Processamento de Linguagens e Compiladores(3º ano de LCC)

Trabalho Prático 2

Relatório de Desenvolvimento

Bruno Jardim	Eduardo Freitas	Pedro Fernandes
(A91680)	(A91643)	(A91699)

15 de janeiro de 2022

Resumo

Este segundo trabalho prático, no âmbito da UC de processamento de linguagens e compiladores aborda a criação de uma linguagem imperativa e de um compilador usando módulos de gramáticas tradutoras do Python. Posto isto, com recurso às ferramentas *yacc* e *lex* do *python*, esta gramática deve ser capaz de gerar instruções assembly a partir de uma linguagem imperativa. Por fim, ao longo deste relatório tentaremos explicar de uma forma sucinta as decisões tomadas, como foi desenvolvido o compilador e as diversas produções implementadas na gramática.

Conteúdo

Capítulo 1

Introdução

Em geral, os compiladores são cruciais na medida que realizam a tradução de uma determinada linguagem de programação de alto nível para código máquina. Sendo assim, com o código máquina em *assebly* pretendemos depois que seja, processado e interpretado pela máquina virtual. Neste caso, a nossa linguagem implementada vai conseguir realizar as suas instruções. Assim sendo, foi implementado um compilador que traduz a nossa linguagem, numa sequência de instruções *assembly* de modo a poder ser interpretada pela VM.

Capítulo 2

Enunciado

Pretende-se que comece por definir uma linguagem de programação imperativa simples, a seu gosto. Apenas deve ter em consideração que essa linguagem terá de permitir:

• declarar variáveis atómicas do tipo inteiro, com os quais se podem realizar as habituais operações aritméticas, relacionais e lógicas.

• efetuar instruções algorítmicas básicas como a atribuição do valor de expressões numéricas a variáveis.

• ler do standard input e escrever no standard output.

• efetuar instruções condicionais para controlo do fluxo de execução.

• efetuar instruções cíclicas para controlo do fluxo de execução, permitindo o seu aninhamento. Note que deve implementar pelo menos o ciclo while-do, repeat-until ou for-do.

Adicionalmente deve ainda suportar, à sua escolha, uma das duas funcionalidades seguintes:

• declarar e manusear variáveis estruturadas do tipo array (a 1 ou 2 dimensões) de inteiros, em relação aos quais é apenas permitida a operação de indexação(índice inteiro).

• definir e invocar subprogramas sem parâmetros mas que possam retornar um resultado do tipo inteiro.

Como é da praxe neste tipo de linguagens, as variáveis deverão ser declaradas no início do programa e não pode haver re-declarações, nem utilizações sem declaração prévia. Se nada for explicitado, o valor da variável após a declaração é 0 (zero).

Desenvolva, então, um compilador para essa linguagem com base na GIC criada acima e com recurso aos módulos Yacc/ Lex do PLY/Python. O compilador deve gerar pseudo-código, Assembly da Máquina Virtual VM. Muito Importante: Para a entrega do TP deve preparar um conjunto de testes (programas-fonte escritos na sua linguagem) e mostrar o código Assembly gerado bem como o programa a correr na máquina virtual VM. Esse conjunto terá de conter, no mínimo, os 4 primeiros exemplos abaixo e um dos 2 últimos conforme a sua escolha acima:

• ler 4 números e dizer se podem ser os lados de um quadrado.

• ler um inteiro N, depois ler N números e escrever o menor deles.

• ler N (constante do programa) números e calcular e imprimir o seu produtório.

• contar e imprimir os números ímpares de uma sequência de números naturais.

• ler e armazenar N números num array; imprimir os valores por ordem inversa.

• invocar e usar num programa seu uma função 'potencia()', que começa por ler do input a base B e o expoente E e retorna o valor BE.

Capítulo 3

Linguagem Desenvolvida

A linguagem que desenvolvemos para este projeto foi nomeada de C+-, para uma utilização mais facilitada optamos por uma sintaxe semelhante à da linguagem de programação C.

3.1 Estrutura de Programas

Um programa desenvolvido com a nossa linguagem de programação deve seguir uma estrutura própria. Inicialmente o nosso programa destina-se à declaração das variáveis, enquanto que num segundo momento temos os restantes comandos implementados na linguagem. Não é permitido declarar novas variáveis depois de executar outros comandos, nem é possível redeclarar variáveis ou usar variáveis que não tenham sido declaradas.

3.2 Declarações

A declaração de variáveis é realizada de uma forma semelhante à usada em outras linguagens imperativas. Ou seja, inicialmente começamos por especificar o tipo de dados que queremos que seja armazenado pela variável, de seguida teremos o nome da variável em si, e depois o seu valor. As variáveis podem ser do tipo string, float ou inteiro(int). Se uma variável do tipo float, int ou se um array for declarado sem um valor inicial, este assumirá um valor nulo(0). É possível ainda declarar mais do que uma variável na mesma linha, assim como declarar um array de valores inteiros, com 1 ou 2 dimensões. Como por exemplo:

```
int a = 6;
float b = 7.4;
string c = "Hello World!";
int d, e = -8, f;
int g[3] = [1,2,3];
int e[9][8];
```

3.3 Input/Output

Para obter input do utilizador é possível usar a função input(t), que recebe como argumento o texto a imprimir antes de obter o input, e devolve uma string com o valor introduzido. Por outro lado, a função print(x) recebe o valor, regista-o, e imprime-o no terminal. Existe ainda a função println(x) que tem um

funcionamento semelhante ao da função print(), visto que imprime o valor recebido tal como a função print(), mas acrescenta um caracter de mudança de linha no final. Como por exemplo:

```
string idade = input("Introduza a sua idade: ");
print("A sua idade é ");
println(idade);
```

Output:

```
Introduza a sua idade: 20
A sua idade é 20
```

3.4 Instruções cíclicas

A linguagem que desenvolvemos possui dois tipos de instruções cíclicas. Inicialmente, temos o "while", esta keyword permite-nos executar um conjunto de instruções enquanto a condição for verdadeira. Por outro lado, existe também o "for"que executa instruções sempre que a condição a verdadeira, mas neste caso, tem associado a ele uma operação que é realizada a cada iteração do ciclo. Exemplo da implementação de um ciclo:

```
int s = 1;
int x;
for(x = 1; x < 3; x++) {
  s = s * x;
}
```

3.5 Instruções condicionais

A sintaxe das instruções condicionais é idêntica à sintaxe utilizada na maioria das outras linguagens de programação. Inicialmente temos a keyword "if", seguida de uma condição e por fim, entre chavetas, o conjunto de instruções que se pretende realizar caso a condição seja verdadeira. A condição não precisa de estar entre parênteses ao contrário do que acontece em linguagens como o C. Podemos também ter ainda, um conjunto de instruções a realizar caso a condição não seja verdadeira, neste caso utilizaremos o operador "else". Exemplos de utilização destes operadores:

```
int alt;
println("Descubra se pode andar na montanha-russa!");
alt = int(input("Introduza a sua altura(cm): "));
if (alt < 170) {
   println("Não pode entrar.");
}
else {
   println("Pode entrar na diversão!.");
}
```

3.6 Operações aritméticas, relacionais, lógicas e de conversão

É possível realizar as operações habituais de subtração, adição, multiplicação e divisão. Essas operações podem ser tanto com valores do tipo float como do tipo int. A ordem das operações e dos parênteses é respeitada. É ainda possível, realizar operações de módulo, mas apenas com valores inteiros. Se um dos operandos for um float e outro um int então, será feita uma conversão implícita do operando int para float. Podemos ainda fazer subtrações e somas do tipo x- - e x++.
Operações lógicas, como a > b, a <= b ou a == b, o resultado nestas operações será sempre um valor int. Nestas operações lógicas podemos ainda utilizar operadores, tais como: &&/and, e!/not e ||/or. As funções float(x) e int(x) permitem-nos fazer a conversão explícita de uma variável para o tipo float ou int, respetivamente. Se por outro lado, o argumento for uma string, é feita uma extração de um valor da mesma. Estas funções, utilizadas com a função input(), dão-nos a capacidade de obter input do utilizador em formato numérico(ex: int idade = int(input("Introduza a sua idade: "));).

Capítulo 4

Regras de Tradução

4.1 Lexer

O código para o Lexer pode ser consultado no Anexo C. Para além da definição dos literals e a definição dos tokens obrigatórios, foram igualmente definidas palavras reservadas. As palavras reservadas evitam conflitos na interpretação de texto. Deste modo, quando o Lexer lê um conjunto de caracteres irá procurar sempre por esse conjunto exato num dicionário de palavras reservadas. Assim sendo, se encontrar uma correspondência, atribuirá ao token lido o tipo correspondente à chave encontrada. Caso contrário, o token irá corresponder a uma variável. Sem esta implementação, se por exemplo existisse uma variável chamada "integrais", nesta situação o Lexer iria interpretar as três primeiras letras, como a keyword para definir um valor inteiro, o que não é o comportamento pretendido. Na regra das variáveis ainda temos (t_ID), com este mecanismo conseguimos detetar se a variável introduzida existe no programa ou não. Desta forma, é possível detetar a variável antes de entrar no Parser, o que mitigará o trabalho do mesmo.

```
def t_ID(self, t):
     r'[a-zA-Z][a-zA-Z0-9_\']*'
     t.type = Lexer.reservadas.get(t.value, 'ID')
     v = self.var.get(t.value, None)
     if v is not None:
         tipo = v[1]
         if tipo == INT:
             t.type = 'VARINT'
         elif tipo == FLOAT:
             t.type = 'VARFLOAT'
         elif tipo == ARRAY:
             t.type = 'VARARRAY'
         elif tipo == STRING:
             t.type = 'VARSTRING'
     return t
```

4.2 Parser

Neste capítulo, dividimos o parser em subcapítulos e iremos fazer uma breve explicação dos mesmos. O código correspondente ao Parser pode ser consultado no Anexo D.

4.2.1 Estrutura Básica

A nossa linguagem deve conter as declarações de variáveis antes do resto das instruções.

4.2.2 Declaração de Variáveis

O conjunto das declarações corresponde a "Atribuicao", que é dividido em por exemplo, atribuições singulares. Cada atribuição contém o identificador do tipo da variável, o seu nome e opcionalmente um valor inicial. Caso optemos por definir um array, só temos que nos preocupar em armazenar o tamanho do mesmo. Em alternativa, para definir um array com valores iniciais, temos primeiro de juntar todos os valores numa lista e depois percorrer a mesma. Existe também a regra "Vazio", esta regra é extremamente útil para ser possível ter valores opcionais nas nossas regras sem ser preciso de definir duas funções diferentes.

4.2.3 Mudança de Valor de Variáveis

É possível fazer uma alteração do valor das variáveis declaradas usando o "Atualiza".
A lógica por detrás das regras que servem para obter os endereços do array é a mesma usada na subsecção das Operações para obter o elemento na posição pretendida.
Conseguimos também utilizar x++ e x- -, que facilita a forma de escrever.

4.2.4 Input/Output

O nosso trabalho tem comandos que permitem imprimir valores no terminal. Como por exemplo, o comando println ou print, ambos imprimem mas no caso do primeiro adiciona também um caracter de newline. Definimos regras diferentes para cada um dos tipos pois os comandos da VM para cada tipo são diferentes. Por fim, é possível ainda ler um valor do terminal, estes valores lidos são reduzidos para uma String, que posteriormente pode ser reduzida para um Valor.

4.2.5 Ciclos

De uma forma muito semelhante, de como nas instruções condicionais, temos contadores que incrementam sempre que o tradutor faz o parsing de um ciclo. A maior diferença neste caso é que, no fim de executar as instruções voltamos à condição, em vez de sair da mesma. Ou seja, saímos do ciclo quando a condição deixar de ser verdadeira. Entre o ciclo "for"e "while", a única diferença no parsing é que para o ciclo "for"temos de incluir a operação de incrementação da variável no conjunto de comandos a executar, para posteriormente iniciar a execução, puxando a variável assinalada para o topo da stack.

4.2.6 Instruções Condicionais

Para compilar instruções condicionais tivemos que utilizar jumps, pois é necessário saltar entre partes do programa. Para fazer a identificação das labels dos salto, temos um contador de labels, que como indica o nome faz uma contagem de cada label que foi usada, e como o valor faz parte do nome da label então nunca teremos labels repetidas. Os contadores são inicializados a 0 ao correr o Parser.

4.2.7 Operações

Primeiro definimos o não-terminal "Valor". O Valor, pode ser formado a partir de uma variável de um valor literal, de uma String ou de uma expressão em float convertida para int. Se pretendermos ir buscar o valor

de um elemento de um array, o processo é mais demorado mas é possível ter acesso à posição do elemento que pretendemos na stack da VM. Para usar estes valores em operações aritméticas e lógicas, mas primeiro iremos reduzi-los para o não-terminal ExpressaoValor. A partir deste não-terminal temos as implementações das operações básicas.
No nosso trabalho prático, optamos por definir manualmente as precendências. As operações de multiplicação/módulo/divisão são reduzidas antes das operações de adição/subtração. Os valores "right"e "left"indicam se se reduz primeiro à direita ou à esquerda do sinal. Nas operações aritméticas primeiro faz-se a operação da esquerda e de seguida a da direita, se a prioridade dos dois lados for a mesma.
Devido a uma limitação da VM, todos os valores de uma operação devem ser do mesmo tipo, sendo assim tivemos que implementar regras que verificassem o tipo da variável e fizessem a transformação para o mesmo tipo para conseguir realizar operações com os valores.
Existem ainda valores lógicos, que se tratam de valores int.
Por fim, definimos o tipo String.

4.3 Main

De forma a correr o ficheiro main.py é necessário executar o seguinte comando:

```
> python main.py [nome ficheiro input] [nome ficheiro output]
```

O nome do ficheiro de output será igual ao do ficheiro de input, com a extensão '.vm', caso não tenha nome.

Capítulo 5

Conclusão

A realização deste projeto permitiu-nos aprofundar e consolidar os conhecimentos leccionados nas aulas de Processamento de Linguagens e Compiladores. Desta forma, todas as decisões e reflexões sobre o problema que nos foi apresentado consideramos que foram cruciais para o desenvolvimento da nossa compreensão sobre o funcionamento de compiladores, algo que nos poderá vir a ser muito útil no nosso futuro profissional e académico. Desta mesma forma, foi importante também rever a linguagem Assebly e a utilização da VM ajudou-nos a relembrar o funcionamento das linguagens máquina. Para concluir, todos os elementos do grupo concordam que apesar do trabalho desenvolvido não ser extraordinário que satisfaz o que nos foi pedido no enunciado, desta forma estamos satisfeitos com o resultado final.

Capítulo 6

A:Exemplos

6.1 Ler 4 números e dizer se podem ser os lados de um quadrado

6.1.1 1-Código C+-

```
int a,b,c,d;
a = int(input());
b = int(input());
c = int(input());
d = int(input());
if a == b && b == c && c == d{
    print("YESSIR");
}
else{
    print("NAO");
}
```

6.1.2 2-Código Máquina

```
pushi 0
pushi 0
pushi 0
pushi 0
start
read
atoi
storeg 0
read
atoi
storeg 1
read
atoi
storeg 2
read
atoi
```

```
storeg 3
pushg 0
pushg 1
equal
pushg 1
pushg 2
equal
pushg 2
pushg 3
equal
mul
mul
jz l0
pushs "YESSIR"
writes
jump le0
l0:
pushs "NAO"
writes
le0:

stop
```

6.2 Ler um inteiro, depois ler N número e escrever o menor

6.2.1 1-Código C+-

```
int tamanho;
int menor;
int valor;
int i;
tamanho = int(input());
for(i = 0; i<tamanho;i++){
    valor = int(input());
    if i == 0{
        menor = valor;
    }

    if valor < menor{
        menor = valor;
    }
}
print(menor);
```

6.2.2 2-Código Máquina

```
pushi 0
pushi 0
```

```
pushi 0
pushi 0
start
read
atoi
storeg 0
pushi 0
storeg 3
lc0:
pushg 3
pushg 0
inf
jz lb0
read
atoi
storeg 2
pushg 3
pushi 0
equal
jz l0
pushg 2
storeg 1
l0:
pushg 2
pushg 1
inf
jz l1
pushg 2
storeg 1
l1:
pushg 3
pushi 1
add
storeg 3
jump lc0
lb0:
pushg 1
writei

stop
```

6.3 Ler N números e calcular e imprimir o seu produtório

6.3.1 1-Código C+-

```
int tamanho;
int produtorio = 1;
int valor;
```

```
int i;
tamanho = int(input());

for(i = 0; i<tamanho;i++){
    valor = int(input());
    produtorio = produtorio * valor;
}
print(produtorio);
```

6.3.2 2-Código Máquina

```
pushi 0
pushi 1
pushi 0
pushi 0
start
read
atoi
storeg 0
pushi 0
storeg 3
lc0:
pushg 3
pushg 0
inf
jz lb0
read
atoi
storeg 2
pushg 1
pushg 2
mul
storeg 1
pushg 3
pushi 1
add
storeg 3
jump lc0
lb0:
pushg 1
writei

stop
```

6.4 Contar e imprimir os números ímpares de uma sequência de números naturais

6.4.1 1-Código C+-

```
int n;
int valor;

valor = int(input());
if valor % 2 == 1{
    n++;
}
while(valor != 0){
    valor = int(input());
    if valor % 2 == 1{
        n++;
    }
}
print(n);
```

6.4.2 2-Código Máquina

```
pushi 0
pushi 0
start
read
atoi
storeg 1
pushg 1
pushi 2
mod
pushi 1
equal
jz l0
pushg 0
pushi 1
add
storeg 0
l0:
lc0:
pushg 1
pushi 0
equal
not
jz lb0
read
atoi
storeg 1
```

```
pushg 1
pushi 2
mod
pushi 1
equal
jz l1
pushg 0
pushi 1
add
storeg 0
l1:
jump lc0
lb0:
pushg 0
writei

stop
```

6.5 Ler e armazenar N números num array; imprimir os valores por ordem inversa

6.5.1 1-Código C+-

```
int i;
int array[7];
int N = 7;
int valor;

for(i = 0; i < N; i++){
    valor = int(input());
    array[i] = valor;

}
for(i = N-1; i>=0;i--){
    println(array[i]);
}
```

6.5.2 2-Código Máquina

```
pushi 0
pushn 7
pushi 7
pushi 0
start
pushi 0
storeg 0
lc0:
```

```
pushg 0
pushg 8
inf
jz lb0
read
atoi
storeg 9
pushgp
pushi 1
padd
pushg 0
pushg 9

storen
pushg 0
pushi 1
add
storeg 0
jump lc0
lb0:
pushg 8
pushi 1
sub
storeg 0
lc1:
pushg 0
pushi 0
supeq
jz lb1
pushgp
pushi 1
padd
pushg 0
loadn
writei
pushs "\n"
writes
pushg 0
pushi 1
 sub
storeg 0
jump lc1
lb1:

stop
```

Capítulo 7

B:Main

```
from parserPLC import Parser

import sys

parser = Parser()

parser.build()

if len(sys.argv) < 2:
    s = ""
    while linha := input():
        s += linha + "\n"

    programa = parser.parser.parse(s)
    print(programa)
else:
    with open(sys.argv[1],"r") as f:
        programa = parser.parser.parse(f.read())
    if programa:
        with open(sys.argv[1].strip('.\\').split('.')[0]+'.vm','w') as maquina:
            maquina.write(programa)
```

Capítulo 8

C:Lexer

```
import ply.lex as lex

INT = 1
FLOAT = 2
ARRAY = 3
STRING = 4



class Lexer:
    def __init__(self, var: dict):
        self.var = var

    reservadas = {
        'int': 'INTR',
        'float': 'FLOATR',
        'str': 'STRR',
        'or': 'OR',
        'and': 'AND',
        'not': 'NOT',
        'if': 'IF',
        'else': 'ELSE',
        'for': 'FOR',
        'while': 'WHILE',
        'print': 'PRINT',
        'println': 'PRINTLN',
        'input': 'INPUT'
    }


    tokens = [
        'INT',
        'FLOAT',
        'ID',
```

```
    'VARINT',
    'VARFLOAT',
    'VARSTRING',
    'VARARRAY',
    'GEQUAL',
    'LEQUAL',
    'EQUAL',
    'DIFF',
    'PP',
    'MM',
    'LINHA'
] + list(reservadas.values())


literals = [
    '+',
    '-',
    '*',
    '/',
    '%',
    '(',
    ')',
    '[',
    ']',
    '{',
    '}',
    '=',
    '>',
    '<',
    ',',
    ';'
]


def t_FLOAT(self, t):
    r'(\d*)?\.\d+(e(?:\+|-)\d+)?'
    t.value = float(t.value)
    return t

def t_INT(self, t):
    r'\d+'
    t.value = int(t.value)
    return t

t_GEQUAL = r'>='
t_LEQUAL = r'<='
t_EQUAL = r'=='
t_DIFF = r'!='
```

```
t_PP = r'\+\+'
t_MM = r'--'
t_AND = r'&&'
t_OR = r'\|\|'

def t_ID(self, t):
    r'[a-zA-Z][a-zA-Z0-9_\']*'
    t.type = Lexer.reservadas.get(t.value, 'ID')
    v = self.var.get(t.value, None)
    if v is not None:
        tipo = v[1]
        if tipo == INT:
            t.type = 'VARINT'
        elif tipo == FLOAT:
            t.type = 'VARFLOAT'
        elif tipo == ARRAY:
            t.type = 'VARARRAY'
        elif tipo == STRING:
            t.type = 'VARSTRING'
    return t
t_LINHA = rf"'(\\'|[^'])*'|\"(\\\"|[^\"])*\""
t_ignore = ' \t'

def t_error(self, t):
    print(rf"Illegal char {t.value[0]}")
    t.lexer.skip(1)

def t_newline(self, t):
    r'\n'
    t.lexer.lineno += 1

def build(self, **kwargs):
    self.lexer = lex.lex(module=self, **kwargs)
```

Capítulo 9

D:Parser

```
import ply.yacc as yacc
from lexerPLC import Lexer

INT = 1
FLOAT = 2
ARRAY = 3
STRING = 4


class Parser:
    tokens = Lexer.tokens

    # Definição de um programa
    # Todas as Atribuições sempre no inicio do programa
    # Seguidas das Instruções
    def p_Programa(self, p):
        "Programa : Atribuicoes Instrucoes"
        p[0] = p[1] + 'start' + '\n' + p[2] + '\nstop' + '\n'

    # Definição de um progama sem Atribuições
    # Apenas Instruções
    def p_Programa_NOATRIB(self, p):
        "Programa : Instrucoes"
        p[0] = 'start' + '\n' + p[1] + '\nstop' + '\n'

    # Definição de um programa sem Instruções
    # Apenas Atribuições
    def p_Programa_NOINST(self, p):
        "Programa : Atribuicoes"
        print('Não foram encontradas Instruções!')
        raise SyntaxError

    # Definição de um programa com erros
    def p_Programa_ERROR(self, p):
```

```
    "Programa : error"
    print('E aprenderes a programar?!')
    raise SystemExit

# Definição de um conjunto de Atribuições genéricas
def p_Atribuicoes(self, p):
    "Atribuicoes : Atribuicoes Atribuicao"
    p[0] = p[1] + p[2]

# Definição de uma Atribuição singular
def p_Atribuicoes_Singular(self, p):
    "Atribuicoes : Atribuicao"
    p[0] = p[1]



# Definição de um conjunto de Instrucoes genéricos
def p_Instrucoes(self, p):
    "Instrucoes : Instrucoes Instrucao"
    p[0] = p[1] + p[2]

# Definição de uma Instrução singular
def p_Intrucoes_Singular(self, p):
    "Instrucoes : Instrucao"
    p[0] = p[1]



#
#   INTEIROS
#

#Atribuicao nao incializada de Inteiros
def p_Atribuicao_Int_NoInit(self,p):
    "Atribuicao : INTR AtribuicoesINT ';'"
    p[0] = p[2]

#Atribuicao de Inteiros Singular
def p_Atribuicao_Int_Singular(self,p):
    "AtribuicoesINT : AtribuicaoINT"
    p[0] = p[1]

# Atribuicao Multipla de Inteiros
def p_Atribuicao_Int_Multipla(self, p):
    "AtribuicoesINT : AtribuicoesINT ',' AtribuicaoINT"
    p[0] = p[1] + p[3]

#Atribuicao inicializada de Inteiros
```

```
def p_Atribuicao_Int_Init(self,p):
    "AtribuicaoINT : ID '=' Expressao"
    if p[1] in self.var:
        #Já existe uma variável com o ID p[1]
        print(rf'A variável {p[1]} declarada na linha {p.lineno} já existe!')
        raise SyntaxError
    else:
        #Adiconar variável à Stack
        self.var[p[1]] = (self.tamanho_stack,INT)
        self.tamanho_stack += 1

    #Atribuição incializada

    p[0] = p[3]


def p_Atribuicao_Int_NOINIT(self,p):
    "AtribuicaoINT : ID"
    if p[1] in self.var:
        #Já existe uma variável com o ID p[1]
        print(rf'A variável {p[1]} declarada na linha {p.lineno} já existe!')
        raise SyntaxError
    else:
        #Adiconar variável à Stack
        self.var[p[1]] = (self.tamanho_stack,INT)
        self.tamanho_stack += 1
    p[0] = 'pushi 0\n'

#
#   FLOATS
#

# Atribuicao nao incializada de Floats
def p_Atribuicao_Float_NoInit(self, p):
    "Atribuicao : FLOATR AtribuicoesFLOAT ';'"
    p[0] = p[2]

# Atribuicao de Floats Singular
def p_Atribuicao_Float_Singular(self, p):
    "AtribuicoesFLOAT : AtribuicaoFLOAT"
    p[0] = p[1]

# Atribuicao Multipla de Floats
def p_Atribuicao_Float_Multipla(self, p):
    "AtribuicoesFLOAT : AtribuicoesFLOAT ',' AtribuicaoFLOAT"
    p[0] = p[1] + p[3]

# Atribuicao inicializada de Floats
```

```
def p_Atribuicao_Float_Init(self, p):
    "AtribuicaoFLOAT : ID '=' Expressao"
    if p[1] in self.var:
        # Já existe uma variável com o ID p[1]
        print(rf'A variável {p[1]} declarada na linha {p.lineno} já existe!')
        raise SyntaxError
    else:
        # Adiconar variável à Stack
        self.var[p[1]] = (self.tamanho_stack, INT)
        self.tamanho_stack += 1

    # Atribuição incializada

    p[0] = p[3]

def p_Atribuicao_Float_NOINIT(self,p):
    "AtribuicaoFLOAT : ID"
    if p[1] in self.var:
        #Já existe uma variável com o ID p[1]
        print(rf'A variável {p[1]} declarada na linha {p.lineno} já existe!')
        raise SyntaxError
    else:
        #Adiconar variável à Stack
        self.var[p[1]] = (self.tamanho_stack,INT)
        self.tamanho_stack += 1
    'pushi 0.0\n'

#
#   STRINGS
#

# Atribuicao nao incializada de Strings
def p_Atribuicao_String_NoInit(self, p):
    "Atribuicao : STRR AtribuicoesSTRING ';'"
    p[0] = p[2]

# Atribuicao de Strings Singular
def p_Atribuicaos_String_Singular(self, p):
    "AtribuicoesSTRING : AtribuicaoSTRING"
    p[0] = p[1]

# Atribuicao Multipla de Strings
def p_Atribuicao_String_Multipla(self, p):
    "AtribuicoesSTRING : AtribuicoesSTRING ',' AtribuicaoSTRING"
    p[0] = p[1] + p[3]

# Atribuicao inicializada de Strings
def p_Atribuicao_String(self, p):
```

```
    "AtribuicaoSTRING : ID '=' String ';'"
    if p[1] in self.var:
        #Já existe uma variável com o ID p[1]
        print(rf'A variável {p[1]} declarada na linha {p.lineno} já existe!')
        raise SyntaxError
    else:
        #Adicionar variável à Stack
        self.var[p[1]] = (self.tamanho_stack,STRING)
        self.tamanho_stack += 1

    p[0] = p[3]

#
#   ARRAYS
#

#Atribuicao nao incializada de um Array
def p_Atribuicao_Array(self,p):
    "Atribuicao : INTR ID '[' INT ']' ';'"
    if p[2] in self.var:
        #Já existe uma variável com o ID p[2]
        print(rf'A variável {p[2]} declarada na linha {p.lineno} já existe!')
        raise SyntaxError
    else:
        self.var[p[2]] = (self.tamanho_stack,ARRAY,p[4])
        self.tamanho_stack += p[4]
    p[0] = f"pushn {p[4]}\n"

#Atribuicao de Arrays Singular
def p_Atribuicao_Array_Singular(self,p):
    "Arrays : Array"
    p[0] = p[1]

#Atribuicao Valorada de Arrays
def p_Atribuicao_Array_Valorada(self,p):
    "Array : '[' Elementos ']'"
    p[0] = p[2]

#Definicao de Elementos
def p_Elementos(self,p):
    "Elementos : Elementos ',' INT"
    p[0] = p[1]
    p[0].append(p[3])

#Definicao Singular de Elementos
def p_Elementos_Singular(self,p):
    "Elementos : INT"
    p[0] = p[1]
```

```
#Definição de Matriz
def p_Atribuicao_Matriz(self,p):
    "Atribuicao : INTR ID '[' INT ']' '[' INT ']' ';'"
    if p[2] not in self.var:
        self.var[p[2]] = (self.tamanho_stack,ARRAY,(p[4],p[7]))
        self.tamanho_stack += p[4] * p[7]
    else:
        print(f"A variável {p[2]} foi já foi declarada na linha {p.lineno(2)}")
        raise SyntaxError
    p[0] = f"pushn {p[4]*p[7]}\n"

#Definição de Matriz por Arrays
def p_Matriz(self,p):
    "Matriz : '[' Arrays ']'"
    p[0] = p[2]

#Definição de Arrays
def p_Arrays(self,p):
    "Arrays : Arrays ',' Array"
    if len(p[3]) != len(p[1][0]):
        print(f"Os arrays têm de ter dimensões iguais! Erro na linha {p.lineno(2)}")
        raise SyntaxError
    p[0] = p[1]
    p[0].append(p[3])

#Definição de Atribuição de Array Valorado
def p_Atribuicao_Array_Valorado_TamanhoDET(self,p):
    "Atribuicao : INTR ID '[' INT ']' '=' Array ';'"
    if p[2] not in self.var:
        self.var[p[2]] = (self.tamanho_stack,ARRAY,p[4])
        self.tamanho_stack += p[4]
    else:
        print(f"A variável {p[2]} foi já foi declarada na linha {p.lineno(2)}")
        raise SyntaxError
    if len(p[7]) != len(p[4]):
        print(f"Os arrays têm de ter dimensões iguais! Erro na linha {p.lineno(2)}")
        raise SyntaxError
    p[0] = ""
    for inteiro in p[7]:
        p[0] += f"pushi {inteiro}\n"
#Definição de Atribuição de um Array Valorado sem tamanho definido
def p_Atribuicao_Array_Valorado_TamanhoNDET(self, p):
    "Atribuicao : INTR ID '[' Vazio ']' '=' Array ';'"

    p[4] = len(p[7])
    if p[2] not in self.var:
        self.var[p[2]] = (self.tamanho_stack, ARRAY, p[4])
```

```
        self.tamanho_stack += p[4]
    else:
        print(f"A variável {p[2]} foi já foi declarada na linha {p.lineno(2)}")
        raise SyntaxError
    p[0] = ""
    for inteiro in p[7]:
        p[0] += f"pushi {inteiro}\n"

#Definicão de Atribuição de uma Matriz Valorada
def p_Atribuicao_Matriz_Valorada(self,p):
    "Atribuicao : INTR ID '[' Vazio ']' '=' Matriz ';'"
    p[4] = len(p[10])
    p[7] = len(p[10][0])

    if p[2] not in self.var:
        self.var[p[2]] = (self.tamanho_stack, ARRAY,(p[4],p[7]))
        self.tamanho_stack += p[4]*p[7]
    else:
        print(f"A variável {p[2]} foi já foi declarada na linha {p.lineno(2)}")
        raise SyntaxError
    p[0] = ""
    for linha in p[10]:
        for inteiro in linha:
            p[0] += f"pushi{inteiro}\n"

#
#   ERRO NAS ATRIBUIÇÕES
#

def p_Atribuicao_ErroINT(self,p):
    "Atribuicao : INTR error ';'"
    p[0] = ""
def p_Atribuicao_ErroFLOAT(self,p):
    "Atribuicao : FLOATR error ';'"
    p[0] = ""
def p_Atribuicao_ErroSTRING(self,p):
    "Atribuicao : STRR error ';'"
    p[0] = ""


#
#   INSTRUCOES
#

#Definição da Instrução Atualiza
def p_Instrucao_Atualiza(self,p):
    "Instrucao : Atualiza ';'"
    p[0] = p[1]
```

```
#Definicao de Atualizar um Inteiro
def p_Atualiza(self,p):
    "Atualiza : VARINT '=' Expressao"
    p[0] = f"{p[3]}storeg {self.var[p[1]][0]}\n"

# Definicao de Atualizar um Float
def p_Atualiza_FLOAT(self, p):
    "Atualiza : VARFLOAT '=' ExpressaoFloat"
    p[0] = f"{p[3]}storeg {self.var[p[1][0]]}\n"

# Definicao de Atualizar uma String
def p_Atualiza_STRING(self, p):
    "Atualiza : VARSTRING '=' String"
    p[0] = f"{p[3]}storeg {self.var[p[1][0]]}\n"

#Definição Atualizacao ++
def p_Atualiza_PP(self,p):
    "Atualiza : VARINT PP"
    val = self.var[p[1]][0]
    p[0] = f"pushg {val}\npushi 1\nadd\nstoreg {val}\n"

# Definição Atualizacao --
def p_Atualiza_MM(self, p):
    "Atualiza : VARINT MM"
    val = self.var[p[1]][0]
    p[0] = f"pushg {val}\npushi 1\n sub\nstoreg {val}\n"

# Definição Atualizacao ++
def p_Atualiza_PP_FLOAT(self, p):
    "Atualiza : VARFLOAT PP"
    val = self.var[p[1]][0]
    p[0] = f"pushg {val}\npushi 1.0\n fadd\nstoreg {val}\n"

# Definição Atualizacao --
def p_Atualiza_MM_FLOAT(self, p):
    "Atualiza : VARFLOAT MM"
    val = self.var[p[1]][0]
    p[0] = f"pushg {val}\npushi 1.0\n fsub\nstoreg {val}\n"

#Definição de Atualizacao de um elemento de um Array
def p_Atualiza_Elem_Array(self,p):
    "Atualiza : VARARRAY '[' Expressao ']' '=' Expressao"
    p[0] = f"pushgp\npushi {self.var[p[1]][0]}\npadd\n{p[3]}{p[6]}\nstoren\n"

# Definição de Atualizacao de um elemento de uma Matriz
def p_Atualiza_Elem_Matriz(self,p):
    "Atualiza : VARARRAY '[' Expressao ']' '[' Expressao ']' '=' Expressao"
```

```
      p[0] = f"pushgp\npushi{self.var[p[1]][0]}\npadd\{p[3]}pushi{self.var[p[1]][2][1]}\
      nmul\n{p[6]}add\n{p[9]}ftoi\nstoreg\n"

#Conversao de Variaveis
def p_INT2FLOAT(self,p):
    "Atualiza : VARFLOAT '=' Expressao"
    p[0] = f"{p[3]}ftoi\nstoreg {self.var[p[1]][0]}\n"

def p_FLOAT2INT(self, p):
    "Atualiza : VARINT '=' ExpressaoFloat"
    p[0] = f"{p[3]}ftoi\nstoreg {self.var[p[1]][0]}\n"

def p_Instrucao_Print(self,p):
    "Instrucao : PRINT '(' Expressao ')' ';'"
    p[0] = p[3] + "writei\n"

def p_Instrucao_Print_ExpLogica(self,p):
    "Instrucao : PRINT '(' ExpLogica ')' ';'"
    p[0] = p[3] + "writei\n"
def p_Instrucao_Print_FLOAT(self,p):
    "Instrucao : PRINT '(' ExpressaoFloat ')' ';'"
    p[0] = p[3] + "writef\n"

def p_Instrucao_Print_String(self,p):
    "Instrucao : PRINT '(' String ')' ';'"
    p[0] = p[3] + "writes\n"



def p_Instrucao_PrintLN(self, p):
    "Instrucao : PRINTLN '(' Expressao ')' ';'"
    p[0] = p[3] + "writei\n" + "pushs \"\\n\"\nwrites\n"

def p_Instrucao_PrintLN_ExpLogica(self, p):
    "Instrucao : PRINTLN '(' ExpLogica ')' ';'"
    p[0] = p[3] + "writei\n" + "pushs \"\\n\"\nwrites\n"

def p_Instrucao_PrintLN_FLOAT(self, p):
    "Instrucao : PRINTLN '(' ExpressaoFloat ')' ';'"
    p[0] = p[3] + "writef\n" + "pushs \"\\n\"\nwrites\n"

def p_Instrucao_PrintLN_String(self, p):
    "Instrucao : PRINTLN '(' String ')' ';'"
    p[0] = p[3] + "writes\n" + "pushs \"\\n\"\nwrites\n"

def p_Instrucao_if(self,p):
    "Instrucao : IF Boolean '{' Instrucoes '}'"
    p[0] = p[2] + f"jz l{self.ifs}\n" + p[4] + f"l{self.ifs}:\n"
```

```
    self.ifs += 1

def p_Instrucao_if_else(self,p):
    "Instrucao : IF Boolean '{' Instrucoes '}' Else"
    p[0] = p[2] + f"jz l{self.ifs}\n" + p[4] + f"jump le{self.if_else}\n" + f"l{self.ifs}:\n
    self.ifs +=1
    self.if_else += 1

def p_Instrucao_Else(self,p):
    "Else : ELSE '{' Instrucoes '}'"
    p[0] = p[3] + f"le{self.if_else}:\n"

def p_Instrucao_else_if(self,p):
    "Else : ELSE IF Boolean '{' Instrucoes '}' Else"
    p[0] = p[3] + f"jz l{self.ifs}\n" + p[5] + f"l{self.ifs}\n" + f"le{self.if_else}:\n"
    self.ifs += 1

def p_Instrucao_For(self,p):
    "Instrucao : FOR '(' Atualiza ';' Boolean ';' Atualiza ')' '{' Instrucoes '}'"
    p[0] = p[3] + f"lc{self.ciclo}:\n" + p[5] + f"jz lb{self.ciclo_fim}\n" + p[10] + p[7] + 
    self.ciclo += 1
    self.ciclo_fim += 1

def p_Instrucao_For_NOINIT(self,p):
    "Instrucao : FOR '(' Vazio ';' Boolean ';' Atualiza ')' '{' Instrucoes '}'"
    p[0] = "" + f"lc{self.ciclo}:\n" + p[5] + f"jz lb{self.ciclo_fim}\n" + p[10] + p[7] + f"

def  p_Instrucao_While(self,p):
    "Instrucao : WHILE Boolean '{' Instrucoes '}'"
    p[0] = f"lc{self.ciclo}:\n" + p[2] + f"jz lb{self.ciclo_fim}\n" + p[4] + f"jump lc{self.
    self.ciclo += 1
    self.ciclo_fim += 1


def p_Boolean(self,p):
    "Boolean : Expressao"
    p[0] = p[1]

def p_Boolean_ExpLogica(self,p):
    "Boolean : ExpLogica"
    p[0] = p[1]

def p_String(self,p):
    "String : LINHA"
    p[0] = f"pushs \"" + p[1].strip('"') + "\"\n"

def p_String_VARSTRING(self,p):
    "String : VARSTRING"
```

```
    p[0] = f"pushg {self.var[p[1]][0]}\n"

def p_String_Input(self,p):
    "String : INPUT '(' ')'"
    p[0] = f"read\n"

def p_Vazio(self,p):
    "Vazio : "
    pass
def p_error(self,p):
    print(f"Isto tá mal men, olha a linha -> {p.lineno}")

##########################################################################################
# Inteiros
# Definição de Soma
def p_Soma(self, p):
    "Expressao : Expressao '+' Expressao"
    p[0] = p[1] + p[3] + "add\n"

# Definição de Subtração
def p_Subtracao(self, p):
    "Expressao : Expressao '-' Expressao"
    p[0] = p[1] + p[3] + "sub\n"

# Definição de Multiplicaçaõ
def p_Multiplicacao(self, p):
    "Expressao : Expressao '*' Expressao"
    p[0] = p[1] + p[3] + "mul\n"

# Defenição de Divisão
def p_Divisao(self, p):
    "Expressao : Expressao '/' Expressao"
    p[0] = p[1] + p[3] + "div\n"

# Definição de Módulo
def p_Modulo(self, p):
    "Expressao : Expressao '%' Expressao"
    p[0] = p[1] + p[3] + "mod\n"

# Dois Float
# Definição de Soma
def p_Soma_Float(self, p):
    "ExpressaoFloat : ExpressaoFloat '+' ExpressaoFloat"
    p[0] = p[1] + p[3] + "fadd\n"

# Definição de Subtração
def p_Subtracao_Float(self, p):
    "ExpressaoFloat : ExpressaoFloat '-' ExpressaoFloat"
```

```
    p[0] = p[1] + p[3] + "fsub\n"

# Definição de Multiplicaçaõ
def p_Multiplicacao_FLoat(self, p):
    "ExpressaoFloat : ExpressaoFloat '*' ExpressaoFloat"
    p[0] = p[1] + p[3] + "fmul\n"

# Defenição de Divisão
def p_Divisao_Float(self, p):
    "ExpressaoFloat : ExpressaoFloat '/' ExpressaoFloat"
    p[0] = p[1] + p[3] + "fdiv\n"

# Um Int e um Float
# Definição de Soma
def p_Soma_Int_Float(self, p):
    "ExpressaoFloat : Expressao '+' ExpressaoFloat"
    p[0] = p[1] + 'itof\n' + p[3] + "fadd\n"

# Definição de Subtração
def p_Subtracao_Int_Float(self, p):
    "ExpressaoFloat : Expressao '-' ExpressaoFloat"
    p[0] = p[1] + 'itof\n' + p[3] + "fsub\n"

# Definição de Multiplicaçaõ
def p_Multiplicacao_Int_FLoat(self, p):
    "ExpressaoFloat : Expressao '*' ExpressaoFloat"
    p[0] = p[1] + 'itof\n' + p[3] + "fmul\n"

# Defenição de Divisão
def p_Divisao_Int_Float(self, p):
    "ExpressaoFloat : Expressao '/' ExpressaoFloat"
    p[0] = p[1] + 'itof\n' + p[3] + "fdiv\n"

# Um Float e um Int
# Definição de Soma
def p_Soma_Float_Int(self, p):
    "ExpressaoFloat : ExpressaoFloat '+' Expressao"
    p[0] = p[1] + p[3] + 'itof\n' + "fadd\n"

# Definição de Subtração
def p_Subtracao_Float_Int(self, p):
    "ExpressaoFloat : ExpressaoFloat '-' Expressao"
    p[0] = p[1] + p[3] + 'itof\n' + "fsub\n"

# Definição de Multiplicaçaõ
def p_Multiplicacao_FLoat_Int(self, p):
    "ExpressaoFloat : ExpressaoFloat '*' Expressao"
    p[0] = p[1] + p[3] + 'itof\n' + "fmul\n"
```

```
# Defenição de Divisão
def p_Divisao_Float_Int(self, p):
    "ExpressaoFloat : ExpressaoFloat '/' Expressao"
    p[0] = p[1] + p[3] + 'itof\n' + "fdiv\n"

##########################################################################################
# Definição de visualização de Variável
def p_Valor_IntID(self, p):
    "Valor : VARINT"
    p[0] = f"pushg {self.var[p[1]][0]}\n"

def p_Valor_FloatID(self, p):
    "ValorFloat : VARFLOAT"
    p[0] = f"pushg {self.var[p[1]][0]}\n"

def p_Valor_ArrayID(self, p):
    "Valor : VARARRAY '[' Expressao ']'"
    p[0] = f"pushgp\npushi {self.var[p[1]][0]}\npadd\n{p[3]}loadn\n"

def p_Valor_2DArrayID(self, p):
    "Valor : VARARRAY '[' Expressao ']' '[' Expressao ']'"
    p[0] = f"pushgp\npushi {self.var[p[1]][0]}\npadd\n{p[3]}pushi {self.var[p[1]][2][1]}\nmul

# Definição do Valor da Variável
# Valor de Int
def p_Valor_Int(self, p):
    "Valor : INT"
    p[0] = f"pushi {p[1]}\n"

# Valor de Float
def p_Valor_FLOAT(self, p):
    "ValorFloat : FLOAT"
    p[0] = f"pushf {p[1]}\n"

# Valor de uma String em Int
def p_Valor_Str_to_Int(self, p):
    "Valor : INTR '(' String ')'"
    p[0] = f"{p[3]}atoi\n"

# Valor de uma String em Float
def p_Valor_str_to_Float(self, p):
    "ValorFloat : FLOATR '(' String ')'"
    p[0] = f"{p[3]}atof\n"

# Valor de um Float em Int
def p_Valor_float_to_int(self, p):
    "Valor : INTR '(' ExpressaoFloat ')'"
```

```
    p[0] = f"{p[3]}ftoi\n"

# Valor de um Int em Float
def p_Valor_int_to_float(self, p):
    "ValorFloat : FLOATR '(' Expressao ')'"
    p[0] = f"{p[3]}itof\n"

##########################################################################################
# Definição de Expressão
def p_ValorExpressao(self, p):
    "Expressao : Valor"
    p[0] = p[1]

def p_ValorExpressao_Float(self, p):
    "ExpressaoFloat : ValorFloat"
    p[0] = p[1]

# Expressão entre parênteses
def p_Expressao_parenteses(self, p):
    "Expressao : '(' Expressao ')'"
    p[0] = p[2]

def p_ExpressaoFloat_parenteses(self, p):
    "ExpressaoFloat : '(' ExpressaoFloat ')'"
    p[0] = p[2]

##########################################################################################
# Expressões Lógicas
# Expressões com Inteiros
def p_Comparacoes_Igual(self, p):
    "ExpLogica : Expressao EQUAL Expressao"
    p[0] = p[1] + p[3] + "equal\n"

def p_Comparacoes_Diferente(self, p):
    "ExpLogica : Expressao DIFF Expressao"
    p[0] = p[1] + p[3] + "equal\nnot\n"

def p_Comparacoes_Maior(self, p):
    "ExpLogica : Expressao '>' Expressao"
    p[0] = p[1] + p[3] + "sup\n"

def p_Comparacoes_Menor(self, p):
    "ExpLogica : Expressao '<' Expressao"
    p[0] = p[1] + p[3] + "inf\n"

def p_Comparacoes_MaiorIgual(self, p):
    "ExpLogica : Expressao GEQUAL Expressao"
    p[0] = p[1] + p[3] + "supeq\n"
```

```
def p_Comparacoes_MenorIgual(self, p):
    "ExpLogica : Expressao LEQUAL Expressao"
    p[0] = p[1] + p[3] + "infeq\n"

# Expressões com dois Floats
def p_ComparacoesFloat_Igual(self, p):
    "ExpLogica : ExpressaoFloat EQUAL ExpressaoFloat"
    p[0] = p[1] + p[3] + "equal\n"

def p_ComparacoesFloat_Diferente(self, p):
    "ExpLogica : ExpressaoFloat DIFF ExpressaoFloat"
    p[0] = p[1] + p[3] + "equal\nnot\n"

def p_ComparacoesFloat_Maior(self, p):
    "ExpLogica : ExpressaoFloat '>' ExpressaoFloat"
    p[0] = p[1] + p[3] + "fsup\nftoi\n"

def p_ComparacoesFloat_Menor(self, p):
    "ExpLogica : ExpressaoFloat '<' ExpressaoFloat"
    p[0] = p[1] + p[3] + "finf\nftoi\n"

def p_ComparacoesFloat_MaiorIgual(self, p):
    "ExpLogica : ExpressaoFloat GEQUAL ExpressaoFloat"
    p[0] = p[1] + p[3] + "fsupeq\nftoi\n"

def p_ComparacoesFLoat_MenorIgual(self, p):
    "ExpLogica : ExpressaoFloat LEQUAL ExpressaoFloat"
    p[0] = p[1] + p[3] + "finfeq\nftoi\n"

# Expressões com um Int e um Float
def p_ComparacoesIntFLoat_Igual(self, p):
    "ExpLogica : Expressao EQUAL ExpressaoFloat"
    p[0] = p[1] + "itof\n" + p[3] + "equal\n"

def p_ComparacoesInt_Float_Diferente(self, p):
    "ExpLogica : Expressao DIFF ExpressaoFloat"
    p[0] = p[1] + "itof\n" + p[3] + "equal\nnot\n"

def p_ComparacoesInt_Float_Maior(self, p):
    "ExpLogica : Expressao '>' ExpressaoFloat"
    p[0] = p[1] + "itof\n" + p[3] + "fsup\nftoi\n"

def p_ComparacoesInt_Float_Menor(self, p):
    "ExpLogica : Expressao '<' ExpressaoFloat"
    p[0] = p[1] + "itof\n" + p[3] + "finf\nftoi\n"

def p_ComparacoesInt_Float_MaiorIgual(self, p):
```

```
    "ExpLogica : Expressao GEQUAL ExpressaoFloat"
    p[0] = p[1] + "itof\n" + p[3] + "fsupeq\nftoi\n"

def p_ComparacoesInt_FLoat_MenorIgual(self, p):
    "ExpLogica : Expressao LEQUAL ExpressaoFloat"
    p[0] = p[1] + "itof\n" + p[3] + "finfeq\nftoi\n"

# Expressões com um Float e um Int
def p_ComparacoesFloat_Int_Igual(self, p):
    "ExpLogica : ExpressaoFloat EQUAL Expressao"
    p[0] = p[1] + p[3] + "itof\n" + "equal\n"

def p_ComparacoesFloat_Int_Diferente(self, p):
    "ExpLogica : ExpressaoFloat DIFF Expressao"
    p[0] = p[1] + p[3] + "itof\n" + "equal\nnot\n"

def p_ComparacoesFloat_Int_Maior(self, p):
    "ExpLogica : ExpressaoFloat '>' Expressao"
    p[0] = p[1] + p[3] + "itof\n" + "fsup\nftoi\n"

def p_ComparacoesFloat_Int_Menor(self, p):
    "ExpLogica : ExpressaoFloat '<' Expressao"
    p[0] = p[1] + p[3] + "itof\n" + "finf\nftoi\n"

def p_ComparacoesFloat_Int_MaiorIgual(self, p):
    "ExpLogica : ExpressaoFloat GEQUAL Expressao"
    p[0] = p[1] + p[3] + "itof\n" + "fsupeq\nftoi\n"

def p_ComparacoesFLoat_Int_MenorIgual(self, p):
    "ExpLogica : ExpressaoFloat LEQUAL Expressao"
    p[0] = p[1] + p[3] + "itof\n" + "finfeq\nftoi\n"

##########################################################################################
# Os valores das ExpLogica tomam valores de 0 ou 1
def p_ExpLogica_Parenteses(self, p):
    "ExpLogica : '(' ExpLogica ')'"
    p[0] = p[2]

def p_ExpLogica_And(self, p):
    "ExpLogica : ExpLogica AND ExpLogica"
    p[0] = p[1] + p[3] + "mul\n"

def p_ExpLogica_Or(self, p):
    "ExpLogica : ExpLogica OR ExpLogica"
    p[0] = p[1] + p[3] + "add\n"

def p_ExpLogica_Not(self, p):
    "ExpLogica : NOT ExpLogica"
```

```
    p[0] = p[2] + "not\n"

def build(self, **kwargs):
    self.var = dict()
    self.tamanho_stack = 0
    self.lexer = Lexer(self.var)
    self.lexer.build()
    self.parser = yacc.yacc(module=self, **kwargs)
    self.ifs = 0
    self.if_else = 0
    self.ciclo = 0
    self.ciclo_fim = 0
```