CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS - CEFET MG
DEPARTAMENTO DE COMPUTAÇÃO - COMPILADORES

RELATÓRIO DO TRABALHO PRÁTICO PARTE 1
ANALISADOR LÉXICO E TABELA DE SÍMBOLOS

Guilherme Moreira de Carvalho

16 de Abril de 2023

INTRODUÇÃO

O analisador léxico implementado recebe como parâmetro um arquivo texto para verificar a concordância com a gramática especificada.

```
> comp [arquivo texto]
```

A implementação, realizada em C++, pode ser compilada com o comando *make*.

O analisador escaneia o arquivo caractere a caractere de acordo com uma máquina de estados a fim de construir um token com as sequências significativas.

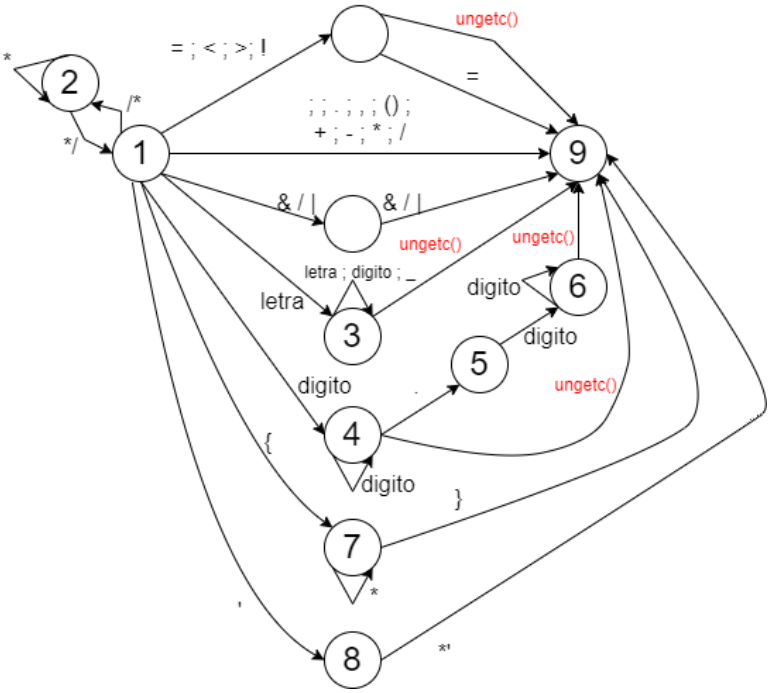

IMAGEM 1 - Representação em Máquina de Estados do analisador.

Como saída, são exibidos todos os tokens encontrados, com seus lexemas e tipos, e a tabela de símbolos construída. Ao encontrar algum token inválido ou chegar no fim do arquivo, a linha da ocorrência também é informada para verificação de erros de escrita.

Já a tabela de símbolos armazena todas as variáveis (tokens de tipo ID) encontradas no arquivo. Uma segunda é mantida com as palavras reservadas de conhecimento prévio da linguagem de modo a permitir a verificação do tipo do token.

CLASSES

- **TAG**

  Define os tipos possíveis para os tokens (46) e sua impressão por extenso.

  Destacam-se os tipos especiais *INVALIDO* de valor -2, *UNEXP_EOF* (fim de arquivo inesperado) de valor -1 e *EOF* (fim de arquivo) de valor 0. Os demais seguem em ordem crescente, indexando variáveis, tipos de dados, palavras reservadas, operadores lógicos e aritméticos e símbolos.

- **TOKEN**

  Definição de um token com lexema e tipo.

- **TS (TABELA DE SÍMBOLOS)**

  Define a tabela de símbolos com seu escopo por meio de uma auto referência. São armazenadas duas tabelas: *m_tab*, tabela com as variáveis presentes no arquivo, e *m_reserved*, tabela com as palavras/símbolos reservados da linguagem.

  Esta tabela é utilizada pelo analisador léxico para definir se um token encontrado é ou não uma variável. Se seu lexema for chave na tabela de reservados, então se trata de uma palavra significativa da gramática e não deve ser adicionada à tabela de símbolos principais. Caso contrário, o token é do tipo ID e deve ser tabelado.

- **LEX (ANALISADOR LÉXICO)**

  Classe principal da primeira parte da análise. O método *next()* lê cada caractere do arquivo de entrada e tenta formar um token.

  Possui como atributos *m_line* para persistir a linha corrente do arquivo, *ch* que armazena o caractere lido do arquivo, *m_reader*, o leitor do arquivo e a tabela de símbolos *m_words*.

  A máquina de estados tem 10 posições: o início 1, que consome todos os delimitadores, tokens de até dois caracteres (símbolos e operadores) e o fim de arquivo;

estado 2 dos comentários, que simplesmente ignora toda leitura até encontrar um delimitador de comentários ou EOF; estado 3 das palavras (reservadas e variáveis); estado 4, 5 e 6 das constantes numéricas inteiras, inteiras seguidas de um ponto e de ponto flutuante respectivamente; estado 7 dos literais, que consome caracteres até que encontre um delimitador *{* ou uma quebra de linha não permitida pela gramática; estado 8 dos caracteres, que são um único caractere entre aspas.

Os estados 9 e 10 são de saída. Aquele quando deve-se antes determinar o tipo do token construído e, se for *ID*, inseri-lo na tabela de símbolos, e este quando esse processo já foi realizado na execução da máquina

RESULTADOS

```
programa teste1

   a, b is int;
   result is int;
   a, x is float;

begin

   a = 12a;
   x = 12.;
   read (a);
   read (b);
   read (c)
   result = (a*b + 1) / (c+2);
   write {Resultado: };
   write (result);

end.
```

```
TOKENS ENCONTRADOS
("programa", ID)       ("teste1", ID)  ("a", ID)
(",", VIRG)    ("b", ID)      ("is", IS)
("int", INT)   (";", SEMI)    ("result", ID)
("is", IS)     ("int", INT)   (";", SEMI)
("a", ID)      (",", VIRG)    ("x", ID)
("is", IS)     ("float", FLOAT)       (";", SEMI)
("begin", BEGIN)       ("a", ID)      ("=", ATRIB)
("12", INT NUM) ("a", ID)      (";", SEMI)
("x", ID)      ("=", ATRIB)   ("12.", INVALIDO)      LINHA: 10
(";", SEMI)    ("read", READ)  ("(", OP_PAR)
("a", ID)      (")", CL_PAR)  (";", SEMI)
("read", READ)  ("(", OP_PAR)  ("b", ID)
(")", CL_PAR)  (";", SEMI)    ("read", READ)
("(", OP_PAR)  ("c", ID)      (")", CL_PAR)
("result", ID)  ("=", ATRIB)   ("(", OP_PAR)
("a", ID)      ("*", MUL)     ("b", ID)
("+", ADD)     ("1", INT NUM)  (")", CL_PAR)
("/", DIV)     ("(", OP_PAR)  ("c", ID)
("+", ADD)     ("2", INT NUM)  (")", CL_PAR)
(";", SEMI)    ("write", WRITE)       ("{Resultado: }", LIT)
(";", SEMI)    ("write", WRITE)       ("(", OP_PAR)
("result", ID)  (")", CL_PAR)  (";", SEMI)
("end", END)   (".", PONTO)   ("", EOF) LINHA: 18

TABELA DE SIMBOLOS
Token: a, Info: ID
Token: b, Info: ID
Token: c, Info: ID
Token: programa, Info: ID
Token: result, Info: ID
Token: teste1, Info: ID
Token: x, Info: ID
```

IMAGEM 2 - Primeiro teste e sua saída

O programa inicia com um erro - a label *program* escrita como *programa* a qualifica como uma variável ao invés de palavra reservada; a atribuição *a = 12a;* não gera erro léxico, pois todos os tokens são identificados corretamente; o token *12.* é corretamente registrado como inválido, pois falta os decimais do número de ponto flutuante; a falta de "()" delimitando a expressão do método *write* também não qualifica erro léxico, mas sim sintático; a falta de *;* ao fim de cada declaração não é reportada por esse tipo de análise.

```
program teste2

      a, b, c:int;
      d, _var: float;

   teste2 = 1;
   Read (a);
   b = a * a;
   c = b + a/2 * (35/b);
   write c;
   val := 34.2
   c = val + 2.2 + a;
   write (val)
end.
```

```
TOKENS ENCONTRADOS
("program", PROGRAM)   ("teste2", ID)  ("a", ID)
(",", VIRG)   ("b", ID)   (",", VIRG)
("c", ID)   (":", INVALIDO)  LINHA: 3
("int", INT)
(";", SEMI)   ("d", ID)   (",", VIRG)
("_", INVALIDO)  LINHA: 4
("var", ID)   (":", INVALIDO)  LINHA: 4
("float", FLOAT)   (";", SEMI)   ("teste2", ID)
("=", ATRIB)   ("1", INT NUM)  (";", SEMI)
("Read", ID)   ("(", OP_PAR)   ("a", ID)
(")", CL_PAR)   (";", SEMI)   ("b", ID)
("=", ATRIB)   ("a", ID)   ("*", MUL)
("a", ID)   (";", SEMI)   ("c", ID)
("=", ATRIB)   ("b", ID)   ("+", ADD)
("a", ID)   ("/", DIV)   ("2", INT NUM)
("*", MUL)   ("(", OP_PAR)   ("35", INT NUM)
("/", DIV)   ("b", ID)   (")", CL_PAR)
(";", SEMI)   ("write", WRITE)   ("c", ID)
(";", SEMI)   ("val", ID)   (":", INVALIDO)  LINHA: 11
("=", ATRIB)   ("34.2", FLOAT NUM)   ("c", ID)
("=", ATRIB)   ("val", ID)   ("+", ADD)
("2.2", FLOAT NUM)   ("+", ADD)   ("a", ID)
(";", SEMI)   ("write", WRITE)   ("(", OP_PAR)
("val", ID)   (")", CL_PAR)   ("end", END)
(".", PONTO)   ("", EOF) LINHA: 14

TABELA DE SIMBOLOS
Token: Read, Info: ID
Token: a, Info: ID
Token: b, Info: ID
Token: c, Info: ID
Token: d, Info: ID
Token: teste2, Info: ID
Token: val, Info: ID
Token: var, Info: ID
```

IMAGEM 3 - Segundo Teste e sua saída

O símbolo *:* não faz parte da linguagem e por isso é considerado inválido; a atribuição *teste2 = 1* se trata de um erro sintático, não léxico; a escrita errônea do método *read* com 'R'

maiúsculo é um erro léxico, pois a linguagem é case sensitive; a falta de *;* ao fim de cada declaração não é reportada por esse tipo de análise.

program
a, aux is int;
b is float
begin
b = 0;
in (a);
in(b);
if (a>b) then //troca variaveis
aux = b;
b = a;
a = aux
end;
write(a;
write(b)
	TOKENS ENCONTRADOS
	("program", PROGRAM) ("a", ID) (",", VIRG)
	("aux", ID) ("is", IS) ("int", INT)
	(";", SEMI) ("b", ID) ("is", IS)
	("float", FLOAT) ("begin", BEGIN) ("b", ID)
	("=", ATRIB) ("0", INT NUM) (";", SEMI)
	("in", ID) ("(", OP_PAR) ("a", ID)
	(")", CL_PAR) (";", SEMI) ("in", ID)
	("(", OP_PAR) ("b", ID) (")", CL_PAR)
	(";", SEMI) ("if", IF) ("(", OP_PAR)
	("a", ID) (">", GRT) ("b", ID)
	(")", CL_PAR) ("then", THEN) ("/", DIV)
	("/", DIV) ("troca", ID) ("variaveis", ID)
	("aux", ID) ("=", ATRIB) ("b", ID)
	(";", SEMI) ("b", ID) ("=", ATRIB)
	("a", ID) (";", SEMI) ("a", ID)
	("=", ATRIB) ("aux", ID) ("end", END)
	(";", SEMI) ("write", WRITE) ("(", OP_PAR)
	("a", ID) (";", SEMI) ("write", WRITE)
	("(", OP_PAR) ("b", ID) (")", CL_PAR)
	("", EOF) LINHA: 14
	TABELA DE SIMBOLOS
	Token: a, Info: ID
	Token: aux, Info: ID
	Token: b, Info: ID
	Token: in, Info: ID
	Token: troca, Info: ID
	Token: variaveis, Info: ID

IMAGEM 4 - Terceiro Teste e sua saída

O uso de variáveis antes da sua declaração não é um erro léxico, por isso a variável *in* é registrada normalmente; a linguagem não suporta comentários de linha, assim a tentativa é considerada como dois sinais de divisão seguidos por duas variáveis; a desordenação dos tokens não é verificada pelo analisador léxico.

```
programa teste4
/* Teste4 do meu compilador
      pontuacao, pontuacaoMaxina, disponibilidade is inteiro;
      pontuacaoMinima is char;
begin
   pontuacaoMinima = 50;
   pontuacaoMaxima = 100;
   write({Pontuacao do candidato: });
   read(pontuacao);
   write({Disponibilidade do candidato: });
   read(disponibilidade);

   while (pontuacao>0 & (pontuacao<=pontuacaoMaxima) do
      if ((pontuação > pontuacaoMinima) && (disponibilidade==1))
then
         write({Candidato aprovado.})
      else
         write({Candidato reprovado.})
      end
      write({Pontuacao do candidato: });
      read(pontuacao);
      write({Disponibilidade do candidato: });
      read(disponibilidade);
   end
end
```

```
TOKENS ENCONTRADOS
("programa", ID)      ("teste4", ID)  ("", EOF) LINHA: 24

TABELA DE SIMBOLOS
Token: programa, Info: ID
Token: teste4, Info: ID
```

IMAGEM 5 - Quarto Teste e sua saída

O comentário não foi fechado, assim todo o corpo do programa não é computado.

```
/* Teste do meu compilador */
program teste5
   a, b, c, maior is int;
   outro is char;
begin
   repeat
      write({A});
      read(a);
      write({B});
      read(b);
      write({C});
      read(c);




      if ( (a>b) && (a>c) ) end
         maior = a

      else
```

```
TOKENS ENCONTRADOS
("program", PROGRAM)   ("teste5", ID)  ("a", ID)
(",", VIRG)    ("b", ID)      (",", VIRG)
("c", ID)      (",", VIRG)    ("maior", ID)
("is", IS)     ("int", INT)   (";", SEMI)
("outro", ID)  ("is", IS)     ("char", CHAR)
(";", SEMI)    ("begin", BEGIN)      ("repeat", REPEAT)
("write", WRITE)      ("(", OP_PAR)  ("{A}", LIT)
(")", CL_PAR)  (";", SEMI)    ("read", READ)
("(", OP_PAR)  ("a", ID)      (")", CL_PAR)
(";", SEMI)    ("write", WRITE)      ("(", OP_PAR)
("{B}", LIT)   (")", CL_PAR)  (";", SEMI)
("read", READ)  ("(", OP_PAR)  ("b", ID)
(")", CL_PAR)  (";", SEMI)    ("write", WRITE)
("(", OP_PAR)  ("{C}", LIT)   (")", CL_PAR)
(";", SEMI)    ("read", READ)  ("(", OP_PAR)
("c", ID)      (")", CL_PAR)  (";", SEMI)
("if", IF)     ("(", OP_PAR)  ("(", OP_PAR)
("a", ID)      (">", GRT)     ("b", ID)
(")", CL_PAR)  ("&&", AND)    ("(", OP_PAR)
("a", ID)      (">", GRT)     ("c", ID)
(")", CL_PAR)  (")", CL_PAR)  ("end", END)
("maior", ID)  ("=", ATRIB)   ("a", ID)
("else", ELSE)  ("if", IF)     ("(", OP_PAR)
```

```
        if (b>c) then
            maior = b;

        else
            maior = c
        end
    end;
    write({Maior valor:}});
    write (maior);
    write ({Outro? (S/N)});
    read(outro);
  until (outro == 'N' || outro == 'n)
end
```

```
("b", ID)      (">", GRT)     ("c", ID)
(")", CL_PAR)  ("then", THEN) ("maior", ID)
("=", ATRIB)   ("b", ID)      (";", SEMI)
("else", ELSE) ("maior", ID)  ("=", ATRIB)
("c", ID)      ("end", END)   ("end", END)
(";", SEMI)    ("write", WRITE)      ("(", OP_PAR)
("{Maior valor:}", LIT) ("}", INVALIDO)  LINHA: 29
(")", CL_PAR)
(";", SEMI)    ("write", WRITE)      ("(", OP_PAR)
("maior", ID)  (")", CL_PAR)  (";", SEMI)
("write", WRITE)      ("(", OP_PAR)  ("{Outro? (S/N)}", LIT)
(")", CL_PAR)  (";", SEMI)    ("read", READ)
("(", OP_PAR)  ("outro", ID)  (")", CL_PAR)
(";", SEMI)    ("until", UNTIL)      ("(", OP_PAR)
("outro", ID)  ("==", EQ)     ("'N'", CHAR)
("||", OR)     ("outro", ID)  ("==", EQ)
("'n", INVALIDO)       LINHA: 33
(")", CL_PAR)  ("end", END)
("", EOF) LINHA: 34

TABELA DE SIMBOLOS
Token: a, Info: ID
Token: b, Info: ID
Token: c, Info: ID
Token: maior, Info: ID
Token: outro, Info: ID
Token: teste5, Info: ID
```

IMAGEM 6 - Quinto Teste e sua saída

Um literal não pode conter um delimitador em seu corpo; o carácter *'n'* não foi devidamente escrito, faltando as aspas de fechamento.