

Processamento de Linguagens e Compiladores (3º Ano)

Trabalho Prático 2

Relatório de Desenvolvimento

Bruno Fernandes
(a95972)

Nelson Almeida
(a97610)

Nuno Costa
(a97610)

15/01/2023

Resumo

O trabalho prático 2, no âmbito da UC de Processamento de Linguagens e Compiladores alude-nos à criação de uma linguagem imperativa a nosso gosto bem como a criação de um compilador usando os módulos de gramáticas tradutoras do Python.

Além disso, esta gramática tem de ser capaz de gerar código assembly a partir da linguagem imperativa, com recurso a algumas ferramentas como o lex e o yacc do Python.

Assim sendo, durante o realizar do relatório tentaremos sempre explicar de uma forma clara e sucinta todas as decisões tomadas por nós, bem como as produções implementadas na gramática e ainda como foi desenvolvido o compilador.

Conteúdo

1	Introdução	3
2	Problema Proposto	5
3	Concepção da Resolução	6
3.1	Organização e estrutura	6
3.2	GIC	6
3.3	Lexer	8
3.4	Parser e geração do código Assembly da VM	9
3.4.1	Algumas notas sobre declaração de variáveis	9
4	Demonstração do Funcionamento	11
4.1	Geração e execução de código Assembly	11
4.2	Teste 1	11
4.2.1	Conteúdo do ficheiro	11
4.2.2	Código assembly gerado	12
4.2.3	Execução da VM com o código gerado	12
4.3	Teste 2	12
4.3.1	Conteúdo do ficheiro	13
4.3.2	Código assembly gerado	13
4.3.3	Código gerado pela VM	14
4.4	Teste 3	14
4.4.1	Conteúdo do ficheiro	14
4.4.2	Código assembly gerado	14
4.4.3	Código gerado pela VM	16
4.5	Teste 4	16
4.5.1	Conteúdo do ficheiro	16
4.5.2	Código assembly gerado	16
4.5.3	Código gerado pela VM	17
4.6	Teste 5	17
4.6.1	Conteúdo do ficheiro	17
4.6.2	Código assembly gerado	17
4.6.3	Execução da VM com o código gerado	21

4.7	Teste 6	21
4.7.1	Conteúdo do ficheiro	21
4.7.2	Código assembly gerado	22
4.7.3	Código gerado pela VM	24
5	Conclusão	25
A	Código do Programa	26

Capítulo 1

Introdução

No âmbito da disciplina de Processamento de Linguagens e Compiladores foi-nos proposto pelo docente Pedro Rangel Henriques um trabalho de grupo cujos objetivos principais são: tornar-nos capazes de escrever gramáticas com uma maior facilidade, sermos capazes de desenvolver um processador de linguagens a partir de uma gramática tradutora e ainda de desenvolver um compilador produzindo código para uma máquina de stack virtual.

A linguagem a ser usada na realização deste projeto, será uma linguagem imperativa simples com regras definidas pelo grupo.

O compilador desenvolvido para a nossa linguagem terá de gerar pseudo-código Assembly para uma VM, com base na gramática independente do contexto(GIC) que definimos.

Neste documento apresentamos a nossa resolução para cada um dos problemas propostos, com recurso aos módulos 'Yacc/Lex' do 'PLY/Python'.

Estrutura do Relatório

O relatório está organizado da seguinte forma:

Começamos por fazer uma pequena introdução, capítulo 1, onde referimos o objetivo do trabalho a desenvolver.

No capítulo 2 apresentamos o enunciado dos problema proposto.

O capítulo 3 demonstramos como está organizado o nosso trabalho.

No capítulo seguinte, demonstramos o funcionamento de vários testes realizados pelo grupo.

E, por fim, o ultimo capítulo contém a conclusão do trabalho realizado.

Capítulo 2

Problema Proposto

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 atômicas 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 de seleção para o controlo do fluxo de execução.
- efetuar instruções de repetição(cíclicas) para o controlo de 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.

Capítulo 3

Concepção da Resolução

3.1 Organização e estrutura

O nosso trabalho pode ser dividido em 4 partes:

- Construção da **GIC** que define a estrutura sintática da nossa linguagem.
- Construção do analisador léxico, **lexer**.
- Construção do analisador sintático, **parser**.
- Conversão das instruções para código **Assembly** da VM.

Todas as funcionalidades descritas neste capítulo podem ser encontradas no anexo A do documento.

3.2 GIC

A nossa linguagem é gerada pela seguinte gramática independente de contexto:

```
Programa : Decls
          | Corpo
          | Decls Corpo

Decls    : Decl
          | Decl Decls

Decl     : VAR ID
          | LISTA ID
          | LISTA ID INT
          | LISTA ID COM lista
          | MATRIZ ID
          | MATRIZ ID INT INT

Atrib    : VAR ID COM expr
```



```

| ALTERNA ID ABREPR expr FECHAPR COM expr
| ALTERNA ID COM expr
| VAR ID COM entradas
| ALTERNA ID COM entradas

Corpo      : Proc
            | Proc Corpo
            | Atrib
            | Atrib Corpo

expr       : INT
            | ID
            | exprArit
            | exprRel

lista      : ABREPR elems FECHAPR

elems      : INT
            | INT VIRG elems

exprArit   : expr SOMA expr
            | expr MENUS expr
            | expr SOMANBEZES expr
            | expr DIBIDE expr
            | expr SOBRAS expr

exprRel    : NOUM ABREPC expr FECHAPC
            | GEMEO ABREPC expr VIRG expr FECHAPC
            | NAOGEMEO ABREPC expr VIRG expr FECHAPC
            | MAISPIQUENO ABREPC expr VIRG expr FECHAPC
            | MAISPIQUENOOUGEMEO ABREPC expr VIRG expr FECHAPC
            | MAISGRANDE ABREPC expr VIRG expr FECHAPC
            | MAISGRANDEOUGEMEO ABREPC expr VIRG expr FECHAPC
            | expr IE expr
            | expr OUE expr

Proc       : if
            | while
            | saidas

if         : SE exprRel LOGO Corpo FIM
            | SE exprRel LOGO Corpo SENAO Corpo FIM

while      : ENQUANTO exprRel FAZ Corpo FIM

saidas     : SAIDAS ID
            | SAIDAS expr

```

3.3 Lexer

O analisador léxico, **lexer**, é o responsável por 'capturar' os símbolos terminais(*tokens*) da nossa linguagem através de expressões regulares. Para a implementação do analisador léxico utilizamos o módulo 'Lex' do 'PLY/Python'.

Os *tokens* e respectivas expressões regulares da nossa linguagem são os seguintes:

```
ABRECHAV : '\{'
FECHACHAV : '\}'
ABREPC : '\('
FECHAPC : '\)'
ABREPR : '\['
FECHAPR : '\]'
VIRG : '\,'
SOMA : '\+'
MENUS : '\-'
SOMANBEZES : '\*'
DIBIDE : '\/'
SOBRAS : '\%'
STRING : '\"w+\"'|\'w+\''
ID : r'\w+'
INT : '\d+'
VAR : 'var'
COM : 'com'
MAISGRANDE : 'maisGrande'
MAISPIQUENO : 'maisPiqueno'
GEMEO : 'gemeo'
NAOGEMEO : 'naogemeo'
MAISGRANDEOUGEMEO : 'maisGrandeOuGemeo'
MAISPIQUENOOUUGEMEO : 'maisPiquenoOuGemeo'
IE : 'ie'
OUE : 'oue'
NOUM : 'noum'
ALTERNA : 'alterna'
LISTA : 'lista'
MATRIZ : 'matriz'
BUSCA : 'busca'
SWAP : 'swap'
SENAO : 'senao'
SE : 'se'
ENTAO : 'entao'
FIM : 'fim'
ENQUANTO : 'enquanto'
FAZ : 'faz'
ENTRADAS : 'entradas'
```

A implementação do analisador léxico pode ser encontrada no anexo A do documento.

3.4 Parser e geração do código Assembly da VM

O analisador sintático, **parser**, é o responsável por verificar se o código escrito na nossa linguagem está correto sintaticamente, isto é, se o código respeita as regras gramaticais definidas.

No caso de não existirem erros sintáticos o **parser** converte o código da nossa linguagem em código **Assembly** da máquina virtual. caso existam erros, então será mostrado ao utilizador uma mensagem do erro sintático produzido.

A implementação do analisador sintático pode ser encontrada no anexo A do nosso relatório.

3.4.1 Algumas notas sobre declaração de variáveis

Na geração do código para declarar uma variavel sem valor fazemos:

```
1 PUSHI 0
```

Sendo a variável predefinida a 0.

Para declarar uma variavel com valor fazemos:

```
1 PUSHI <valor>
2 STOREG <endereco>
```

Para declarar uma lista de valores temos de fazer sempre:

```
1 PUSHN <tamanho>
```

Inicializando todos os valores da lista a 0.

Seja a uma lista de tamanho 3, por exemplo:

```
1 lista a tamanho
2 lista a com [valor1,valor2,valor3]
```

Para atribuir valores à lista fazemos:

```
1 PUSHN <tamanho>
2 PUSHGP
3 PUSHI 0
4 PUSHI <valor1>
5 STOREN
6 PUSHGP
7 PUSHI 1
8 PUSHI <valor2>
9 STOREN
10 PUSHGP
11 PUSHI 2
12 PUSHI <valor3>
13 STOREN
```

No caso das matrizes, para declarar uma matriz fazemos:

```
1 matriz <nomeDaMatriz> <tamanho1> <tamanho2>
```

Caso a matriz m seja de tamanho 2x2, então gera-se:

```
1 PISHN 4
```

Inicializando todos os valores a 0.

Para alterar os valores de uma matriz temos duas maneiras de o fazer:

1. Alteramos uma posição em específico:

```
1 alterna <nomeDaMatriz> [<indice1>] [<indice2>] com <valor>
```

2. Alteramos uma linha da matriz passando uma lista:

```
1 alterna <nomeDaMatriz> [<indice1>] com [valor1,valor2]
```

Tomando como exemplo uma matriz 2x2, ao fazer a alteração de valores pela 1ª opção:

```
1 alterna m [0][1] com <valor>
```

Gera-se o seguinte:

```
1 PUSHN 4
2 PUSHGP
3 PUSHI 0
4 PADD
5 PUSHI 0
6 PUSHI <valor>
7 STOREN
```

Fazendo o mesmo mas de acordo com a 2ª opção:

```
1 alterna m [0] com [valor1,valor2]
```

Gera-se o seguinte:

```
1 PUSHN 4
2 PUSHGP
3 PUSHI 0
4 PADD
5 PUSHI 0
6 PUSHI 2
7 MUL
8 PADD
9 PUSHI 0
10 PUSHI <valor1>
11 STOREN
12 PUSHGP
13 PUSHI 0
14 PADD
15 PUSHI 0
16 PUSHI 2
17 MUL
18 PADD
19 PUSHI 1
20 PUSHI <valor2>
21 STOREN
```

Capítulo 4

Demonstração do Funcionamento

4.1 Geração e execução de código Assembly

Para utilizar a nossa linguagem, o utilizador tem 3 opções:

1. Escrever instruções de acordo com as regras gramaticais da linguagem.

```
>> python3 yacc.py
```

2. Escrever e guardar as instruções num ficheiro .plo de acordo com as regras gramaticais da linguagem.

```
>> python3 yacc.py <ficheiro de input>
```

3. Escrever e guardar as instruções num ficheiro .plo de acordo com as regras gramaticais da linguagem e escolher o ficheiro de saída.

```
>> python3 yacc.py <ficheiro de input> <ficheiro de output>
```

Por exemplo:

```
>> python3 yacc.py .\testes\factorial.plo output.vm
```

Nota: Caso o utilizador escolha fazer a opção 1 ou 2 é criado um ficheiro "a.vm" onde será guardado o código Assembly gerado.

4.2 Teste 1

Calcula o fatorial de um número passado como input.

Ficheiro de input: 'factorial.plo'

4.2.1 Conteúdo do ficheiro

```
1 saidas "Factorial: "  
2 var n com entradas  
3 saidas n  
4 var res com 1
```

```

5
6 enquanto (maisGrande(n,0)) faz {
7     alterna res com res * n
8     alterna n com n - 1
9 } fim
10
11 saidas "\nResultado: "
12 saidas res

```

4.2.2 Código assembly gerado

```

1 START
2 PUSHHS "Factorial: "
3 WRITES
4 READ
5 ATOI
6 STOREG 0
7 PUSHG 0
8 WRITEI
9 PUSHI 1
10 STOREG 1
11 loc: NOP
12 PUSHG 0
13 PUSHI 0
14 SUP
15 JZ 10f
16 PUSHG 1
17 PUSHG 0
18 MUL
19 STOREG 1
20 PUSHG 0
21 PUSHI 1
22 SUB
23 STOREG 0
24 JUMP 10c
25 10f: NOP
26 PUSHHS "\nResultado: "
27 WRITES
28 PUSHG 1
29 WRITEI
30 STOP

```

4.2.3 Execução da VM com o código gerado

```

1 Factorial: 5
2 Resultado: 120

```

4.3 Teste 2

Procura determinado número pelo seu índice.
Ficheiro de input: 'busca_no_array.plo'.

4.3.1 Conteúdo do ficheiro

```
1 lista a 10
2 lista a com [1,2,3,4,5,6,7,8,9,10]
3
4 saidas "Introduza um indice do array:\n"
5 var i com entradas
6
7 var x com busca a[i]
8
9 saidas "Valor: "
10 saidas x
```

4.3.2 Código assembly gerado

```
1 PUSHN 10
2 START
3 PUSHGP
4 PUSHI 0
5 PUSHI 1
6 STOREN
7 PUSHGP
8 PUSHI 1
9 PUSHI 2
10 STOREN
11 PUSHGP
12 PUSHI 2
13 PUSHI 3
14 STOREN
15 PUSHGP
16 PUSHI 3
17 PUSHI 4
18 STOREN
19 PUSHGP
20 PUSHI 4
21 PUSHI 5
22 STOREN
23 PUSHGP
24 PUSHI 5
25 PUSHI 6
26 STOREN
27 PUSHGP
28 PUSHI 6
29 PUSHI 7
30 STOREN
31 PUSHGP
32 PUSHI 7
33 PUSHI 8
34 STOREN
35 PUSHGP
36 PUSHI 8
37 PUSHI 9
38 STOREN
39 PUSHGP
40 PUSHI 9
41 PUSHI 10
42 STOREN
43 PUSHHS "Introduza um indice do array:\n"
```

```

44 WRITES
45 READ
46 ATOI
47 STOREG 10
48 PUSHGP
49 PUSHI 0
50 PADD
51 PUSHG 10
52 LOADN
53 STOREG 11
54 PUSHG "Valor: "
55 WRITES
56 PUSHG 11
57 WRITEI
58 STOP

```

4.3.3 Código gerado pela VM

```

1 Introduza um índice do array: 3
2 Valor: 4

```

4.4 Teste 3

Lê os 5 valores de um array passados como input.
Ficheiro de input: 'read_array.plo'.

4.4.1 Conteúdo do ficheiro

```

1 var n com 5
2 var i com 0
3 lista a 5
4
5 enquanto (maisPiqueno(i,n)) faz {
6     alterna a [i] com entradas
7     alterna i com i + 1
8 } fim
9
10 saidas "Array gerado:\n"
11 saidas a

```

4.4.2 Código assembly gerado

```

1 PUSHI 5
2 STOREG 0
3 START
4 PUSHI 0
5 STOREG 1
6 PUSHN 5
7 10c: NOP
8 PUSHG 1
9 PUSHG 0
10 INF
11 JZ 10f

```



```

12 PUSHGP
13 PUSHI 2
14 PADD
15 PUSHG 1
16 READ
17 ATOI
18 STOREN
19 PUSHG 1
20 PUSHI 1
21 ADD
22 STOREG 1
23 JUMP 10c
24 10f: NOP
25 PUSHHS "Array gerado:\n"
26 WRITES
27 PUSHHS "["
28 WRITES
29 PUSHGP
30 PUSHI 2
31 PADD
32 PUSHI 0
33 LOADN
34 WRITEI
35 PUSHHS ", "
36 WRITES
37 PUSHGP
38 PUSHI 2
39 PADD
40 PUSHI 1
41 LOADN
42 WRITEI
43 PUSHHS ", "
44 WRITES
45 PUSHGP
46 PUSHI 2
47 PADD
48 PUSHI 2
49 LOADN
50 WRITEI
51 PUSHHS ", "
52 WRITES
53 PUSHGP
54 PUSHI 2
55 PADD
56 PUSHI 3
57 LOADN
58 WRITEI
59 PUSHHS ", "
60 WRITES
61 PUSHGP
62 PUSHI 2
63 PADD
64 PUSHI 4
65 LOADN
66 WRITEI
67 PUSHHS "]"
68 WRITES
69 STOP

```

4.4.3 Código gerado pela VM

```
1 Array gerado:
2 [1,2,3,4,5]
```

4.5 Teste 4

Realiza o produto de vários números passados como input.
Ficheiro de input: 'produtorio.plo'.

4.5.1 Conteúdo do ficheiro

```
1 saidas "Quantos numeros? "
2 var n com entradas
3 saidas n
4 var res com 1
5 var x com 1
6
7 enquanto (maisGrande(n,0)) faz {
8     alterna x com entradas
9     alterna res com res * x
10    alterna n com n - 1
11 } fim
12
13 saidas "\nResultado: "
14 saidas res
```

4.5.2 Código assembly gerado

```
1 START
2 PUSHES "Quantos numeros? "
3 WRITES
4 READ
5 ATOI
6 STOREG 0
7 PUSHG 0
8 WRITEI
9 PUSHI 1
10 STOREG 1
11 PUSHI 1
12 STOREG 2
13 loc: NOP
14 PUSHG 0
15 PUSHI 0
16 SUP
17 JZ 10f
18 READ
19 ATOI
20 STOREG 2
21 PUSHG 1
22 PUSHG 2
23 MUL
24 STOREG 1
25 PUSHG 0
```

```

26 PUSHI 1
27 SUB
28 STOREG 0
29 JUMP 10c
30 10f: NOP
31 PUSHHS "\nResultado: "
32 WRITES
33 PUSHG 1
34 WRITEI
35 STOP

```

4.5.3 Código gerado pela VM

```

1 Quantos numeros? 5
2 Resultado: 120

```

4.6 Teste 5

A partir de 3 arrays de tamanho 3, crai uma matriz de tamanho 3x3.
Ficheiro de input: 'matriz.plo'.

4.6.1 Conteúdo do ficheiro

```

1 matriz m 3 3
2
3 alterna m [0] com [1,2,3]
4 alterna m [1] com [4,5,6]
5 alterna m [2] com [7,8,9]
6
7 saidas m

```

4.6.2 Código assembly gerado

```

1 PUSHN 9
2 START
3 PUSHGP
4 PUSHI 0
5 PADD
6 PUSHI 0
7 PUSHI 3
8 MUL
9 PADD
10 PUSHI 0
11 PUSHI 1
12 STOREN
13 PUSHGP
14 PUSHI 0
15 PADD
16 PUSHI 0
17 PUSHI 3
18 MUL
19 PADD
20 PUSHI 1

```

```
21 PUSHI 2
22 STOREN
23 PUSHGP
24 PUSHI 0
25 PADD
26 PUSHI 0
27 PUSHI 3
28 MUL
29 PADD
30 PUSHI 2
31 PUSHI 3
32 STOREN
33 PUSHGP
34 PUSHI 0
35 PADD
36 PUSHI 1
37 PUSHI 3
38 MUL
39 PADD
40 PUSHI 0
41 PUSHI 4
42 STOREN
43 PUSHGP
44 PUSHI 0
45 PADD
46 PUSHI 1
47 PUSHI 3
48 MUL
49 PADD
50 PUSHI 1
51 PUSHI 5
52 STOREN
53 PUSHGP
54 PUSHI 0
55 PADD
56 PUSHI 1
57 PUSHI 3
58 MUL
59 PADD
60 PUSHI 2
61 PUSHI 6
62 STOREN
63 PUSHGP
64 PUSHI 0
65 PADD
66 PUSHI 2
67 PUSHI 3
68 MUL
69 PADD
70 PUSHI 0
71 PUSHI 7
72 STOREN
73 PUSHGP
74 PUSHI 0
75 PADD
76 PUSHI 2
77 PUSHI 3
78 MUL
79 PADD
```

```

80 PUSHI 1
81 PUSHI 8
82 STOREN
83 PUSHGP
84 PUSHI 0
85 PADD
86 PUSHI 2
87 PUSHI 3
88 MUL
89 PADD
90 PUSHI 2
91 PUSHI 9
92 STOREN
93 PUSHHS "["
94 WRITES
95 PUSHHS "["
96 WRITES
97 PUSHGP
98 PUSHI 0
99 PADD
100 PUSHGP
101 PUSHI 0
102 PUSHI 3
103 MUL
104 PADD
105 PUSHI 0
106 LOADN
107 WRITEI
108 POP 1
109 PUSHHS ", "
110 WRITES
111 PUSHGP
112 PUSHI 0
113 PADD
114 PUSHGP
115 PUSHI 0
116 PUSHI 3
117 MUL
118 PADD
119 PUSHI 1
120 LOADN
121 WRITEI
122 POP 1
123 PUSHHS ", "
124 WRITES
125 PUSHGP
126 PUSHI 0
127 PADD
128 PUSHGP
129 PUSHI 0
130 PUSHI 3
131 MUL
132 PADD
133 PUSHI 2
134 LOADN
135 WRITEI
136 POP 1
137 PUSHHS "]"
138 WRITES

```

```
139 PUSHHS ",",  
140 WRITES  
141 PUSHHS "["  
142 WRITES  
143 PUSHGP  
144 PUSHI 0  
145 PADD  
146 PUSHGP  
147 PUSHI 1  
148 PUSHI 3  
149 MUL  
150 PADD  
151 PUSHI 0  
152 LOADN  
153 WRITEI  
154 POP 1  
155 PUSHHS ",",  
156 WRITES  
157 PUSHGP  
158 PUSHI 0  
159 PADD  
160 PUSHGP  
161 PUSHI 1  
162 PUSHI 3  
163 MUL  
164 PADD  
165 PUSHI 1  
166 LOADN  
167 WRITEI  
168 POP 1  
169 PUSHHS ",",  
170 WRITES  
171 PUSHGP  
172 PUSHI 0  
173 PADD  
174 PUSHGP  
175 PUSHI 1  
176 PUSHI 3  
177 MUL  
178 PADD  
179 PUSHI 2  
180 LOADN  
181 WRITEI  
182 POP 1  
183 PUSHHS "]"  
184 WRITES  
185 PUSHHS ",",  
186 WRITES  
187 PUSHHS "["  
188 WRITES  
189 PUSHGP  
190 PUSHI 0  
191 PADD  
192 PUSHGP  
193 PUSHI 2  
194 PUSHI 3  
195 MUL  
196 PADD  
197 PUSHI 0
```

```

198 LOADN
199 WRITEI
200 POP 1
201 PUSHHS ",", "
202 WRITES
203 PUSHGP
204 PUSHI 0
205 PADD
206 PUSHGP
207 PUSHI 2
208 PUSHI 3
209 MUL
210 PADD
211 PUSHI 1
212 LOADN
213 WRITEI
214 POP 1
215 PUSHHS ",", "
216 WRITES
217 PUSHGP
218 PUSHI 0
219 PADD
220 PUSHGP
221 PUSHI 2
222 PUSHI 3
223 MUL
224 PADD
225 PUSHI 2
226 LOADN
227 WRITEI
228 POP 1
229 PUSHHS "]" "
230 WRITES
231 PUSHHS "]" "
232 WRITES
233 STOP

```

4.6.3 Execução da VM com o código gerado

```

1 [[1,2,3],[4,5,6],[7,8,9]]

```

4.7 Teste 6

Troca a posição de um certo valor do array por um outro, tendo em conta os índices .
 Ficheiro de input: 'swap_array.plo'.

4.7.1 Conteúdo do ficheiro

```

1 lista a 5
2 lista a com [1,2,3,4,5]
3
4 saidas "Array inicial:\n"
5 saidas a
6

```

```

7  saidas "\nTroca do indice 1 com indice 3."
8
9  swap a [1] com [3]
10
11 saidas "Array inicial:\n"
12 saidas a

```

4.7.2 Código assembly gerado

```

1  PUSHN 5
2  START
3  PUSHGP
4  PUSHI 0
5  PUSHI 1
6  STOREN
7  PUSHGP
8  PUSHI 1
9  PUSHI 2
10 STOREN
11 PUSHGP
12 PUSHI 2
13 PUSHI 3
14 STOREN
15 PUSHGP
16 PUSHI 3
17 PUSHI 4
18 STOREN
19 PUSHGP
20 PUSHI 4
21 PUSHI 5
22 STOREN
23 PUSHHS "Array inicial:\n"
24 WRITES
25 PUSHHS "["
26 WRITES
27 PUSHGP
28 PUSHI 0
29 PADD
30 PUSHI 0
31 LOADN
32 WRITEI
33 PUSHHS ", "
34 WRITES
35 PUSHGP
36 PUSHI 0
37 PADD
38 PUSHI 1
39 LOADN
40 WRITEI
41 PUSHHS ", "
42 WRITES
43 PUSHGP
44 PUSHI 0
45 PADD
46 PUSHI 2
47 LOADN
48 WRITEI
49 PUSHHS ", "

```



```

50 WRITES
51 PUSHGP
52 PUSHI 0
53 PADD
54 PUSHI 3
55 LOADN
56 WRITEI
57 PUSHS ","
58 WRITES
59 PUSHGP
60 PUSHI 0
61 PADD
62 PUSHI 4
63 LOADN
64 WRITEI
65 PUSHS "]"
66 WRITES
67 PUSHS "\nTroca do indice 1 com indice 3."
68 WRITES
69 PUSHG 1
70 PUSHG 3
71 STOREG 1
72 STOREG 3
73 PUSHS "Array inicial:\n"
74 WRITES
75 PUSHS "["
76 WRITES
77 PUSHGP
78 PUSHI 0
79 PADD
80 PUSHI 0
81 LOADN
82 WRITEI
83 PUSHS ","
84 WRITES
85 PUSHGP
86 PUSHI 0
87 PADD
88 PUSHI 1
89 LOADN
90 WRITEI
91 PUSHS ","
92 WRITES
93 PUSHGP
94 PUSHI 0
95 PADD
96 PUSHI 2
97 LOADN
98 WRITEI
99 PUSHS ","
100 WRITES
101 PUSHGP
102 PUSHI 0
103 PADD
104 PUSHI 3
105 LOADN
106 WRITEI
107 PUSHS ","
108 WRITES

```

```
109 PUSHGP
110 PUSHI 0
111 PADD
112 PUSHI 4
113 LOADN
114 WRITEI
115 PUSHs "]"
116 WRITES
117 STOP
```

4.7.3 Código gerado pela VM

```
1 Array inicial:
2 [1,2,3,4,5]
3 Troca do indice 1 com indice 3.Array inicial:
4 [1,4,3,2,5]
```

Capítulo 5

Conclusão

No decorrer deste trabalho, tentamos sempre aplicar todo e qualquer conhecimento adquirido em aulas, o que nos permitiu aprofundar e consolidar melhor a matéria lecionada nesta UC.

Consideramos que, no geral, conseguimos alcançar os objetivos esperados e desta forma temos mais bagagem no que toca à escrita de gramáticas e no desenvolvimento de compiladores de linguagens. Este trabalho levou-nos também a obter um maior conhecimento no que diz respeito à máquina virtual e a uma maior compreensão da escrita em Assembly.

Em suma, todo o trabalho aplicado na realização deste projeto foi bastante útil para consolidar as nossas bases e dar-nos também alguma naturalidade na abordagem de certas temáticas da UC que poderão vir a ser necessárias no nosso futuro profissional.

Apêndice A

Código do Programa

Ficheiro lex2.py

```
1 import ply.lex as lex
2
3 tokens = [
4     "ASPA",
5     "ID",
6     "VAR",
7     "COM",
8
9     "ABREPC",
10    "FECHAPC",
11    "ABREPR",
12    "FECHAPR",
13    "ABREHAV",
14    "FECHACHAV",
15    "VIRG",
16
17    "INT",
18
19    'SOMA',
20    'MENUS',
21    'SOMANBEZES',
22    'DIBIDE',
23    'SOBRAS',
24
25    'MAISGRANDE',
26    'MAISPIQUENO',
27    'GEMEO',
28    'NAOGEMEO',
29    'MAISGRANDEOUGEMEO',
30    'MAISPIQUENOUGEMEO',
31
32    'IE',
33    'OUE',
34    'NOUM',
35
36    "ALTERNA",
37
38    "LISTA",
39    "MATRIZ",
40    "BUSCA",
41    "SWAP",
```

```

42
43     "SENAO",
44     "SE",
45     "ENTAO",
46     "FIM",
47
48     "ENQUANTO",
49     "FAZ",
50
51     "ENTRADAS",
52     "SAIDAS"
53 ]
54
55 t_ignore = ' \r\n\t'
56
57 t_ABRECHAV = r'\{'
58 t_FECHACHAV = r'\}'
59 t_ABREPC = r'\('
60 t_FECHAPC = r'\)'
61 t_ABREPR = r'\['
62 t_FECHAPR = r'\]'
63 t_VIRG = r'\,'
64 t_SOMA = r'\+'
65 t_MENUS = r'\-'
66 t_SOMANBEZES = r'\*'
67 t_DIBIDE = r'\/'
68 t_SOBRAS = r'\%'
69
70
71 t_ASPA= r'\".*\\"'
72 t_ID = r"\w+"
73
74 def t_INT(t):
75     r'\d+'
76     t.type = "INT"
77     return t
78
79
80 def t_COMENTARIO(t):
81     r'comentario'
82     t.type = "COMENTARIO"
83     return t
84
85
86 def t_VAR(t):
87     r'var'
88     t.type = "VAR"
89     return t
90
91
92 def t_COM(t):
93     r'com'
94     t.type = "COM"
95     return t
96
97
98 def t_ALTERNA(t):
99     r'alterna'
100    t.type = "ALTERNA"

```

```

101     return t
102
103
104 def t_MAISGRANDE(t):
105     r"maisGrande"
106     t.type = "MAISGRANDE"
107     return t
108
109
110 def t_MAISPIQUENO(t):
111     r"maisPiqueno"
112     t.type = "MAISPIQUENO"
113     return t
114
115
116 def t_NAOGEMEO(t):
117     r"naogemeo"
118     t.type = "NAOGEMEO"
119     return t
120
121
122 def t_GEMEO(t):
123     r"gemeo"
124     t.type = "GEMEO"
125     return t
126
127
128 def t_MAISGRANDEOUGEMEO(t):
129     r"maisGrandeOuGemeo"
130     t.type = "MAISGRANDEOUGEMEO"
131     return t
132
133
134 def t_MAISPIQUENOUGEMEO(t):
135     r"maisPiquenoOuGemeo"
136     t.type = "MAISPIQUENOUGEMEO"
137     return t
138
139
140 def t_IE(t):
141     r"ie"
142     t.type = "IE"
143     return t
144
145
146 def t_OUE(t):
147     r"oue"
148     t.type = "OUE"
149     return t
150
151
152 def t_NOUM(t):
153     r"noum"
154     t.type = "NOUM"
155     return t
156
157
158 def t_LISTA(t):
159     r'lista'

```

```

160     t.type = "LISTA"
161     return t
162
163
164 def t_MATRIZ(t):
165     r'matriz'
166     t.type = "MATRIZ"
167     return t
168
169
170 def t_BUSCA(t):
171     r'busca'
172     t.type = "BUSCA"
173     return t
174
175
176 def t_SWAP(t):
177     r'swap'
178     t.type = "SWAP"
179     return t
180
181
182 def t_SENAO(t):
183     r'senao'
184     t.type = "SENAO"
185     return t
186
187
188 def t_SE(t):
189     r'se'
190     t.type = "SE"
191     return t
192
193
194 def t_ENTAO(t):
195     r'entao'
196     t.type = "ENTAO"
197     return t
198
199
200 def t_ENQUANTO(t):
201     r'enquanto'
202     t.type = "ENQUANTO"
203     return t
204
205
206 def t_FAZ(t):
207     r'faz'
208     t.type = "FAZ"
209     return t
210
211
212 def t_FIM(t):
213     r'fim'
214     t.type = "FIM"
215     return t
216
217
218 def t_ENTRADAS(t):

```

```

219     r"entradas"
220     t.type = "ENTRADAS"
221     return t
222
223
224 def t_SAIDAS(t):
225     r"saidas"
226     t.type = "SAIDAS"
227     return t
228
229
230 def t_error(t):
231     print('Illegal character: ' + t.value[0])
232     t.lexer.skip(1)
233     return
234
235
236 lexer = lex.lex()

```

Ficheiro yacc3.py

```

1  import ply.yacc as yacc
2  import random as rd
3  import sys
4  import os
5  import difflib
6  import random as rd
7
8  from lex import *
9
10
11 def p_Programa_Empty(p):
12     '''
13     Programa : Decls
14               | Atrib
15     '''
16     parser.assembly = f'{p[1]}'
17
18
19 def p_Programa(p):
20     '''
21     Programa : Decls Corpo
22               | Atrib Corpo
23     '''
24     parser.assembly = f'{p[1]}START\n{p[2]}STOP\n'
25
26
27 def p_Programa_Corpo(p):
28     '''
29     Programa : Corpo
30     '''
31     parser.assembly = f"START\n{p[1]}STOP\n"
32
33
34 def p_Corpo(p):
35     '''
36     Corpo :Codigo
37     '''
38     p[0] = f"{p[1]}"
39

```



```

40
41 def p_Codigo_Rec(p):
42     '''
43     Codigo : Proc Codigo
44             | Atrib Codigo
45     '''
46     p[0] = f"{p[1]}{p[2]}"
47
48
49 def p_Codigo(p):
50     '''
51     Codigo : Proc
52             | Atrib
53     '''
54     p[0] = f"{p[1]}"
55
56
57 def p_Decls(p):
58     "Decls : Decl"
59     p[0] = f'"{p[1]}"'
60
61
62 def p_DeclsRec(p):
63     "Decls : Decl Decls"
64     p[0] = f'"{p[1]}{p[2]}"'
65
66
67 def p_expr_arit(p):
68     '''
69     expr : exprArit
70           | exprRel
71     '''
72     p[0] = p[1]
73
74
75 def p_Proc(p):
76     '''
77     Proc : if
78           | while
79           | saidas
80     '''
81     p[0] = p[1]
82
83
84 # Declara o de uma variavel sem valor
85 def p_Decl(p):
86     "Decl : VAR ID"
87     varName = p[2]
88     if varName not in parser.variaveis:
89         parser.variaveis[varName] = (parser.stackPointer, None)
90         p[0] = "PUSHI 0\n"
91         parser.stackPointer += 1
92     else:
93         parser.exito = False
94         parser.error = f"Vari vel com o nome {varName} j existe"
95         parser.linhaDeCodigo +=1
96
97
98 # Declara o de uma vari vel com atribui o de um valor

```

```

99 def p_Atrib_expr(p):
100     "Atrib : VAR ID COM expr"
101     varName = p[2]
102     if varName not in parser.variaveis:
103         value = p[4]
104         parser.variaveis[varName] = (parser.stackPointer, None)
105         p[0] = f"{value}STOREG {parser.stackPointer}\n"
106         parser.stackPointer += 1
107     else:
108         parser.exito = False
109         parser.error = f"Vari vel com o nome {varName} j existe"
110     parser.linhaDeCodigo +=1
111
112
113 # Altera valor de um vari vel
114 def p_alterna_var(p):
115     "Atrib : ALTERNA ID COM expr"
116     varName = p[2]
117     if varName in parser.variaveis:
118         p[0] = f"{p[4]}STOREG {parser.variaveis[varName][0]}\n"
119     parser.linhaDeCodigo +=1
120
121
122 def p_expr(p):
123     "expr : INT"
124     p[0] = f"PUSHI {int(p[1])}\n"
125
126
127 def p_expr_var(p):
128     "expr : ID"
129     varName = p[1]
130     if varName in parser.variaveis:
131         p[0] = f"PUSHG {parser.variaveis[varName][0]}\n"
132
133
134 def p_expr_entradas(p):
135     "expr : ENTRADAS"
136     p[0] = f"READ\nATOI\n"
137
138
139 # Declara lista sem tamanho
140 def p_Decl_Lista_NoSize(p):
141     "Atrib : LISTA ID"
142     listName = p[2]
143     if listName not in parser.variaveis:
144         parser.variaveis[listName] = (parser.stackPointer, 0)
145         p[0] = f"PUSHN 0\n"
146         parser.stackPointer += 1
147     else:
148         parser.error = (
149             f"Vari vel com o nome {listName} j definida anteriormente.")
150         parser.exito = False
151     parser.linhaDeCodigo +=1
152
153
154 # Declara lista com tamanho INT
155 def p_DeclLista_Size(p):
156     "Atrib : LISTA ID INT"
157     listName = p[2]

```

```

158 size = int(p[3])
159 if listName not in parser.variaveis:
160     if size > 0:
161         parser.variaveis[listName] = (parser.stackPointer, size)
162         p[0] = f"PUSHN {size}\n"
163         parser.stackPointer += size
164     else:
165         parser.error = f"Imposs vel declarar um array de tamanho {size}"
166         parser.exito = False
167 else:
168     parser.error = (
169         f"Vari vel com o nome {listName} j definida anteriormente.")
170     parser.exito = False
171     parser.linhaDeCodigo +=1
172
173
174 # Atribui valores lista com outra lista
175 def p_AtribLista_lista(p):
176     "Atrib : LISTA ID COM lista"
177     lista = p[4]
178     varName = p[2]
179     print(lista)
180     if varName in parser.variaveis:
181         if len(lista) == parser.variaveis[varName][1]:
182             assm = ""
183             for i in range(len(lista)):
184                 assm += f"PUSHGP\nPUSHI {parser.variaveis[varName][0]+i}\nPUSHI {int(lista
185 [i])}\nSTOREN\n"
186             p[0] = assm
187         else:
188             print("stackOverflow")
189     else:
190         parser.error = f"Vari vel com o nome {varName} n o definida"
191         parser.exito = False
192         parser.linhaDeCodigo +=1
193
194
195 # Altera valor de um indice da lista
196 def p_AlternaLista_elem(p):
197     "Atrib : ALTERNA ID ABREPR expr FECHAPR COM expr"
198     varName = p[2]
199     pos = p[4]
200     if varName in parser.variaveis:
201         p[0] = f"PUSHGP\nPUSHI {parser.variaveis[varName][0]}\nPADD\n{p[4]}\n{p[7]}\nSTOREN\n"
202     else:
203         parser.error = f"Vari vel com o nome {varName} n o definida"
204         parser.exito = False
205         parser.linhaDeCodigo +=1
206
207
208 # Declara lista sem tamanho
209 def p_Decl_Matriz_NoSize(p):
210     "Decl : MATRIZ ID"
211     listName = p[2]
212     if listName not in parser.variaveis:
213         parser.variaveis[listName] = (parser.stackPointer, 0, 0)
214         p[0] = f"PUSHN 0\n"
215         parser.stackPointer += 1
216     else:

```

```

216         parser.error = (
217             f"Vari vel com o nome {listName} j      definida anteriormente.")
218         parser.exito = False
219         parser.linhaDeCodigo +=1
220
221
222 # Declara matriz com tamanho INT INT
223 def p_DeclMatriz_Size(p):
224     "Decl : MATRIZ ID INT INT"
225     listName = p[2]
226     size = int(p[3])
227     size1 = int(p[4])
228     if listName not in parser.variaveis:
229         parser.variaveis[listName] = (parser.stackPointer, size, size1)
230         p[0] = f"PUSHN {size*size1}\n"
231         parser.stackPointer += size*size1
232     else:
233         parser.error = (
234             f"Vari vel com o nome {listName} j      definida anteriormente.")
235         parser.exito = False
236         parser.linhaDeCodigo +=1
237
238
239 # Fun    o que altera o valor de um indice da matriz por outro
240 def p_AtribMatriz_comExpr(p):
241     "Atrib : ALTERNA ID ABREPR expr FECHAPR ABREPR expr FECHAPR COM expr"
242     matName = p[2]
243     indice1 = p[4]
244     indice2 = p[7]
245     valor = p[10]
246     if matName in parser.variaveis:
247         if len(parser.variaveis[matName]) == 3:
248             p[0] = f"PUSHGP\nPUSHI {parser.variaveis[matName][0]}\nPADD\n{indice1}PUSHI {
249                 parser.variaveis[matName][2]}\nMUL\nPADD\n{indice2}{valor}STOREN\n"
250         else:
251             parser.error = f"Opera    o inv lida, vari vel {matName} n o      uma matriz"
252             parser.exito = False
253     else:
254         parser.error = f"Vari vel n o declarada anteriormente"
255         parser.exito = False
256         parser.linhaDeCodigo +=1
257
258 # Fun    o que altera uma lista da matriz por outra
259 def p_AtribMatriz_comLista(p):
260     "Atrib : ALTERNA ID ABREPR expr FECHAPR COM lista"
261     matName = p[2]
262     if matName in parser.variaveis:
263         if len(parser.variaveis[matName]) == 3:
264             if len(p[7]) <= parser.variaveis[matName][2]:
265                 assm = ""
266                 j = 0
267                 for i in p[7]:
268                     assm += f'''PUSHGP\nPUSHI {parser.variaveis[matName][0]}\nPADD\n{p[4]}
269                     PUSHI {parser.variaveis[matName][2]}\nMUL\nPADD\nPUSHI {j}\nPUSHI {i}\nSTOREN\n'''
270                     j += 1
271                 p[0] = f'{assm}'
272             else:
273                 parser.error = f"Tamanho da lista maior do que o alocado"

```

```

273         parser.exito = False
274     else:
275         parser.error = f"Opera o inv lida, vari vel {matName} n o uma matriz"
276         parser.exito = False
277     else:
278         parser.error = f"Vari vel n o declarada anteriormente"
279         parser.exito = False
280     parser.linhaDeCodigo +=1
281
282
283 # Fun o que vai buscar o valor do indice na lista
284 def p_AtribBusca_Lista(p):
285     "expr : BUSCA ID ABREPR expr FECHAPR"
286     varName = p[2]
287     indice = p[4]
288     if varName in parser.variaveis:
289         p[0] = f"PUSHGP\nPUSHI {parser.variaveis[varName][0]}\nPADD\n{indice}LOADN\n"
290     else:
291         parser.error = (
292             f"Vari vel com o nome {varName} n o definida anteriormente.")
293         parser.exito = False
294     parser.linhaDeCodigo +=1
295
296
297 # Fun o que vai buscar o valor do indice na matriz
298 def p_AtribBusca_Matriz(p):
299     "expr : BUSCA ID ABREPR expr FECHAPR ABREPR expr FECHAPR"
300     varName = p[2]
301     indice1 = p[4]
302     indice2 = p[7]
303     if varName in parser.variaveis:
304         p[0] = f"PUSHGP\nPUSHI {parser.variaveis[varName][0]}\nPADD\n{indice1}PUSHI {
305 parser.variaveis[varName][2]}\nMUL\nPADD\n{indice2}LOADN\n"
306     else:
307         parser.error = f"Vari vel com o nome {varName} n o definida"
308         parser.exito = False
309     parser.linhaDeCodigo +=1
310
311
312 # Fun o swap entre elementos do mesmo array
313 def p_ProcSwap_Lista(p):
314     "Proc : SWAP ID ABREPR INT FECHAPR COM ABREPR INT FECHAPR"
315     varName = p[2]
316     indice1 = p[4]
317     indice2 = p[8]
318     if varName in parser.variaveis:
319         p[0] = f"PUSHG {indice1}\nPUSHG {indice2}\nSTOREG {indice1}\nSTOREG {indice2}\n"
320     else:
321         parser.error = (
322             f"Vari vel com o nome {varName} n o definida anteriormente.")
323         parser.exito = False
324     parser.linhaDeCodigo +=1
325
326
327 # Express o Aritm tica Soma
328 def p_soma(p):
329     "exprArit : expr SOMA expr"
330     p[0] = f"{p[1]}{p[3]}ADD\n"

```

```

331
332 # Express o Aritm tica Subtra o
333 def p_sub(p):
334     "exprArit : expr MENUS expr"
335     p[0] = f"{p[1]}{p[3]}SUB\n"
336
337
338 # Express o Aritm tica Multiplica o
339 def p_mult(p):
340     "exprArit : expr SOMANBEZES expr"
341     p[0] = f"{p[1]}{p[3]}MUL\n"
342
343
344 # Express o Aritm tica Divis o
345 def p_div(p):
346     "exprArit : expr DIBIDE expr"
347     p[0] = f"{p[1]}{p[3]}MUL\n"
348
349
350 # Express o Aritm tica Resto da divis o
351 def p_rem(p):
352     "exprArit : expr SOBRAS expr"
353     p[0] = f"{p[1]}{p[3]}MOD\n"
354
355
356 # Express o Relativa N o
357 def p_not(p):
358     "exprRel : NOUM ABREPC expr FECHAPC"
359     p[0] = f"{p[3]}NOT\n"
360
361
362 # Express o Relativa Igual
363 def p_gemeo(p):
364     "exprRel : GEMEO ABREPC expr VIRG expr FECHAPC"
365     p[0] = f"{p[3]}{p[5]}EQUAL\n"
366
367
368 # Express o Relativa Diferente
369 def p_naogemeo(p):
370     "exprRel : NAOGEMEO ABREPC expr VIRG expr FECHAPC"
371     p[0] = f"{p[3]}{p[5]}NOT\nEQUAL\n"
372
373
374 # Express o Relativa Menor
375 def p_inf(p):
376     "exprRel : MAISPIQUENO ABREPC expr VIRG expr FECHAPC"
377     p[0] = f"{p[3]}{p[5]}INF\n"
378
379
380 # Express o Relativa Menor ou Igual
381 def p_infeq(p):
382     "exprRel : MAISPIQUENOOUGEMEO ABREPC expr VIRG expr FECHAPC"
383     p[0] = f"{p[3]}{p[5]}INFEQ\n"
384
385
386 # Express o Relativa Maior
387 def p_sup(p):
388     "exprRel : MAISGRANDE ABREPC expr VIRG expr FECHAPC"
389     p[0] = f"{p[3]}{p[5]}SUP\n"

```

```

390
391
392 # Express o Relativa Maior ou Igual
393 def p_supeq(p):
394     "exprRel : MAISGRANDEOUGEMEO ABREPC expr VIRG expr FECHAPC"
395     p[0] = f"{p[3]}{p[5]}SUPEQ\n"
396
397
398 # Express o Relativa E
399 def p_ie(p):
400     "exprRel : expr IE expr"
401     p[0] = f"{p[1]}{p[3]}ADD\nPUSHI 2\nEQUAL\n"
402
403
404 # Express o Relativa OU
405 def p_oue(p):
406     "exprRel : expr OUE expr"
407     p[0] = f"{p[1]}{p[3]}ADD\nPUSHI 1\nSUPEQ\n"
408
409
410 # Controlo de fluxo (if then)
411 def p_if_Then(p):
412     "if : SE ABREPC exprRel FECHAPC ENTAO ABRECHAV Codigo FECHACHAV FIM"
413     p[0] = f"{p[3]}JZ 1{parser.labels}\n{p[7]}1{parser.labels}: NOP\n"
414     parser.labels += 1
415     parser.linhaDeCodigo+=1
416
417
418 # Controlo de fluxo (if then else)
419 def p_if_Then_Else(p):
420     "if : SE ABREPC exprRel FECHAPC ENTAO ABRECHAV Codigo FECHACHAV SENAO ABRECHAV Codigo FECHACHAV FIM"
421     p[0] = f"{p[3]}JZ 1{parser.labels}\n{p[7]}JUMP 1{parser.labels}f\n1{parser.labels}: NOP\n{p[11]}1{parser.labels}f: NOP\n"
422     parser.labels += 1
423     parser.linhaDeCodigo+=1
424
425
426 # Ciclo (while)
427 def p_while(p):
428     "while : ENQUANTO ABREPC exprRel FECHAPC FAZ ABRECHAV Codigo FECHACHAV FIM"
429     p[0] = f'1{parser.labels}c: NOP\n{p[3]}JZ 1{parser.labels}f\n{p[7]}JUMP 1{parser.labels}c\n1{parser.labels}f: NOP\n'
430     parser.labels += 1
431     parser.linhaDeCodigo+=1
432
433
434 def p_saidas_STRING(p):
435     '''saidas : SAIDAS ASPA'''
436     p[0] = f'PUSHS {p[2]}\nWRITES\n'
437     parser.linhaDeCodigo+=1
438
439
440 def p_saidas_lista(p):
441     "saidas : SAIDAS ID"
442     if len(parser.variaveis[p[2]]) == 3:
443         listas = parser.variaveis[p[2]]
444         initLista = listas[0]
445         numeroListas = listas[1]

```

```

446     tamanhoListas = listas[2]
447     assm = "PUSHS \"[\"\\nWRITES\\n"
448     for i in range(numeroListas):
449         assm += "PUSHS \"[\"\\nWRITES\\n"
450         for j in range(tamanhoListas):
451             assm += f"PUSHGP\\nPUSHI {initLista}\\nPADD\\nPUSHGP\\nPUSHI {i}\\nPUSHI {
tamanhoListas}\\nMUL\\nPADD\\nPUSHI {j}\\nLOADN\\nWRITEI\\nPOP 1\\nPUSHS \",\"\\nWRITES\\n"
452             rm = "PUSHS \",\"\\nWRITES"
453             assm = assm[:-len(rm)-1]
454             assm += "PUSHS \"]\"\\nWRITES\\n"
455             assm += "PUSHS \",\"\\nWRITES\\n"
456         rm = "PUSHS \",\"\\nWRITES"
457         assm = assm[:-len(rm)-1]
458         assm += "PUSHS \"]\"\\nWRITES\\n"
459         p[0] = assm
460
461     elif len(parser.variaveis[p[2]]) == 2:
462         if parser.variaveis[p[2]][1] != None:
463             listas = parser.variaveis[p[2]]
464             initLista = listas[0]
465             tamanhoListas = listas[1]
466             assm = "PUSHS \"[\"\\nWRITES\\n"
467             for j in range(tamanhoListas):
468                 assm += f"PUSHGP\\nPUSHI {initLista}\\nPADD\\nPUSHI {j}\\nLOADN\\nWRITEI\\nPUSHS
\",\"\\nWRITES\\n"
469             rm = "PUSHS \",\"\\nWRITES"
470             assm = assm[:-len(rm)-1]
471             assm += "PUSHS \"]\"\\nWRITES\\n"
472             p[0] = assm
473         else:
474             p[0] = f"PUSHG {parser.variaveis[p[2]][0]}\\nWRITEI\\n"
475
476     else:
477         parser.error = ""
478         parser.exito = False
479         parser.linhaDeCodigo+=1
480
481
482 # Fun es auxiliares
483
484 def p_lista(p):
485     "lista : ABREPR elems FECHAPR"
486     p[0] = p[2]
487
488 def p_elems(p):
489     "elems : INT"
490     p[0] = [int(p[1])]
491
492
493 def p_elems_rec(p):
494     "elems : elems VIRG INT"
495     p[0] = p[1]+p[3]
496
497
498
499 # -----
500
501 def p_error(p):
502     print(p)

```



```

503     try:
504         helper(p.value)
505         parser.exito = False
506     except:
507         print(p)
508         parser.exito=False
509
510
511 def helper(syntaxError):
512     error = syntaxError.upper()
513     matches = difflib.get_close_matches(error, tokens, n=2, cutoff=0.6)
514     if matches != []:
515         parser.error = f"Syntax error na linha {parser.linhaDeCodigo}: Querias dizer {
matches[0]}"
516
517 # -----
518
519
520 parser = yacc.yacc()
521 parser.exito = True
522 parser.error = ""
523 parser.assembly = ""
524 parser.variaveis = {}
525 parser.stackPointer = 0
526 parser.linhaDeCodigo = 0
527 parser.labels = 0
528
529 assembly = ""
530
531
532 if len(sys.argv) == 3:
533     inputFileName = sys.argv[1]
534     if inputFileName[-4:] == ".plo":
535         file = open(inputFileName, "r")
536         content = file.read()
537         parser.parse(content)
538         if parser.exito:
539             assembly += parser.assembly
540         else:
541             print("-----")
542             print(parser.error)
543             print("-----")
544             sys.exit()
545         file.close()
546
547     arr = os.listdir()
548     outputFileName = sys.argv[2]
549
550     while outputFileName in arr:
551         outputFileName = outputFileName.split(".")[0]
552         outputFileName += "_copy.vm"
553
554     if ".vm" not in outputFileName:
555         outputFileName+=".vm"
556
557     outputFile = open(outputFileName, "w")
558     outputFile.write(assembly)
559     outputFile.close()
560

```

```

561         print("File saved successfully")
562
563     else:
564         print("Invalid file extension")
565
566
567 if len(sys.argv) == 2:
568     inputFileName = sys.argv[1]
569     if inputFileName[-4:] == ".plo":
570         file = open(inputFileName, "r")
571         content = file.read()
572         parser.parse(content)
573         if parser.exito:
574             assembly += parser.assembly
575             print(parser.variaveis)
576         else:
577             print("-----")
578             print(parser.error)
579             print(parser.variaveis)
580             print("-----")
581             sys.exit()
582         file.close()
583         outputFileName = "a.vm"
584
585         arr = os.listdir()
586
587         while outputFileName in arr:
588             outputFileName = outputFileName.split(".")[0]
589             outputFileName += "_copy.vm"
590
591         outputFile = open(outputFileName, "w")
592         outputFile.write(assembly)
593         outputFile.close()
594
595         print("File saved successfully")
596
597     else:
598         print("Invalid file extension")
599
600 if len(sys.argv) == 1:
601     line = input(">")
602     while line:
603         parser.exito = True
604         parser.parse(line)
605         if parser.exito:
606             assembly += parser.assembly
607         else:
608             print("-----")
609             print(parser.error)
610             print("-----")
611             sys.exit()
612         line = input(">")
613
614     saveMachineCode = input(
615         "Do you want to save the code that you generated?[y/n]")
616     if saveMachineCode.lower() == "y":
617         path = input("Where do you want to save it?")
618         if path:
619             if ".vm" not in path:

```

```

620         file = open(f"{path}.vm", "w")
621         file.write(assembly)
622     else:
623         file = open(f"{path}.vm", "w")
624         file.write(assembly)
625
626     else:
627         file = open("./a.vm", "w")
628         file.write(assembly)
629
630     file.close()
631     print("File saved successfully")
632
633 else:
634     print("Bye Bye")
635     quit

```

Ficheiro de teste factorial.plo

```

1  saidas "Factorial: "
2  var n com entradas
3  saidas n
4  var res com 1
5
6  enquanto (maisGrande(n,0)) faz {
7      alterna res com res * n
8      alterna n com n - 1
9  } fim
10
11 saidas "\nResultado: "
12 saidas res

```

Ficheiro de teste busca_no_array.plo

```

1  lista a 10
2  lista a com [1,2,3,4,5,6,7,8,9,10]
3
4  saidas "Introduza um indice do array:\n"
5  var i com entradas
6
7  var x com busca a[i]
8
9  saidas "Valor: "
10 saidas x

```

Ficheiro de teste read_array.plo

```

1  var n com 5
2  var i com 0
3  lista a 5
4
5  enquanto (maisPiqueno(i,n)) faz {
6      alterna a [i] com entradas
7      alterna i com i + 1
8  } fim
9
10 saidas "Array gerado:\n"
11 saidas a

```

Ficheiro de teste produtorio.plo

```

1  saidas "Quantos numeros? "

```

```

2 var n com entradas
3 saidas n
4 var res com 1
5 var x com 1
6
7 enquanto (maisGrande(n,0)) faz {
8     alterna x com entradas
9     alterna res com res * x
10    alterna n com n - 1
11 } fim
12
13 saidas "\nResultado: "
14 saidas res

```

Ficheiro de teste matriz.plo

```

1 matriz m 3 3
2
3 alterna m [0] com [1,2,3]
4 alterna m [1] com [4,5,6]
5 alterna m [2] com [7,8,9]
6
7 saidas m

```

Ficheiro de teste swap_array.plo

```

1 lista a 5
2 lista a com [1,2,3,4,5]
3
4 saidas "Array inicial:\n"
5 saidas a
6
7 saidas "\nTroca do indice 1 com indice 3."
8
9 swap a [1] com [3]
10
11 saidas "Array inicial:\n"
12 saidas a

```