Processsamento de Linguagens e Compiladores (3º ano de Curso)

Trabalho Prático 2 - Grupo 3

Relatório de Desenvolvimento

André Neves da Costa
(A95869)

Tiago Emanuel Lemos Teixeira
(A97666)

16 de janeiro de 2023

Resumo

Isto é o relatório do trabalho prático 2 de Processsamento de Linguagens e Compiladores, onde vai ser explicado o método de trabalho por trás do seu desenvolvimento.

Conteúdo

Capítulo 1

Introdução

No âmbito da disciplina de Processamento de Linguagens e Compiladores foi-nos proposto pelo docente Pedro Manuel Rangel Santos Henriques um trabalho cuja realização tem os seguintes objetivos primários:

- aumentar a experiência em engenharia de linguagens e em programação generativa (gramatical), reforçando a capacidade de escrever gramáticas, quer independentes de contexto (GIC), quer tradutoras (GT);
- desenvolver processadores de linguagens segundo o método da tradução dirigida pela sintaxe, a partir de uma gramática tradutora;
- desenvolver um compilador gerando código para uma máquina de stack virtual, no ano corrente será usada a VM, Virtual Machine;
- utilizar geradores de compiladores baseados em gramáticas tradutoras, concretamente o Yacc, versão PLY do Python, completado pelo gerador de analisadores léxicos Lex, também versão PLY do Python;

E objetivos secundários:

- rever e aumentar a capacidade de escrever gramáticas independentes de contexto que satisfaçam a condição LR() usando BNF-puro
- criar o hábito de escrever a documentação (os relatórios dos trabalhos práticos e projectos) em LATEX.

Capítulo 2

Enunciado

Neste trabalho pretende-se que seja definida uma linguagem de programação imperativa simples que permita:

1. declarar variáveis atómicas do tipo inteiro, com os quais se podem realizar as habituais operações aritméticas, relacionais e lógicas.
2. efetuar instruções algorítmicas básicas como a atribuição do valor de expressões numéricas a variáveis.
3. ler do standard input e escrever no standard output.
4. efetuar instruções de seleção para controlo do fluxo de execução.
5. efetuar instruções de repetição (cíclicas) para controlo do fluxo de execução, permitindo o seu aninhamento.
6. 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).
7. 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.

Capítulo 3

Decisões Tomadas

3.1 Funcionamento da Linguagem

3.1.1 Declarações e Atribuições

Comando	Função
Int x	Declarar x como inteiro
Arr x[2]	Declarar x como array com 2 posições
Mat x[2][2]	Declarar x como matriz com 2 linhas e 2 colunas
Int x is 12	Declarar x como um inteiro atribuindo-lhe o valor inicial de 12
x is 3	Atribuir ao x o valor 3

3.1.2 Operações

Operações aritméticas

Comando	Função
(x + y)	Somar x a y
(x - y)	Subtrair y a x
(x / y)	Dividir x por y
(x >< y)	Multiplicar x por y
(x % y)	Calcular o resto da divisão inteira de x por y

Operações relacionais

Comando	Função
(x^y)	x maior que y
(x _ y)	x menor que y
(x <-> y)	x igual a y
(x # y)	x diferente de y
(x i y)	x maior ou igual a y
(x _i y)	x menor ou igual a y

Operações lógicas

Comando	Função
(x n y)	Disjunção de x com y
(x u y)	Conjunção de x com y
(~ x)	Negação de x

3.1.3 Input-Output

Comando	Função
output(x)	Escreve o valor de x no standard output
outputAM(x)	Escreve os valores do array ou matriz x no standard output
input	Lê um valor do standard input

3.1.4 Instruções Condicionais

If-then

Seja x a condição que fará executar ... se for verdadeira:

```
1 se x
2 | ...
3 $
```

If-then-else

Seja x a condição que fará executar ... se for verdadeira ou ,,, se for falsa:

```
1 se x
2 | ...
3 se!
4 | ,,,
5 $
```

3.1.5 Instruções Cíclicas

While-do

Seja x a condição que, enquanto for verdadeira, fará executar ...:

```
eqnt x ->
...
$
```

3.1.6 Sub-programas

Seja s o nome do subprograma e ... o que ele executa:

```
def s()$
...
$
```

3.1.7 Indexação

Sejam i e j números inteiros:

Comando	Função
x[i][j]	Aceder ao inteiro na linha i e coluna j da matriz x
x[i]	Aceder ao inteiro na posição i do array x

3.2 Gramática

A gramática implementada é constituida pelas seguintes regras de derivação:

```
Script  : Decls SubPs Corpo
        | Decls Corpo
        | SubPs Corpo
        | Corpo

Decls   : Decl
        | Decls Decl

Decl    : MATRIZ NOME SA Exp SF SA Exp SF
        | ARRAY NOME SA Exp SF
        | INT NOME
        | INT NOME ART Exp

Atr     : NOME SA Exp SF SA Exp SF ATR Exp
        | NOME SA Exp SF ATR Exp
        | NOME ATR Exp
```

```
17            | NOME SA Exp SF ATR INPUT
18            | NOME SA Exp SF SA Exp SF ATR INPUT
19            | NOME ATR INPUT
20
21 SubPs    : SubP
22            | SubPs SubP
23
24 SubP     : SUBP NOME PA PF DELIM Decls Corpo DELIM
25            | SUBP NOME PA PF DELIM Corpo DELIM
26
27 Corpo    : Proc
28            | Corpo Proc
29
30 Proc     : Exp
31            | Atr
32            | Output
33            | Se
34            | Enquanto
35
36 Exp      : PA Exp PF
37            | Var
38            | NUM
39            | Cond
40            | Call
41            | Exp SOMA Exp
42            | Exp MULT Exp
43            | Exp SUB Exp
44            | Exp DIV Exp
45            | Exp MOD Exp
46
47 Output   : OUTPUT PA Exp PF
48            | OUTPUTAM PA NOME PF
49
50 Se       : SE Cond ENTAO Corpo DELIM
51            | SE Cond ENTAO Corpo SENAO ENTAO Corpo DELIM
52
53 Enquanto: ENQUANTO Cond FAZ Corpo DELIM
54
55 Var      : NOME SA Exp SF SA Exp SF
56            | NOME SA Exp SF
57            | NOME
58
59 Cond     : PA Cond PF
60            | Exp MAIOR Exp
61            | Exp MENOR Exp
62            | Exp IGUAL Exp
63            | Exp MENORIG Exp
64            | Exp MAIORIG Exp
65            | Exp DIF Exp
66            | Exp E Exp
67            | Exp OU Exp
68            | NO Exp
69
70 Call     : CALL PA NOME PF
```

3.3 Variáveis usadas no compilador

1. **parser.success**

   É uma variável booleana que é declarada com o valor True e mudará o valor para False caso haja um erro na sintaxe do código.

2. **parser.assembly**

   Contém o string com a tradução do código para assembly.

3. **parser.log**

   É um dicionário que vai guardar o nome das variáveis e memorizar a posição do seu valor na stack.

4. **parser.labels**

   É um inteiro que vai corresponder ao label atual, ao ser usado um JUMP ou JZ usa-se como label este inteiro e de seguida é incrementado.

5. **parser.gp**

   É um inteiro que corresponde à posição da stack em que o compilador se encontra.

6. **parser.vars**

   É uma lista que vai ter o nome de todas as variáveis declaradas.

7. **parser.subP**

   É um dicionário que vai ter o nome de todos os sub-programas associados ao seu código respetivos.

Capítulo 4

Exemplos de utilização

4.1 Construção de uma matriz 3x3 apartir do stdin

```
1  Mat ord[3][3]
2  Int lin
3  Int col
4
5
6  eqnt (lin _i 2) ->
7      eqnt (col _i 2) ->
8          ord[lin][col] is input
9          col is (col + 1)$
10     lin is (lin + 1)$
11
12 outputAM(ord)
```

Para Assembly:

```
PUSHN 9
PUSHI 0
PUSHI 0

start
l1w: NOP
PUSHG 9
PUSHI 2
INFEQ
JZ l1e
l0w: NOP
PUSHG 10
PUSHI 2
INFEQ
JZ l0e
PUSHGP
PUSHI 0
```

```
PADD
PUSHG 9
PUSHI 3
MUL
PUSHG 10
ADD
READ
ATOI
STOREN
PUSHG 10
PUSHI 1
ADD

STOREG 10
JUMP l0w
l0e: NOP
PUSHG 9
PUSHI 1
ADD

STOREG 9
JUMP l1w
l1e: NOP
PUSHGP
PUSHI 0
PADD
PUSHI 0
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 1
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 2
LOADN
WRITEI
PUSHS
WRITES
```

```
PUSHS ""
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 3
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 4
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 5
LOADN
WRITEI
PUSHS
WRITES
PUSHS ""
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 6
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 7
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 8
```

```
LOADN
WRITEI
PUSHS
WRITES
PUSHS ""
WRITES
stop
```

4.2 Array de fatoriais das respetivas posições

```
1  Arr facts[11]
2
3  Int f is 1
4  Int numero
5
6  eqnt (numero _ 11) ->
7          se (~(numero ^ 1)) | facts[numero] is f
8          se!                |
9                              f is (f >< numero)
10                             facts[numero] is f $
11                             numero is (numero+1) $
12
13 outputAM(facts)
```

Para Assembly:

```
PUSHN 11
PUSHI 1
PUSHI 0

start
l1w: NOP
PUSHG 12
PUSHI 11
INF
JZ l1e
PUSHG 12
PUSHI 1
SUP
NOT
JZ l0
PUSHGP
PUSHI 0
PADD
PUSHG 12
PUSHG 11
STOREN
```

```
JUMP l0e
l0: NOP
PUSHG 11
PUSHG 12
MUL

STOREG 11
PUSHGP
PUSHI 0
PADD
PUSHG 12
PUSHG 11
STOREN
l0e: NOP
PUSHG 12
PUSHI 1
ADD

STOREG 12
JUMP l1w
l1e: NOP
PUSHGP
PUSHI 0
PADD
PUSHI 0
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 1
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 2
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
```

```
PADD
PUSHI 3
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 4
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 5
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 6
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 7
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 8
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
```

```
PADD
PUSHI 9
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 10
LOADN
WRITEI
PUSHS
WRITES
PUSHS ""
WRITES
stop
```

4.3 Incrementar na matriz até haver igualdade

```
Arr x[4]
Mat y[2][2]

def maior()$
se (x[2] ^ y[1][1]) | x[2] se! | y[1][1] $
$

x[2] is 12
y[1][1] is 2

eqnt (call(maior) # y[1][1]) -> y[1][1] is (y[1][1] + 1)$

outputAM(x)
outputAM(y)
```

Para Assembly:

```
PUSHN 4
PUSHN 4

START
PUSHGP
PUSHI 0
PADD
PUSHI 2
PUSHI 12
STOREN
```

```
PUSHGP
PUSHI 4
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 1
ADD
PUSHI 2
STOREN
l1w: NOP
PUSHGP
PUSHI 0
PADD
PUSHI 2
LOADN
PUSHGP
PUSHI 4
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 1
ADD
LOADN
SUP
JZ l0
PUSHGP
PUSHI 0
PADD
PUSHI 2
LOADN
JUMP l0e
l0: NOP
PUSHGP
PUSHI 4
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 1
ADD
LOADN
l0e: NOP
PUSHGP
PUSHI 4
PADD
PUSHI 1
```

```
PUSHI 2
MUL
PUSHI 1
ADD
LOADN
EQUAL
NOT
JZ l1e
PUSHGP
PUSHI 4
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 1
ADD
PUSHGP
PUSHI 4
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 1
ADD
LOADN
PUSHI 1
ADD
STOREN
JUMP l1w
l1e: NOP
PUSHGP
PUSHI 0
PADD
PUSHI 0
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 1
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
```

```
PADD
PUSHI 2
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 3
LOADN
WRITEI
PUSHS
WRITES
PUSHS ""
WRITES
PUSHGP
PUSHI 4
PADD
PUSHI 0
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 4
PADD
PUSHI 1
LOADN
WRITEI
PUSHS
WRITES
PUSHS ""
WRITES
PUSHGP
PUSHI 4
PADD
PUSHI 2
LOADN
WRITEI
PUSHS
WRITES
PUSHGP
PUSHI 4
PADD
PUSHI 3
LOADN
WRITEI
```

```
PUSHS
WRITES
PUSHS ""
WRITES
STOP
```

Capítulo 5

Conclusão

Durante a realização deste trabalho utilizamos os conhecimentos sobre GIC's, YACC e Lex que fomos vindo a acimentar ao longo do semestre, sendo que o próprio trabalho nos fortaleceu nesse sentido.

Tivemos em conta os objetivos, que no fim foram alcaçados com sucesso. Obtivemos a experiência de construir uma linguagem só nossa, e poder gerar código a partir da escrita de uma gramática, dando assim a conhecer um pouco do que está por de trás de grandes linguagens de programação como C e Python.

Não deixando de lado, a oportunidade de sermos capazes de compreender o funcionamento de uma máquina virtual através de código Assembly.

Para finalizar, acabamos por dizer que este trabalho foi muito interessante e por sua vez motivador, pois foi capaz de nos cativar pela parte imaginativa da estruturação de uma linguagem, e que sem dúvida nos ajudará no percurso académico e/ou profissional.

Apêndice A

Código

A.1 Analisador Léxico

```
1
2     import ply.lex as lex
3     import sys
4
5     tokens = (
6        'PA',
7        'PF',
8        'NUM',
9        'NOME',
10       'INT',
11       'SOMA',
12       'SUB',
13       'MULT',
14       'DIV',
15       'MOD',
16       'ATR',
17       'SA',
18       'SF',
19       'OUTPUT',
20       'INPUT',
21       'MENOR',
22       'MENORIG',
23       'MAIOR',
24       'MAIORIG',
25       'IGUAL',
26       'DIF',
27       'SE',
28       'ENTAO',
29       'SENAO',
30       'ENQUANTO',
31       'FAZ',
32       'OU',
33       'E',
34       'NO',
35       'ARRAY',
36       'MATRIZ',
```

```
37          'OUTPUTAM',
38          'SUBP',
39          'DELIM',
40          'CALL',
41      )
42
43      t_PA = r'\('
44      t_PF = r'\)'
45      t_ENTAO = r'\|'
46      t_SA = r'\['
47      t_SF = r'\]'
48      t_DELIM = r'$'
49
50      def t_NUM(t):
51          r"\d+"
52          return t
53
54      def t_SUBP(t):
55          r"def"
56          return t
57
58      def t_FAZ(t):
59          r"->"
60          return t
61
62      def t_CALL(t):
63          r"call"
64          return t
65
66      def t_MATRIZ(t):
67          r"Mat"
68          return t
69
70      def t_INT(t):
71          r"Int"
72          return t
73
74      def t_OUTPUTAM(t):
75          r"outputAM"
76          return t
77
78      def t_OUTPUT(t):
79          r"output"
80          return t
81
82      def t_INPUT(t):
83          r"input"
84          return t
85
86      def t_ATR(t):
87          r"is"
88          return t
89
90      def t_SOMA(t):
```

```
91          r"\+"
92          return t
93
94      def t_SUB(t):
95          r"-"
96          return t
97
98      def t_MOD(t):
99          r"%"
100         return t
101
102     def t_MULT(t):
103         r">"
104         return t
105
106     def t_DIV(t):
107         r"/"
108         return t
109
110     def t_MENORIG(t):
111         r"\_i"
112         return t
113
114     def t_MAIORIG(t):
115         r"\^i"
116         return t
117
118     def t_MENOR(t):
119         r"\_"
120         return t
121
122     def t_MAIOR(t):
123         r"\^"
124         return t
125
126     def t_DIF(t):
127         r"\#"
128         return t
129
130     def t_IGUAL(t):
131         r"<->"
132         return t
133
134
135     def t_OU(t):
136         r"u\s"
137         return t
138
139     def t_E(t):
140         r"n\s "
141         return t
142
143     def t_NO(t):
144         r"\~"
```

```
145         return t
146 
147     def t_SENAO (t):
148         r"se!"
149         return t
150 
151     def t_SE (t):
152         r"se"
153         return t
154 
155     def t_ENQUANTO (t):
156         r"eqnt"
157         return t
158 
159     def t_ARRAY(t):
160         r"Arr"
161         return t
162 
163     def t_NOME(t):
164         r"\w+"
165         return t
166 
167     t_ignore = ' \r\t\n'
168 
169     def t_error(t):
170         print("Illegal character '%s'" % t.value[0])
171         t.lexer.skip(1)
172 
173     lexer = lex.lex()
```

A.2 Compilador

```
1  import ply.yacc as yacc
2  import re
3  import sys
4  
5  from lex import tokens
6  
7  def p_Script(p):
8      "Script : Corpo"
9      parser.assembly = f'start\n{p[1]}stop'
10 
11 def p_Script_Decls_SubPs(p):
12     "Script : Decls SubPs Corpo"
13     parser.assembly = f'{p[1]}\nstart\n{p[3]}stop'
14 
15 def p_Script_Decls(p):
16     "Script : Decls Corpo"
17     parser.assembly = f'{p[1]}\nstart\n{p[2]}stop'
18 
19 def p_Script_SubPs(p):
20     "Script : SubPs Corpo"
```

```
21      parser.assembly = f'start\n{p[2]}stop'
22
23  def p_Decls(p):
24      "Decls : Decl"
25      p[0] = f'{p[1]}'
26
27  def p_Decls_Varias(p):
28      "Decls : Decls Decl"
29      p[0] = f'{p[1]}{p[2]}'
30
31  def p_Decl_Matriz(p):
32      "Decl : MATRIZ NOME SA NUM SF SA NUM SF"
33      if p[2] not in p.parser.log:
34          p.parser.log.update({p[2] : (p.parser.gp, int(p[4]), int(p[7]))})
35          tam = int(p[4])*int(p[7])
36          p[0] = f'PUSHN {str(tam)}\n'
37          p.parser.gp += tam
38      else:
39          print("Erro! Vari vel j  existe.")
40          parser.success = False
41
42
43  def p_Decl_Array(p):
44      "Decl : ARRAY NOME SA NUM SF"
45      if p[2] not in p.parser.log:
46          p.parser.log.update({p[2] : (p.parser.gp, int(p[4]))})
47          p[0] = f'PUSHN {p[4]}\n'
48          p.parser.gp += int(p[4])
49      else:
50          print("Erro! Vari vel {p[2]} j  existe.")
51          parser.success = False
52
53
54  def p_Decl_Int(p):
55      "Decl : INT NOME"
56      if p[2] not in p.parser.log:
57          p.parser.log.update({p[2] : p.parser.gp})
58          p[0] = 'PUSHI 0\n'
59          p.parser.vars.append(p[2])
60          p.parser.gp += 1
61      else:
62          print(f"ERRO! Vari vel {p[2]} j  existe.")
63          parser.success = False
64
65  def p_Decl_Int_Atr(p):
66      "Decl : INT NOME ATR Exp"
67      if p[2] not in p.parser.log:
68          p.parser.log.update({p[2] : p.parser.gp})
69          p[0] = p[4]
70          p.parser.vars.append(p[2])
71          p.parser.gp += 1
72      else:
73          print(f"ERRO! Vari vel {p[2]} j  existe.")
74          parser.success = False
```

```
75
76 def p_Matriz_Atr(p):
77     "Atr : NOME SA Exp SF SA Exp SF ATR Exp"
78     if p[1] in p.parser.log:
79         if p[1] not in p.parser.vars and len(p.parser.log.get(p[1])) == 3:
80             c = p.parser.log.get(p[1])[2]
81             p[0] = f'PUSHGP\nPUSHI {p.parser.log.get(p[1])[0]}\nPADD\n{p[3]}
                PUSHI {c}\nMUL\n{p[6]}ADD\n{p[9]}STOREN\n'
82         else:
83             print(f"Erro: Vari vel {p[1]} n o  uma matriz.")
84             parser.success = False
85     else:
86         print("Erro: Vari vel ainda n o existe.")
87         parser.success = False
88
89 def p_Array_Atr(p):
90     "Atr : NOME SA Exp SF ATR Exp"
91     if p[1] in p.parser.log:
92         if p[1] not in p.parser.vars and len(p.parser.log.get(p[1])) == 2:
93             p[0] = f'PUSHGP\nPUSHI {p.parser.log.get(p[1])[0]}\nPADD\n{p[3]}{p
                [6]}STOREN\n'
94         else:
95             print(f"Erro: Vari vel {p[1]} n o  um array.")
96             parser.success = False
97     else:
98         print("Erro: Vari vel ainda n o existe.")
99         parser.success = False
100
101 def p_Atr_Exp(p):
102     "Atr : NOME ATR Exp"
103     if p[2] not in p.parser.log:
104         p[0] = f"{p[3]}\nSTOREG {p.parser.log.get(p[1])}\n"
105         p.parser.vars.append(p[1])
106         p.parser.gp += 1
107     else:
108         print(f"ERRO! Vari vel {p[1]} j  existe.")
109         parser.success = False
110
111 def p_Atr_Array_I(p):
112     "Atr : NOME SA Exp SF ATR INPUT"
113     if p[1] in p.parser.log:
114         if p[1] not in p.parser.vars and len(p.parser.log.get(p[1])) == 2:
115             p[0] = f'PUSHGP\nPUSHI {p.parser.log.get(p[1])[0]}\nPADD\n{p[3]}READ
                \nATOI\nSTOREN\n'
116         else:
117             print(f"Erro: Vari vel {p[1]} n o  um array.")
118             parser.success = False
119     else:
120         print("Erro: Vari vel ainda n o existe.")
121         parser.success = False
122
123
124 def p_Atr_Matriz_I(p):
125     "Atr : NOME SA Exp SF SA Exp SF ATR INPUT"
```

```
126     if p[1] in p.parser.log:
127         if p[1] not in p.parser.vars and len(p.parser.log.get(p[1])) == 3:
128             c = p.parser.log.get(p[1])[2]
129             p[0] = f'PUSHGP\nPUSHI {p.parser.log.get(p[1])[0]}\nPADD\n{p[3]}
                    PUSHI {c}\nMUL\n{p[6]}ADD\nREAD\nATOI\nSTOREN\n'
130         else:
131             print(f"Erro: Vari vel {p[1]} n o   uma matriz.")
132             parser.success = False
133     else:
134         print("Erro: Vari vel ainda n o existe.")
135         parser.success = False
136
137
138 def p_Atr_Input(p):
139     "Atr : NOME_ATR_INPUT"
140     if p[1] in p.parser.log:
141         p[0] = f'READ\nATOI\nSTOREG {p.parser.log.get(p[1])}\n'
142     else:
143         print("Erro: Vari vel n o definida.")
144         parser.success = False
145
146 def p_SubPs(p):
147     "SubPs : SubP"
148     p[0] = f'{p[1]}'
149
150 def p_SubPs_Varias(p):
151     "SubPs : SubPs SubP"
152     p[0] = f'{p[1]}{p[2]}'
153
154 def p_SubP_C(p):
155     "SubP : SUBP NOME_PA PF DELIM Corpo DELIM"
156     if p[2] not in p.parser.subP:
157         p.parser.subP.update({p[2] : f'{p[6]}'})
158     else:
159         print(f"ERRO! SubPrograma com o nome {p[2]} j  existe.")
160         parser.success = False
161
162 def p_SubP_DC(p):
163     "SubP : SUBP NOME_PA PF DELIM Decls Corpo DELIM"
164     if p[2] not in p.parser.subP:
165         p.parser.subP.update({p[2] : f'{p[6]}{p[7]}'})
166     else:
167         print(f"ERRO! SubPrograma com o nome {p[2]} j  existe.")
168         parser.success = False
169
170 def p_Corpo(p):
171     "Corpo : Proc"
172     p[0] = p[1]
173
174 def p_Corpo_Varias(p):
175     "Corpo : Corpo Proc"
176     p[0] = f'{p[1]}{p[2]}'
177
178 def p_Proc_Exp(p):
```

```
179     "Proc : Exp"
180     p[0] = p[1]
181
182 def p_Proc_Atr(p):
183     "Proc : Atr"
184     p[0] = p[1]
185
186 def p_Proc_Output(p):
187     "Proc : Output"
188     p[0] = p[1]
189
190 def p_Proc_Se(p):
191     "Proc : Se"
192     p[0] = p[1]
193
194 def p_Proc_Enquanto(p):
195     "Proc : Enquanto"
196     p[0] = p[1]
197
198 def p_Se(p):
199     "Se : SE Cond ENTAO Corpo DELIM"
200     p[0] = f'{p[2]}JZ l{p.parser.labels}\n{p[4]}l{p.parser.labels}: NOP\n'
201     p.parser.labels += 1
202
203 def p_Se_Senao(p):
204      "Se : SE Cond ENTAO Corpo SENAO ENTAO Corpo DELIM"
205      p[0] = f'{p[2]}JZ l{p.parser.labels}\n{p[4]}JUMP l{p.parser.labels}e\nl{p.
             parser.labels}: NOP\n{p[7]}l{p.parser.labels}e: NOP\n'
206      p.parser.labels += 1
207
208 def p_Enquanto(p):
209     "Enquanto : ENQUANTO Cond FAZ Corpo DELIM"
210     p[0] = f'l{p.parser.labels}w: NOP\n{p[2]}JZ l{p.parser.labels}e\n{p[4]}JUMP
            l{p.parser.labels}w\nl{p.parser.labels}e: NOP\n'
211     p.parser.labels += 1
212
213 def p_Output_Exp(p):
214     "Output : OUTPUT PA Exp PF"
215     p[0] = f'{p[3]}WRITEI\nPUSHS "\\n"\nWRITES\n'
216
217 def p_Output_AM(p):
218     "Output : OUTPUTAM PA NOME PF"
219     if p[3] in p.parser.log:
220         if p[3] not in p.parser.vars:
221             if len(p.parser.log.get(p[3])) == 2:
222                 array = ""
223                 for i in range(p.parser.log.get(p[3])[1]):
224                     array += f'PUSHGP\nPUSHI {p.parser.log.get(p[3])[0]}\nPADD\
                            nPUSHI {i}\nLOADN\nWRITEI\nPUSHS " "\nWRITES\n'
225                 p[0] = array + 'PUSHS "\\n"\nWRITES\n'
226             else:
227                 matriz = ""
228                 for l in range(p.parser.log.get(p[3])[1]):
229                     for c in range(p.parser.log.get(p[3])[2]):
```

```
230                             matriz += f'PUSHGP\nPUSHI {p.parser.log.get(p[3])[0]}\
                                    nPADD\nPUSHI {p.parser.log.get(p[3])[2] * l + c}\
                                    nLOADN\nWRITEI\nPUSHS " "\nWRITES\n'
231                         matriz += 'PUSHS "\\n"\nWRITES\n'
232                     p[0] = matriz
233
234             else:
235                 print("Erro! Vari vel n o  um array.")
236                 parser.success = False
237         else:
238             print("Erro! Vari vel n o definida.")
239             parser.success = False
240
241 def p_Exp_Var(p):
242     "Exp : Var"
243     p[0] = p[1]
244
245 def p_Exp_P(p):
246     "Exp : PA Exp PF"
247     p[0] = p[2]
248
249 def p_Exp_NUM(p):
250     "Exp : NUM"
251     p[0] = f'PUSHI {p[1]}\n'
252
253 def p_Exp_SOMA(p):
254     "Exp : Exp SOMA Exp"
255     p[0] = f'{p[1]}{p[3]}ADD\n'
256
257 def p_Exp_MULT(p):
258     "Exp : Exp MULT Exp"
259     p[0] = f'{p[1]}{p[3]}MUL\n'
260
261 def p_Exp_DIV(p):
262     "Exp : Exp DIV Exp"
263     p[0] = f'{p[1]}{p[3]}DIV\n'
264
265 def p_Exp_SUB(p):
266     "Exp : Exp SUB Exp"
267     p[0] = f'{p[1]}{p[3]}SUB\n'
268
269 def p_Exp_MOD(p):
270     "Exp : Exp MOD Exp"
271     p[0] = f'{p[1]}{p[3]}MOD\n'
272
273 def p_Exp_Cond(p):
274     "Exp : Cond"
275     p[0] = p[1]
276
277 def p_Exp_Call(p):
278     "Exp : Call"
279     p[0] = p[1]
280
281 def p_Call(p):
```

```
282     "Call : CALL PA NOME PF"
283     if p[3] in p.parser.subP:
284         p[0] = p.parser.subP[p[3]]
285     else:
286         print(f"Erro! SubPrograma com o nome {p[3]} n o existe.")
287
288 def p_Var_Matriz(p):
289     "Var : NOME SA Exp SF SA Exp SF"
290     if p[1] in p.parser.log:
291         if p[1] not in p.parser.vars and len(p.parser.log.get(p[1])) == 3:
292             c = p.parser.log.get(p[1])[2]
293             p[0] = f'PUSHGP\nPUSHI {p.parser.log.get(p[1])[0]}\nPADD\n{p[3]}
                    PUSHI {c}\nMUL\n{p[6]}ADD\nLOADN\n'
294         else:
295             print(f"Erro! Vari vel {p[1]} n o  uma matriz.")
296             parser.success = False
297     else:
298         print("Erro! Vari vel n o definida.")
299         parser.success = False
300
301 def p_Var_Array(p):
302     "Var : NOME SA Exp SF"
303     if p[1] in p.parser.log:
304         if p[1] not in p.parser.vars and len(p.parser.log.get(p[1])) == 2:
305             p[0] = f'PUSHGP\nPUSHI {p.parser.log.get(p[1])[0]}\nPADD\n{p[3]}
                    LOADN\n'
306         else:
307             print(f"Erro! Vari vel {p[1]} n o  um array.")
308             parser.success = False
309     else:
310         print("Erro! Vari vel n o definida.")
311         parser.success = False
312
313 def p_Var_Int(p):
314     "Var : NOME"
315     if p[1] in p.parser.log:
316         p[0] = f'PUSHG {p.parser.log.get(p[1])}\n'
317     else:
318         print("ERRO! Vari vel n o definida.")
319         parser.success = False
320
321 def p_Cond_P(p):
322     "Cond : PA Cond PF"
323     p[0] = p[2]
324
325 def p_Cond_Maior(p):
326     "Cond : Exp MAIOR Exp"
327     p[0] = f'{p[1]}{p[3]}SUP\n'
328
329 def p_Cond_Menor(p):
330     "Cond : Exp MENOR Exp"
331     p[0] = f'{p[1]}{p[3]}INF\n'
332
333 def p_Cond_Igual(p):
```

```
334     "Cond : Exp IGUAL Exp"
335     p[0] = f'{p[1]}{p[3]}EQUAL\n'
336
337 def p_Cond_Menori(p):
338     "Cond : Exp MENORIG Exp"
339     p[0] = f'{p[1]}{p[3]}INFEQ\n'
340
341 def p_Cond_Maiori(p):
342     "Cond : Exp MAIORIG Exp"
343     p[0] = f'{p[1]}{p[3]}SUPEQ\n'
344
345 def p_Cond_Dif(p):
346     "Cond : Exp DIF Exp"
347     p[0] = f'{p[1]}{p[3]}EQUAL\nNOT\n'
348
349 def p_Cond_E(p):
350     "Cond : Exp E Exp"
351     p[0] = f'{p[1]}{p[3]}ADD\nPUSHI 2\nEQUAL\n'
352
353 def p_Cond_Ou(p):
354     "Cond : Exp OU Exp"
355     p[0] = f'{p[1]}{p[3]}ADD\nPUSHI 1\nSUPEQ\n'
356
357 def p_Cond_Nao(p):
358     "Cond : NO Exp"
359     p[0] = f'{p[2]}NOT\n'
360
361 def p_error(p):
362     print('Syntax error: ', p)
363     parser.success = False
364
365 parser = yacc.yacc()
366
367 parser.success = True
368 parser.assembly = ""
369 parser.log = {}
370 parser.labels = 0
371 parser.gp = 0
372 parser.vars = []
373 parser.subP = {}
374
375 f = open('script.txt')
376 content = f.read()
377
378 parser.parse(content)
379
380 if parser.success:
381   print(parser.assembly)
```