Universidade do Minho

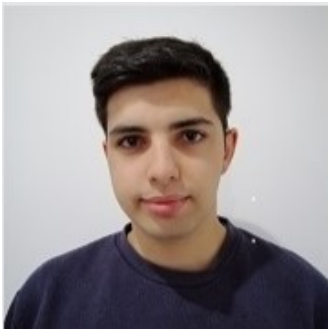
Licenciatura em Ciências da Computação

Processamento de Linguagens e Compiladores
Relatório

Grupo 13

Miguel Rego (a94017) Paulo Freitas (a100053)
Pedro Santos (a100110)

Ano Letivo 23/24


Conteúdo

Capítulo 1

Introdução

No âmbito da unidade curricular de Processamento de Linguagens e Compiladores, foi proposto a realização de um trabalho prático que consiste na implementação de programa capaz de produzir um código Assembly a partir de um dado input de declarações de variáveis e uma função.

Através de uma Gramática Independente do contexto (GIC) deve ser possível estabelecer um conjunto de regras e, posteriormente, elaborar um analisador léxico (Lexer) e um gerador de analise sintática (Yacc). O analisador léxico deve ser capaz de identificar tokens e conjunto de símbolos que invocam regras da gramática GIC. O analisador sintático deve definir, em código, tais regras e gerar o código Assembly. Tanto o GIC como o Lexer e o Yacc foram desenvolvidos em linguagem Python com recurso ao modulo "PLY".

O seguinte relatório vem esclarecer a nossa abordagem ao problema proposto, explicar a estrutura do projeto elaborado, e também demonstrar os conhecimentos adquiridos ao longo do semestre, nomeadamente sobre expressões regulares, gramática independente do contexto, analisadores léxicos e analisadores sintáticos .

Capítulo 2

Descrição do Problema

O presente trabalho foi proposto de modo a mostrar os nossos conhecimentos sobre o conteúdo lecionado ao longo do semestre, nomeadamente, a nossa aptidão para escrever gramáticas independente do contexto (GIC) e tradutoras (GT), de modo a garantir a condição LR() e com recurso a BNF puro.

Assim, neste trabalho prático temos como objetivos:

- o desenvolvimento de processadores de linguagens segundo o método da tradução dirigida pela sintaxe, a partir de uma GT;
- o desenvolvimento de um compilador que gera código para a **VM (Virtual Machine**);
- a utilização de geradores de compiladores baseados em GT, em concreto o **Yacc**,e também pelo gerador de analisadores léxicos **Lex**, ambos na versão **PLY** do **Pyhton**.

Temos ainda que desenvolver uma linguagem imperativa a ser usada pelas gramáticas.

- 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;
- declarar e manusear variáveis estruturadas do tipo array de inteiros, em relação aos quais é apenas permitida a operação de indexação;

De forma a manter o bom funcionamento do programa, a declaração de variáveis sem indicação do respetivo valor, será assumido o valor 0 na variável em questão, como também não é possível re-declarar variáveis.

Capítulo 3

Implementação da Gramática

Para alem dos requisitos essenciais propostos no enunciado apresentado anteriormente, adicionamos outras funcionalidades à gramática, das quais estão organizadas nos seguintes sub-tópicos.

3.1 Linguagem

Semelhante a outras linguagens imperativas populares, a nossa gramática é *Case Sensitve*, isto é, a palavras que diferem de caracteres maiúsculos ou minúsculos, remetem a significados diferentes.

3.2 Declaração de Variáveis

Possibilidade de declarar variáveis inteiras, array de inteiros e um array bidimensionais de inteiros;

- Um array bidimensional (Matriz), no momento da declaração, é necessário colocar seguido do nome, dois pares de parênteses reto, nos quais consta as dimensões do array;
- É possível fazer atribuições a variáveis, como por exemplo, atribuição de um valor inteiro a uma variável, através do uso do símbolo "=", após o nome da variável sendo colocado a frente do símbolo o valor pretendido;
- Caso uma variável seja declarada sem valor, essa tomará valor 0 no caso de int, array de 0's caso array ou array de arrays de 0's caso matriz.
- Possibilidade de definir o valor de uma posição do array. Para isso é necessário indicar o nome do array seguido da posição a definir entre parêntesis reto, seguido de um símbolo '=' e o valor a definir.
- Possibilidade de definir o valor de uma posição da matriz. Para isso é necessário indicar o nome do array bidimensional seguido da posição a definir entre dois pares de parêntesis reto, seguido de um símbolo '=' e o valor a definir.

3.3 Operações

São aceites na gramática, operações aritméticas comuns (soma, diferença, multiplicação, divisão e resto de divisão inteira (mod)), e operações lógicas (e,ou,maior, menor, igual, negação, maior ou igual e menor ou igual)

3.4 Comandos

É possível, criar funções com um ou mais comandos.

O comando *Write* escreve no output o conteúdo imediatamente seguinte da sua invocação.

O comando *Read* lê do input um valor e atribui à variável imediatamente a seguir à invocação do comando

3.5 Comandos Condicionais

São aceites na gramática, comandos comummente conhecidos como If-Then-Else, onde executa um comando/expressão caso uma certa condição seja verdadeira, ou outro comando/expressão caso contrário.

3.6 Comandos Cíclicos

É aceite na gramática, o comando While, onde dada uma condição e um comando/expressão, executa um comando/expressão enquanto tal condição for verdadeira.

3.7 Indexação

Para aceder a um elemento na posição "i"de um dado array "a", basta escrever "a[i]".

Para aceder a um elemento na posição "i"e "j"de um dado array bidimensional "a", basta escrever "a[i][j]".

3.8 Gramática independente do Contexto (GIC)

Para a implementação das funcionalidades referidas anteriormente, foi criada a seguinte GIC:

```
Programa    : Declaracoes Funcao

Funcao      : Funcao Cmds
            | Cmds

Declaracoes : Declaracao Declaracoes
            | Declaracao

Declaracao  : INT ID
            | INT ID '=' NUM
            | INT ID '[' NUM ']'
            | INT ID '[' NUM ']''['NUM ']'

Arrays      : '[' Array ']' ',' Arrays
            | €

Array       : NUM ',' Array
            | €

Cmds        : Cmd Cmds
            | €

Cmd         : Cmd_If
            | Cmd_If_Else
            | Cmd_While
            | Cmd_Writes
            | Cmd_Read
            | Id '=' Exp
            | ID '[' NUM ']' '=' Exp
            | ID '[' NUM ']' '[' NUM ']' '=' Exp

Cmd_If      : IF LPAREN Condicao RPAREN '{' Cmds '}'

Cmd_If_Else : IF LPAREN Condicao RPAREN '{' Cmds '}' ELSE '{' Cmds '}'

Cmd_While   : WHILE LPAREN Condicao RPAREN '{' Cmds '}'

Cmd_Writes  : WRITE LPAREN Cmd_Write Writes RPAREN
            | WRITE LPAREN Cmd_Write RPAREN

Writes      : ',' Cmd_Write
            | ',' Cmd_Write Writes

Cmd_Write   : Exp
            | ID '[' Factor ']'
            | ID '[' Factor ']' '[' Factor ']'

Cmd_Read    : READ ID
            | READ ID '[' NUM ']'
```

```
            | READ ID '[' NUM ']' '[' NUM ']'

Condicao    : LPAREN Cond RPAREN
            | NOT Cond
            | Condicao AND Condicao
            | Condicao OR Condicao
            | Cond

Cond        : Exp '==' Exp
            | Exp '>' Exp
            | Exp '<' Exp
            | Exp '>=' Exp
            | Exp '<=' Exp
            | Condicao

Exp         : Exp '+' Term
            | Exp '-' Term
            | Term

Term        : Term '/' Factor
            | Term '*' Factor
            | Term '%' Factor
            | Factor

Factor      : NUM
            | Id
            | ID '[' NUM ']' '=' Exp
            | ID '[' NUM ']' '[' NUM ']' '=' Exp
            | '-' Exp

Id          : ID
```

Capítulo 4

Exemplos de Aplicação

De modo a mostrar o bom funcionamento do código e a exemplificar o uso do mesmo, seguem-se alguns exemplos do input do código e o seu código Assembly gerado pelo programa.

4.1 Exemplo 1 - Declaração de Variáveis e Operações aritméticas

Dado 4 variáveis de input, realiza operações aritméticas.

INPUT

```
INT a = 10
INT resultado
INT x
INT y
INT z
INT w
READ x
READ y
READ z
READ w
resultado = ((a * a * a * x) + (a * a * y) + (a * z))
WRITE (resultado)
resultado = (a * w) - (y / z) % 2
WRITE (resultado)
```

OUTPUT

```
PUSHI 10
PUSHI 0
PUSHI 0
PUSHI 0
PUSHI 0
PUSHI 0
```

```
START
READ
ATOI
STOREG 2
READ
ATOI
STOREG 3
READ
ATOI
STOREG 4
READ
ATOI
STOREG 5
PUSHG 0
PUSHG 0
MUL
PUSHG 0
MUL
PUSHG 2
MUL
PUSHG 0
PUSHG 0
MUL
PUSHG 3
MUL
ADD
PUSHG 0
PUSHG 4
MUL
ADD
STOREG 1
PUSHG 1
WRITEF
PUSHS "\n"
WRITES
PUSHG 0
PUSHG 5
MUL
PUSHG 3
PUSHG 4
DIV
PUSHI 2
MOD
SUB
STOREG 1
PUSHG 1
WRITEF
PUSHS "\n"
WRITES
STOP
```

4.2 Exemplo 2 - Comando Condicional

Verifica se dado os comprimentos *a,b e c*, verifica se é possível fazer um triângulo.

INPUT

```
INT a
INT b
INT c
READ a
READ b
READ c
IF (((a == 0) OR (b == 0) OR (c == 0)) OR ((a + b <= c) OR (a + c <= b) OR (b + c <= a)))
{
        WRITE (a)
} ELSE{
    a=a+1
    WRITE(a)
}
```

OUTPUT

```
PUSHI 0
PUSHI 0
PUSHI 0
START
READ
ATOI
STOREG 0
READ
ATOI
STOREG 1
READ
ATOI
STOREG 2
PUSHG 0
PUSHI 0
EQUAL
PUSHG 1
PUSHI 0
EQUAL
PUSHG 2
PUSHI 0
EQUAL
OR
OR
PUSHG 0
PUSHG 1
ADD
PUSHG 2
INFEQ
```

```
PUSHG 0
PUSHG 2
ADD
PUSHG 1
INFEQ
PUSHG 1
PUSHG 2
ADD
PUSHG 0
INFEQ
OR
OR
OR
JZ l0
PUSHG 0
WRITEF
PUSHS "\n"
WRITES
JUMP l0f
l0: NOP
PUSHG 0
PUSHI 1
ADD
STOREG 0
PUSHG 0
WRITEF
PUSHS "\n"
WRITES
l0f: NOP
STOP
```

4.3 Exemplo 3 - While e Comando Condicional

Enquanto a variável i for menor que 5, imprime o resultado de uma expressão, caso tal resultado for igual a 15/2, caso contrário, imprime o valor de i.

INPUT

```
INT a = 7
INT b = 4
INT result = 0
INT i = 1
WHILE (i <= 5) {
    result = ((a * i) + (b * 2)) / (i + 1)
    IF (result == 15/2){
        WRITE(result)
    } ELSE{
        WRITE(i)
    }
     i = i + 1
```

```
}
```

OUTPUT

```
PUSHI 7
PUSHI 4
PUSHI 0
PUSHI 1
START
l1c: NOP
PUSHG 3
PUSHI 5
INFEQ
JZ l1f
PUSHG 0
PUSHG 3
MUL
PUSHG 1
PUSHI 2
MUL
ADD
PUSHG 3
PUSHI 1
ADD
DIV
STOREG 2
PUSHG 2
PUSHI 15
PUSHI 2
DIV
EQUAL
JZ l0
PUSHG 2
WRITEF
PUSHS "\n"
WRITES
JUMP l0f
l0: NOP
PUSHG 3
WRITEF
PUSHS "\n"
WRITES
l0f: NOP
PUSHG 3
PUSHI 1
ADD
STOREG 3
JUMP l1c
l1f: NOP
STOP
```

4.4 Exemplo 4 - Mudança de valor em posições de um Array

INPUT

```
INT i = 0
INT b [ 10 ]
INT a [ 3 ] [ 3 ]
a [ 2 ] [ 2 ] = 3
WRITE (a [ 2 ] [ 2 ])
WHILE (i < 10){
    b[ i ] = i
    i = i+1
    }
WRITE ( b [ 10 ] )
```

OUTPUT

```
PUSHI 0
PUSHN 10
PUSHN 9
START
PUSHGP
PUSHI 11
PADD
PUSHI 2
PUSHI 3
MULPUSHI 2

ADD
PUSHI 3
STOREN
PUSHGP
PUSHI 11
PADD
PUSHI 0
PUSHI 3
MUL
PUSHI 0
ADD
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 11
PADD
PUSHI 0
PUSHI 3
MUL
PUSHI 1
ADD
```

```
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 11
PADD
PUSHI 0
PUSHI 3
MUL
PUSHI 2
ADD
LOADN
WRITEI
PUSHS " "
WRITES
PUSHS "\n"
WRITES
PUSHGP
PUSHI 11
PADD
PUSHI 1
PUSHI 3
MUL
PUSHI 0
ADD
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 11
PADD
PUSHI 1
PUSHI 3
MUL
PUSHI 1
ADD
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 11
PADD
PUSHI 1
PUSHI 3
MUL
PUSHI 2
ADD
LOADN
WRITEI
PUSHS " "
WRITES
```

```
PUSHS "\n"
WRITES
PUSHGP
PUSHI 11
PADD
PUSHI 2
PUSHI 3
MUL
PUSHI 0
ADD
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 11
PADD
PUSHI 2
PUSHI 3
MUL
PUSHI 1
ADD
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 11
PADD
PUSHI 2
PUSHI 3
MUL
PUSHI 2
ADD
LOADN
WRITEI
PUSHS " "
WRITES
PUSHS "\n"
WRITES
l0c: NOP
PUSHG 0
PUSHI 10
INF
JZ l0f
PUSHGP
PUSHI 1
PADD
PUSHG 0
PUSHG 0
STOREN
PUSHG 0
PUSHI 1
ADD
```

```
STOREG 0
JUMP l0c
l0f: NOP
PUSHGP
PUSHI 1
PADD
PUSHI 0
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 1
PADD
PUSHI 1
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 1
PADD
PUSHI 2
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 1
PADD
PUSHI 3
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 1
PADD
PUSHI 4
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 1
PADD
PUSHI 5
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 1
PADD
```

```
PUSHI 6
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 1
PADD
PUSHI 7
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 1
PADD
PUSHI 8
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 1
PADD
PUSHI 9
LOADN
WRITEI
PUSHS " "
WRITES
PUSHS "\n"
WRITES
STOP
```

Capítulo 5

Conclusão

De modo geral, o presente trabalho permitiu a melhor consolidação do conteúdo lecionado pela UC, tanto na questão da criação da Gramática como no Analisador Léxico. Num momento inicial, o grupo viu-se com bastantes dificuldades em compreender o enunciado inicial, o que levou a várias retificações da gramática independente do contexto. Após esse entrave inicial, o grupo conseguiu progredir na elaboração. Consideramos que o grupo obteu sucesso na resolução das funcionalidades propostas no enunciado. Em uma possível reintervenção, consideramos que novas funcionalidades poderiam ser adicionada, como a aceitação de declaração de variáveis do tipo *Float*, *Char* e *string*.

Capítulo 6

Apêndice: A

Analisador Léxico

Segue-se o código do Analisador Léxico, em Python e com recurso ao modulo PLY, usado na elaboração do presente trabalho.

```
import ply.lex as lex

tokens = (
   # TIPOS - feito em yacc
   'INT','NUM','STR','ID',

   # OPERACOES FUNCOES
   'WRITE','READ','IF','ELSE',
   'WHILE',

   # OPERACOES LOGICAS - feito em yacc
   'LPAREN','RPAREN','AND','OR',
   'NOT','EQUALS','GT','LT',
   'GE','LE',

   # OPERAÇOES ARITMETICAS - feito em yacc
   'PLUS','MINUS','TIMES','DIVIDE',
   'RESTO'

)

#################

# literals - simbolos que têm um significado
literals = ['+','-','*','/','%','=',
            '(',')','.',',',',','<','>',
            '!','{','}','[',']']


#################
```

```
# DEFINICAO TOKENS

# TIPOS
def t_INT(t):
   r'INT'
   return t

def t_NUM(t):
   r'-?\d+'
   t.value = int(t.value)
   return t
   return t

# OPERACOES FUNCOES
def t_WRITE(t):
   r'WRITE'
   return t


def t_READ(t):
   r'READ'
   return t

def t_IF(t):
   r'IF'
   return t

def t_ELSE(t):
   r'ELSE'
   return t

def t_WHILE(t):
   r'WHILE'
   return t

# OPERACOES LOGICAS

t_LPAREN = r'\('
t_RPAREN = r'\)'
t_EQUALS = r'=='

def t_AND(t):
   r'AND'
   return t

def t_OR(t):
   r'OR'
   return t

def t_NOT(t):
   r'NOT'
   return t

t_GT = r'>'
```

```
t_LT = r'<'
t_GE = r'>='
t_LE = r'<='

# OPERACOES ARITMETICAS
t_PLUS = r'\+'
t_MINUS = r'-'
t_TIMES = r'\*'
def t_DIVIDE(t):
   r'/'
   return t
t_RESTO = r'%'

# CASOS ESPECIAIS
def t_ID(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
    return t

t_ignore = " \t"

def t_newline(t):
   r'\n+'
   t.lexer.lineno += len(t.value)

def t_error(t):
   print('Illegal character: ', t.value[0])
   t.lexer.skip(1)

lexer = lex.lex()
```

Capítulo 7

Apêndice: B

Compilador Yacc

Segue-se o código do Compilador Yacc, em Python e com recurso ao modulo PLY, usado na elaboração do presente trabalho.

```
from lex import tokens
import ply.yacc as yacc

precedence = (('right', 'UMINUS'),)

def p_init(p):
    '''Init : Declaracoes Funcao
            | Funcao
    '''
    if len(p) == 2:
        parser.assembly = f'START\n{p[1]}STOP'
    else:
        parser.assembly = f'{p[1]}START\n{p[2]}STOP'

def p_funcao(p):
    'Funcao      : Cmds'
    p[0] = f'{p[1]}'

def p_funcao_rec(p):
    'Funcao      : Funcao Cmds'
    p[0] = f'{p[1]}{p[2]}'

def p_Arrays(p):
    '''Arrays : Array ',' Arrays
              | Array
    '''
    if len(p) > 2:
        p[0] = f'{p[1]}{p[3]}'
    else:
        p[0] = f'{p[1]}'
```

```
def p_Array(p):
    '''Array : NUM ',' Array
             | NUM
    '''
    if len(p) > 2:
        p[0] = f'{p[1]}'
    else:
        p[0] = f'{p[1]}'

def p_cmds(p):
    '''Cmds : Cmds Cmd
            | Cmd
    '''
    if len(p) > 2:
        p[0] = f'{p[1]}{p[2]}'
    else:
        p[0] = f'{p[1]}'

def p_cmd_writes(p):
    '''Cmd_Writes : WRITE LPAREN Cmd_Write Writes RPAREN
                  | WRITE LPAREN Cmd_Write RPAREN
    '''
    if len(p) > 5:
        p[0] = f'{p[3]}{p[4]}'
    else:
        p[0] = f'{p[3]}'

def p_writes(p):
    '''Writes : ',' Cmd_Write
              | ',' Cmd_Write Writes
    '''
    if len(p) > 3:
        p[0] = f'{p[2]}{p[3]}'
    else:
        p[0] = f'{p[2]}'


def p_cmd_write(p):
    '''Cmd_Write : Exp
    '''
    p[0] = f'{p[1]}WRITEF\nPUSHS "\\n"\nWRITES\n'


def p_cmd_writeArray(p):
    '''Cmd_Write : ID '[' Factor ']'
    '''
    if p[1] in p.parser.registers:
        if p[1] not in p.parser.ints:
            if len(p.parser.registers.get(p[1])) == 2:
                array = ""
                for i in range(p.parser.registers.get(p[1])[1]):
                    array += f'PUSHGP\nPUSHI {p.parser.registers.get(p[1])[0]} \nPADD\nPUSHI
                    array += f'WRITEI\n'
                    array += f'PUSHS " "\nWRITES\n'
```

```
                p[0] = array + f'PUSHS "\\n"\nWRITES\n'
        else:
            print(f"Erro: {p[1]} não é um array.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_cmd_writeMatrix(p):
    '''Cmd_Write : ID '[' Factor ']' '[' Factor ']'
    '''
    if p[1] in p.parser.registers:
        if p[1] not in p.parser.ints and len(p.parser.registers.get(p[1])) == 3:
            rows = p.parser.registers.get(p[1])[1]
            cols = p.parser.registers.get(p[1])[2]

            matrix_output = ""
            for i in range(rows):
                for j in range(cols):
                    matrix_output += f'PUSHGP\nPUSHI {p.parser.registers.get(p[1])[0]}\nPADD\
                    matrix_output += f'PUSHI {i}\nPUSHI {cols}\nMUL\n'  # Corrected multiplic
                    matrix_output += f'PUSHI {j}\nADD\nLOADN\n'
                    matrix_output += 'WRITEI\nPUSHS " "\nWRITES\n'
                matrix_output += 'PUSHS "\\n"\nWRITES\n'
            p[0] = matrix_output
        else:
            print(f"Erro: {p[1]} não é uma matriz.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False



def p_Declaracoes(p):
    '''Declaracoes : Declaracao
                   | Declaracoes Declaracao
    '''
    if len(p) == 2:
        p[0] = f'{p[1]}'
    else:
        p[0] = f'{p[1]}{p[2]}'

def p_Declaracao_Int(p):
    '''Declaracao : INT ID
                  | INT ID '=' NUM
    '''
    num = 0
    if len(p) > 3:
        num = p[4]
    if p[2] not in p.parser.registers:
        p.parser.registers.update({p[2] : p.parser.gp})
        p[0] = f'PUSHI {num}\n'
        p.parser.ints.append(p[2])
```

```
        p.parser.gp += 1
    else:
        print("Error: Variável já existe.")
        parser.success = False

def p_Declaracao_Array(p):
    '''Declaracao : INT ID '[' NUM ']'
    '''
    #

    if p[2] not in p.parser.registers:
        p.parser.registers.update({p[2]: (p.parser.gp, int(p[4]))})
        p[0] = f'PUSHN {p[4]}\n'
        p.parser.gp += int(p[4])
    else:
        print("Erro: Variável já existe.")
        parser.success = False

def p_Declaracao_Matriz(p):
    '''Declaracao : INT ID '[' NUM ']' '[' NUM ']'
    '''
    if p[2] not in p.parser.registers:
        p.parser.registers.update({p[2]: (p.parser.gp, int(p[4]), int(p[7]))})
        size = int(p[4]) * int(p[7])
        p[0] = f'PUSHN {str(size)}\n'
        p.parser.gp += size
    else:
        print("Erro: Variável já existe.")
        parser.success = False

def p_cmd_atr(p):
    '''Cmd : ID '=' Exp'''
    if p[1] in p.parser.registers:
        if p[1] in p.parser.ints:
            p[0] = f'{p[3]}STOREG {p.parser.registers.get(p[1])}\n'
        else:
            print("Error: Variável não é int")
            parser.success = False
    else:
        print("Error: Variável não definida.")
        parser.success = False

#o padd do 1 adicionado antees é aqui
def p_cmd_array(p):
    '''
    Cmd : ID '[' Factor ']' '=' Exp
    '''
    if p[1] in p.parser.registers:
        if p[1] not in p.parser.ints and len(p.parser.registers.get(p[1])) == 2:
            p[0] = f'PUSHGP\nPUSHI {p.parser.registers.get(p[1])[0]}\nPADD\n{p[3]}{p[6]}STORE
        else:
            print(f"Erro: Variável {p[1]} não é um array.")
            parser.success = False
    else:
```

```
        print("Erro: Variável não definida.")
        parser.success = False

#falta ver isto
def p_cmd_Matriz(p):
    '''
    Cmd : ID '[' Factor ']' '[' Factor ']' '=' Exp
    '''
    if p[1] in p.parser.registers:
        if p[1] not in p.parser.ints and len(p.parser.registers.get(p[1])) == 3:
            c = p.parser.registers.get(p[1])[2]
            p[0] = f'PUSHGP\nPUSHI {p.parser.registers.get(p[1])[0]}\nPADD\n{p[3]}PUSHI {c} \
        else:
            print(f"Erro: Variável {p[1]} não é uma matriz.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_factor_par(p):
    'Factor : LPAREN Exp RPAREN'
    p[0] = p[2]

def p_exp_operations(p):
    '''Exp : Exp PLUS Term
           | Exp MINUS Term
           | Term
    '''
    if len(p) == 4:
        if p[2] == '+':
            p[0] = f'{p[1]}{p[3]}ADD\n'
        elif p[2] == '-':
            p[0] = f'{p[1]}{p[3]}SUB\n'
    else:
        p[0] = p[1]

def p_term_factor(p):
    '''Term : Term DIVIDE Factor
            | Term TIMES Factor
            | Term RESTO Factor
            | Factor
    '''
    if len(p) > 2:
        if p[2] == '/':
            p[0] = f'{p[1]}{p[3]}DIV\n'
        elif p[2] == '*':
            p[0] = f'{p[1]}{p[3]}MUL\n'
        elif p[2] == '%':
            p[0] = f'{p[1]}{p[3]}MOD\n'
    else:
        p[0] = p[1]

def p_factor_id(p):
    '''Factor : ID
```

```
    '''
    if p[1] in p.parser.registers:
        if p[1] in p.parser.ints:
            p[0] = f'PUSHG {p.parser.registers.get(p[1])}\n'
        else:
            print("Erro: Variável não é de tipo inteiro.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_factor_Array(p):
    '''Factor : ID '[' Exp ']'
    '''
    if p[1] in p.parser.registers:
        if p[1] not in p.parser.ints and len(p.parser.registers.get(p[1])) == 2:
            p[0] = f'PUSHGP\nPUSHI {p.parser.registers.get(p[1])[0]}\nPADD\n{p[3]}LOADN\n'
        else:
            print(f"Erro: Variável {p[1]} não é um array.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_factor_Matrix(p):
    '''Factor : ID '[' Exp ']' '[' Exp ']'
    '''
    if p[1] in p.parser.registers:
        if p[1] not in p.parser.ints and len(p.parser.registers.get(p[1])) == 3:
            c = p.parser.registers.get(p[1])[2]
            p[0] = f'PUSHGP\nPUSHI {p.parser.registers.get(p[1])[0]}\nPADD\n{p[3]}PUSHI {c}\n
        else:
            print(f"Erro: Variável {p[1]} não é uma matriz.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_factor_not(p):
    'Factor : MINUS Exp %prec UMINUS'
    p[0] = f'PUSHI 0\n{p[2]}SUB\n'

#aqui esta a ser adicionado o 1antes do PUSHI

def p_factor_num(p):
    '''Factor : NUM
    '''
    p[0] = f'PUSHI {p[1]}\n'

def p_cmd(p):
    '''
    Cmd : Cmd_Writes
        | Cmd_Read
        | Cmd_If
```

```
        | Cmd_If_Else
        | Cmd_While
    '''
    p[0] = p[1]

def p_cmd_read(p):
    '''Cmd_Read : READ ID
    '''
    if p[2] in p.parser.registers:
        p[0] = f'READ\nATOI\nSTOREG {p.parser.registers.get(p[2])}\n'
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_cmd_readArray(p):
    '''Cmd_Read : READ ID '[' NUM ']'
    '''
    if p[2] in p.parser.registers:
        if p[2] not in p.parser.ints and len(p.parser.registers.get(p[2])) == 2:
            p[0] = f'PUSHGP\nPUSHI {p.parser.registers.get(p[2])[0]}\nPADD\nPUSHI{p[4]} \nREA
        else:
            print(f"Erro: Variável {p[2]} não é um array.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_cmd_readMatrix(p):
    '''Cmd_Read : READ ID '[' NUM ']' '[' NUM ']'
    '''
    if p[2] in p.parser.registers:
        if p[2] not in p.parser.ints and len(p.parser.registers.get(p[2])) == 3:
            c = p.parser.registers.get(p[2])[2]
            p[0] = f'PUSHGP\nPUSHI {p.parser.registers.get(p[2])[0]}\nPADD\nPUSHI {p[4]}\nPUS
        else:
            print(f"Erro: Variável {p[2]} não é uma matriz.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_cmd_if(p):
    "Cmd_If        : IF LPAREN Condicao RPAREN '{' Cmds '}'"
    p[0] = f'{p[3]}JZ l{p.parser.labels}\n{p[6]}l{p.parser.labels}: NOP\n'
    p.parser.labels += 1

def p_condicao(p):
    '''Condicao : Condicao AND Condicao
                | Condicao OR Condicao
                | NOT Condicao
                | LPAREN Condicao RPAREN
                | Cond
    '''
    if len(p) == 4:
```

```
        if p[2] == 'AND':
            p[0] = f'{p[1]}{p[3]}AND\n'
        elif p[2] == 'OR':
            p[0] = f'{p[1]}{p[3]}OR\n'
        else:
            p[0] = p[2]
    elif p[1] == 'NOT':
        p[0] = f'{p[2]}PUSHI 0\nEQUAL\n'
    else:
        p[0] = p[1]

def p_cond(p):
    '''Cond : Exp EQUALS Exp
            | Exp LT Exp
            | Exp LE Exp
            | Exp GT Exp
            | Exp GE Exp
            | Exp
    '''
    if len(p) > 2:
        if p[2] == '==':
            p[0] = f'{p[1]}{p[3]}EQUAL\n'
        elif p[2] == '<':
            p[0] = f'{p[1]}{p[3]}INF\n'
        elif p[2] == '<=':
            p[0] = f'{p[1]}{p[3]}INFEQ\n'
        elif p[2] == '>':
            p[0] = f'{p[1]}{p[3]}SUP\n'
        elif p[2] == '>=':
            p[0] = f'{p[1]}{p[3]}SUPEQ\n'
    else:
        p[0] = p[1]

def p_cmd_if_else(p):
    "Cmd_If_Else        : IF LPAREN Condicao RPAREN '{' Cmds '}' ELSE '{' Cmds '}'"
    p[0] = f'{p[3]}JZ l{p.parser.labels}\n{p[6]}JUMP l{p.parser.labels}f\nl{p.parser.labels}:
    p.parser.labels += 1

def p_cmd_while(p):
    "Cmd_While   : WHILE LPAREN Condicao RPAREN '{' Cmds '}'"
    p[0] = f'l{p.parser.labels}c: NOP\n{p[3]}JZ l{p.parser.labels}f\n{p[6]}JUMP l{p.parser.la
    p.parser.labels += 1

def p_error(p):
    print('Syntax error: ', p)
    parser.success = False

parser = yacc.yacc(debug=True)

parser.success = True
parser.registers = {}
parser.labels = 0
parser.gp = 0
parser.ints = []
```

```
parser.assembly = ""

try:
    with open('./testes/run.txt','r') as file:
        inp = file.read()
        parser.parse(inp)
        if parser.success:
            with open('./testes/run.vm', 'w') as output:
                output.write(parser.assembly)
                print(parser.assembly)
        else:
            print("<><><><><><><><><><><><><><><><><><><><><><><>")
            print("Erro")
            print("<><><><><><><><><><><><><><><><><><><><><><><>")
except KeyboardInterrupt:
    print()
```