**Análise Sintática**

**Mateus Tomoo Yonemoto Peixoto**

DACOM – Universidade Tecnológica Federal do Paraná (UTFPR)

Caixa Postal 271 – 87301-899 – Campo Mourão – PR – Brazil

{mateustomoo}@gmail.com

**Abstract.** This paper describes the development of the sintactic analysis for a compiler being designed for the T++ programming language. In the future, it will so also approach the development of the semantic analysis.

**Resumo.** Este artigo descreve o desenvolvimento da análise sintática para um compilador projetado para a linguagem de programação T++. No futuro, também abordará o desenvolvimento da análise semântica.

# Introdução

Um compilador exige alguns passos até que um código final seja obtido. Esses passos são divididos em análise léxica, análise sintática, análise semântica e geração de código.

A Análise Sintática é responsável por determinar a estrutura sintática de um programa a partir dos tokens separados na primeira etapa (análise léxica). Ou seja, irá verificar se as sentenças são válidas para a linguagem de programação, no caso T++.

Para o desenvolvimento deste trabalho, foi utilizado a linguagem de programação Python (versão 3.6.2), onde possui a biblioteca chamada PLY, que contém ferramentas léxicas e sintáticas. Também foi utilizado o Python YACC.

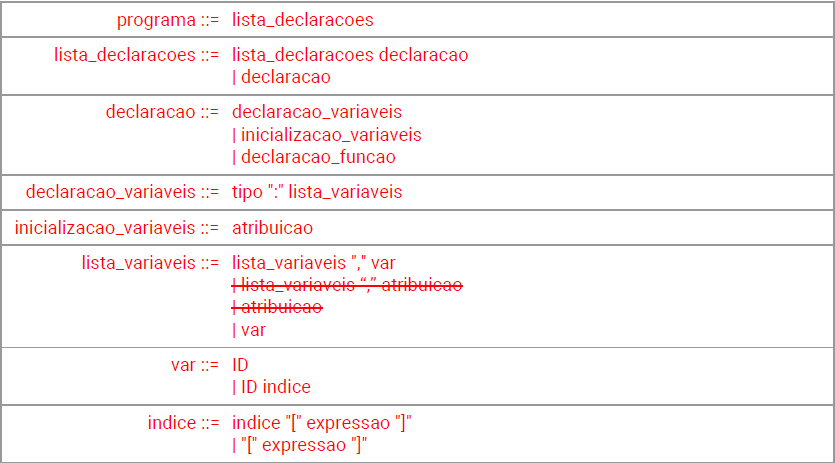
# 2. A linguagem

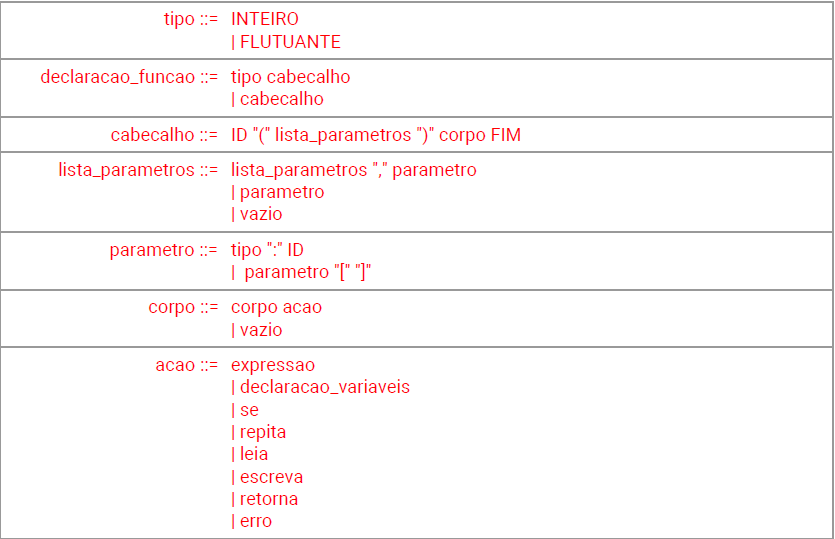
A linguagem T++ possui algumas características, dentre elas são:

* Tipos básicos de dados suportado: inteiro e flutuante;
* Suporte a arranjos uni e bidimensionais (arrays);
* Variáveis globais e locais devem ter um dos tipos especificados;
* Linguagem quase fortemente tipificada: nem todos os erros são especificados, mas sempre deve ocorrer avisos.

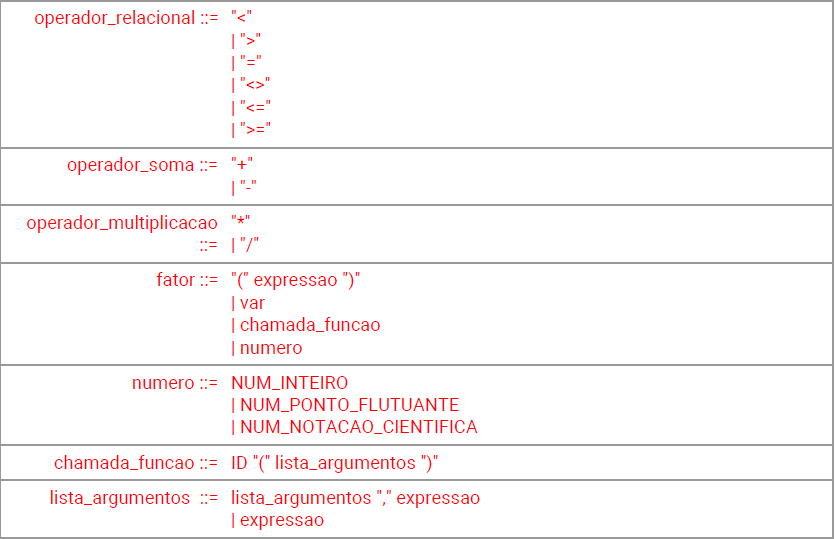
## 3. Análise Sintática

A gramática no padrão BNF utilizada foi:











**3.1. Formato da Análise Sintática**

O formato na Análise Sintática realizado pela ferramenta é a LR.

O formato LR é do tipo bottom-up, onde o L significa que o analisador lê o texto de entrada da esquerda para a direita e o R significa que o analisador produz uma derivação mais a direita em sentido inverso, ou seja, faz uma análise bottom-up. O padrão LR normalmente é seguido por um número, como por exemplo LR(1) e muitas vezes é precedido por outros qualificadores, como o LALR.

O padrão LR é determinístico, ou seja, produz uma única análise sem adivinhações. Isso ocorre pois ele aguarda até ter visto uma instância completa de alguma gramática antes de se comprometer com o que encontrou, diferente de um LL que deve decidir ou adivinhar o que está vendo muito mais cedo.

Outra característica do LR é que lidam com uma gama maior de linguagens e gramáticas em relação à por exemplo o LL (top-down).

**3.2. YACC**

A ferramenta YACC é uma ferramenta que a biblioteca PLY nos permite utilizar como analisador sintático. Para implementação de tal, foi utilizado os seguintes comandos:



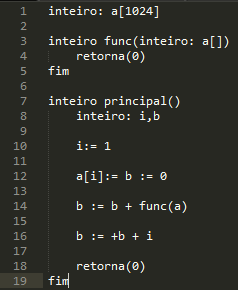
A primeira linha em questão se remete a invocação da ferramenta YACC, gerando o “parser” e armazenando na variável parser. A segunda linha utiliza a função parse da variável parser, passando como parâmetro um código, onde esse código será analisado sintaticamente e armazenado.

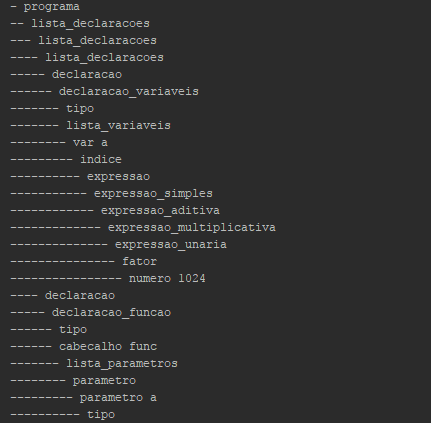
**3.3. Árvore Sintática**

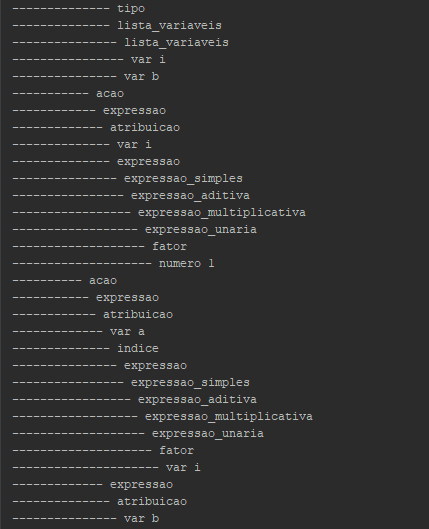
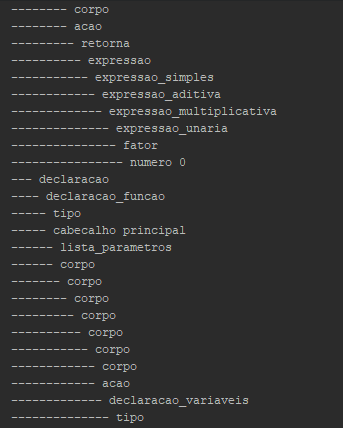
Após a utilização da ferramenta YACC e implementar a gramática da linguagem, uma árvore sintática é gerada. Essa árvore possui tipos de nós, os filhos e seus valores. Para a impressão da árvore, foi criado uma função print\_tree, onde por parâmetro é passado os nós dessa árvore, no caso a análise sintática feita pelo parser mostrado na sessão anterior.

**3.4. Saída**

A seguir é mostrado um exemplo de código em T++ e sua respectiva saída sintática.







## 4. Referências

LOUDEN, Kenneth C. Compiladores: princípios e práticas. São Paulo, SP:

Thomson, c2004. xiv, 569 p. ISBN 8522104220.

http://hackingoff.com/compilers/regular-expression-to-nfa-dfa

PLY (Python Lex-Yacc). Disponível em: <http://www.dabeaz.com/ply/>.