TRABALHO DE COMPILADORES
Análise Sintática

Grupo: Ana Cláudia Gomes de Mendonça, Bruno Marques Maciel e Matheus Martins Aguiar

1. INTRODUÇÃO

Este relatório contém os detalhes a respeito da segunda etapa do trabalho da disciplina de Compiladores, realizado no primeiro semestre de 2017. Esta etapa aborda o Analisador sintático deste compilador, mostrando suas características.

O compilador em questão está sendo implementado em JAVA.

2. COMO USAR O COMPILADOR

Bom, caso deseje utilizar o código não compilado, basta alterar o caminho na linha 26 do arquivo Main.java para o caminho completo do arquivo texto com o código a ser compilado e não informar nenhum parâmetro ao executar o compilador.

Porém, caso deseje utilizar o arquivo JAR, basta colocar o código a ser analisado no arquivo entrada.txt que se encontra na mesma pasta do arquivo JAR. Nesse caso, ao executar o programa, não é preciso informar nenhum parâmetro na linha de comando.

Para a execução do JAR via terminal do Linux, basta utilizar o seguinte comando: java -jar NomeDoJar.jar 'parâmetro' , de modo que o parâmetro deve ser o caminho do arquivo de texto que será utilizado como entrada (caso deseje informá-lo, senão lembre-se que o arquivo "entrada.txt" será considerado como entrada default, daí basta executar com java -jar NomeDoJar.jar).

3. IMPLEMENTAÇÃO

Para o desenvolvimento desta etapa do trabalho foi necessário a eliminação de toda recursão à esquerda das regras de linguagem, assim como a identificação dos conjuntos first e follows da mesma, uma vez que são necessários para que se identifique qual regra será acionada por aquele token. Tendo isto em mente, implementamos a classe **Syntactic.java**, a qual possui um método para cada símbolo não terminal listado na tabela de First e Follow, dentro de cada um deles há um switch-case que é responsável por definir a ação a ser tomada com base no token identificado pelo lexer.

Como há duas regras no método responsável pela regra inicial que são acionadas pelo mesmo first foi necessário a leitura de mais tokens para que a decisão fosse tomada, uma vez que o first não é capaz de determinar isso.

1. Gramática Original

1) program ::= init [decl-list] stmt-list stop
2) decl-list ::= decl ";" { decl ";"}

```
3)  decl ::= ident-list is type
4)  ident-list ::= identifier {"," identifier}
5)  type ::= integer |
             string
6)  stmt-list ::= stmt ";" { stmt ";"}
7)  stmt ::= assign-stmt |
             if-stmt |
             do-stmt |
             read-stmt |
             write-stmt
8)  assign-stmt ::= identifier ":=" simple_expr
9)  if-stmt ::= if "(" condition ")" begin stmt-list end |
                if "(" condition ")" begin stmt-list end else begin stmt-list end
10) condition ::= expression
11) do-stmt ::= do stmt-list do-suffix
12) do-suffix ::= while "(" condition ")"
13) read-stmt ::= read "(" identifier ")"
14) write-stmt ::= write "(" writable ")"
15) writable ::= simple-expr
16) expression ::= simple-expr |
                   simple-expr relop simple-expr
17) simple-expr ::= term |
                    simple-expr addop term
18) term ::= factor-a |
             term mulop factor-a
19) factor-a ::= factor |
                 not factor |
                 "-" factor
20) factor ::= identifier |
               constant |
               "(" expression ")"
21) relop ::= "=" |
              ">" |
              ">=" |
              "<" |
              "<=" |
              "<>"
22) addop ::= "+" |
              "-" |
              or
23) mulop ::= "*" |
              "/" |
              and
24) constant ::= integer_const |
                 literal
25) integer_const ::= nozero {digit} |
```

“0”

26) literal ::= " “ " {caractere} " ” "
27) identifier ::= (letter) {letter | digit | " _ " }
28) letter ::= [A-Za-z]
29) digit ::= [0-9]
30) nozero ::= [1-9]
31) caractere ::= um dos 256 caracteres do conjunto ASCII, exceto “ " ” e quebra de linha

2. Casos de Fatoração à Esquerda

- 9) if-stmt ::= if "(" condition ")" begin stmt-list end | if "(" condition ")" begin stmt-list end else begin stmt-list end

  Reescrevendo como:

  9) if-stmt ::= if "(" condition ")" begin stmt-list end if-stmt’
  32) if-stmt’ ::= else begin stmt-list end | λ

- 16) expression ::= simple-expr | simple-expr relop simple-expr

  Reescrevendo como:

  16) expression ::= simple-expr expression'
  33) expression' ::= relop simple-expr | λ

3. Casos de Recursão a Esquerda

- 17) simple-expr ::= term | simple-expr addop term

  Reescrevendo como:

  17) simple-expr ::= term simple-expr'
  34) simple-expr' ::= addop term simple-expr' | λ

- 18) term ::= factor-a | term mulop factor-a

  Reescrevendo como:

  18) term ::= factor-a term'
  35) term' ::= mulop factor-a term' | λ

4. Gramática após alterações

1) program ::= init [decl-list] stmt-list stop

2) decl-list ::= decl ";" { decl ";"}
3) decl ::= ident-list is type
4) ident-list ::= identifier {"," identifier}
5) type ::= integer |
string
6) stmt-list ::= stmt ";" { stmt ";"}
7) stmt ::= assign-stmt |
if-stmt |
do-stmt |
read-stmt |
write-stmt
8) assign-stmt ::= identifier ":=" simple_expr
9) if-stmt ::= if "(" condition ")" begin stmt-list end if-stmt'
10) condition ::= expression
11) do-stmt ::= do stmt-list do-suffix
12) do-suffix ::= while "(" condition ")"
13) read-stmt ::= read "(" identifier ")"
14) write-stmt ::= write "(" writable ")"
15) writable ::= simple-expr
16) expression ::= simple-expr expression'
17) simple-expr ::= term simple-expr'
18) term ::= factor-a term'
19) factor-a ::= factor |
not factor |
"-" factor
20) factor ::= identifier |
constant |
"(" expression ")"
21) relop ::= "=" | ">" | ">=" | "<" | "<=" | "<>"
22) addop ::= "+" |
"-" |
or
23) mulop ::= "*" |
"/" |
and
24) constant ::= integer_const |
literal
25) integer_const ::= nozero {digit} |
“0”
26) literal ::= " “ " {caractere} " ” "
27) identifier ::= (letter) {letter | digit | " _ " }
28) letter ::= [A-Za-z]
29) digit ::= [0-9]
30) nozero ::= [1-9]
31) caractere ::= um dos 256 caracteres do conjunto ASCII, exceto “ " ” e quebra de linha

32) if-stmt' ::= else begin stmt-list end | λ
33) expression' ::= relop simple-expr | λ
34) simple-expr' ::= addop term simple-expr' | λ
35) term' ::= mulop factor-a term' | λ

5. Tabela de FIRST e FOLLOW dos símbolos não-terminais

Símbolo Não Terminal	FIRST	FOLLOW
program	init	$
decl-list	letter	letter,if,do,read,write
decl	letter	;
ident-list	letter	is
type	integer, string	;
stmt-list	letter,if,do,read,write	stop,end,while
stmt	letter,if,do,read,write	;
assign-stmt	letter	;
if-stmt	if	;
condition	not, -, (, letter, nozero, 0, " [abre]	)
do-stmt	do	;
do-suffix	while	;
read-stmt	read	;
write-stmt	write	;
writeable	not, -, (, letter, nozero, 0, " [abre]	)
expression	not, -, (, letter, nozero, 0, " [abre]	)
simple-expr	not, -, (, letter, nozero, 0, " [abre]	),=, <, <=, <>,>,>=
term	not, -, (, letter, nozero, 0, " [abre]	+,-,or
factor-a	not, -, (, letter, nozero, 0, " [abre]	*, /, and
factor	(, letter, nozero, 0, " [abre]	*, /, and
relop	=,>,>=,<,<=,<>	not, -, (, letter, nozero, 0, " [abre]
addop	+,-,or	not, -, (, letter, nozero, 0, " [abre]

mulop	*,/,and	not, -, (, letter, nozero, 0, " [abre]
constant	nozero, 0, " [abre]	*, /, and
integer_const	nozero, 0	*, /, and
literal	" [abre]	*, /, and
identifier	letter	‘,’, letter, is, :=,), *, /, and
if-stmt'	else, λ	;
expression'	λ,=,>,>=,<,<=,<>	)
simple-expr'	+,-,or, λ	),=, <, <=, <>,>,>=
term'	*,/,and, λ	+,-,or

6. Tabela de Parser Preditivo

Conforme tabela Anexa do Excel, podemos perceber que não há células com mais de um elemento e, portanto, o Analisador Sintático Descendente pode ser implementado com sucesso.

7. Alterações adicionais na gramática

Na produção 01 ocorre uma inconsistência para o desenvolvimento do analisador sintático LL(1) que não foi demonstrada na tabela acima devido ao erro ocorrer por uma produção opcional, o que originalmente não observamos.

Para corrigir este erro, as seguintes alterações foram feitas: após o init, se for lido um identificador (letter), caso seja lido "is" ou "," trata-se de uma declaração e, portanto, implementamos todas as produções seguintes que são abertas normalmente na árvore de decl-list após o "is" ou "," até voltar para program; caso seja lido ":=", trata-se de uma atribuição e, portanto, fizemos o mesmo porém para a árvore de stmt-list após o ":=".

4. CASOS DE TESTE

- **Caso de teste 1:**

Original:
```
init
a, b, c, result is integer;
read (a);
read (c);
b := 10;
result := (a * c)/(b + 5 - 345);
```

```
write(result);
stop
```

Saída:
Compilação concluída com sucesso!

- **Caso de teste 2:**

Original:

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

Saída:
Erro na linha 1: Token <id,'a'> não esperado.

Corrigido:

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

Saída:
Erro na linha 2: Token <atr_op,':='> não esperado.

Corrigido:

```
init
a, valor, b is integer;
read (a);
b := a * a;
write (b);
b = b + a/2 * (a + 5);
Write (b);
stop
```

Saída:
Erro na linha 6: Token <rel_op,'='> não esperado.

Corrigido:
```
init
a, valor, b is integer;
read (a);
b := a * a;
write (b);
b := b + a/2 * (a + 5);
Write (b);
stop
```

Saída:

Compilação concluída com sucesso!

- **Caso de teste 3:**

Original:
```
{ Programa de Teste
Calculo de idade }

init
  cont_ is integer;
  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
```

Saída:
Erro na linha 7: Token <begin> não esperado.

Corrigido:

```
{ Programa de Teste
Calculo de idade }

init
  cont_ is integer;
  media, idade, soma_ is integer;
  if (idade > 10)
  begin
    cont_ := 5;
    soma := 0;
    do
      write("Altura:" );
      read (altura);
      soma := soma + altura;
      cont_ := cont_ - 1;
    while(cont_ > 0)

    write("Media: ");
    write (soma / qtd);
  end;
stop
```

Saída:

Erro na linha 18: Token <write> não esperado.

Corrigido:

```
{ Programa de Teste
Calculo de idade }

init
  cont_ is integer;
  media, idade, soma_ is integer;
  if (idade > 10)
  begin
    cont_ := 5;
    soma := 0;
    do
      write("Altura:" );
      read (altura);
      soma := soma + altura;
      cont_ := cont_ - 1;
    while(cont_ > 0);

    write("Media: ");
```

```
   write (soma / qtd);
  end;
stop
```

Saída:
Compilação concluída com sucesso!

- **Caso de teste 4:**

Original:

```
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);
```

Saída:
Erro na linha 4: Token <read> não esperado.

Corrigido:

```
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);
```

Saída:
Erro na linha 4: Token <simb_pont,';'> não esperado.

Corrigido:

```
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);
```

Saída:
Erro na linha 10: Final de arquivo inesperado

Corrigido:

```
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
```

Saída:
Compilação concluída com sucesso!

- **Caso de teste 5:**

Original:

```
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);
```

Saída:
Erro na linha 8: Token <rel_op,'='> não esperado.

Corrigido:

```
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);
```

Saída:
Erro na linha 9: Token <end> não esperado.

Corrigido

```
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);
```

Saída:
Erro na linha 15: Token <write> não esperado.

Corrigido:

```
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);
```

Saída:
Erro na linha 15: Final de arquivo inesperado

Corrigido:

```
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
```

- **Caso de teste 6:**

Original:

```
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
```

Saída:

Erro na linha 5: Token <simb_pont,';'> não esperado.

Corrigido:

```
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
```

Saída:
Erro na linha 7: Token <rel_op,'>'> não esperado.

Corrigido:

```
init
 a, b, c, maior is integer;
 read(a);
 read(b);
 read(c);
 maior := 0;
 if ( a>b and c )
          maior := a;
 else
          if (b>c)
                   maior := b;
          else
                   maior := c;
 write(“Maior idade: ”);
 write(maior);
end
```

Saída:
Erro na linha 8: Token <id,'maior'> não esperado.

Corrigido:

```
init
 a, b, c, maior is integer;
 read(a);
 read(b);
 read(c);
 maior := 0;
 if ( a>b and c ) begin
          maior := a;
 end else
          if (b>c)
                   maior := b;
          else
                   maior := c;
       end;
 write(“Maior idade: ”);
```

```
write(maior);
end
```

Saída:
Erro na linha 10: Token <if> não esperado.

Corrigido:

```
init
 a, b, c, maior is integer;
 read(a);
 read(b);
 read(c);
 maior := 0;
 if ( a>b and c ) begin
          maior := a;
 end else begin
          if (b>c)
                   maior := b;
          else
                   maior := c;
       end;
 write(“Maior idade: ”);
 write(maior);
end
```

Saída:
Erro na linha 11: Token <id,'maior'> não esperado.

Corrigido:

```
init
 a, b, c, maior is integer;
 read(a);
 read(b);
 read(c);
 maior := 0;
 if ( a>b and 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);
end
```

Saída:
Erro na linha 18: Token <end> não esperado.

Corrigido:

```
init
 a, b, c, maior is integer;
 read(a);
 read(b);
 read(c);
 maior := 0;
 if ( a>b and 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
```

Saída:

Compilação concluída com sucesso!

- **Caso de teste 7:**

Original:

```
init

x, y, z is integer;

  read (x);
  read (y);
  z := x+y;
  write("SOMA:");
```

```
  write (z);

end
```

Saída:
Erro na linha 11: Token <end> não esperado.

Corrigido:

```
init

x, y, z is integer;

  read (x);
  read (y);
  z := x+y;
  write("SOMA:");
  write (z);

stop
```

Saída:
Compilação concluída com sucesso!

- **Caso de teste 8:**

Original:

```
init
a, b, c is integer;
  read (a);
  b := a+0;
  write(b);
end
```

Saída:
Erro na linha 6: Token <end> não esperado.

Corrigido:

```
init
a, b, c is integer;
  read (a);
  b := a+0;
  write(b);
```

stop

Saída:
Compilação concluída com sucesso!