'read', 'write', 'end', 'while', 'stop'.
Análise sintática terminada com erro(s).

4.5.2 Segunda execução

Código:

```
init
// Programa com if
  j, k, m is integer;
  a, j is string;

  read(j);
  read(k);

  if (j = "ok")
  begin
    result := k/m
  end
  else
  begin
    result := 0;
    write ("Invalid entry");
  end;

  write(result);
stop
```

Resultado:

Erro sintático na linha 13: Token 'end' inesperado. Esperando '*', '/', 'and', 'or', '+', '-', ';', ')', '>', '=', '>=', '<', '<=', '<>'.
Análise sintática terminada com erro(s).

4.5.3 Terceira execução

Código:

```
init
// Programa com if
  j, k, m is integer;
  a, j is string;

  read(j);
  read(k);

  if (j = "ok")
  begin
    result := k/m
  end
  else
  begin
    result := 0;
    write ("Invalid entry");
  end;

  write(result);
stop
```

Resultado:

Análise sintática terminada com sucesso.

4.6 Teste 6

4.6.1 Primeira execução

Código:

```
init
  a, b, c, maior is integer;

  read(a);
  read(b);
  read(c;

  maior := 0;
  if ( a>b and a>c )
    maior := a;

  else
    if (b>c)
      maior := b;

    else
      maior := c;

  write("Maior idade: ");
  write(maior);
end
```

Resultado:

Erro sintático na linha 6: Token ';' inesperado. Esperando ')'.
Erro sintático na linha 10: Token 'maior' inesperado. Esperando ';'.
Erro sintático na linha 12: Token 'else' inesperado. Esperando 'ID', 'if', 'do',
'read', 'write', 'end', 'while', 'stop'.
Erro sintático na linha 22: Token 'end' inesperado. Esperando 'stop'.
Análise sintática terminada com erro(s).

4.6.2 Segunda execução

Código:

```
init
  a, b, c, maior is integer;

  read(a);
  read(b);
  read(c);

  maior := 0;
  if ( a>b and a>c )
  begin
    maior := a;
  end
  else
    if (b>c)
      maior := b;

    else
      maior := c;

  write("Maior idade: ");
  write(maior);
stop
```

Resultado:

Erro sintático na linha 9: Token '>' inesperado. Esperando ')'.
Erro sintático na linha 14: Token 'if' inesperado. Esperando 'begin'.
Erro sintático na linha 24: Fim de arquivo inesperado. Esperando 'ID', 'if', 'do',
'read', 'write'.
Análise sintática terminada com erro(s).

4.6.3 Terceira execução

Código:

```
init
  a, b, c, maior is integer;

  read(a);
  read(b);
  read(c);

  maior := 0;
  if ((a > b) and (a > c))
  begin
    maior := a;
  end
  else
  begin
    if (b > c)
      maior := b;

    else
      maior := c;
    end;
  end;

  write("Maior idade: ");
  write(maior);
stop
```

Resultado:

Erro sintático na linha 16: Token 'maior' inesperado. Esperando 'begin'.
Erro sintático na linha 24: Fim de arquivo inesperado. Esperando 'ID', 'if', 'do',
'read', 'write'.
Análise sintática terminada com erro(s).

4.6.4 Quarta execução

Código:

```
init
  a, b, c, maior is integer;

  read(a);
  read(b);
  read(c);

  maior := 0;
  if ((a > b) and (a > c))
  begin
    maior := a;
  end
  else
  begin
    if (b > c)
    begin
      maior := b;
    end
    else
      maior := c;
    end;

  write("Maior idade: ");
  write(maior);
stop
```

Resultado:

Erro sintático na linha 20: Token 'maior' inesperado. Esperando 'begin'.
Erro sintático na linha 26: Fim de arquivo inesperado. Esperando 'ID', 'if', 'do',
'read', 'write'.
Análise sintática terminada com erro(s).

4.6.5 Quinta execução

Código:

```
init
  a, b, c, maior is integer;

  read(a);
  read(b);
  read(c);

  maior := 0;
  if ((a > b) and (a > c))
  begin
    maior := a;
  end
  else
  begin
    if (b > c)
    begin
      maior := b;
    end
    else
    begin
      maior := c;
    end;
  end;

  write("Maior idade: ");
  write(maior);
stop
```

Resultado:

Análise sintática terminada com sucesso.

4.7 Teste 7

4.7.1 Primeira execução

Código:

```
init
  a is int;

  begin
    read(A);

  DO
    A := A - 2
  WHILE (A >= 2);

  iF (a = 0)
    write(A);
    write(" é par");
  ELSE
    write(" é ímpar.");
```

Resultado:

Erro sintático na linha 2: Token 'int' inesperado. Esperando 'integer', 'string'.
Erro sintático na linha 4: Token 'begin' inesperado. Esperando 'ID', 'if', 'do', 'read', 'write'.
Erro sintático na linha 16: Fim de arquivo inesperado. Esperando 'stop'.
Análise sintática terminada com erro(s).

4.7.2 Segunda execução

Código:

```
init
  a is integer;

  read(A);

  DO
    A := A - 2
  WHILE (A >= 2);

  iF (a = 0)
    write(A);
    write(" é par");
  ELSE
    write(" é ímpar.");
stop
```

Resultado:

Erro sintático na linha 8: Token 'while' inesperado. Esperando '*', '/', 'and', 'or', '+', '-', ';', ')', '>', '=', '>=', '<', '<=', '<>'.
Erro sintático na linha 8: Token '>=' inesperado. Esperando ';'.
Erro sintático na linha 11: Token 'write' inesperado. Esperando 'begin'.
Erro sintático na linha 15: Fim de arquivo inesperado. Esperando 'ID', 'if', 'do', 'read', 'write'.
Análise sintática terminada com erro(s).

4.7.3 Terceira execução

Código:

```
init
  a is integer;

  read(A);

  DO
    A := A - 2;
  WHILE (A >= 2);

  iF (a = 0)
  begin
    write(A);
    write(" é par");
  end
  ELSE
    write(" é ímpar.");
stop
```

Resultado:

Erro sintático na linha 16: Token 'write' inesperado. Esperando 'begin'.
Erro sintático na linha 17: Fim de arquivo inesperado. Esperando 'ID', 'if', 'do', 'read', 'write'.
Análise sintática terminada com erro(s).

4.7.4 Quarta execução

Código:

```
init
  a is integer;

  read(A);

  DO
    A := A - 2;
  WHILE (A >= 2);

  iF (a = 0)
  begin
    write(A);
    write(" é par");
  end
  ELSE
  begin
    write(" é ímpar.");
  end;
stop
```

Resultado:

Análise sintática terminada com sucesso.

4.8 Teste 8

4.8.1 Primeira execução

Código:

```
init
  n is integer;
  anterior, proximo, aux, i is integer;

  begin
    write("Digite a posicao: ");
    read(n);

    if (n == 1)

      proximo := 0;

    else

      if (n == 2)
        begin
          proximo := 1;

        else

          anterior := 1;
          proximo := 1
          i := 3;
          do
            aux := proximo;
            proximo := anterior + proximo;
            anterior := aux;
            i := i + 1;
          while (i < n)

        write("O termo: ");
        write(proximo);
stop
```

Resultado:

Erro sintático na linha 5: Token 'begin' inesperado. Esperando 'ID', 'if', 'do', 'read', 'write'.

Análise sintática terminada com erro(s).

4.8.2 Segunda execução

Código:

```
init
  n is integer;
  anterior, proximo, aux, i is integer;

  write("Digite a posicao: ");
  read(n);

  if (n == 1)

    proximo := 0;

  else

    if (n == 2)
      begin
        proximo := 1;

      else

        anterior := 1;
        proximo := 1
        i := 3;
        do
          aux := proximo;
          proximo := anterior + proximo;
          anterior := aux;
          i := i + 1;
        while (i < n)

      write("O termo: ");
      write(proximo);
stop
```

Resultado:

Erro sintático na linha 8: Token '=' inesperado. Esperando 'ID', 'NUM', 'STRING', '(', 'not', '- '.

Erro sintático na linha 10: Token 'proximo' inesperado. Esperando 'begin'.

Erro sintático na linha 18: Token 'else' inesperado. Esperando 'ID', 'if', 'do', 'read', 'write', 'end', 'while', 'stop'.

Erro sintático na linha 28: Token 'while' inesperado. Esperando 'end'.

Erro sintático na linha 34: Fim de arquivo inesperado. Esperando 'else', ';'.

Análise sintática terminada com erro(s).

4.8.3 Terceira execução

Código:

```
init
  n is integer;
  anterior, proximo, aux, i is integer;

  write("Digite a posicao: ");
  read(n);

  if (n = 1)
  begin
    proximo := 0;
  end
  else

    if (n == 2)
    begin
      proximo := 1;
    end
    else

      anterior := 1;
      proximo := 1
      i := 3;
      do
        aux := proximo;
        proximo := anterior + proximo;
        anterior := aux;
        i := i + 1;
      while (i < n)

    write("O termo: ");
    write(proximo);
  stop
```

Resultado:

Erro sintático na linha 14: Token 'if' inesperado. Esperando 'begin'.
Erro sintático na linha 18: Token 'else' inesperado. Esperando ';'.'
Erro sintático na linha 22: Token 'i' inesperado. Esperando '*', '/', 'and', 'or', '+', '-', ';', ')', '>', '=', '>=', '<', '<=', '<>'.
Erro sintático na linha 31: Token 'write' inesperado. Esperando ';'.'
Análise sintática terminada com erro(s).

4.8.4 Quarta execução

Código:

```
init
  n is integer;
  anterior, proximo, aux, i is integer;

  write("Digite a posicao: ");
  read(n);

  if (n = 1)
  begin
    proximo := 0;
  end
  else
  begin
    if (n == 2)
    begin
      proximo := 1;
    end
    else
      anterior := 1;
      proximo := 1;
      i := 3;
      do
        aux := proximo;
        proximo := anterior + proximo;
        anterior := aux;
        i := i + 1;
      while (i < n);

      write("O termo: ");
      write(proximo);
  stop
```

Resultado:

Erro sintático na linha 14: Token '=' inesperado. Esperando 'ID', 'NUM', 'STRING', '(', 'not', '-'.
Erro sintático na linha 20: Token 'anterior' inesperado. Esperando 'begin'.

Erro sintático na linha 34: Fim de arquivo inesperado. Esperando 'ID', 'if', 'do', 'read', 'write'.

Análise sintática terminada com erro(s).

4.8.5 Quinta execução

Código:

```
init
  n is integer;
  anterior, proximo, aux, i is integer;

  write("Digite a posicao: ");
  read(n);

  if (n = 1)
  begin
    proximo := 0;
  end
  else
  begin
    if (n == 2)
    begin
      proximo := 1;
    end
    else
      anterior := 1;
      proximo := 1;
      i := 3;
      do
        aux := proximo;
        proximo := anterior + proximo;
        anterior := aux;
        i := i + 1;
      while (i < n);

      write("O termo: ");
      write(proximo);
stop
```

Resultado:

Erro sintático na linha 33: Token 'stop' inesperado. Esperando 'end'.
Erro sintático na linha 34: Fim de arquivo inesperado. Esperando ';'.
Análise sintática terminada com erro(s).

4.8.6 Sexta execução

Código:

```
init
  n is integer;
  anterior, proximo, aux, i is integer;

  write("Digite a posicao: ");
  read(n);

  if (n = 1)
  begin
    proximo := 0;
  end
  else
  begin
    if (n = 2)
    begin
      proximo := 1;
    end
    else
    begin
      anterior := 1;
      proximo := 1;
      i := 3;
      do
        aux := proximo;
        proximo := anterior + proximo;
        anterior := aux;
        i := i + 1;
      while (i < n);
    end;
  end;

  write("O termo: ");
  write(proximo);
stop
```

Resultado:

Análise sintática terminada com sucesso.