

# Análise Sintática na implementação de um compilador para a Linguagem T++

Guilherme Vasco da Silva

Departamento de Ciências da Computação  
Universidade Tecnológica Federal do Paraná (UTFPR) – Campo Mourão, PR – Brazil  
guilhermevasco@alunos.utfpr.edu.br

**Resumo.** Este artigo é um trabalho com o intuito de documentar um programa/ferramenta para realização da análise sintática de códigos na linguagem de programação T++.

## 1. Linguagem T++

A linguagem a ser lida pelo analisador sintático será a T++, que suporta dados inteiros e flutuantes e necessidade da especificação de tipos das variáveis locais e globais, porém as funções podem ter tipo omitido. A linguagem possui também operações de adição, multiplicação, subtração, divisão e também operadores lógicos E, OU e NÃO.

## 2. A Ferramenta

O analisador sintático para a linguagem T++ é composto dos arquivos *tppparser.py*, *tpplex.py*, *mytree.py* e *lextrab.py*, sendo o primeiro citado o principal para a execução. Os arquivos *tpplex* e *lextrab* já foram utilizados anteriormente e são responsáveis pela parte da Análise Léxica do código, o arquivo *tppparser* é responsável pela Análise Sintática, sendo o *mytree* um auxiliar para a geração das árvores sintáticas.

A ferramenta utiliza as bibliotecas do Python Anytree, PLY e Yacc, a importação das mesmas é necessária para a compilação, a Yacc vem inclusa com a PLY.




## 3. Gramática de acordo com o padrão BNF

De acordo com o padrão BNF, a linguagem T++ possui a seguinte gramática:

**Tabela 1. Gramática no padrão BNF da linguagem T++.**

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   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 "(" var ")"
escreva ::=	ESCREVA "(" expressao ")"
retorna ::=	RETORNA "(" expressao ")"
expressao ::=	expressao_logica   atribuicao

expressao_logica ::=	expressao_simples   expressao_logica operador_logico expressao_simples
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_negacao fator
operador_relacional ::=	"<"   ">"   "="   "<="    "<="    ">=" 
operador_soma ::=	"+"   "-"
operador_logico ::=	"&&"   "  "
operador_negacao ::=	"!"
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

É importante notar que é possível ter uma lista de variáveis do mesmo tipo.

#### 4. Formato na Análise Sintática realizado pela ferramenta

A ferramenta realiza a análise sintática de forma ascendente LALR(1), ou seja, estados iguais na tabela são juntados em relação aos itens. O Yacc suporta esse tipo de análise.

Outra funcionalidade da ferramenta é apresentar onde estão erros sintáticos nos códigos passados para ela onde não é possível gerar árvore sintática.

#### 5. Implementação da ferramenta utilizando Yacc

Na implementação no Python foi utilizado o Yacc (Yet Another Compiler Compiler), que é um gerador de analisador sintático. Tal analisador sintático requer também um analisador léxico, que foi fornecido a partir do criado na primeira parte do desenvolvimento do compilador proposto para T++. O analisador léxico é responsável por retornar os tokens que o analisador sintático necessita para a operação de parse, assim utilizando uma gramática formal para realizar a análise sintática das entradas. Como citado anteriormente, o Yacc suporta a forma LALR(1), por isso ela está sendo utilizada.

#### 6. Árvore Sintática

A ferramenta utiliza o Mytree (parte do Anytree) para a geração gráfica da árvore sintática, ele é responsável por desenhar gráficos a partir de arquivos DOT, por isso, a ferramenta cria arquivos desse tipo e os mesmos são convertidos em versões gráficas da árvore, exportada em PNG.

A árvore sintática é composta por nós, esses nós são criados na ferramenta dentro de um construtor na classe MyNode, para ele é passado nome, pai, ID, tipo, label e filhos e o construtor cria o nó a partir dessas informações.

```
class MyNode(NodeMixin): # Add Node feature

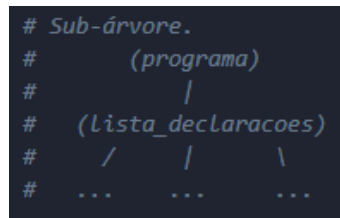
    def __init__(self, name, parent=None, id=None, type=None, label=None, children=None):
        super(MyNode, self).__init__()
        global node_sequence

        if (id):
            self.id = id
        else:
            self.id = str(node_sequence) + ': ' + str(name)

        self.label = name
        # self.name = name + '_' + str(node_sequence)
        self.name = name
        node_sequence = node_sequence + 1
        self.type = type
        self.parent = parent
        if children:
            self.children = children
```

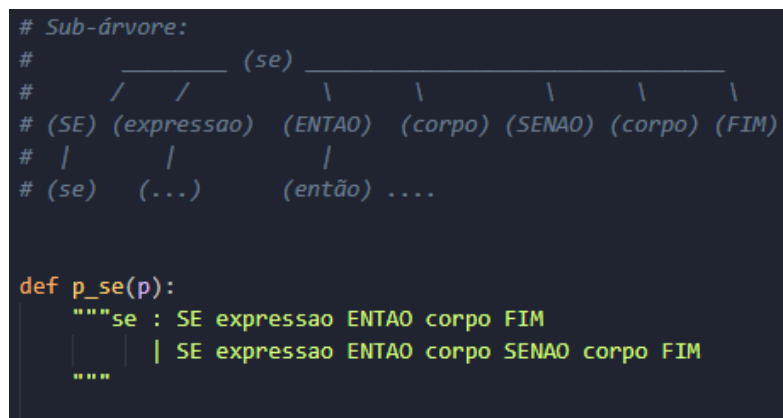
Figura 1. Criação de um nó (MyNode) dentro do arquivo mytree.py.

O processo de criação da árvore utiliza a criação de sub-árvores com programa e lista de declarações, de acordo com a regra do T++. O programa é tipo como raiz, que é ligado à lista de declarações, essa por sua vez é ligada a novas sub-árvores abaixo.



**Figura 2. Modelo da sub-árvore.**

Para cada regra existente na linguagem é montado um tipo de sub-árvore de acordo com suas listas de declarações e variáveis. A regra *SE*, por exemplo, possui até 8 partes que podem precisar ser posicionadas na sub-árvore.



**Figura 3. Modelo de sub-árvore possível para a regra SE (cima) acompanhada da declaração da função da regra (baixo).**

Ao longo da ferramenta as sub-árvores se juntam, resultando na árvore sintática do programa passado, o DotExporter (presente no Anytree) é utilizado ao final para gerar a versão gráfica da árvore.

```

if root and root.children != ():
    print("Generating Syntax Tree Graph...")
    DotExporter(root).to_picture(argv[1] + ".ast.png")
    UniqueDotExporter(root).to_picture(argv[1] + ".unique.ast.png")
    DotExporter(root).to_dotfile(argv[1] + ".ast.dot")
    UniqueDotExporter(root).to_dotfile(argv[1] + ".unique.ast.dot")
    print(RenderTree(root, style=AsciiStyle()).by_attr())
    print("Graph was generated.\nOutput file: " + argv[1] + ".ast.png")

    DotExporter(root, graph="graph",
                 nodelabelfunc=MyNode.nodelabelfunc,
                 nodeattrfunc=lambda node: 'label=%s' % (node.type),
                 edgeattrfunc=MyNode.edgeattrfunc,
                 edgetypefunc=MyNode.edgetypefunc).to_picture(argv[1] + ".ast2.png")

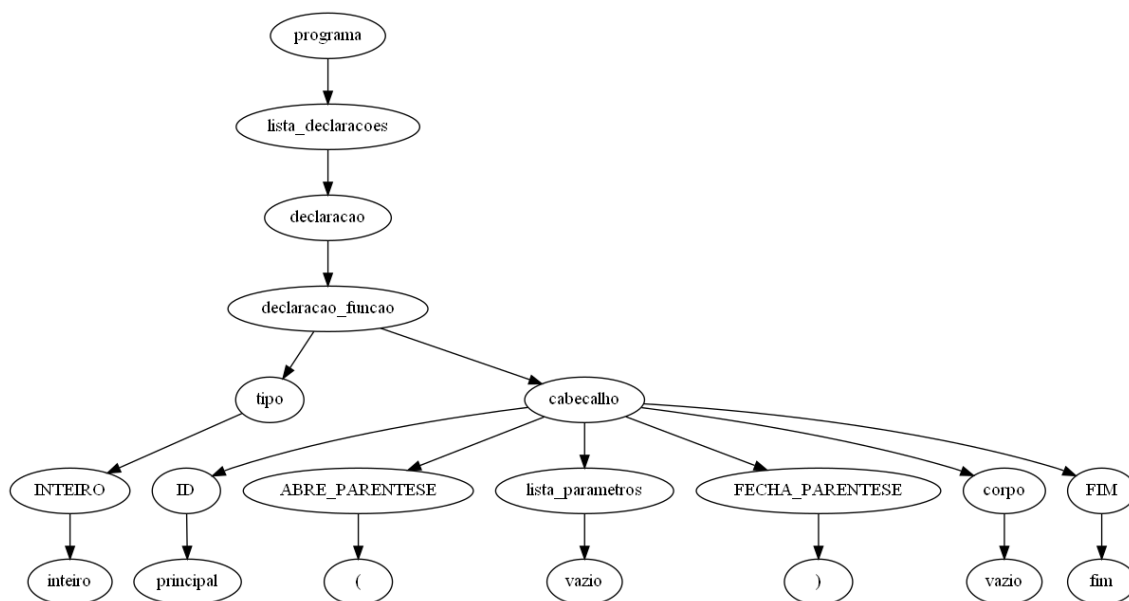
```

**Figura 4. Uso do DotExporter do Anytree para geração gráfica da árvore em PNG.**

## 7. Testes

Junto aos arquivos da ferramenta está presente a pasta *sintatica-testes*, que fornece diversos arquivos de programas escritos em T++ para a realização de testes do conjunto de analisador léxico e analisador sintático desenvolvidos para o compilador.

Alguns dos arquivos de testes funcionam e geram árvores perfeitamente, outros possuem erros que serão apontados na análise sintática pela ferramenta. O arquivo *erro-003.tpp* é um que funciona e possui geração de árvore, pode ser executado através do comando `python tppparser.py sintatica-testes/erro-003.tpp`.



**Figura 5. Imagem da árvore sintática do programa erro-003.tpp (erro-003.tpp.unique.ast.png), gerada pela ferramenta.**

Já o arquivo *fat.tpp* possui problemas sintáticos, por isso não será possível gerar a árvore e a ferramenta retorna um *log* dos problemas encontrados. Isso pode ser visto tentando executar a ferramenta no arquivo com o comando `python tppparser.py sintatica-testes/fat.tpp`.

```
Generating LALR tables
WARNING: 47 shift/reduce conflicts
Erro:[5,5]: Erro próximo ao token 'entÃ'
Caracter inválido 'f'
Caracter inválido '@'
Caracter inválido 'f'
Erro:[14,14]: Erro próximo ao token 'fim'
Erro:[18,18]: Erro próximo ao token 'leia'
Erro:[21,21]: Erro próximo ao token 'fim'
Unable to generate Syntax Tree.
```

**Figura 6. Log no terminal dos erros na execução do arquivo fat.tpp pela ferramenta.**

## 8. Resultados obtidos

Os resultados obtidos foram satisfatórios, as árvores foram geradas completamente e o analisador sintático trabalhou bem em conjunto com o analisador léxico, dando boa continuidade no desenvolvimento do compilador. As árvores geradas não possuem qualquer tipo de redução ou simplificação, resultando em todos os nós necessários. A visualização pode ser difícil por conta do número de nós que são resultados, mas a geração do arquivo DOT, juntamente à PNG, faz com que seja possível abrir a visualização da árvore em outros ambientes, como o Graphviz.

O log gerado para os arquivos que não podem ter árvores geradas aponta caracteres inválidos e linhas, bem como proximidade, onde estão os possíveis erros, o que auxilia na correção.

## Referências

- Gonçalves, R. A. (2017) “Documentação online da Gramática da TPP”, [https://docs.google.com/document/d/1oYX-5ipzL\\_izj\\_hO8s7axuo2OyA279YEhnAItgXzXAAQ](https://docs.google.com/document/d/1oYX-5ipzL_izj_hO8s7axuo2OyA279YEhnAItgXzXAAQ), Novembro.
- Louden, K. C. (2004) “Análise Sintática Ascendente”, Em: Compiladores: princípios e práticas., EUA.
- Beazley, D. M. (2021) “PLY (Python Lex-Yacc)”, <https://www.dabeaz.com/ply/ply.html>, Novembro.
- Johnson, S. C. (1979) Yacc: Yet Another Compiler-Compiler.