

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

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
          | Atrib
          | Corpo
          | Decls Corpo
          | Atrib Corpo

Corpo    :Codigo

Codigo   : Proc Codigo
          | Atrib Codigo
          | Proc
          | Atrib

Decls    : Decl
          | Decl Decls
```

```

Decl      : VAR ID
           | MATRIZ ID
           | MATRIZ ID INT INT

Atrib     : VAR ID COM expr
           | ALTERNA ID COM expr
           | LISTA ID
           | LISTA ID INT
           | LISTA ID COM lista
           | ALTERNA ID ABREPR expr FECHAPR ABREPR expr FECHAPR COM expr
           | ALTERNA ID ABREPR expr FECHAPR COM lista

lista     : ABREPR elems FECHAPR

elems     : INT
           | elems VIRG INT

expr      : INT
           | ID
           | ENTRADAS
           | BUSCA ID ABREPR expr FECHAPR
           | BUSCA ID ABREPR expr FECHAPR ABREPR expr FECHAPR

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
           | SWAP ID ABREPR INT FECHAPR COM ABREPR INT FECHAPR

if        : SE ABREPC exprRel FECHAPC ENTAO ABRECHAV Codigo FECHACHAV FIM
           | SE ABREPC exprRel FECHAPC ENTAO ABRECHAV Codigo FECHACHAV SENAO ABRECHAV Codigo FECHACHAV FIM

```



```
while      : ENQUANTO ABREPC exprRel FECHAPC FAZ ABRECHAVCodigo FECHACHAV FIM

saidas     : SAIDAS ASPA
            | SAIDAS ID
```

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 : 'gemo'
NAOGEMEO : 'naogemo'
MAISGRANDEOUGEMEO : 'maisGrandeOuGemo'
MAISPIQUENOOUUGEMEO : 'maisPiquenoOuGemo'
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, vamos alterar 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 2
7 MUL
8 PADD
9 PUSHI 1
10 PUSHI <valor>
11 STOREN
```

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

```

```
109 PUSHGP
110 PUSHI 0
111 PADD
112 PUSHI 4
113 LOADN
114 WRITEI
115 PUSH5 "]"
116 WRITES
117 STOP
```

4.7.3 Código gerado pela VM

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

```

42     "SENAO",
43     "SE",
44     "ENTAO",
45     "FIM",
46
47     "ENQUANTO",
48     "FAZ",
49
50     "ENTRADAS",
51     "SAIDAS",
52     "STRING"
53 ]
54
55 t_ABRECHAV = r"\{"
56 t_FECHACHAV = r"\}"
57 t_ABREPC = r'\('
58 t_FECHAPC = r'\)'
59 t_ABREPR = r'\['
60 t_FECHAPR = r'\]'
61 t_VIRG = r'\,'
62 t_SOMA = r'\+'
63 t_MENUS = r'\-'
64 t_SOMANBEZES = r'\*'
65 t_DIBIDE = r'\/'
66 t_SOBRAS = r'\%'
67 t_STRING = r"\w+\" | \"\w+\" "
68
69 t_ignore = ' \r\n\t'
70
71 t_ID = r"\w+"
72
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 def t_ALTERNA(t):
98     r'alterna'
99     t.type = "ALTERNA"
100    return t

```

```

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

```

```

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

```



```

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

```

Ficheiro yacc3.py

```

1  import ply.yacc as yacc
2  import random as rd
3
4  from lex2 import *
5  import sys
6
7
8  def p_Programa_Empty(p):
9      '''
10     Programa : Decls
11               | Atrib
12     '''
13     parser.assembly = f'{p[1]}'
14
15
16  def p_Programa(p):
17      '''
18     Programa : Decls Corpo
19     '''
20     parser.assembly = f'{p[1]}START\n{p[2]}STOP\n'
21
22
23  def p_Programa_Corpo(p):
24      '''
25     Programa : Corpo
26     '''
27     parser.assembly = f"START\n{p[1]}STOP\n"
28
29
30  def p_Corpo(p):
31      '''
32     Corpo : Proc
33            | Atrib
34     '''
35     p[0] = f"{p[1]}"
36
37
38  def p_Corpo_Rec(p):
39      '''
40     Corpo : Proc Corpo

```

```

41         | Atrib Corpo
42     '''
43     p[0] = f"{p[1]}{p[2]}"
44
45
46 def p_Decls(p):
47     "Decls : Decl"
48     p[0] = f'{p[1]}'
49
50
51 def p_DeclsRec(p):
52     "Decls : Decls Decl"
53     p[0] = f'{p[1]}{p[2]}'
54
55
56 def p_expr_arit(p):
57     '''
58     expr : exprArit
59         | exprRel
60     '''
61     p[0] = p[1]
62
63
64 def p_Proc(p):
65     '''
66     Proc : if
67         | while
68         | saidas
69     '''
70     p[0] = p[1]
71
72
73 # Declara o de uma variavel sem valor
74 def p_Decl(p):
75     "Decl : VAR ID"
76     varName = p[2]
77     if varName not in parser.variaveis:
78         parser.variaveis[varName] = (parser.stackPointer, None)
79         p[0] = "PUSHI 0\n"
80         parser.stackPointer += 1
81     else:
82         parser.exito = False
83         parser.error = f"Vari vel com o nome {varName} j existe"
84
85
86 # Declara o de uma vari vel com atribui o de um valor
87 def p_Atrib_expr(p):
88     "Atrib : VAR ID COM expr"
89     varName = p[2]
90     if varName not in parser.variaveis:
91         value = p[4]
92         parser.variaveis[varName] = (parser.stackPointer, None)
93         p[0] = f"{value}STOREG {parser.stackPointer}\n"
94         parser.stackPointer += 1
95     else:
96         parser.exito = False
97         parser.error = f"Vari vel com o nome {varName} j existe"
98
99

```

```

100 # Altera valor de um variavel
101 def p_alterna_var(p):
102     '''Atrib : ALTERNA ID COM expr'''
103     varName = p[2]
104     if varName in parser.variaveis:
105         #parser.variaveis[varName] = (p[4], parser.variaveis[varName][0])
106         p[0] = f"{p[4]}STOREG {parser.variaveis[varName][0]}\n"
107
108
109 def p_expr(p):
110     "expr : INT"
111     p[0] = f"PUSHI {int(p[1])}\n"
112
113
114 def p_expr_var(p):
115     "expr : ID"
116     varName = p[1]
117     if varName in parser.variaveis:
118         p[0] = f"PUSHG {parser.variaveis[varName][0]}\n"
119
120
121 def p_expr_entradas(p):
122     "expr : ENTRADAS"
123     p[0] = f"READ\nATOI\n"
124
125
126 # Declara lista sem tamanho
127 def p_Decl_Lista_NoSize(p):
128     "Decl : LISTA ID"
129     listName = p[2]
130     if listName not in parser.variaveis:
131         parser.variaveis[listName] = (parser.stackPointer, 0)
132         p[0] = f"PUSHN 0\n"
133         parser.stackPointer += 1
134     else:
135         parser.error = (
136             f"Vari vel com o nome {listName} j definida anteriormente.")
137         parser.exito = False
138
139
140 # Declara lista com tamanho INT
141 def p_DeclLista_Size(p):
142     "Decl : LISTA ID INT"
143     listName = p[2]
144     size = int(p[3])
145     if listName not in parser.variaveis:
146         if size > 0:
147             parser.variaveis[listName] = (parser.stackPointer, size)
148             p[0] = f"PUSHN {size}\n"
149             parser.stackPointer += size
150         else:
151             parser.error = f"Imposs vel declarar um array de tamanho {size}"
152             parser.exito = False
153     else:
154         parser.error = (
155             f"Vari vel com o nome {listName} j definida anteriormente.")
156         parser.exito = False
157
158

```

```

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

```

```

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

```

```

274     p[0] = f"PUSHGP\nPUSHI {parser.variaveis[varName][0]}\nPADD\n{indice}LOADN\n"
275 else:
276     parser.error = (
277         f"Vari vel com o nome {varName} n o definida anteriormente.")
278     parser.exito = False
279
280
281 # Fun o que vai buscar o valor do indice na matriz
282 def p_ProcBusca_Matriz(p):
283     "Proc : BUSCA ID ABREPR expr FECHAPR ABREPR expr FECHAPR"
284     varName = p[2]
285     indice1 = p[4]
286     indice2 = p[7]
287     if varName in parser.variaveis:
288         p[0] = f"PUSHGP\nPUSHI {parser.variaveis[varName][0]}\nPADD\n{indice1}PUSHI {
parser.variaveis[varName][2]}\nMUL\nPADD\n{indice2}LOADN\n"
289     else:
290         parser.error = f"Vari vel com o nome {varName} n o definida"
291         parser.exito = False
292
293
294 # Fun o swap entre elementos do mesmo array
295 def p_ProcSwap_Lista(p):
296     "Proc : SWAP ID ABREPR expr FECHAPR COM ABREPR expr FECHAPR"
297     varName = p[2]
298     indice1 = p[4]
299     indice2 = p[8]
300     if varName in parser.variaveis:
301         p[0] = f"PUSHG {parser.variaveis[varName][0]}"
302     else:
303         parser.error = (
304             f"Vari vel com o nome {varName} n o definida anteriormente.")
305         parser.exito = False
306
307
308 # Express o Aritm tica Soma
309 def p_soma(p):
310     "exprArit : expr SOMA expr"
311     p[0] = f"{p[1]}{p[3]}ADD\n"
312
313
314 # Express o Aritm tica Subtra o
315 def p_sub(p):
316     "exprArit : expr MENUS expr"
317     p[0] = f"{p[1]}{p[3]}SUB\n"
318
319
320 # Express o Aritm tica Multiplica o
321 def p_mult(p):
322     "exprArit : expr SOMANBEZES expr"
323     p[0] = f"{p[1]}{p[3]}MUL\n"
324
325
326 # Express o Aritm tica Divis o
327 def p_div(p):
328     "exprArit : expr DIBIDE expr"
329     p[0] = f"{p[1]}{p[3]}MUL\n"
330
331

```

```

332 # Express o Aritm tica Resto da divis o
333 def p_rem(p):
334     "exprArit : expr SOBRAS expr"
335     p[0] = f"{p[1]}{p[3]}MOD\n"
336
337
338 # Express o Relativa N o
339 def p_not(p):
340     "exprRel : NOUM ABREPC expr FECHAPC"
341     p[0] = f"{p[3]}NOT\n"
342
343
344 # Express o Relativa Igual
345 def p_gemeo(p):
346     "exprRel : GEMEO ABREPC expr VIRG expr FECHAPC"
347     p[0] = f"{p[3]}{p[5]}EQUAL\n"
348
349
350 # Express o Relativa Diferente
351 def p_naogemeo(p):
352     "exprRel : NAOGEMEO ABREPC expr VIRG expr FECHAPC"
353     p[0] = f"{p[3]}{p[5]}NOT\nEQUAL\n"
354
355
356 # Express o Relativa Menor
357 def p_inf(p):
358     "exprRel : MAISPIQUENO ABREPC expr VIRG expr FECHAPC"
359     p[0] = f"{p[3]}{p[5]}INF\n"
360
361
362 # Express o Relativa Menor ou Igual
363 def p_infeq(p):
364     "exprRel : MAISPIQUENOUGEMEO ABREPC expr VIRG expr FECHAPC"
365     p[0] = f"{p[3]}{p[5]}INFEQ\n"
366
367
368 # Express o Relativa Maior
369 def p_sup(p):
370     "exprRel : MAISGRANDE ABREPC expr VIRG expr FECHAPC"
371     p[0] = f"{p[3]}{p[5]}SUP\n"
372
373
374 # Express o Relativa Maior ou Igual
375 def p_supeq(p):
376     "exprRel : MAISGRANDEOUGEMEO ABREPC expr VIRG expr FECHAPC"
377     p[0] = f"{p[3]}{p[5]}SUPEQ\n"
378
379
380 # Express o Relativa E
381 def p_ie(p):
382     "exprRel : expr IE expr"
383     p[0] = f"{p[1]}{p[3]}ADD\nPUSHI 2\nEQUAL\n"
384
385
386 # Express o Relativa OU
387 def p_oue(p):
388     "exprRel : expr OUE expr"
389     p[0] = f"{p[1]}{p[3]}ADD\nPUSHI 1\nSUPEQ\n"
390

```

```

391
392 # Controlo de fluxo (if then)
393 def p_if_Then(p):
394     "if : SE ABREPC exprRel FECHAPC ENTAO ABRECHAV Corpo FECHACHAV FIM"
395     p[0] = f"{p[3]}JZ 1{parser.labels}\n{p[7]}1{parser.labels}: NOP\n"
396     parser.labels += 1
397
398
399 # Controlo de fluxo (if then else)
400 def p_if_Then_Else(p):
401     "if : SE ABREPC exprRel FECHAPC ENTAO ABRECHAV Corpo FECHACHAV SENA O ABRECHAV Corpo
FECHACHAV FIM"
402     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"
403     parser.labels += 1
404
405
406 # Ciclo (while)
407 def p_while(p):
408     "while : ENQUANTO ABREPC exprRel FECHAPC FAZ ABRECHAV Corpo FECHACHAV FIM"
409     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"
410     parser.labels += 1
411
412
413 def p_saidas_STRING(p):
414     "saidas : SAIDAS STRING"
415     p[0] = f'PUSHS {p[2]}\nWRITES\n'
416
417
418 def p_saidas_lista(p):
419     "saidas : SAIDAS ID"
420     if len(parser.variaveis[p[2]])==3 :
421         listas = parser.variaveis[p[2]]
422         initLista = listas[0]
423         numeroListas = listas[1]
424         tamanhoListas = listas[2]
425         assm = ""
426         for i in range(numeroListas):
427             for j in range(tamanhoListas):
428                 assm+=f"PUSHGP\nPUSHI {initLista}\nPADD\nPUSHI {i}\nPUSHI {tamanhoListas}\
nPADD\nPUSHI {j}\nLOADN\nWRITEI\n"
429
430     if len(parser.variaveis[p[2]])==2:
431         listas = parser.variaveis[p[2]]
432         initLista = listas[0]
433         tamanhoListas = listas[1]
434         assm = ""
435         for j in range(tamanhoListas):
436             assm+=f"PUSHGP\nPUSHI {initLista}\nPADD\nPUSHI {j}\nLOADN\nWRITEI\n"
437         p[0]=assm
438
439 def p_saidas_expr(p):
440     "saidas : SAIDAS expr"
441     p[0] = f"{p[2]}WRITEI\n"
442
443
444 # Fun es auxiliares
445

```



```

446 def p_lista(p):
447     "lista : ABREPR elems FECHAPR"
448     p[0] = p[2]
449
450
451 def p_elems(p):
452     "elems : INT"
453     p[0] = [int(p[1])]
454
455
456 def p_elems_rec(p):
457     "elems : elems VIRG INT"
458     p[0] = p[1]+[p[3]]
459
460
461 # -----
462
463 def p_error(p):
464     print(f'Erro na linha {parser.linhaDeCodigo}: {parser.error}')
465     parser.exito = False
466
467 # -----
468
469
470 parser = yacc.yacc(outputdir="./cache")
471 parser.exito = True
472 parser.error = ""
473 parser.assembly = ""
474 parser.variaveis = {}
475 parser.stackPointer = 0
476 parser.linhaDeCodigo = 0
477 parser.labels = 0
478
479 assembly = ""
480
481 if len(sys.argv)==3:
482     print(f"{sys.argv[0]}, {sys.argv[1]}, {sys.argv[2]}")
483
484 if len(sys.argv)==2:
485     print(f"{sys.argv[0]}, {sys.argv[1]}")
486
487 if len(sys.argv)==1:
488     line = input(">")
489     while line:
490         parser.exito = True
491         parser.parse(line)
492         if parser.exito:
493             assembly += parser.assembly
494             print(parser.variaveis)
495         else:
496             print("-----")
497             print(parser.error)
498             print("-----")
499             quit
500         line = input(">")
501
502 saveMachineCode=input("Do you want to save the code that you generated?[y/n]")
503 if saveMachineCode.lower()=="y":
504     path = input("Where do you want to save it?")

```

```

505         if path:
506             if ".vm" not in path:
507                 file = open(f"{path}.vm", "w")
508                 file.write(assembly)
509             else:
510                 file = open(f"{path}.vm", "w")
511                 file.write(assembly)
512
513         else:
514             file = open("./a.vm", "w")
515             file.write(assembly)
516
517         file.close()
518         print("File saved successfully")
519
520     else:
521         print("Bye Bye")
522         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

```