Compiladores
2017/01

Relatório 01:
Análise Léxica

Arley Ribeiro
Saulo Antuness
Belo Horizonte, 8 de junho de 2017

1 Como usar o compilador

Pelo terminal, no diretório raiz, execute os comandos abaixo:

- `javac main/Main.java`
  Este comando compila a classe em um bytecode.
- `java main.Main tests/test1.lang`
  Executa o Main criado pela etapa anterior, passando como parâmetro um arquivo de teste do diretório tests.

2 Analisador Sintático

Para implementar o parser para está linguagem utilizaremos o parser recursivo descendente ou preditivo, no entanto, este parser funciona apenas em gramáticas nas quais o primeiro símbolo terminal de cada sub-cadeia provê informação suficiente para escolher a próxima produção a ser utilizada.
No entanto, a gramática apresentada necessita de algumas modificações, pois, quando duas ou mais produções de um símbolo não terminal começam com a mesma forma sentencial, não é claro qual das produções deve ser escolhida. Este problema pode ser resolvido reescrevendo-se as produções de forma a adiar a decisão até que se tenha lido a cadeia de entrada o suficiente, iremos utilizar há fatoração à esquerda, que gera uma transformação na gramática útil na produção de uma gramática adequada para o parser preditivo. Para eliminar a recursão à esquerda, reescrevemos a gramática utilizando se recursão à direita.

2.1 Decisões de Projeto

De acordo com o objetivo definido, de implementar o parser preditivo descendente, além da remoção de recurção à esquerda e a fatoração à esquerda, eliminamos algumas produções da linguagem, pois ambas as produções eram idênticas, portanto, para facilitar a implementação optamos por remover essas produções como writable e condition que ser referiam respectivamente a simple-expr e expression. Na gramática o termo decl-list poderia ser opcional, portanto, inserimos a produção λ em decl-list.

3 Resultados

Considerando a implementação do analisador léxico, nesta fase identificamos os tokens do programa fonte e, para identificar os caracteres como '(' utilizaremos o valor deste na tabela ASCII, com isso, obtivemos a seguinte saída para os testes executados.

3.1 Teste 1

```
1  init
2      a, b, c, result is integer;
3
4      read (a);
5      read (c);
6      b := 10;
7      result := (a * c)/(b + 5 - 345);
8      write(result);
9
10 stop
```

Saída:
Programa correto!

3.2 Teste 2

```
1      a, _valor, b : integer;
2
3  init
4      read (a);
5      b := a * a;
6      write (b);
7      b = b + a/2 * (a + 5);
8      Write (b);
9  stop
```

Saída:
Erro!!! – linha 1 – token inválido: a

Este erro indica que a linguagem não deve ter variáveis declaradas antes da palavra **init**. Corrigindo esse erro e executando novamente.

```
1 init
2     a, _valor, b : integer;
3     read (a);
4     b := a * a;
5     write (b);
6     b = b + a/2 * (a + 5);
7     Write (b);
8 stop
```

Saída:
Erro!!! – linha 2 – token inválido: :

Este erro indica que a linguagem não permite variáveis que começem com _, corrigindo este erro e executando novamente.

Para corrigir este erro substituimos **:** por **is**.

```
1
2 init
3     a, _valor, b : integer;
4     read (a);
5     b := a * a;
6     write (b);
7     b = b + a/2 * (a + 5);
8     Write (b);
9 stop
```

Saída:
Erro!!! – linha 6 – token inválido: =

Nesta linguagem atribuição é representada pelo símbolo **:=**.

```
1 init
2     a, valor, b is integer;
3     read (a);
4     b := a * a;
5     write (b);
6     b := b + a/2 * (a + 5);
7     Write (b);
8 stop
```

Saída:
Programa correto!

3.3 Teste 3

```
1  { Programa de Teste
2  Calculo de idade
3  init
4      cont_ is int;
5      media, idade, soma_ is integer;
6  begin
7      cont_ = 5;
8      soma = 0;
9
10     do
11         write( Altura :    );
12         read (altura);
13         soma := soma altura;
14         cont_ := cont_ - 1;
15     while(cont_ > 0)
16
17     write( Media :    );
18     write (soma / qtd);
19
20 stop
```

Saída:
Erro!!! – linha 20 – token inválido: :

Neste teste temos um código que têm uma abertura de comentário de bloco que não é fechado, por isso, corrigindo este erro.

```
1  { Programa de Teste
2  Calculo de idade}
3  init
4      cont_ is int;
5      media, idade, soma_ is integer;
6  begin
7      cont_ = 5;
8      soma = 0;
```

```
9
10      do
11          write( Altura :    );
12          read (altura);
13          soma := soma altura;
14          cont_ := cont_ - 1;
15      while(cont_ > 0)
16
17      write( Media :    );
18      write (soma / qtd);
19
20  stop
```

Saída:
Erro!!! – linha 4 – token inválido: int
Na linguagem inteiros são representados pela palavra **integer**.

```
1   { Programa de Teste
2   Calculo de idade}
3   init
4       cont_ is integer;
5       media, idade, soma_ is integer;
6   begin
7       cont_ = 5;
8       soma = 0;
9
10      do
11          write( Altura :    );
12          read (altura);
13          soma := soma altura;
14          cont_ := cont_ - 1;
15      while(cont_ > 0)
16
17      write( Media :    );
18      write (soma / qtd);
19
20  stop
```

Saída:
Erro!!! – linha 6 – token inválido: begin

A **begin** é necessária apenas no bloco do comando **if**.

```
1  { Programa de Teste
2  Calculo de idade}
3  init
4      cont_ is integer;
5      media, idade, soma_ is integer;
6
7      cont_ = 5;
8      soma = 0;
9
10     do
11         write( Altura :    );
12         read (altura);
13         soma := soma altura;
14         cont_ := cont_ - 1;
15     while(cont_ > 0)
16
17     write( Media :    );
18     write (soma / qtd);
19
20 stop
```

Saída:
Erro!!! – linha 7 – token inválido: =
Nesta linguagem atribuição é representada pelo símbolo **:=**.

```
1  { Programa de Teste
2  Calculo de idade}
3  init
4      cont_ is integer;
5      media, idade, soma_ is integer;
6
7      cont_ := 5;
8      soma = 0;
9
10     do
11         write( Altura :    );
12         read (altura);
13         soma := soma altura;
```

```
14          cont_ := cont_ - 1;
15      while(cont_ > 0)
16
17      write( Media :    );
18      write (soma / qtd);
19
20  stop
```

Saída:
Erro!!! – linha 7 – token inválido: :=

Obtivemos um novo erro na linha 7, no entanto, nesta linguagem atribuições são feitas em blocos **stmt**, corrigindo este erro temos:

3.4 Teste 4

```
1  init
2
3      i, j, k, @total, 1soma is integer
4
5          read (I);
6          k := i * (5-i * 50 / 10;
7          j := i * 10;
8          k := i* j / k;
9          k := 4 + a $;
10         write(i);
11         write(j);
12         write(k);
```

Saída:
Erro!!! – linha 3 – token inválido: @
Nesta linguagem identificadores começam com [A-Za-z].

```
1  init
2
3      i, j, k, total, 1soma is integer
4
5          read (I);
6          k := i * (5-i * 50 / 10;
7          j := i * 10;
8          k := i* j / k;
9          k := 4 + a $;
10         write(i);
11         write(j);
12         write(k);
```

Saída:
Erro!!! – linha 3 – token inválido: 1
Nesta linguagem identificadores começam com [A-Za-z].

```
1  init
2
3      i, j, k, total, soma is integer
4
```

```
5         read (I);
6         k := i * (5-i * 50 / 10;
7         j := i * 10;
8         k := i* j / k;
9         k := 4 + a $;
10        write(i);
11        write(j);
12        write(k);
```

Saída:
Erro!!! – linha 5 – token inválido: read

As declarações devem terminar com **;**.

```
1  init
2
3     i, j, k, total, soma is integer;
4
5         read (I);
6         k := i * (5-i * 50 / 10);
7         j := i * 10;
8         k := i* j / k;
9         k := 4 + a $;
10        write(i);
11        write(j);
12        write(k);
```

Saída:
Erro!!! – linha 6 – token inválido: ;
Na linha 6 temos uma expressão que abre parêntesis mas não fecha, corrigindo este erro.

```
1  init
2
3     i, j, k, total, soma is integer;
4
5         read (I);
6         k := i * (5-i * 50 / 10);
7         j := i * 10;
8         k := i* j / k;
```

```
9         k := 4 + a $;
10        write(i);
11        write(j);
12        write(k);
```

Saída:
Erro!!! – linha 9 – token inválido: $

Na linha 9 temos uma expressão incompleta, nesse caso optamos por remover o carácter $.

```
1  init
2
3      i, j, k, total, soma is integer;
4
5          read (I);
6          k := i * (5-i * 50 / 10);
7          j := i * 10;
8          k := i* j / k;
9          k := 4 + a;
10         write(i);
11         write(j);
12         write(k);
```

Saída:
Erro!!! – linha 12 – token inválido:

Na linha 12 podemos perceber o fim de arquivo, no entanto, a palavra **stop** não foi declarada para indicar o fim do programa.

```
1  init
2
3      i, j, k, total, soma is integer;
4
5          read (I);
6          k := i * (5-i * 50 / 10);
7          j := i * 10;
8          k := i* j / k;
9          k := 4 + a;
10         write(i);
11         write(j);
```

```
12          write(k);
13
14  stop
```

Saída:
Programa correto!

3.5 Teste 5

```
1   init
2   // Programa com if
3
4           j, k, m is integer;
5           a, j is string;
6
7       read(j);
8       read(k);
9
10      if (j ==   ok  )
11      begin
12          result = k/m
13      end
14      else
15      begin
16          result := 0;
17          write (Invalid  entry );
18      end
19
20
21      write(result);
```

Saída Erro!!! – linha 10 – token inválido: =

Nesta linguagem para verificar igualdade é necessário apenas um símbolo =.

```
1   init
2   // Programa com if
3
4           j, k, m is integer;
5           a, j is string;
```

```
6
7       read(j);
8       read(k);
9
10      if (j ==   ok  )
11      begin
12          result = k/m
13      end
14      else
15      begin
16          result := 0;
17          write (Invalid  entry );
18      end
19
20
21      write(result);
```

Saída:
Erro!!! – linha 12 – token inválido: =

Nesta linguagem para atribuição usamos o símbolo **:=**.

```
1   init
2   // Programa com if
3
4           j, k, m is integer;
5           a, j is string;
6
7       read(j);
8       read(k);
9
10      if (j =   ok  )
11      begin
12          result := k/m
13      end
14      else
15      begin
16          result := 0;
17          write (Invalid  entry );
18      end
```

```
19
20
21      write(result);
```

Saída:
Erro!!! – linha 12 – token inválido: end

É necessário um ponto e virgula **;** após uma expressão.

```
1  init
2  // Programa com if
3
4          j, k, m is integer;
5          a, j is string;
6
7      read(j);
8      read(k);
9
10     if (j =   ok  )
11     begin
12         result := k/m
13     end
14     else
15     begin
16         result := 0;
17         write (Invalid  entry );
18     end
19
20
21     write(result);
```

Saída:
Erro!!! – linha 21 – token inválido: write

Após o end é necessário um ponto e virgula **;**.

```
1  init
2  // Programa com if
3
4          j, k, m is integer;
```

```
5         a, j is string;
6
7     read(j);
8     read(k);
9
10    if (j =   ok  )
11    begin
12        result := k/m;
13    end
14    else
15    begin
16        result := 0;
17        write (Invalid  entry );
18    end;
19
20
21    write(result);
```

Saída:
Erro!!! – linha 21 – token inválido: write

Para terminar o programa é necessário a palavra **stop**.

```
1  init
2  // Programa com if
3
4         j, k, m is integer;
5         a, j is string;
6
7     read(j);
8     read(k);
9
10    if (j =   ok  )
11    begin
12        result := k/m;
13    end
14    else
15    begin
16        result := 0;
17        write (Invalid  entry );
```

```
18      end;
19
20
21      write(result);
22  stop
```

Saída:
Programa correto!

3.6 Teste 6

```
1  init
2      a, b, c, maior is integer;
3
4          read(a);
5          read(b);
6          read(c;
7          maior := 0;
8          if ( a>b and a>c )
9              maior := a;
10
11         else
12             if (b>c)
13                 maior := b;
14
15             else
16                 maior := c;
17
18
19
20         write(Maior idade:   );
21         write(maior);
22 end
```

Saída:
Erro!!! – linha 6 – token inválido: ;

Nessa linha temos um declaração com um parêntesis que não foi fechado.

```
1  init
2      a, b, c, maior is integer;
3
4          read(a);
5          read(b);
6          read(c);
7          maior := 0;
8          if ( a>b and a>c )
9              maior := a;
10
```

```
11          else
12              if (b>c)
13                  maior := b;
14
15              else
16                  maior := c;
17
18
19
20          write( Maior  idade:   );
21          write(maior);
22  end
```

Saída:
Erro!!! – linha 9 – token inválido: maior

Na declaração de um **if** é necessário usar **begin** e **end**.

```
1   init
2       a, b, c, maior is integer;
3
4           read(a);
5           read(b);
6           read(c);
7           maior := 0;
8           if ( a>b and a>c )
9           begin
10              maior := a;
11          end
12          else
13              if (b>c)
14                  maior := b;
15
16              else
17                  maior := c;
18
19
20
21          write( Maior  idade:   );
22          write(maior);
```

```
23 end
```

Saída:
Erro!!! – linha 13 – token inválido: if

Na declaração de **if** é necessário usar **begin** e **end**.

```
1  init
2      a, b, c, maior is integer;
3
4          read(a);
5          read(b);
6          read(c);
7          maior := 0;
8          if ( a>b and a>c )
9          begin
10             maior := a;
11         end
12         else
13             begin
14                 if (b>c)
15                     maior := b;
16
17                 else
18                     maior := c;
19             end
20
21
22
23         write( Maior  idade:   );
24         write(maior);
25 end
```

Saída:
Erro!!! – linha 15 – token inválido: maior

Novamente temos uma declaração de **if** que não usa **begin** e **end**.

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

```
3
4           read(a);
5           read(b);
6           read(c);
7           maior := 0;
8           if ( a>b and a>c )
9               begin
10                  maior := a;
11              end
12          else
13              begin
14                  if (b>c)
15                      begin
16                          maior := b;
17                      end
18                  else
19                      maior := c;
20              end
21
22
23
24          write( Maior  idade:   );
25          write(maior);
26  end
```

Saída:
Erro!!! – linha 19 – token inválido: maior

Novamente temos uma declaração de **if** que não usa **begin** e **end**.

```
1   init
2       a, b, c, maior is integer;
3
4           read(a);
5           read(b);
6           read(c);
7           maior := 0;
8           if ( a>b and a>c )
9               begin
10                  maior := a;
```

```
11              end
12          else
13              begin
14                  if (b>c)
15                      begin
16                          maior := b;
17                      end
18                  else
19                      begin
20                          maior := c;
21                      end
22              end
23
24
25
26          write( Maior  idade:   );
27          write(maior);
28  end
```

Saída:
Erro!!! – linha 22 – token inválido: end

Após a declaração do bloco **if** é necessário um ponto e virgula **;**, assim nas linhas 21 e 22 será acrescentado o **;**.

```
1   init
2       a, b, c, maior is integer;
3
4           read(a);
5           read(b);
6           read(c);
7           maior := 0;
8           if ( a>b and a>c )
9               begin
10                  maior := a;
11              end
12          else
13              begin
14                  if (b>c)
15                      begin
```

```
16                          maior := b;
17                      end
18                  else
19                      begin
20                          maior := c;
21                      end;
22              end;
23
24
25
26          write( Maior  idade:   );
27          write(maior);
28  end
```

Saída:
Erro!!! – linha 28 – token inválido: end

No final do arquivo é necessário a palavra **stop**, substituiremos **end** por **stop**.

```
1   init
2       a, b, c, maior is integer;
3
4           read(a);
5           read(b);
6           read(c);
7           maior := 0;
8           if ( a>b and a>c )
9               begin
10                  maior := a;
11              end
12          else
13              begin
14                  if (b>c)
15                      begin
16                          maior := b;
17                      end
18                  else
19                      begin
20                          maior := c;
21                      end;
```

```
22              end;
23
24
25
26          write( Maior  idade:   );
27          write(maior);
28  stop
```

Saída:
Programa correto!

4 Conclusões:

Nesta primeira etapa do processo de compilação, agrupamos os caracteres em lexemas para produzir uma sequência de símbolos léxicos que são os tokens. Estes tokens foram inseridos na tabela de símbolos com suas informações, como os identificadores. Esta etapa é muito prematura para identificar erros de compilação e, na análise léxica não é possível identificar erros de instruções como palavra reservada escrita incorretamente, essa verificação será feita na análise sintática. Um erro léxico que pode ser identificado nessa etapa são caracteres que não pertencem a linguagem, como caracteres. Nesse caso o analisador léxico pode sinalizar um erro informando a posição desse caractere.

Referências

[1] V. A. Alfred, S. L. Monica, S. Ravi, and J. D. Ullman. *Compiladores princípios, técnicas e ferramentas*. Pearson Addison-Wesley, São Paulo, 2. edition, 2008.

1