Trabalho Prático 1:

Implementação de um Analisador Léxico

Disciplina: Compiladores

Data: 2018/02

1 Especificação da Linguagem TINY

<PROGRAMA> ::= <DECL_SEQUENCIA>

<DECL_SEQUENCIA> ::= <DECL_SEQUENCIA> **;** <DECLARACAO> | <DECLARACAO>

<DECLARACAO> ::= <COND_DECL> | <REPET_DECL> | <ATRIB_DECL> |

<LEIT_DECL> | <ESCR_DECL>

<COND_DECL> ::= **if** <EXP> **then** <DECL_SEQUENCIA |

if <EXP> **then** <DECL_SEQUENCIA> **else** <DECL_SEQUENCIA> **end**

<REPET_DECL> ::= **repeat** <DECL_SEQUENCIA> **until** <EXP>

<ATRIB_DECL> ::= **identificador :=** <EXP>

<LEIT_DECL> ::= **read identificador**

<ESCR_DECL> ::= **write** <EXP>

<EXP> ::= <EXP_SIMPLES> <COMP_OP> <EXP_SIMPLES> | <EXP_SIMPLES>

<COMP_OP> ::= **<** | **=**

<EXP_SIMPLES> ::= <EXP_SIMPLES> <SOMA> <TERMO> | <TERMO>

<SOMA> ::= **+** | **-**

<TERMO> ::= <TERMO> <MULT> <FATOR> | <FATOR>

<MULT> ::= * | /

<FATOR> ::= **(**<EXP>**) | número | identificador**

COMENTÁRIOS

Uma vez que os comentários servem apenas como documentação do código fonte, ao realizar a compilação deste código faz-se necessário eliminar todo o conteúdo entre seus delimitadores

{ }

2. Palavras Reservadas e Tokens da Linguagem TINY

Tokens da linguagem TINY	
Lexema	**TOKEN**
If	IF
then	THEN
else	ELSE
end	END
repeat	REPEAT
until	UNTIL
read	READ
write	WRITE
+	PLUS
-	MINUS
*	TIMES
/	DIV
=	EQUAL
<	LESS
(	LBRACKET
)	RBRACKET
;	DOTCOMA
:=	ATRIB
número (1 ou mais dígitos [0..9])	NUM
Identificador (1 ou mais letras [a..zA..Z])	ID

3. Exemplo de Programa

```
{programa de exemplo  na linguagem TINY – computa o fatorial }
read x ;              {entrada de um inteiro}
if 0 < x then      {não calcula se x <= 0}
    fact := 1 ;
    repeat
       fact :=  fact * x ;
       x := x - 1
     until x = 0 ;
write fact         {saída do fatorial de x}
end
```

4. Analisador Léxico

O analisador léxico é a primeira fase de um compilador. Sua tarefa principal é a de ler os caracteres de entrada e produzir uma sequência de *token*s que o analisador sintático utilizará.

4.1 Os Algoritmos do Analisador Léxico

Uma vez definida a estrutura de dados do analisador léxico, é possível descrever seu algoritmo básico. No nível mais alto de abstração, o funcionamento do analisador léxico pode ser definido pelo algoritmo:

```
Algoritmo Analisador Léxico (Nível 0)
Inicio
      Abre arquivo fonte
      Enquanto não acabou o arquivo fonte
      Faça {
            Trata Comentário e Consome espaços
                  Pega Token
                  Coloca Token na Lista de Tokens
            }
       Fecha arquivo fonte
 Fim
```

```
Algoritmo Pega Token
Inicio
   Se caracter é digito
        Então Trata Digito
        Senão Se caracter é letra
                  Então Trata Identificador e Palavra Reservada
                  Senão ERRO
Fim.
```

```
Algoritmo Trata Dígito
Def num : Palavra
Inicio
    num ← caracter
    Ler(caracter)
    Enquanto caracter é dígito
       Faça {
                num ← num ++ caracter
                Ler(caracter)
              }
    token.símbolo ← snúmero
    token.lexema ← num
Fim.
```

```
Algoritmo Trata Identificador e Palavra Reservada
Def id: Palavra
Inicio
    id ← caracter
    Ler(caracter)
    Enquanto caracter é letra ou dígito
        Faça { id ← id ++ caracter
                Ler(caracter)
        }
    token.lexema ← id
    caso
        id = "programa": token.símbolo  ← sprograma
        id = "se": token.símbolo  ← sse
        id = "entao": token.símbolo  ← sentao
        id = "senao": token.símbolo  ← ssenao
        id = "enquanto": token.símbolo  ← senquanto
        id = "faca": token.símbolo  ← sfaca
        id = "Início": token.símbolo  ← sinício
        id = "fim": token.símbolo  ← sfim
        id = "escreva": token.símbolo  ← sescreva
        id = "leia":token.símbolo  ← sleia
        id = "var": token.símbolo  ← svar
        id = "Inteiro": token.símbolo  ← sinteiro
        id = "booleano": token.símbolo  ← sbooleano
        id = "verdadeiro": token.símbolo  ← sverdadeiro
        id = "falso": token.símbolo  ← sfalso
        id = "procedimento": token.símbolo  ← sprocedimento
        id = "funcao": token.símbolo  ← sfuncao
        id = "div": token.símbolo  ← sdiv
        id = "e": token.símbolo  ← se
        id = "ou": token.símbolo  ← sou
        id = "nao": token.símbolo  ← snao
        senão : token.símbolo ← sidentificador
Fim.
```

5. Trabalho Prático

5.1) Objetivo:

Implementar o analisador léxico para a linguagem TINY descrita na BNF da seção 1.

O analisador léxico receberá como entrada um arquivo no formato texto (código fonte da linguagem TINY) e retornará uma lista com os *tokens* reconhecidos da linguagem.

Por exemplo:

Entrada:

```
{ exemplo de programa na linguagem TINY}

read x ;
read y ;

soma := x + y ;

write soma
```

Saída:

1) Com sucesso:

```
<READ, 3 >
<ID, 3>
<DOTCOMA, 3>
<READ, 4>
<ID, 4>
<DOTCOMA, 4>
<ID, 6>
<ATRIB, 6>
<ID, 6>
<PLUS, 6>
<ID, 6>
<DOTCOMA, 6>
<WRITE, 8>
<ID, 8>
```

2) Com falha:
Indicar o tipo e localização (linha) do erro:
2.1) Identificador inválido;
2.2) Número inválido.

```
r@&d x ;
read y ;

soma := x + 5.0 ;
write soma
```

Erro linha 1: Identificador inválido!
Erro linha 4: Número inválido!

Lista de tokens reconhecidos com sucesso:
<ID, 1>
<DOTCOMA, 1>
<ID, 2>
<ATRIB, 4>
<PLUS, 4>
<DOTCOMA, 4>
<WRITE, 5>
<ID, 5>

Data de entrega: a definir

Trabalho Individual

Referências

→ **Compiladores Princípios e Práticas**. Kenneth C. Louden.

Compiladores. Princípios, Técnicas e Ferramentas. Alfred V. Aho, Ravi Sethi and Jeffrey D. Ullman.

Implementação de Linguagens de Programação: Compiladores. Ana Maria de Alencar Price e Simão Sirineo Toscani