TRABALHO DE COMPILADORES
Análise Léxica e Tabela de Símbolos

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

1. INTRODUÇÃO

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

O compilador em questão será implementado em JAVA.

2. COMO USAR O COMPILADOR

Na pasta contendo os arquivos-fonte do Analisador Léxico, é necessário abrir o arquivo *Main.java*, e na linha 24, alterar o caminho com o arquivo de entrada que deseja utilizar. A entrada deve ser um arquivo de texto, e o caminho deverá estar entre aspas. Esta classe é a principal e ela deverá ser executada.

Além disso, também geramos um arquivo .jar que pode ser utilizado para a execução. Para a execução 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. Outra opção é adicionar a entrada a um arquivo *entrada.txt* na mesma pasta do arquivo .jar. Nesse caso, não é preciso passar nenhum parâmetro para o comando *java -jar NomeDoJar.jar*.

3. IMPLEMENTAÇÃO

A partir da gramática da linguagem, os seguintes tokens foram identificados:

- Palavras reservadas:
  - init
  - stop
  - is
  - integer
  - if
  - string
  - begin
  - end
  - else
  - do
  - while
  - read
  - write
  - and
  - or

    - not
- Literais: “{caractere}”; caractere: 256 símbolos ASCII, exceto aspas e /n
- Constantes numéricas: [1-9]{[0-9]} | 0
- Operadores
    - >
    - >=
    - <
    - <=
    - <>
    - =
    - :=
    - +
    - -
    - *
    - /
- Símbolos de Pontuação
    - (
    -)
    - ;
    - ,
- Identificadores:([A-Za-z]) { [A-Za-z] | [0-9] | ’_’}

A classe *Token.java* especifica um token, definindo uma string tipo, para impressão do token, e uma constante numérica “tag” que representa o tipo do token. O valor que será atribuído para esse atributo é, em geral, aquele definido na classe Tag. Além disso, define um formato para a impressão na tela do token.

Já a classe *Tag.java* especifica diferentes constantes inteiras para diferenciar os diversos tipos de tokens presentes na linguagem. Há uma tag para cada palavra reservada, para cada operador aritmético, operador relacional, operador de atribuição, símbolos de pontuação e também uma tag para constante numérica, uma para identificador e uma para literal.

A classe *Word.java* é filha da classe Token, ela adiciona uma string ao token para armazenar o lexema. Identificadores, operadores, símbolos de pontuação e literais serão tokens do tipo Word. A classe *Num.java* também herda da classe Token. Ela, por sua vez, possui um atributo inteiro *value* para armazenar os valores das constantes numéricas. Ambas as classes filhas sobrescrevem o método *toString()* da pai para adaptar o formato de impressão do token.

A classe *Lexer.java* analisa o arquivo de entrada, caractere a caractere. Assim, ela identifica lexemas e, quando termina a verificação de um, identifica o token correspondente de acordo com as especificações da gramática. Ela faz isso até identificar o fim do arquivo. Também é sua função desconsiderar comentários de linha e comentários de bloco, assim como delimitadores.

A tabela de símbolos também está declarada na classe Lexer, trata-se de um HashMap chamado *words*. O método *reserve()* é utilizado para adicionar tokens a

tabela de símbolos. As palavras-chave da linguagem, como são palavras reservadas, são adicionadas ao *words* logo no construtor dessa classe.

A classe principal, *Main.java* recebe o caminho do arquivo de entrada, chama a classe Lexer para identificar todos os tokens, que são retornados e impressos, até o fim do arquivo. O método da classe Lexer que é invocado é chamado *scan()*, e, além de retornar os tokens, ele também verifica erros e, caso encontre-os, invoca o método *error()* que exibe uma mensagem de erro indicando a linha do erro, o caractere em análise quando o erro foi encontrado e finaliza a execução do programa.

4. RESULTADOS DOS TESTES ESPECIFICADOS

- **teste1.txt**

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

stop
```

Saída:

```
<init>
<id,'a'>
<simb_pont,','>
<id,'b'>
<simb_pont,','>
<id,'c'>
<simb_pont,','>
<id,'result'>
<is>
<integer>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'a'>
<simb_pont,')'>
<simb_pont,';'>
<read>
```

<simb_pont,'('>
<id,'c'>
<simb_pont,')'>
<simb_pont,';'>
<id,'b'>
<atr_op,':='>
<num,10>
<simb_pont,';'>
<id,'result'>
<atr_op,':='>
<simb_pont,'('>
<id,'a'>
<arit_op,'*'>
<id,'c'>
<simb_pont,')'>
<arit_op,'/'>
<simb_pont,'('>
<id,'b'>
<arit_op,'+'>
<num,5>
<arit_op,'-'>
<num,345>
<simb_pont,')'>
<simb_pont,';'>
<write>
<simb_pont,'('>
<id,'result'>
<simb_pont,')'>
<simb_pont,';'>
<stop>

Neste primeiro teste, que foi passado na especificação do trabalho, não foram encontrados erros, e os Tokens foram exibidos corretamente na tela.

- **teste2.txt**

```
  a, _valor, b : integer;

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

```
  Write (b);
stop
```

Saída:

```
<id,'a'>

Erro na linha 1: Token não reconhecido - caractere com provável erro: _
<simb_pont,','>
```

Na primeira execução deste teste, a mensagem de erro informou que havia um "_" (underscore) causando problemas. Isso é porque um identificador não pode começar com este caractere. De acordo com a gramática indicada na especificação, todo identificador deve começar com uma letra. Para resolver este problema, havia duas soluções: acrescentar uma (ou mais) letra(s) e dígitos(s) antes do underscore (desde que contenha uma letra no começo do identificador), ou removê-lo. A solução apresentada aqui foi a remoção do underscore.

Correção:

```
a, valor, b : integer;

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

Saída após correção:

```
<id,'a'>
<simb_pont,','>
<id,'valor'>
<simb_pont,','>
<id,'b'>

Erro na linha 1: Token não reconhecido - caractere com provável erro:
```

Após a primeira correção, a mensagem de erro mostra um espaço em branco como a causa do problema. Isso aconteceu porque havia algum caractere que deveria trazer outro caractere específico logo em seguida. Como esse caractere não foi encontrado, a mensagem exibiu um espaço em branco. No caso, a declaração do tipo dos identificadores deveria trazer um símbolo de igualdade logo após os dois pontos (:=). Para resolver este problema, também havia duas soluções: a primeira era acrescentar o símbolo de igualdade após os dois pontos, e a segunda era substituir essa atribuição por "is". Em nossa resolução, optamos por acrescentar o símbolo de igualdade.

Segunda correção:

```
a, valor, b := integer;

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

Saída após segunda correção:

```
<id,'a'>
<simb_pont,','>
<id,'valor'>
<simb_pont,','>
<id,'b'>
<atr_op,':='>
<integer>
<simb_pont,';'>
<init>
<read>
<simb_pont,'('>
<id,'a'>
<simb_pont,')'>
<simb_pont,';'>
<id,'b'>
<atr_op,':='>
<id,'a'>
```

<arit_op,'*'>
<id,'a'>
<simb_pont,';'>
<write>
<simb_pont,'('>
<id,'b'>
<simb_pont,')'>
<simb_pont,';'>
<id,'b'>
<rel_op,'='>
<id,'b'>
<arit_op,'+'>
<id,'a'>
<arit_op,'/'>
<num,2>
<arit_op,'*'>
<simb_pont,'('>
<id,'a'>
<arit_op,'+'>
<num,5>
<simb_pont,')'>
<simb_pont,';'>
<write>
<simb_pont,'('>
<id,'b'>
<simb_pont,')'>
<simb_pont,';'>
<stop>

- **teste3.txt**

```
{ 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
```

Saída:
Comentário de bloco iniciado na linha 1 sem fim!

Neste caso de teste, a mensagem de erro informou que um bloco de comentários não havia fim. Isso aconteceu porque de acordo com a especificação desta gramática, um bloco de comentários deve ser iniciado com um abre-chaves ({) e finalizado com um fecha-chaves (}). Quando o comentário ocupar apenas uma linha, pode-se usar "//" logo no início da linha. Como o comentário ocupava mais de uma linha, era necessário finalizá-lo com um fecha-chaves.

Correção:

```
{ 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
```

Saída após correção:
<init>

<id,'cont_'>
<is>
<id,'int'>
<simb_pont,';'>
<id,'media'>
<simb_pont,','>
<id,'idade'>
<simb_pont,','>
<id,'soma_'>
<is>
<integer>
<simb_pont,';'>
<begin>
<id,'cont_'>
<rel_op,'='>
<num,5>
<simb_pont,';'>
<id,'soma'>
<rel_op,'='>
<num,0>
<simb_pont,';'>
<do>
<write>
<simb_pont,'('>
<lit,'"altura:"'>
<simb_pont,')'>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'altura'>
<simb_pont,')'>
<simb_pont,';'>
<id,'soma'>
<atr_op,':='>
<id,'soma'>
<id,'altura'>
<simb_pont,';'>
<id,'cont_'>
<atr_op,':='>
<id,'cont_'>
<arit_op,'-'>
<num,1>

<simb_pont,';'>
<while>
<simb_pont,'('>
<id,'cont_'>
<rel_op,'>'>
<num,0>
<simb_pont,')'>
<write>
<simb_pont,'('>
<lit,'"media: "'>
<simb_pont,')'>
<simb_pont,';'>
<write>
<simb_pont,'('>
<id,'soma'>
<arit_op,'/'>
<id,'qtd'>
<simb_pont,')'>
<simb_pont,';'>
<stop>

- **teste4.txt**
  init

```
i, j, k, @total, 1soma is integer
  read (l);
  k := i * (5-i * 50 / 10;
  j := i * 10;
  k := i* j / k;
  k := 4 + a $;
  write(i);
  write(j);
  write(k);
```

Saída:
<init>
<id,'i'>
<simb_pont,','>
<id,'j'>
<simb_pont,','>
<id,'k'>
<simb_pont,','>

Erro na linha 3: Token não reconhecido - caractere com provável erro: @

Na primeira execução deste teste, a mensagem de erro informou que o caractere "@" era a provável causa do problema. Isso aconteceu porque o arroba (@) não pertence à gramática desta linguagem. Portanto, a solução foi remover este caractere do arquivo de entrada.

Correção:

```
init

  i, j, k, total, 1soma is integer
    read (l);
    k := i * (5-i * 50 / 10;
    j := i * 10;
    k := i* j / k;
    k := 4 + a $;
    write(i);
    write(j);
    write(k);
```

Saída após correção:

```
<init>
<id,'i'>
<simb_pont,','>
<id,'j'>
<simb_pont,','>
<id,'k'>
<simb_pont,','>
<id,'total'>
<simb_pont,','>
```

Erro na linha 3: Token não reconhecido - caractere com provável erro: s

Na segunda execução do teste, a mensagem de erro apontou o caractere "s" como causa do problema. Esse erro foi apontado porque, como explicado no teste 2, todo identificador deve começar com uma letra. Como foi encontrado um número (no caso, "1"). A solução para este problema foi remover o "1" do identificador.

Segunda correção:

```
init
```

```
 i, j, k, total, soma is integer
   read (l);
   k := i * (5-i * 50 / 10;
   j := i * 10;
   k := i* j / k;
   k := 4 + a $;
   write(i);
   write(j);
   write(k);
```

Saída após segunda correção:

```
<init>
<id,'i'>
<simb_pont,','>
<id,'j'>
<simb_pont,','>
<id,'k'>
<simb_pont,','>
<id,'total'>
<simb_pont,','>
<id,'soma'>
<is>
<integer>
<read>
<simb_pont,'('>
<id,'i'>
<simb_pont,')'>
<simb_pont,';'>
<id,'k'>
<atr_op,':='>
<id,'i'>
<arit_op,'*'>
<simb_pont,'('>
<num,5>
<arit_op,'-'>
<id,'i'>
<arit_op,'*'>
<num,50>
<arit_op,'/'>
<num,10>
<simb_pont,';'>
```

<id,'j'>
<atr_op,':='>
<id,'i'>
<arit_op,'*'>
<num,10>
<simb_pont,';'>
<id,'k'>
<atr_op,':='>
<id,'i'>
<arit_op,'*'>
<id,'j'>
<arit_op,'/'>
<id,'k'>
<simb_pont,';'>
<id,'k'>
<atr_op,':='>
<num,4>
<arit_op,'+'>
<id,'a'>

Erro na linha 8: Token não reconhecido - caractere com provável erro: $

Após duas correções, a mensagem de erro apresentou o caractere "$" como a provável causa do erro. Isso ocorreu porque o "$" não pertence à linguagem. A solução encontrada foi remover este caractere.

Terceira correção:

```
init

  i, j, k, total, soma is integer
    read (l);
    k := i * (5-i * 50 / 10;
    j := i * 10;
    k := i* j / k;
    k := 4 + a;
    write(i);
    write(j);
    write(k);
```

Saída após terceira correção:
<init>
<id,'i'>

<simb_pont,','>
<id,'j'>
<simb_pont,','>
<id,'k'>
<simb_pont,','>
<id,'total'>
<simb_pont,','>
<id,'soma'>
<is>
<integer>
<read>
<simb_pont,'('>
<id,'i'>
<simb_pont,')'>
<simb_pont,';'>
<id,'k'>
<atr_op,':='>
<id,'i'>
<arit_op,'*'>
<simb_pont,'('>
<num,5>
<arit_op,'-'>
<id,'i'>
<arit_op,'*'>
<num,50>
<arit_op,'/'>
<num,10>
<simb_pont,';'>
<id,'j'>
<atr_op,':='>
<id,'i'>
<arit_op,'*'>
<num,10>
<simb_pont,';'>
<id,'k'>
<atr_op,':='>
<id,'i'>
<arit_op,'*'>
<id,'j'>
<arit_op,'/'>
<id,'k'>
<simb_pont,';'>

<id,'k'>
<atr_op,':='>
<num,4>
<arit_op,'+'>
<id,'a'>
<simb_pont,';'>
<write>
<simb_pont,'('>
<id,'i'>
<simb_pont,')'>
<simb_pont,';'>
<write>
<simb_pont,'('>
<id,'j'>
<simb_pont,')'>
<simb_pont,';'>
<write>
<simb_pont,'('>
<id,'k'>
<simb_pont,')'>
<simb_pont,';'>

- **teste5.txt**

```
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:

<init>
<id,'j'>
<simb_pont,','>
<id,'k'>
<simb_pont,','>
<id,'m'>
<is>
<integer>
<simb_pont,';'>
<id,'a'>
<simb_pont,','>
<id,'j'>
<is>
<string>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'j'>
<simb_pont,')'>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'k'>
<simb_pont,')'>
<simb_pont,';'>
<if>
<simb_pont,'('>
<id,'j'>
<rel_op,'='>
<rel_op,'='>
<lit,'"ok"'>
<simb_pont,')'>
<begin>
<id,'result'>
<rel_op,'='>
<id,'k'>
<arit_op,'/'>
<id,'m'>
<end>
<else>

<begin>
<id,'result'>
<atr_op,':='>
<num,0>
<simb_pont,';'>
<write>
<simb_pont,'('>
<lit,'"invalid entry"'>
<simb_pont,')'>
<simb_pont,';'>
<end>
<write>
<simb_pont,'('>
<id,'result'>
<simb_pont,')'>
<simb_pont,';'>

Do ponto de vista do Analisador Léxico, esta entrada não apresenta erros.

- **teste6.txt**

```
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:
<init>

<id,'a'>
<simb_pont,','>
<id,'b'>
<simb_pont,','>
<id,'c'>
<simb_pont,','>
<id,'maior'>
<is>
<integer>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'a'>
<simb_pont,')'>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'b'>
<simb_pont,')'>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'c'>
<simb_pont,';'>
<id,'maior'>
<atr_op,':='>
<num,0>
<simb_pont,';'>
<if>
<simb_pont,'('>
<id,'a'>
<rel_op,'>'>
<id,'b'>
<and>
<id,'a'>
<rel_op,'>'>
<id,'c'>
<simb_pont,')'>
<id,'maior'>
<atr_op,':='>
<id,'a'>
<simb_pont,';'>

<else>
<if>
<simb_pont,'('>
<id,'b'>
<rel_op,'>'>
<id,'c'>
<simb_pont,')'>
<id,'maior'>
<atr_op,':='>
<id,'b'>
<simb_pont,';'>
<else>
<id,'maior'>
<atr_op,':='>
<id,'c'>
<simb_pont,';'>
<write>
<simb_pont,'('>
<lit,'"maior idade: "'>
<simb_pont,')'>
<simb_pont,';'>
<write>
<simb_pont,'('>
<id,'maior'>
<simb_pont,')'>
<simb_pont,';'>
<end>

Do ponto de vista do Analisador Léxico, esta entrada não apresenta erros.

- **teste7.txt**
  init

  x, y, z is integer;

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

  end

Saída:

<init>
<id,'x'>
<simb_pont,','>
<id,'y'>
<simb_pont,','>
<id,'z'>
<is>
<integer>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'x'>
<simb_pont,')'>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'y'>
<simb_pont,')'>
<simb_pont,';'>
<id,'z'>
<atr_op,':='>
<id,'x'>
<arit_op,'+'>
<id,'y'>
<simb_pont,';'>
<write>
<simb_pont,'('>
<id,'soma'>

Erro na linha 8: Token não reconhecido - caractere com provável erro:)

Este teste, criado por nós, traz uma escrita de string que não contém aspas. Desta forma, como a string contém o símbolo de dois pontos, o Lexer espera que venha o símbolo de igualdade logo em seguida. Porém, o caractere após os dois pontos é um parênteses, sendo este caractere apontado como a causa do erro. Logo, a solução foi acrescentar as aspas para que seja reconhecido como string, e não como um identificador seguido de uma atribuição.

Correção:

```
init

x, y, z is integer;

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

end
```

Saída após correção:

```
<init>
<id,'x'>
<simb_pont,','>
<id,'y'>
<simb_pont,','>
<id,'z'>
<is>
<integer>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'x'>
<simb_pont,')'>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'y'>
<simb_pont,')'>
<simb_pont,';'>
<id,'z'>
<atr_op,':='>
<id,'x'>
<arit_op,'+'>
<id,'y'>
<simb_pont,';'>
<write>
<simb_pont,'('>
<lit,'"soma:"'>
<simb_pont,')'>
```

<simb_pont,';'>
<write>
<simb_pont,'('>
<id,'z'>
<simb_pont,')'>
<simb_pont,';'>
<end>

- **teste8.txt**

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

Saída:

<init>
<id,'a'>
<simb_pont,','>
<id,'b'>
<simb_pont,','>
<id,'c'>
<is>
<integer>
<simb_pont,';'>
<read>
<simb_pont,'('>
<id,'a'>
<simb_pont,')'>
<simb_pont,';'>
<id,'b'>
<atr_op,':='>
<id,'a'>
<arit_op,'+'>
<num,0>
<simb_pont,';'>
<write>
<simb_pont,'('>
<id,'b'>
<simb_pont,')'>
<simb_pont,';'>

<end>

Este caso de teste, criado por nós, foi feito para mostrar que este analisador léxico é capaz de reconhecer o 0 (zero) e o ponto e vírgula separadamente. Por exemplo, no teste 4, foi mostrado que o símbolo "$" vindo logo após uma operação aritmética, gerou uma mensagem de erro. O ponto e vírgula é reconhecido corretamente como caractere que mostra o final da operação aritmética.