Projeto de Implementação de um Compilador para a Linguagem T++
Análise Sintatica (Trabalho – 2ª parte)

Gabriel Negrão Silva[1]
[1] **Universidade Tecnológica Federal do Paraná - Campo Mourão (UTFPR-CM)**
Campo Mourão – PR – Brasil, 27 de Outubro de 2017

itsg_negrao@hotmail.com

Resumo. *Este artigo descreve como ocorre o processo de análise léxica de um compilador construindo um analisador para a linguagem* ***T++*** *com o auxílio da ferramento PLY, tal como procedimentos, alterações, instruções, exemplos e análise dos resultados.*

1. Introdução

O trabalho proposto foi aplicado de forma gradual, sendo constituído de 4 partes principais, neste artigo discorreremos sobre a primeira parte deste trabalho proposto. Este relatório tem como objetivo mostrar como foi projetado e construído o algoritmo com o auxílio de uma ferramenta (PLY) que discorreremos sobre posteriormente junto com uma conclusão final. O código está identado e comentado, instruções auxiliares são dadas pelo no decorrer do artigo anexado.

2. Descrição e Procedimentos

2.1. Descrição da Gramática no padrão BNF

programa ::=	lista_declaracoes
lista_declaracoes ::=	lista_declaracoes declaracao | declaracao
declaracao ::=	declaracao_variaveis | inicializacao_variaveis | declaracao_funcao
declaracao_variaveis ::=	tipo ":" lista_variaveis
inicializacao_variaveis ::=	atribuicao
lista_variaveis ::=	lista_variaveis "," var | lista_variaveis “,” atribuicao | atribuicao | var
var ::=	ID | ID indice
indice ::=	indice "[" expressao "]" | "[" expressao "]"
tipo ::=	INTEIRO | FLUTUANTE
declaracao_funcao ::=	tipo cabecalho | cabecalho

cabecalho ::=	ID "(" lista_parametros ")" corpo FIM
lista_parametros ::=	lista_parametros "," parametro | parametro | vazio
parametro ::=	tipo ":" ID | parametro "[" "]"
corpo ::=	corpo acao | vazio
acao ::=	expressao | declaracao_variaveis | se | repita | leia | escreva | retorna | erro
se ::=	SE expressao ENTAO corpo FIM | SE expressao ENTAO corpo SENAO corpo FIM
repita ::=	REPITA corpo ATE expressao
atribuicao ::=	var ":=" expressao
leia ::=	LEIA "(" ID ")"
escreva ::=	ESCREVA "(" expressao ")"
retorna ::=	RETORNA "(" expressao ")"
expressao ::=	expressao_simples | atribuicao
expressao_simples ::=	expressao_aditiva | expressao_simples operador_relacional expressao_aditiva
expressao_aditiva ::=	expressao_multiplicativa

	| expressao_aditiva operador_soma expressao_multiplicativa
expressao_multiplicativa ::=	expressao_unaria | expressao_multiplicativa operador_multiplicacao expressao_unaria
expressao_unaria ::=	fator | operador_soma fator
operador_relacional ::=	"<" | ">" | "=" | "<>" | "<=" | ">="
operador_soma ::=	"+" | "-"
operador_multiplicacao ::=	"*" | "/"
fator ::=	"(" expressao ")" | var | chamada_funcao | numero
numero ::=	NUM_INTEIRO | NUM_PONTO_FLUTUANTE | NUM_NOTACAO_CIENTIFICA
chamada_funcao ::=	ID "(" lista_argumentos ")"
lista_argumentos ::=	lista_argumentos "," expressao | expressao | vazio

Tabela 1. Gramática da Linguagem T++ no padrão EBNF.

3. Resultados e Análise

Dado o algoritmo de teste escrito na linguagem T++ o qual recebe um número inteiro e escreve seu fatorial na tela como entrada para o analisador léxico programando em Python para a linguagem. (Figura 1).

O algoritmo a seguir (Figura 1) escrito na linguagem T++ anexado ao projeto, o qual escreve na tela o fatorial de um número inteiro n, é a entrada de teste para o analisador sintático programado em Python com auxílio da ferramenta PLY executado via terminal com Python 3.

```
1   inteiro: n
2
3   inteiro fatorial(inteiro: n)
4       inteiro: fat
5       se n > 0 então {não calcula se n > 0}
6           fat := 1
7           repita
8               fat := fat * n
9               n := n - 1
10          até n = +1.32e110
11          retorna(fat) {retorna o valor do fatorial de n}
12      senão
13          retorna(0)
14      fim
15  fim
16
17  inteiro principal()
18      leia(n)
19      escreva(fatorial(n))
20      retorna(0)
21  fim
```

Figura 1. Algoritmo de calculo fatorial em T++.

A saída do programa criado na linguagem Python para a análise sintática da linguagem T++ usando o programa de teste (Figura 1) escrito na linguagem T++ incorporado ao projeto é dada uma Árvore Sintática, a saída final executando o programa '*Analisador_Sintatico.py*' via terminal de comando utilizando Python 3 e passando o programa fat.tpp como parâmetro foi a seguinte (Figura 2).

Exemplo de execução via terminal:

---

root@root-PC:~/Yacc$ python3 Analisador_Sintatico.py sintatica-testes/fat.tpp
O Analisador Sintático Finalizou com Sucesso.

```
- programa
-- lista_declaracoes
--- lista_declaracoes
---- lista_declaracoes
----- declaracao
------ declaracao_variaveis
------- tipo
------- lista_variaveis
-------- var n
---- declaracao
----- declaracao_funcao
------ tipo
------ cabecalho fatorial
------- lista_parametros
-------- parametro n
--------- tipo
------- corpo
-------- corpo
--------- corpo
--------- acao
---------- declaracao_variaveis
----------- tipo
----------- lista_variaveis
------------ var fat
-------- acao
--------- se
---------- expressao
----------- expressao_simples
------------ expressao_simples
------------- expressao_aditiva
-------------- expressao_multiplicativa
--------------- expressao_unaria
---------------- fator
----------------- var n
------------ operador_relacional
------------ expressao_aditiva
------------- expressao_multiplicativa
-------------- expressao_unaria
--------------- fator
---------------- numero 0
```

Figura 2. Trecho da Árvore de Saída.

4. Discussão

A saída dos teste executados foram satisfatórias, tal como este demonstrado anteriormente. Há mais algoritmos programados em T++ para testes do analisador e o resultado obtido com a execução dos mesmos foram satisfatoriamente boas (cumpriram os requisitos mínimos).

Os experimentos foram realizados em computador pessoal móvel (Notebook) próprio com processador intel quad core e seu tempo de execução foi baixo e olho nú aparentemente de término instantâneo dado tamanho do teste de entrada.

A ferramenta de análise sintática PLY foi de grande ajuda e serviu como base para inicio deste projeto, encapsulando certas funções autônomas das quais o programador não precisa se preocupar como funcionam internamente, não necessitando implementação própria do programador também.

Discutiremos a seguir sobre o formato na Análise Sintática realizado pela ferramenta PLY, sobre a implementação e utilização da ferramenta Yacc.

O trecho de código a seguir (Figura 3) define os tokens para o yacc atribuindo a classe Analisador_Sintático como módulo 'self' carregada a precedência e os tokens e chama a árvore sintática abstrata *'ast'* que recebera a execução final do parser, yacc inicializado.

```
26 - class Analisador_Sintatico:
27
28 -     def __init__(self, code):
29           al = Analisador_Lexico(sys.argv[1])
30           self.tokens = al.tokens
31 -         self.precedence = (
32               ('left', 'IGUALDADE', 'MAIORIGUAL', 'MAIOR', 'MENORIGUAL', 'MENOR'),
33               ('left', 'SOMA', 'SUBTRACAO'),
34               ('left', 'MULTIPLICACAO', 'DIVISAO'),
35           )
36           arq = open(code,'r')
37           data = arq.read()
38           parser = yacc.yacc(debug=False, module=self, optimize=False)
39           self.ast = parser.parse(data)
```

Figura 3. Trecho da Inicialização '*PLY.Yacc*'.

4.1. Discussão sobre implementação da Árvore Sintática

A Classe Tree implementa a *'ast' (abstract syntax tree)*, também chamada de Árvore Sintática Abstrata (Figura 4). A ferramenta PLY utiliza de uma syntax 'p_' anterior as funções para definir a BNF, a construção da árvore ocorre na volta da recursão da chamada das funções (Figura 5) e são printadas usando a função 'print_tree' (Figura 6) ao fim do programa (Figura 7).

```
8   # Importação ply lexer
9   from graphviz import Graph
10  import ply.yacc as yacc
11  from Analisador_Lexico import Analisador_Lexico
12  import sys
13  import os
14
15  class Tree:
16
17      def __init__(self, type_node='', child=[], value=''):
18          self.type = type_node
19          self.child = child
20          self.value = value
21
22      def __str__(self):
23          return self.type
```

Figura 4. Imports e Implementação da Classe Tree.

```
40
41      def p_programa(self,p):
42          '''programa : lista_declaracoes'''
43          p[0] = Tree('programa', [p[1]])
44
45      def p_lista_declaracoes(self,p):
46          '''lista_declaracoes : lista_declaracoes declaracao
47                               | declaracao'''
48          if len(p) == 3:
49              p[0] = Tree('lista_declaracoes', [p[1], p[2]])
50          elif len(p) == 2:
51              p[0] = Tree('lista_declaracoes', [p[1]])
52
53      def p_declaracao(self,p):
54          ''' declaracao : declaracao_variaveis
55                         | inicializacao_variaveis
56                         | declaracao_funcao '''
57          p[0] = Tree('declaracao',[p[1]])
58
```

Figura 5. Construção da Árvore (Tree) e funções BNF PLY yacc.

```
296 - def print_tree(node, level="-"):
297 -     if node != None:
298           print("%s %s %s" %(level, node.type, node.value))
299 -         for son in node.child:
300               print_tree(son, level+"-")
```

Figura 6. Função print_tree.

```
302 - if __name__ == '__main__':
303       Analisador_Sintatico.teste()
304       f = Analisador_Sintatico(sys.argv[1])
305
306       #inicia a arvore sintatica de saida
307       g = Graph('Arvore Sintatica',format='png')
308       #salvando imagem da arvore sintatica.
309       g.render('ArvoreSintatica')
310
311       print("_____")
312       print("O Analisador Sintático Finalizou com Sucesso.\n")
313       print("_____")
314       print_tree(f.ast)
```

Figura 7. Chamada do Analisador e Print da Árvore.

5. Conclusão

A parte complicada, se é que houve, é a de integração do conteúdo com o projeto implementado, tal como, BNF e Árvore Sintática vistos em aula e implementados com a ferramenta PLY, após ocorrido o entendimento do mesmo, a programação se torna mais fluida. Este projeto teve como contribuição o entendimento da parte Sintática de um compilador referindo a disciplina de Compiladores, das funcionalidades e construção de um compilador e das ferramentas de suporte ao projeto, mais especificamente a parte de um analisador Sintático, parte integrante de um compilador.