Analisa Léxica da Linguagem T++

Humberto Moreira Gonçalves

November 24, 2016

1 Introdução

Este documento descreve como foi implementado o analisador léxico da linguagem T++ proposta na disciplina de Compiladores.

2 Linguagem T++

Esse projeto consiste na implementação da análise léxica de um compilador para a linguagem de programação T++. Será demonstrado as palavras reservadas e expressões regulares da análise léxica. Um compilador é um programa que possui a capacidade de converter a implementação de uma determinada linguagem para código de máquina, muito compiladores geram código em Assembly, dado que é uma linguagem mais próxima do binário utilizado no código de máquina.

T++ é uma linguagem de programação simples que foi construída para intuito de aprendizado de compiladores e para realizações de operações simples de matemática, possuindo operadores matemáticos, como adição, subtração, multiplicação e divisão, e operadores lógicos, como maior, menor, etc. Fornecendo número inteiros e de ponto flutuante.

3 Análise Léxica

Essa análise se baseia em um conjunto de tokens de palavras reservadas e símbolos. Também possuindo número inteiro ou ponto flutuante

Palavras reservadas	Símbolos	Outros
se	+ soma	número
então	- subtração	identificador
senão	* multiplicação	comentário
fim	/ divisão	nova linha
repita	= igualdade	
flutuante	, vírgula	
retorna	:= atribuição	
até	< menor	
leia	> maior	
escreve	<= menor-igual	
inteiro	>= maior-igual	
	(abre-par	
	) fecha-par	
	: dois-pontos	

Table 1: Classes de tokens e suas definições

Com base na tabela 1, foi criado o automâto para cada token, da seguinte forma:

Desta forma, a implementação dos tokens na tabela 1, ficou como a figura 2:

Para cada token é necessário a utilização de de expressões regulares, para que possam ser reconhecidos na passada de leitura do código, como na figura 4:

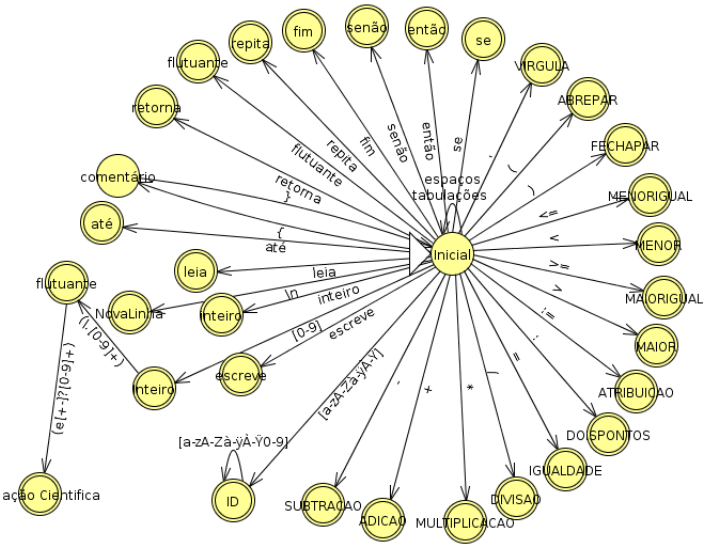

Figure 1: Automato dos tokens

```
# Dicionario reservadas
keywords = {
    u'se': 'SE',
    u'então': 'ENTAO',
    u'senão': 'SENAO',
    u'fim': 'FIM',
    u'repita': 'REPITA',
    u'flutuante': 'FLUTUANTE',
    u'retorna': 'RETORNA',
    u'até': 'ATE',
    u'leia': 'LEIA',
    u'escreve': 'ESCREVE',
    u'inteiro': 'INTEIRO',
    u'principal': 'PRINCIPAL',
    }
```

Figure 2: Implementação das palavras reservadas

```
# Lista de Tokens
tokens = ['ADICAO', 'SUBTRACAO', 'MULTIPLICACAO', 'DIVISAO', 'IGUALDADE',
          'VIRGULA', 'ATRIBUICAO', 'MENOR', 'MAIOR', 'MENORIGUAL', 'MAIORIGUAL',
           'ABREPAR', 'FECHAPAR', 'DOISPONTOS','NUMERO', 'ID'] + list(keywords.values())
```

Figure 3: Implementação dos tokens

```
# Expressões simples
t_ADICAO = r'\+'
t_SUBTRACAO = r'\-'
t_MULTIPLICACAO = r'\*'
t_DIVISAO = r'/'
t_IGUALDADE = r'='
t_VIRGULA = r','
t_ATRIBUICAO = r':='
t_MENOR  = r'<'
t_MAIOR = r'>'
t_MENORIGUAL = r'<='
t_MAIORIGUAL = r'>='
t_ABREPAR = r'\('
t_FECHAPAR = r'\)'
t_DOISPONTOS = r':'
t_NUMERO = r'[0-9]+(\.[0-9]+)?'


def t_ID(self, t):
    r'[a-zA-Zà-ú][a-zA-Zà-úÀ-Ú0-9]*'
    t.type = self.keywords.get(t.value, 'ID')
    return t

def t_COMMENT(self, t):
    r'\{[^\{^\}]*\}'
    pass

def t_NEWLINE(self, t):
    r'\n+'
    t.lexer.lineno += len(t.value)
#Ignora tabs e espacos
t_ignore = ' \t'
```

Figure 4: Automato dos tokens

4 Análise Sintática

A gramática é essencial para o compilador, visto que ela é responsável pelo conjunto de regras de produção da linguagem, que descreve como formar a implementação, sendo validada pela análise sintática da

linguagem. Portanto, é preciso criar a linguagem utilizando as palavras reservadas e os tokens utilizados anteriormente na análise léxica para montar um código completo. Para esse compilador, deve-se considerar o fato do código possuir, variáveis globais, funções e função principal. O código por possuir qualquer função e qualquer varáveis global e função principal independente da ordem. A tabela a seguir, apresenta a implementação da gramática para a análise sintática:

<programa> ::= <statement> <programa>
| <statement>

<statement> ::= <declaracao_de_funcao>
| <declara_var>

<declaracao_de_funcao> ::= <tipo> IDENTIFICADOR (<declaracao_param>) <sequencia_de_declaracao> FIM
| <tipo> IDENTIFICADOR (<declaracao_param>) FIM
| <tipo> IDENTIFICADOR ( ) FIM
| <tipo> IDENTIFICADOR ( ) <sequencia_de_declaracao> FIM

<declaracao_param> ::= <declaracao_param> VIRGULA <tipo> : IDENTIFICADOR
| <tipo> : IDENTIFICADOR

<sequencia_de_declaracao> ::= <declaracao>
| <declaracao> <sequencia_de_declaracao>

<declaracao> ::= <expressao_condicional>
| <expressao_iteracao>
| <expressao_:=>
| <expressao_leitura>
| <expressao_escreva>
| <declara_var>
| <retorna>
| <chamada_de_funcao>

<expressao_condicional> ::= SE <expressao> ENTAO <sequencia_de_declaracao> SENAO <sequencia_de_declaracao> FIM
| SE <expressao> ENTAO <sequencia_de_declaracao> FIM

<expressao_iteracao> ::= REPITA <sequencia_de_declaracao> ATE <expressao>
<expressao_:=> ::= IDENTIFICADOR := <expressao>
<expressao_leitura ::= LEIA (IDENTIFICADOR)
<expressao_escreva ::= ESCREVE (<expressao>)
<declara_var ::= <tipo> : IDENTIFICADOR
| <tipo> : IDENTIFICADOR VIRGULA <declara_outra_var>
| <tipo> : IDENTIFICADOR := <expressao>
<declara_outra_var> ::= IDENTIFICADOR

<retorna> ::= RETORNA (<expressao>)
<chamada_de_funcao> ::= IDENTIFICADOR (<param_chama_funcao>)

<param_chama_funcao> ::= <param_chama_funcao> VIRGULA <expressao>
| <expressao>
<expressao> ::= <expressao_simples>
| <expressao_simples> <comparacao_operador> <expressao_simples>

<comparacao_operador ::= MAIOR
| MAIORIGUAL
| MENOR
| MENORIGUAL
| IGUALDADE

<expressao_simples> ::= <expressao_simples> <soma> <termo>
| <termo>

<soma> ::= ADICAO
| SUBTRACAO
<mult> ::= MULTIPLICACAO
| DIVISAO
<termo> ::= <fator>
| <termo> <mult> <fator>
<fator> ::= (<expressao>)
| <chamada_de_funcao>
| <expressao_numero>
| <expressao_identificador>
<expressao_identificador> ::= IDENTIFICADOR

<expressao_numero> ::= <numero>

<tipo> ::= INTEIRO
| FLUTUANTE

<numero> ::= INTEIRO
| FLUTUANTE

Como é necessário a criação de um nó da árvore para cada derivação da análise sintática, ao final de cada derivação é adicionado um nó na árvore com a seguinte chamada ArvoreSintatica("tipo", filho, folha). O tipo seria quem produziu, o parâmetro filho é vazio caso a produção não gere nenhuma outro produção, caso contrário é necessário passar a posição da chamada na gramática, e o folha é vazio se a produção não gerar nenhum simbolo terminal, se gerar, é necessário passar a posição do símbolo terminal na produção. O segundo e o terceiro parâmetro é uma lista, portanto é necessário passar os nós filho e folha como lista, ficando da seguinte forma ArvoreSintatica('tipo', [folha], [filho]).

5 Análise Semântica

Após a análise sintática, é necessário fazer uma análise semântica , a fase onde erros semânticos são tratados. Com saída ArvoreSintatica gerada pela fase anterior, agora é necessário recebendo como entrada nessa fase e processar para gerar uma forma mais simples do código e aproximar para geração de código, que é a próxima fase.

Aqui, é visto análise de tipo de variáveis, análise de retorno de função, atribuição em variáveis não declaradas e afins, por exemplo.

Para o trabalho foi desenvolvido *Warnings* e Erros, alguns são:

1. Erro: Atribuição em varíável não declarada;
2. Erro: Múltiplas declarações como mesmo nome;
3. Erro: Tipo de retorno de função diferente do tipo chamada da função;
4. Erro: Função não declarada;
5. *Warnings*: Atribuição de tipo flutuante em uma variável inteiro, e vice-versa.

Para manter informações, foi utilizado a tabela de símbolos, uma estrutura em forma de *Hash*. Nessa tabela foi armazenado o escopo de cada variável e função.

```
from parser import Tree
from parser import AnaliseSintatica

class Semantica():

    def __init__(self,codigo):
        self.tabela ={}
        self.escopo = "global"
        self.tree = AnaliseSintatica().parser_codigo(codigo)
```

```
def raiz(self):
    if(self.tree.type == "statement_loop"):
        self.statement(self.tree.child[0])
        self.programa(self.tree.child[1])

    if(self.tree.type == "statement_sem_loop"):
        self.statement(self.tree.child[0])

def programa(self,node):
    if(node.type == "statement_loop"):
        self.statement(node.child[0])
        self.programa(node.child[1])
    if(node.type == "statement_sem_loop"):
        self.statement(node.child[0])

def statement(self,node):
    if(node.type == "statement_declaracao_de_funcao"):
        self.declaracao_de_funcao(node.child[0])

    if(node.type == "statement_declara_var"):
        self.declara_var(node.child[0])


def  declaracao_de_funcao(self,node):
    if(node.value in self.tabela.keys()): #se ja tem funcao com esse nome na tabela
        print("Erro Semântico, o nome "+  node.value + " já esta sendo utilizado")
        exit(1)


    # print(node.type)
    if(node.type == "declaracao_de_funcao_td"):

        self.tabela[node.value] = {}
        self.tabela[node.value]["variavel"] = False
        self.tabela[node.value]["tipo"] = self.tipo(node.child[0])
        self.tabela[node.value]["num_parametros"] = 0

        self.escopo = node.value
        self.tabela[node.value]["num_parametros"] = self.declaracao_param(node.child[1]) #rece
        # self.tabela[node.value]["parametros"] =[]
        self.tabela[node.value]["parametros"] = self.param(node.child[1],[])
        print(self.param(node.child[1],[]),"oi")
        self.sequencia_de_declaracao(node.child[2])
        self.escopo = "global"



    if(node.type == "declaracao_de_funcao_sem_corpo"):
        self.tabela[node.value] = {}
        self.tabela[node.value]["variavel"] = False
        self.tabela[node.value]["tipo"] = self.tipo(node.child[0])
        self.tabela[node.value]["num_parametros"] = 0

        self.escopo = node.value  #escopo nome da função
        self.tabela[node.value]["num_parametros"] = self.declaracao_param(node.child[1]) #rece
        # self.tabela[node.value]["parametros"] =[]
        self.tabela[node.value]["parametros"] = self.param(node.child[1],[])
        print(self.param(node.child[1],[]),"oi")
        self.escopo = "global"
```

```
        if(self.tree.type == "declaracao_de_funcao_sem_corpo_sem_parametros"):

            self.tabela[node.value] = {}
            self.tabela[node.value]["variavel"] = False
            self.tabela[node.value]["tipo"] = self.tipo(node.child[0])
            self.tabela[node.value]["num_parametros"] = 0
            self.tabela[node.value]["parametros"] =[]

            self.escopo = node.value  #escopo nome da função
            self.escopo = "global"

        if(node.type == "declaracao_de_funcao_sem_param_com_corpo"):

            self.tabela[node.value] = {}
            self.tabela[node.value]["variavel"] = False
            self.tabela[node.value]["tipo"] = self.tipo(node.child[0])
            self.tabela[node.value]["num_parametros"] = 0
            self.tabela[node.value]["parametros"] =[]
            self.escopo = node.value
            self.sequencia_de_declaracao(node.child[1])

            self.escopo = "global"

    def declara_var(self,node):

        if(self.escopo + "." + node.value in self.tabela.keys()): #se ja tem variavel com esse nom
            print("Erro Semântico, nome já utilizado : " + node.value )
            exit(1)
        if(node.value in self.tabela.keys()): #se ja tem variavel com esse nome na tabela
            print("Erro Semântico, nome já utilizado : " + node.value )
            exit(1)

        if(node.type == "declara_var_so_declara"):
            self.tabela[self.escopo + "." + node.value] = {}
            self.tabela[self.escopo + "." + node.value]["variavel"] = True
            self.tabela[self.escopo + "." + node.value]["inicializada"] = False
            self.tabela[self.escopo + "." + node.value]["tipo"] = self.tipo(node.child[0])
            self.tabela[self.escopo + "." + node.value]["valor"] = None

        if(node.type == "declara_var_loop"):
            self.tabela[self.escopo + "." + node.value] = {}
            self.tabela[self.escopo + "." + node.value]["variavel"] = True
            self.tabela[self.escopo + "." + node.value]["inicializada"] = False
            self.tabela[self.escopo + "." + node.value]["tipo"] = self.tipo(node.child[0])
            self.tabela[self.escopo + "." + node.value]["valor"] = None

            self.declara_outra_var(node.child[1], self.tipo(node.child[0]))

    def declara_outra_var(self,node,tipo):
        if(self.escopo + "." + node.value in self.tabela.keys()): #se ja tem variavel com esse nom
            print("Erro Semântico, nome já utilizado : " + node.value )
            exit(1)
        if(node.type == "declara_outra_var_1"):
            self.tabela[self.escopo + "." + node.value] = {}
            self.tabela[self.escopo + "." + node.value]["variavel"] = True
            self.tabela[self.escopo + "." + node.value]["inicializada"] = False
            self.tabela[self.escopo + "." + node.value]["tipo"] = tipo
            self.tabela[self.escopo + "." + node.value]["valor"] = None
            self.declara_outra_var(node.child[0],tipo)

        if(node.type == "declara_outra_var_2"):
```

```
            self.tabela[self.escopo + "." + node.value] = {}
            self.tabela[self.escopo + "." + node.value]["variavel"] = True
            self.tabela[self.escopo + "." + node.value]["inicializada"] = False
            self.tabela[self.escopo + "." + node.value]["tipo"] = tipo
            self.tabela[self.escopo + "." + node.value]["valor"] = None

    def declaracao_param(self,node):
        if( self.escopo + "." + node.value in self.tabela.keys()): #se ja tem variavel com esse no
            print("Erro Semântico, nome já utilizado :" + node.value )
            exit(1)

        if(node.type == "declaracao_param_loop"):
            # print(node.value)
            self.tabela[self.escopo + "." + node.value] = {}
            self.tabela[self.escopo + "." + node.value]["variavel"] = True
            self.tabela[self.escopo + "." + node.value]["inicializada"] = True
            self.tabela[self.escopo + "." + node.value]["tipo"] = self.tipo(node.child[0])
            self.tabela[self.escopo + "." + node.value]["valor"] = None

            return self.declaracao_param(node.child[1]) + 1

        else:
            self.tabela[self.escopo + "." + node.value] = {}
            self.tabela[self.escopo + "." + node.value]["variavel"] = True
            self.tabela[self.escopo + "." + node.value]["inicializada"] = True
            self.tabela[self.escopo + "." + node.value]["tipo"] = self.tipo(node.child[0])
            self.tabela[self.escopo + "." + node.value]["valor"] = None

            return 1

    def param(self,node,lista):
        # print(node.type+"oi7777")
        # print(self.tipo[node.child[0]])
        if(node.type == "declaracao_param_loop"):
            lista.append(self.tipo(node.child[0]))
            print(lista)
            return self.param(node.child[1],lista)
        else:
            lista.append(self.tipo(node.child[0]))
            return lista

    def sequencia_de_declaracao(self,node):
        if(node.type == "sequencia_de_declaracao_sem_loop") :
            self.declaracao(node.child[0])
        else :
            self. declaracao(node.child[0])
            self.sequencia_de_declaracao(node.child[1])

    def declaracao(self,node):
        if(node.type == "declaracao_expressao_condicional") :
            self.expressao_condicional(node.child[0])

        if(node.type == "declaracao_expressao_iteracao") :
            self.expressao_iteracao(node.child[0])

        if(node.type == "declaracao_expressao_atribuicao") :

            self.expressao_atribuicao(node.child[0])

        if(node.type == "declaracao_expressao_leitura") :
            self.expressao_leitura(node.child[0])
```

```
    if(node.type == "declaracao_expressao_escreva") :
        self.expressao_escreva(node.child[0])

    if(node.type == "declaracao_declara_var") :
        self.declara_var(node.child[0])

    if(node.type == "declaracao_retorna") :
        self.expressao_retorna(node.child[0])

    if(node.type == "declaracao_chamada_de_funcao") :
        self.chamada_de_funcao(node.child[0])


def expressao_condicional(self,node):
    if(node.type=="expressao_condicional_com_senao"):
        self.expressao(node.child[0],None)
        self.sequencia_de_declaracao(node.child[1])
        self.sequencia_de_declaracao(node.child[2])
    else:
        self.expressao(node.child[0])
        self.sequencia_de_declaracao(node.child[1])

def expressao_iteracao(self, node):
    self.sequencia_de_declaracao(node.child[0])
    self.expressao(node.child[1],None)

def expressao_atribuicao(self, node):
    if (self.escopo + "." + node.value not in self.tabela.keys() and "global." + node.value no
        print("Erro Semântico. Variável " + node.value + " não encontrada")
        exit(1)

    else :
        if self.escopo + "." + node.value in self.tabela.keys():
            # print(self.escopo + "." + node.value)
            tipo = self.expressao(node.child[0],node.value)
            # print(tipo)

            if self.tabela[self.escopo + '.' + node.value]["tipo"] != tipo:
                print("WARNING atribuição: variavel '" + node.value + "' é do tipo '" + self.t

            self.tabela[self.escopo + "." + node.value]["inicializada"] = True
            self.expressao(node.child[0], node.value) #passa o nome da variável

        elif "global." + node.value in self.tabela.keys():
            tipo = self.expressao(node.child[0],node.value)

            self.tabela["global." + node.value]["inicializada"] = True
            self.expressao(node.child[0], node.value)

def expressao_leitura(self,node):
    # print('global.' + node.value)
    # print( self.escopo + '.' + node.value)
    if self.escopo + "." + node.value not in self.tabela.keys() and "global." + node.value not
        print("Erro Semântico. Variável " + node.value + " não encontrada")
        exit(1)

    elif self.escopo + '.' + node.value in self.tabela.keys():
        # print(self.escopo + '.' + node.value+"n")
        self.tabela[self.escopo + '.' + node.value]["inicializada"] = True
    elif "global."+ node.value in self.tabela.keys():
```

```
            # print("global."+ node.value+"m")
            self.tabela['global.' + node.value]["inicializada"] = True

    def expressao_escreva(self,node):
        self.expressao(node.child[0], None)

    def expressao_retorna(self,node):
        novo = self.expressao(node.child[0], None)
        if(self.escopo+"."+novo in self.tabela.keys() ):
            tipo  = self.tabela[self.escopo+"."+novo]["tipo"]
            # print(tipo)
            if(tipo != self.tabela[self.escopo]["tipo"]):
                print("Erro Semantico. Retorno de funcao "+self.escopo+" o certo eh "+self.tabela[

    def chamada_de_funcao(self,node):

        if(node.value not in self.tabela.keys()) :
            print("Erro Semântico, nome de funcao não declarado : " + node.value )
            exit(1)

        if self.param_chama_funcao(node.child[0],0) != self.tabela[node.value]["num_parametros"]:
            print("Erro Semântico, número de parametros não correspondem com os da função : " + no
            exit(1)



# 1,5 + 2 + 1,5 + 2 + 1 + 2
    def param_chama_funcao(self,node,level):

        if node.type == "param_chama_funcao_loop":
            # print(node.child[0])
            if(self.tabela[self.escopo]["parametros"]!=[]):
                if(self.tabela[self.escopo]["parametros"][level+1]!=self.expressao(node.child[0],N
                    print("Erro Semantico: tipo de parametros incompativel")
                    exit(1)
            level = level+1

            # self.tabela[self.escopo+"."+node.value]["inicializada"]=True
            return self.param_chama_funcao(node.child[1],level) + 1
        else:
            return 1


    def expressao(self,node,nomeVariavel):

        if( node.type == "expressao_simples_composta" ):
            esquerda = self.expressao_simples(node.child[0],nomeVariavel)
            self.comparacao_operador(node.child[1])
            direita = self.expressao_simples(node.child[2],nomeVariavel)
            if esquerda == direita:
                return esquerda

            elif esquerda == "flutuante" or direita == "flutuante":
                return "flutuante"
            else:
                return "inteiro"
        else:
            # print(node.child[0])
            return self.expressao_simples(node.child[0],nomeVariavel)
```

```
def comparacao_operador(self, node):
        return node.value

def expressao_simples(self, node,nomeVariavel) :
    if(node.type == "expressao_simples_termo_com_soma"):
        esquerda =self.expressao_simples(node.child[0],nomeVariavel)
        self.soma(node.child[1])
        direita =self.termo(node.child[2],nomeVariavel)

        if esquerda == direita:
            return esquerda

        elif esquerda == "flutuante" or direita == "flutuante":
            return "flutuante"
        else:
            return "inteiro"
    else :
        return self.termo(node.child[0], nomeVariavel)

def soma(self,node):
    return node.value

def mult(self,node):
    return node.value

def termo(self,node,nomeVariavel):
    if(node.type == "fator"):
       return self.fator(node.child[0],nomeVariavel)
    else:
        esquerda = self.termo(node.child[0],nomeVariavel)
        direita = self.fator(node.child[2],nomeVariavel)

        if self.mult(node.child[1]) == "/":
            if self.escopo + "." + nomeVariavel in self.tabela.keys():
                 if self.tabela[self.escopo+"."+nomeVariavel]["tipo"]!= "flutuante":
                    print("Warning: ")

        if esquerda == direita:
            return esquerda

        elif esquerda == "flutuante" or direita == "flutuante":
            return "flutuante"
        else:
            return "inteiro"

def fator(self,node, nomeVariavel):
    if(node.type == "fator_expressao"):
        return self.expressao(node.child[0],nomeVariavel)
    if(node.type == "fator_chamada_de_funcao"):
        return self.chamada_de_funcao(node.child[0])
    if(node.type == "fato_expressao_numero"):
        return self.expressao_numero(node.child[0], nomeVariavel,False)

        if(nomeVariavel!=None):
            if self.escopo + "." + node.expressao_numero(node.child[0],nomeVariavel) not in se
               print("Erro semântico : Variável não declarada : " + node.value)
               exit(1)

    if(node.type == "fato_expressao_identificador"):
        return self.expressao_identificador(node.child[0],nomeVariavel)
```

```
    def expressao_identificador(self,node,nomeVariavel):
        self.tabela[self.escopo+"."+node.value]["inicializada"]=True
        return node.value

    def expressao_numero(self,node,nomeVariavel,boole):
        if(node.type=="expressao_numero_composta"):
            if(node.value=="+"):
                return self.expressao_numero(node.child[0],nomeVariavel,False)
            else:
                return self.expressao_numero(node.child[0],nomeVariavel,True)

        if(node.type=="expressao_numero"):
            return self.numero(node.child[0],boole)
    def tipo(self, node) :
        # print(node.value+"oi")

        return node.value

    def numero(self, node,boole):
        x = node.value.isdigit()
        if x is True:
            return "inteiro"
        else:
            return "flutuante"

if __name__ == "__main__":
    import sys
    code = open(sys.argv[1])
    s = Semantica(code.read())
    s.raiz()
    print("tabela de simbolos")
    for keys,values in s.tabela.items():
        if(values["variavel"]==True and values["inicializada"]==False):
            print("Warning: Variavel " +  keys + " nao esta sendo utilizada\n" )
        print(keys,values)
```