Universidade Tecnológica Federal do Paraná
Curso de Engenharia da Computação
Compiladores
Prof. Marco Antonio Barbosa

Trabalho Prático de Implementação 1.

Objetivo: Construção de um analisador léxico simplificado.

Data de entrega/apresentação: a definir

1) A Construção do Analisador Léxico

A função do analisador léxico é reconhecer as palavras que fazem parte de uma linguagem (lexemas) e classificar estar palavras (determinar os tokens). Portanto, caberá ao analisador léxico *ler um arquivo texto como entrada* e *fornecer como saída a lista dos tokens reconhecidos.*

Para esta etapa do trabalho não será necessário o uso da ***tabela de símbolos*** portanto a lista de tokens será formada por uma tupla:

<nome_do_token, posição_do_token>

Onde:
nome_do_token – indica a classe do token reconhecido
posição do token - indica a linha em que determinada palavra foi reconhecida.

Por exemplo:

Entrada: valor := 50 ;
Saída: <Id, 1><Atrib, 1><Num, 1><Scolon, 1>

Sugestão: Como ponto de partida estabelecer os tokens para todos os lexemas da linguagem. Por exemplo:

Lexema	Token
identificador	Id
:=	Atrib
Numero	Num
;	Scolon
programa	Prog
Se	if
...	...

A escolha pela estratégia de implementação fica a cargo do aluno, portanto, poderá ser implementada uma das três alternativas vistas:
código,
tabela de transição ou
geradores automáticos (FLEX, JFLEX, etc.).

Universidade Tecnológica Federal do Paraná
Curso de Engenharia da Computação
Compiladores
Prof. Marco Antonio Barbosa

A gramática da linguagem **Small L**

<programa> ::= **programa** <identificador> **;** <bloco> <comandos>

<bloco> ::= **var** <declaracao>

<declaracao> ::= <nome_var> **:** <tipo> **;** | <nome_var > **:** <tipo> **;** <declaracao>

<nome_var> ::= <identificador> | <identificador> **,** <nome_var>

<tipo> ::= **inteiro** | **real** | **booleano**

<comandos> ::= <comando> | **inicio** <comando> **;** <comandos> **fim**

<comando> ::= <atribuicao> | <condicional> | <enquanto> | <leitura> | <escrita>

<atribuicao> ::= <identificador> **:=** <expressão>

<condicional> ::= **se** <expressão> **entao** <comandos> |
se <expressão> **entao** <comandos> **senao** <comandos>

<enquanto> ::= **enquanto** <expressão> **faca** <comando>

<leitura> ::= **leia (** <identificador> **)**

<escrita> ::= **escreva (** <identificador> **)**

<expressão> ::= <simples> | <simples> <op_relacional> <simples>

<op_relacional> ::= **<>** | **=** | **< | > | <= | >=**

<simples> ::= <termo> <operador> <termo> | <termo>

<operador> ::= **+** | **-** | **ou**

<termo> ::= <fator> | <fator> <op> <fator>

<op> ::= ***** | **div** | **e**

<fator> ::= <identificador> | <numero> | **(**<expressão>**)** | **verdadeiro** | **falso | nao** <fator>

<identificador> ::= **id**

<número> ::= **num**

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: **{ }**

Tipos Numéricos: Inteiros ({naturais positivos e negativos}) e Reais (float)

Identificadores: Letras seguidas de zero ou mais letras ou dígitos

Exemplo de um programa na linguagem **Small L**
Entrada: test.l

```
programa test ;
 var
    v : inteiro ;
    i , max , juro : inteiro ;
 inicio
     enquanto v <> -1 faca
     inicio
         leia ( v ) ;        { leia o valor inicial }
         leia ( juro ) ;    { leia a taxa de juros }
         leia ( max ) ;    { leia o periodo }
         valor := 1 ;
         i := 1 ;
         enquanto i <= max { (1+juro) elevado a n } faca
         inicio
            valor := valor * ( 1 + juro ) ;
            i := i + 1 ;
         fim
         escreva ( valor ) ;
     fim
 fim
```

saída:
[PROG, 1][Id, 1][PVIRG, 1][VAR, 2][ID, 3]...[FIM, 20]

Os Algoritmos do Analisador Léxico

Uma vez defina 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
```

Na tentativa de aproximar o algoritmo acima de um código executável, são feitos refinamentos sucessivos do mesmo. Durante este processo, surgem novos procedimentos, que são refinados na medida do necessário.

```
 Algoritmo Analisador Léxico (Nível 1)
 Def. token: TipoToken
 Inicio
    Abre arquivo fonte
    Ler(caracter)
    Enquanto não acabou o arquivo fonte
    Faça {Enquanto ((caracter = "{")ou
                        (caracter = espaço)) e
                         (não acabou o arquivo fonte)
            Faça { Se caracter = "{"
                    Então {Enquanto (caracter ≠ "}" ) e
                                (não acabou o arquivo fonte)
                      Faça Ler(caracter)
                      Ler(caracter)}
                    Enquanto (caracter = espaço) e
                            (não acabou o arquivo fonte)
                    Faça Ler(caracter)
                   }
            se caracter <> fim de arquivo
            então {Pega Token
                    Insere Lista}
             }
    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 Se caracter = ":"
                        Então Trata Atribuição
                        Senão Se caracter ∈ {+,-,*}
                                Então Trata Operador Aritmético
                                Senão Se caracter ∈ {<,>,=}
                                        EntãoTrataOperadorRelacional
                                        Senão Se caracter ∈ {; ,",", (, ), .}
                                                Então Trata Pontuação
                                                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 ou "_"
        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.
```

Referências

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

Compiladores Princípios e Práticas. Kenneth C. Louden.

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