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 Linguaguens 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 suncinta todas as decisões tomadas pos 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 complilador 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 aprensentamos 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 at ómicas 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 divido em **4** partes:

- Cosntrução da **GIC** que define a estrutura sintática da nossa liguaguem.
- Construção do analizador léxico, **lexer** .
- Construção do analizador 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 grámatica 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 FECHA
```

```
while    : ENQUANTO ABREPC exprRel FECHAPC FAZ ABRECHAV Codigo FECHACHAV FIM

saidas   : SAIDAS ASPA
         | SAIDAS ID
```

3.3 Lexer

O analisador léxico, **lexer**, é o responsável por 'capturar' os simbolos 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 respetivas 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'
MAISPIQUENOOUGEMEO : '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 **Assemby** 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:

```
PUSHI 0
```

Sendo a variável predefinida a 0.

Para declarar uma variavel com valor fazemos:

```
PUSHI <valor>
STOREG <endereco>
```

Para declarar uma lista de valores temos de fazer sempre:

```
PUSHN <tamanho>
```

Inicializando todos os valores da lista a 0.

Seja a uma lista de tamanho 3, por exemplo:

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

Para atribuir valores à lista fazemos:

```
PUSHN <tamanho>
PUSHGP
PUSHI 0
PUSHI <valor1>
STOREN
PUSHGP
PUSHI 1
PUSHI <valor2>
STOREN
PUSHGP
PUSHI 2
PUSHI <valor3>
STOREN
```

No caso das matrizes, para declarar uma matriz fazemos:

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

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

```
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 especifico:

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

2. Alteramos uma linha da matriz passando uma lista:

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

Tomando como exemplo uma matriz 2x2, vamos alterar valores.
Pela 1ª opção:

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

Gera-se o seguinte:

```
PUSHN 4
PUSHGP
PUSHI 0
PADD
PUSHI 0
PUSHI 2
MUL
PADD
PUSHI 1
PUSHI <valor>
STOREN
```

Pela 2ª opção:

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

Gera-se o seguinte:

```
PUSHN 4
PUSHGP
PUSHI 0
PADD
PUSHI 0
PUSHI 2
MUL
PADD
PUSHI 0
PUSHI <valor1>
STOREN
PUSHGP
PUSHI 0
PADD
PUSHI 0
PUSHI 2
```

```
MUL
PADD
PUSHI 1
PUSHI <valor2>
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  PUSHS "Factorial: "
3  WRITES
4  READ
5  ATOI
6  STOREG 0
7  PUSHG 0
8  WRITEI
9  PUSHI 1
10 STOREG 1
11 l0c: NOP
12 PUSHG 0
13 PUSHI 0
14 SUP
15 JZ l0f
16 PUSHG 1
17 PUSHG 0
18 MUL
19 STOREG 1
20 PUSHG 0
21 PUSHI 1
22 SUB
23 STOREG 0
24 JUMP l0c
25 l0f: NOP
26 PUSHS "\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

```
lista a 10
lista a com [1,2,3,4,5,6,7,8,9,10]

saidas "Introduza um indice do array:\n"
var i com entradas

var x com busca a[i]

saidas "Valor: "
saidas x
```

4.3.2 Código assembly gerado

```
PUSHN 10
START
PUSHGP
PUSHI 0
PUSHI 1
STOREN
PUSHGP
PUSHI 1
PUSHI 2
STOREN
PUSHGP
PUSHI 2
PUSHI 3
STOREN
PUSHGP
PUSHI 3
PUSHI 4
STOREN
PUSHGP
PUSHI 4
PUSHI 5
STOREN
PUSHGP
PUSHI 5
PUSHI 6
STOREN
PUSHGP
PUSHI 6
PUSHI 7
STOREN
PUSHGP
PUSHI 7
PUSHI 8
STOREN
PUSHGP
PUSHI 8
PUSHI 9
STOREN
PUSHGP
PUSHI 9
PUSHI 10
STOREN
PUSHS "Introduza um indice do array:\n"
```

```
WRITES
READ
ATOI
STOREG 10
PUSHGP
PUSHI 0
PADD
PUSHG 10
LOADN
STOREG 11
PUSHS "Valor: "
WRITES
PUSHG 11
WRITEI
STOP
```

4.3.3 Código gerado pela VM

```
Introduza um indice do array: 3
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

```
var n com 5
var i com 0
lista a 5

enquanto (maisPiqueno(i,n)) faz {
    alterna a [i] com entradas
    alterna i com i + 1
} fim

saidas "Array gerado:\n"
saidas a
```

4.4.2 Código assembly gerado

```
PUSHI 5
STOREG 0
START
PUSHI 0
STOREG 1
PUSHN 5
l0c: NOP
PUSHG 1
PUSHG 0
INF
JZ l0f
```

```
PUSHGP
PUSHI 2
PADD
PUSHG 1
READ
ATOI
STOREN
PUSHG 1
PUSHI 1
ADD
STOREG 1
JUMP l0c
l0f: NOP
PUSHS "Array gerado:\n"
WRITES
PUSHS "["
WRITES
PUSHGP
PUSHI 2
PADD
PUSHI 0
LOADN
WRITEI
PUSHS ","
WRITES
PUSHGP
PUSHI 2
PADD
PUSHI 1
LOADN
WRITEI
PUSHS ","
WRITES
PUSHGP
PUSHI 2
PADD
PUSHI 2
LOADN
WRITEI
PUSHS ","
WRITES
PUSHGP
PUSHI 2
PADD
PUSHI 3
LOADN
WRITEI
PUSHS ","
WRITES
PUSHGP
PUSHI 2
PADD
PUSHI 4
LOADN
WRITEI
PUSHS "]"
WRITES
STOP
```

4.4.3 Código gerado pela VM

```
Array gerado:
[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

```
saidas "Quantos numeros? "
var n com entradas
saidas n
var res com 1
var x com 1

enquanto (maisGrande(n,0)) faz {
    alterna x com entradas
    alterna res com res * x
    alterna n com n - 1
} fim

saidas "\nResultado: "
saidas res
```

4.5.2 Código assembly gerado

```
START
PUSHS "Quantos numeros? "
WRITES
READ
ATOI
STOREG 0
PUSHG 0
WRITEI
PUSHI 1
STOREG 1
PUSHI 1
STOREG 2
l0c: NOP
PUSHG 0
PUSHI 0
SUP
JZ l0f
READ
ATOI
STOREG 2
PUSHG 1
PUSHG 2
MUL
STOREG 1
PUSHG 0
```

```
PUSHI 1
SUB
STOREG 0
JUMP l0c
l0f: NOP
PUSHS "\nResultado: "
WRITES
PUSHG 1
WRITEI
STOP
```

4.5.3 Código gerado pela VM

```
Quantos numeros? 5
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

```
matriz m 3 3

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

saidas m
```

4.6.2 Código assembly gerado

```
PUSHN 9
START
PUSHGP
PUSHI 0
PADD
PUSHI 0
PUSHI 3
MUL
PADD
PUSHI 0
PUSHI 1
STOREN
PUSHGP
PUSHI 0
PADD
PUSHI 0
PUSHI 3
MUL
PADD
PUSHI 1
```

```
PUSHI 2
STOREN
PUSHGP
PUSHI 0
PADD
PUSHI 0
PUSHI 3
MUL
PADD
PUSHI 2
PUSHI 3
STOREN
PUSHGP
PUSHI 0
PADD
PUSHI 1
PUSHI 3
MUL
PADD
PUSHI 0
PUSHI 4
STOREN
PUSHGP
PUSHI 0
PADD
PUSHI 1
PUSHI 3
MUL
PADD
PUSHI 1
PUSHI 5
STOREN
PUSHGP
PUSHI 0
PADD
PUSHI 1
PUSHI 3
MUL
PADD
PUSHI 2
PUSHI 6
STOREN
PUSHGP
PUSHI 0
PADD
PUSHI 2
PUSHI 3
MUL
PADD
PUSHI 0
PUSHI 7
STOREN
PUSHGP
PUSHI 0
PADD
PUSHI 2
PUSHI 3
MUL
PADD
```

```
PUSHI 1
PUSHI 8
STOREN
PUSHGP
PUSHI 0
PADD
PUSHI 2
PUSHI 3
MUL
PADD
PUSHI 2
PUSHI 9
STOREN
PUSHS "["
WRITES
PUSHS "["
WRITES
PUSHGP
PUSHI 0
PADD
PUSHGP
PUSHI 0
PUSHI 3
MUL
PADD
PUSHI 0
LOADN
WRITEI
POP 1
PUSHS ","
WRITES
PUSHGP
PUSHI 0
PADD
PUSHGP
PUSHI 0
PUSHI 3
MUL
PADD
PUSHI 1
LOADN
WRITEI
POP 1
PUSHS ","
WRITES
PUSHGP
PUSHI 0
PADD
PUSHGP
PUSHI 0
PUSHI 3
MUL
PADD
PUSHI 2
LOADN
WRITEI
POP 1
PUSHS "]"
WRITES
```

```
PUSHS ","
WRITES
PUSHS "["
WRITES
PUSHGP
PUSHI 0
PADD
PUSHGP
PUSHI 1
PUSHI 3
MUL
PADD
PUSHI 0
LOADN
WRITEI
POP 1
PUSHS ","
WRITES
PUSHGP
PUSHI 0
PADD
PUSHGP
PUSHI 1
PUSHI 3
MUL
PADD
PUSHI 1
LOADN
WRITEI
POP 1
PUSHS ","
WRITES
PUSHGP
PUSHI 0
PADD
PUSHGP
PUSHI 1
PUSHI 3
MUL
PADD
PUSHI 2
LOADN
WRITEI
POP 1
PUSHS "]"
WRITES
PUSHS ","
WRITES
PUSHS "["
WRITES
PUSHGP
PUSHI 0
PADD
PUSHGP
PUSHI 2
PUSHI 3
MUL
PADD
PUSHI 0
```

```
LOADN
WRITEI
POP 1
PUSHS ","
WRITES
PUSHGP
PUSHI 0
PADD
PUSHGP
PUSHI 2
PUSHI 3
MUL
PADD
PUSHI 1
LOADN
WRITEI
POP 1
PUSHS ","
WRITES
PUSHGP
PUSHI 0
PADD
PUSHGP
PUSHI 2
PUSHI 3
MUL
PADD
PUSHI 2
LOADN
WRITEI
POP 1
PUSHS "]"
WRITES
PUSHS "]"
WRITES
STOP
```

4.6.3 Execução da VM com o código gerado

```
[[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

```
lista a 5
lista a com [1,2,3,4,5]

saidas "Array inicial:\n"
saidas a

```

```
saidas "\nTroca do indice 1 com indice 3."

swap a [1] com [3]

saidas "Array inicial:\n"
saidas a
```

4.7.2 Código assembly gerado

```
PUSHN 5
START
PUSHGP
PUSHI 0
PUSHI 1
STOREN
PUSHGP
PUSHI 1
PUSHI 2
STOREN
PUSHGP
PUSHI 2
PUSHI 3
STOREN
PUSHGP
PUSHI 3
PUSHI 4
STOREN
PUSHGP
PUSHI 4
PUSHI 5
STOREN
PUSHS "Array inicial:\n"
WRITES
PUSHS "["
WRITES
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
PUSHS "\nTroca do indice 1 com indice 3."
WRITES
PUSHG 1
PUSHG 3
STOREG 1
STOREG 3
PUSHS "Array inicial:\n"
WRITES
PUSHS "["
WRITES
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
```

```
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 gramaticas e no desenvolvimento de compiladores de linguagens. Este trabalho levou-nos também a obter um maior conhecimento no que diz respeito à máquina virtuale a uma maior entendimento 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

```
import ply.lex as lex

tokens = [
    "ID",
    "VAR",
    "COM",

    "ABREPC",
    "FECHAPC",
    "ABREPR",
    "FECHAPR",
    "ABRECHAV",
    "FECHACHAV",
    "VIRG",

    "INT",

    'SOMA',
    'MENUS',
    'SOMANBEZES',
    'DIBIDE',
    'SOBRAS',

    'MAISGRANDE',
    'MAISPIQUENO',
    'GEMEO',
    'NAOGEMEO',
    'MAISGRANDEOUGEMEO',
    'MAISPIQUENOOUGEMEO',

    'IE',
    'OUE',
    'NOUM',

    "ALTERNA",

    "LISTA",
    "MATRIZ",
    "BUSCA",
    "SWAP",

```

```
    "SENAO",
    "SE",
    "ENTAO",
    "FIM",

    "ENQUANTO",
    "FAZ",

    "ENTRADAS",
    "SAIDAS",
    "STRING"
]

t_ABRECHAV = r"\{"
t_FECHACHAV = r"\}"
t_ABREPC = r'\('
t_FECHAPC = r'\)'
t_ABREPR = r'\['
t_FECHAPR = r'\]'
t_VIRG = r'\,'
t_SOMA = r'\+'
t_MENUS = r'\-'
t_SOMANBEZES = r'\*'
t_DIBIDE = r'\/'
t_SOBRAS = r'\%'
t_STRING = r"\"w+\"|\'w+\'"

t_ignore = ' \r\n\t'

t_ID = r"\w+"


def t_INT(t):
    r'\d+'
    t.type = "INT"
    return t


def t_COMENTARIO(t):
    r'comentario'
    t.type = "COMENTARIO"
    return t


def t_VAR(t):
    r'var'
    t.type = "VAR"
    return t


def t_COM(t):
    r'com'
    t.type = "COM"
    return t

def t_ALTERNA(t):
    r'alterna'
    t.type = "ALTERNA"
    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_MAISPIQUENOOUGEMEO(t):
134     r"maisPiquenoOuGemeo"
135     t.type = "MAISPIQUENOOUGEMEO"
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"
```

```
    t.type = "ENTRADAS"
    return t


def t_SAIDAS(t):
    r"saidas"
    t.type = "SAIDAS"
    return t


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


lexer = lex.lex()
```

Ficheiro yacc3.py

```
import ply.yacc as yacc
import random as rd

from lex2 import *
import sys


def p_Programa_Empty(p):
    '''
    Programa : Decls
             | Atrib
    '''
    parser.assembly = f'{p[1]}'


def p_Programa(p):
    '''
    Programa : Decls Corpo
    '''
    parser.assembly = f'{p[1]}START\n{p[2]}STOP\n'


def p_Programa_Corpo(p):
    '''
    Programa : Corpo
    '''
    parser.assembly = f"START\n{p[1]}STOP\n"


def p_Corpo(p):
    '''
    Corpo : Proc
          | Atrib
    '''
    p[0] = f"{p[1]}"


def p_Corpo_Rec(p):
    '''
    Corpo : Proc Corpo
```

```
          | Atrib Corpo
    '''
    p[0] = f"{p[1]}{p[2]}"


def p_Decls(p):
    "Decls : Decl"
    p[0] = f'{p[1]}'


def p_DeclsRec(p):
    "Decls : Decls Decl"
    p[0] = f'{p[1]}{p[2]}'


def p_expr_arit(p):
    '''
    expr : exprArit
         | exprRel
    '''
    p[0] = p[1]


def p_Proc(p):
    '''
    Proc : if
         | while
         | saidas
    '''
    p[0] = p[1]


# Declara  o de uma variavel sem valor
def p_Decl(p):
    "Decl : VAR ID"
    varName = p[2]
    if varName not in parser.variaveis:
        parser.variaveis[varName] = (parser.stackPointer,None)
        p[0] = "PUSHI 0\n"
        parser.stackPointer += 1
    else:
        parser.exito = False
        parser.error = f"Vari vel com o nome {varName} j  existe"


# Declara  o de uma vari vel com atribui  o de um valor
def p_Atrib_expr(p):
    "Atrib : VAR ID COM expr"
    varName = p[2]
    if varName not in parser.variaveis:
        value = p[4]
        parser.variaveis[varName] = (parser.stackPointer, None)
        p[0] = f"{value}STOREG {parser.stackPointer}\n"
        parser.stackPointer += 1
    else:
        parser.exito = False
        parser.error = f"Vari vel com o nome {varName} j  existe"


```

```
# Altera valor de um vari vel
def p_alterna_var(p):
    '''Atrib : ALTERNA ID COM expr'''
    varName = p[2]
    if varName in parser.variaveis:
        #parser.variaveis[varName] = (p[4], parser.variaveis[varName][0])
        p[0] = f"{p[4]}STOREG {parser.variaveis[varName][0]}\n"


def p_expr(p):
    "expr : INT"
    p[0] = f"PUSHI {int(p[1])}\n"


def p_expr_var(p):
    "expr : ID"
    varName = p[1]
    if varName in parser.variaveis:
        p[0] = f"PUSHG {parser.variaveis[varName][0]}\n"


def p_expr_entradas(p):
    "expr : ENTRADAS"
    p[0] = f"READ\nATOI\n"


# Declara lista sem tamanho
def p_Decl_Lista_NoSize(p):
    "Decl : LISTA ID"
    listName = p[2]
    if listName not in parser.variaveis:
        parser.variaveis[listName] = (parser.stackPointer, 0)
        p[0] = f"PUSHN 0\n"
        parser.stackPointer += 1
    else:
        parser.error = (
            f"Vari vel com o nome {listName} j  definida anteriormente.")
        parser.exito = False


# Declara lista com tamanho INT
def p_DeclLista_Size(p):
    "Decl : LISTA ID INT"
    listName = p[2]
    size = int(p[3])
    if listName not in parser.variaveis:
        if size > 0:
            parser.variaveis[listName] = (parser.stackPointer, size)
            p[0] = f"PUSHN {size}\n"
            parser.stackPointer += size
        else:
            parser.error = f"Imposs vel declarar um array de tamanho {size}"
            parser.exito = False
    else:
        parser.error = (
            f"Vari vel com o nome {listName} j  definida anteriormente.")
        parser.exito = False


```

```
# Atribui valores    lista com outra lista
def p_AtribLista_lista(p):
    "Atrib : LISTA ID COM lista"
    lista = p[4]
    varName = p[2]
    if varName in parser.variaveis:
        if len(lista)<parser.variaveis[varName][1]:
            assm = ""
            for i in lista:
                assm += f"PUSHI {i}\n"

            parser.variaveis[varName] = (parser.stackPointer, parser.variaveis[varName
    ][1])
            parser.stackPointer += len(lista)
            p[0] = assm
        else:
            parser.error = f"Vari vel com o nome {varName} n o definida"
            parser.exito = False
    else:
        assm = ""
        for i in lista:
            assm += f"PUSHI {i}\n"

        parser.variaveis[varName] = (parser.stackPointer, len(lista))
        parser.stackPointer += len(lista)
        p[0] = assm



# Altera valor de um indice da lista
def p_AlternaLista_elem(p):
    "Atrib : ALTERNA ID ABREPR expr FECHAPR COM expr"
    varName = p[2]
    pos = p[4]
    if varName in parser.variaveis:
        p[0] = f"PUSHGP\nPUSHI {parser.variaveis[varName][0]}\nPADD\n{p[4]}{p[7]}STOREN\n"
    else:
        parser.error = f"Vari vel com o nome {varName} n o definida"
        parser.exito = False


# Declara lista sem tamanho
def p_Decl_Matriz_NoSize(p):
    "Decl : MATRIZ ID"
    listName = p[2]
    if listName not in parser.variaveis:
        parser.variaveis[listName] = (parser.stackPointer, 0, 0)
        p[0] = f"PUSHN 0\n"
        parser.stackPointer += 1
    else:
        parser.error = (
            f"Vari vel com o nome {listName} j  definida anteriormente.")
        parser.exito = False


# Declara matriz com tamanho INT INT
def p_DeclMatriz_Size(p):
    "Decl : MATRIZ ID INT INT"
    listName = p[2]
```

```
    size = int(p[3])
    size1 = int(p[4])
    if listName not in parser.variaveis:
        parser.variaveis[listName] = (parser.stackPointer, size, size1)
        p[0] = f"PUSHN {size*size1}\n"
        parser.stackPointer += size*size1
    else:
        parser.error = (
            f"Vari vel com o nome {listName} j  definida anteriormente.")
        parser.exito = False


# Fun  o que altera o valor de um indice da matriz por outro
def p_AtribMatriz_comExpr(p):
    "Atrib : ALTERNA ID ABREPR expr FECHAPR ABREPR expr FECHAPR COM expr"
    matName = p[2]
    if matName in parser.variaveis:
        if len(parser.variaveis[matName]):
            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'''
        else:
            parser.error = f"Opera  o inv lida, vari vel {matName} n o  uma matriz"
            parser.exito = False
    else:
        parser.error = f"Vari vel n o declarada anteriormente"
        parser.exito = False


# Fun  o que altera uma lista da matriz por outra
def p_AtribMatriz_comLista(p):
    "Atrib : ALTERNA ID ABREPR expr FECHAPR COM lista"
    matName = p[2]
    if matName in parser.variaveis:
        if len(parser.variaveis[matName]) == 3:
            if len(p[7]) <= parser.variaveis[matName][2]:
                assm = ""
                j = 0
                for i in p[7]:
                    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'''
                    j += 1
                p[0] = f'{assm}'
            else:
                parser.error = f"Tamanho da lista maior do que o alocado"
                parser.exito = False
        else:
            parser.error = f"Opera  o inv lida, vari vel {matName} n o  uma matriz"
            parser.exito = False
    else:
        parser.error = f"Vari vel n o declarada anteriormente"
        parser.exito = False


# Fun  o que vai buscar o valor do indice na lista
def p_ProcBusca_Lista(p):
    "Proc : BUSCA ID ABREPR expr FECHAPR"
    varName = p[2]
    indice = p[4]
    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 
```

```
# Express o Aritm tica Resto da divis o
def p_rem(p):
    "exprArit : expr SOBRAS expr"
    p[0] = f"{p[1]}{p[3]}MOD\n"


# Express o Relativa N o
def p_not(p):
    "exprRel : NOUM ABREPC expr FECHAPC"
    p[0] = f"{p[3]}NOT\n"


# Express o Relativa Igual
def p_gemeo(p):
    "exprRel : GEMEO ABREPC expr VIRG expr FECHAPC"
    p[0] = f"{p[3]}{p[5]}EQUAL\n"


# Express o Relativa Diferente
def p_naogemeo(p):
    "exprRel : NAOGEMEO ABREPC expr VIRG expr FECHAPC"
    p[0] = f"{p[3]}{p[5]}NOT\nEQUAL\n"


# Express o Relativa Menor
def p_inf(p):
    "exprRel : MAISPIQUENO ABREPC expr VIRG expr FECHAPC"
    p[0] = f"{p[3]}{p[5]}INF\n"


# Express o Relativa Menor ou Igual
def p_infeq(p):
    "exprRel : MAISPIQUENOOUGEMEO ABREPC expr VIRG expr FECHAPC"
    p[0] = f"{p[3]}{p[5]}INFEQ\n"


# Express o Relativa Maior
def p_sup(p):
    "exprRel : MAISGRANDE ABREPC expr VIRG expr FECHAPC"
    p[0] = f"{p[3]}{p[5]}SUP\n"


# Express o Relativa Maior ou Igual
def p_supeq(p):
    "exprRel : MAISGRANDEOUGEMEO ABREPC expr VIRG expr FECHAPC"
    p[0] = f"{p[3]}{p[5]}SUPEQ\n"


# Express o Relativa E
def p_ie(p):
    "exprRel : expr IE expr"
    p[0] = f"{p[1]}{p[3]}ADD\nPUSHI 2\nEQUAL\n"


# Express o Relativa OU
def p_oue(p):
    "exprRel : expr OUE expr"
    p[0] = f"{p[1]}{p[3]}ADD\nPUSHI 1\nSUPEQ\n"

```

```
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 l{parser.labels}\n{p[7]}l{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 SENAO ABRECHAV Corpo
    FECHACHAV FIM"
402     p[0] = f"{p[3]}JZ l{parser.labels}\n{p[7]}JUMP l{parser.labels}f\nl{parser.labels}:
    NOP\n{p[11]}l{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'l{parser.labels}c: NOP\n{p[3]}JZ l{parser.labels}f\n{p[7]}JUMP l{parser.
    labels}c\nl{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?")
```

```
        if path:
            if ".vm" not in path:
                file = open(f"{path}.vm", "w")
                file.write(assembly)
            else:
                file = open(f"{path}.vm", "w")
                file.write(assembly)

        else:
            file = open("./a.vm","w")
            file.write(assembly)

        file.close()
        print("File saved successfully")

    else:
        print("Bye Bye")
        quit
```

Ficheiro de teste factorial.plo

```
saidas "Factorial: "
var n com entradas
saidas n
var res com 1

enquanto (maisGrande(n,0)) faz {
    alterna res com res * n
    alterna n com n - 1
} fim

saidas "\nResultado: "
saidas res
```

Ficheiro de teste busca_no_array.plo

```
lista a 10
lista a com [1,2,3,4,5,6,7,8,9,10]

saidas "Introduza um indice do array:\n"
var i com entradas

var x com busca a[i]

saidas "Valor: "
saidas x
```

Ficheiro de teste read_array.plo

```
var n com 5
var i com 0
lista a 5

enquanto (maisPiqueno(i,n)) faz {
    alterna a [i] com entradas
    alterna i com i + 1
} fim

saidas "Array gerado:\n"
saidas a
```

Ficheiro de teste produtorio.plo

```
saidas "Quantos numeros? "
var n com entradas
saidas n
var res com 1
var x com 1

enquanto (maisGrande(n,0)) faz {
    alterna x com entradas
    alterna res com res * x
    alterna n com n - 1
} fim

saidas "\nResultado: "
saidas res
```

Ficheiro de teste matriz.plo

```
matriz m 3 3

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

saidas m
```

Ficheiro de teste swap_array.plo

```
lista a 5
lista a com [1,2,3,4,5]

saidas "Array inicial:\n"
saidas a

saidas "\nTroca do indice 1 com indice 3."

swap a [1] com [3]

saidas "Array inicial:\n"
saidas a
```