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

Trabalho Prático 2

Objetivo: Construção de um Analisador Sintático.

Data de entrega/apresentação: 07/06/2022

Desenvolvimento: Para a implementação deste trabalho pode-se utilizar qualquer uma das técnicas vistas em aula que seja possível de implementar a linguagem: Análise sintática descente (*top down*) ou ascendente *(bottom up*) ou, utilizar uma ferramenta geradora como o ANTLR, Bison ou qualquer outra similar.

1) A Construção do Analisador Léxico

O analisador léxico desenvolvido no trabalho 1 deverá agora sofrer algumas modificações:

- O analisador léxico agora é visto como uma função do analisador sintático e, sempre que invocada por este, retorna um *token*.

2) A construção do analisador sintático

- O analisador sintático deverá ter a entrada do arquivo a ser analisado, passada como argumento, ou seja, não poderá ser fixado um nome de arquivo a ser analisado.
- A saída do analisador léxico é binária, isto é, o resultado é:
- Sintaticamente Correto ou Sintaticamente Incorreto
- Para fins de verificação de execução, o analisador deverá imprimir as produções que estiverem sendo executadas. Por exemplo:

```
C:\Compiladores\testes aula\exemplo LALR>.\exerc
( a )
 E -> a
 E -> ( A )
Ap -> A
 Sintaticamente Correto
```

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>

<bloco> ::= **var** <declaracao> **inicio** <comandos> **fim**

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

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

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

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

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

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

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

<enquanto> ::= **enquanto** <expressao> **faca** <comandos>

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

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

<expressao> ::= <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> | **(**<expressao>**)** | **verdadeiro** | **falso | nao** <fator>

<identificador> ::= **id**

<numero> ::= **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**

1) maior.sml

```
programa maior ;
var
 x , y : inteiro ;
 inicio
  leia ( x ) ;
  leia ( y ) ;
  se ( x <= y ) entao
     escreva ( x )
  senao
     escreva ( y )
fim
```

2) fatorial.sml

```
programa fatorial ;
 var
   i , fat , n : inteiro ;
   inicio
      leia ( n ) ;
      i := 1 ;
      fat := 1 ;
      enquanto ( i <= n ) faca
           fat := fat * i ;
           i := i + 1 ;
      escreva ( valor )
   fim
```

Referências

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

Compiladores Princípios e Práticas. Kenneth C. Louden.

Flex&Bison–JohnLevine

https://simran2607.medium.com/compiler-design-using-flex-and-bison-in-windows-a9642ebd0a43

http://dinosaur.compilertools.net/

http://alumni.cs.ucr.edu/~lgao/teaching/bison.html

https://www.skenz.it/compilers/flex_bison

https://www.antlr.org/