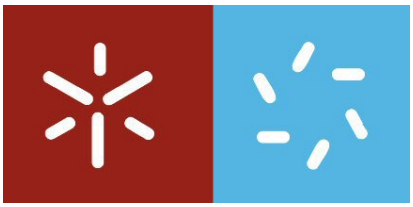


Universidade do Minho

Licenciatura em Ciências da Computação

PLC - Trabalho Prático 2

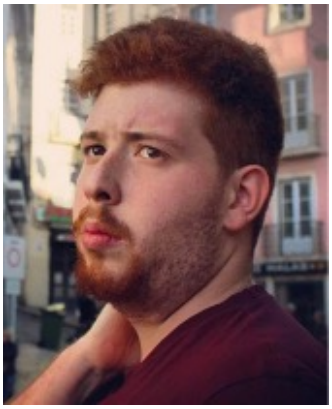
Grupo nº16

Francisco José Pereira Teófilo
(A93741)

Inês Pires Presa
(A90355)

Tiago dos Santos Silva Peixoto Carriço
(A91695)

16 de janeiro de 2022



Conteúdo

Capítulo 1

Introdução

No âmbito da disciplina de Processamento de Linguagens e Compiladores foi-nos proposto pelo docente Pedro Manuel Rangel Santos Henriques um trabalho onde temos de definir uma linguagem de programação imperativa simples que permita declarar e efetuar instruções básicas.

Com este trabalho pretendemos aumentar o nosso conhecimento sobre a engenharia de linguagens em programação gramatical, de modo a reforçar a escrita de GIC e GT, ou seja, gramáticas independentes de contexto e tradutoras, respetivamente. Desenvolver um compilador que gera código para uma "virtual machine".

Através deste documento queremos apresentar a nossa solução do problema, começando por discriminar as decisões tomadas na criação de uma linguagem e implementação da GIC que a reconhece. Apresentando, seguidamente, alguns exemplos de programas-fonte e os respetivos códigos *Assembly* produzidos.

Capítulo 2

Enunciado

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 atómicas do tipo *inteiro*, com os quais se podem realizar as habituais operações aritméticas, relacionais e lógicas.
- *efetuar* instruções algorítmicas básicas como a *atribuição do valor de expressões numéricas a variáveis.*
- *ler* do *standard input* e *escrever* no *standard output.*
- *efetuar* instruções *condicionais* para controlo do fluxo de execução. *e*fetuar instruções cíclicas para controlo do 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.

Como é da praxe neste tipo de linguagens, as variáveis deverão ser declaradas no início do programa e não pode haver re-declarações, nem utilizações sem declaração prévia. Se nada for explicitado, o valor da variável após a declaração é 0 (zero).
Desenvolva, então, um compilador para essa linguagem com base na `GIC` criada acima e com recurso aos módulos `Yacc/ Lex` do `PLY/Python`.
O compilador deve gerar **pseudo-código**, Assembly da Máquina Virtual `VM`.
Muito Importante:
Para a entrega do TP deve preparar um conjunto de testes (programas-fonte escritos na sua linguagem) e mostrar o código `Assembly` gerado bem como o programa a correr na máquina virtual `VM`. Esse conjunto terá de conter, no mínimo, os 4 primeiros exemplos abaixo e um dos 2 últimos conforme a sua escolha acima:

- ler 4 números e dizer se podem ser os lados de um quadrado.

- ler um inteiro N, depois ler N números e escrever o menor deles.
- ler N (constante do programa) números e calcular e imprimir o seu produtório.
- contar e imprimir os números impares de uma sequêcia de números naturais.
- ler e armazenar N números num array; imprimir os valores por ordem inversa.
- invocar e usar num programa seu uma função 'potencia()', que começa por ler do input a base B e o expoente E e retorna o valor B^E.

Capítulo 3

Decisões Tomadas

3.1 Desenho da linguagem

Para a implementação da nossa linguagem decidimos utilizar uma notação *case insensitive* na qual os diferentes nomes das operações estão definidos em português, como será constatado na descrição detalhada efetuada seguidamente.

3.1.1 Declarar variáveis

Para a declaração de variáveis decidimos utilizar a seguinte notação:

Comando	Função
int x	Criar inteiro "x"
int x <- v	Criar inteiro "x"com valor v
array a t	Criar array de inteiros "a"de tamanho t
matriz m t1 t2	Criar matriz de inteiros "m"de tamanho t1 * t2

3.1.2 Operações

Para as diferentes operações utilizamos a notação de função, onde o nome da função sugere uma determinada operação e os argumentos são os seus operandos. As tabelas seguintes mostram os nomes usados para cada operação.

- Operações aritméticas:

Comando	Função
soma (x, y)	Somar "x"a "y"
sub (x, y)	Subtrair "y"a "x"
mult (x, y)	Multiplicar "x"por "y"
div (x, y)	Dividir "x"por "y"
mod (x, y)	Calcular resto da divisao inteira de "x"por "y"

- Operações relacionais:

Comando	Função
maior (x, y)	x >y
menor (x, y)	x <y
igual (x, y)	x == y
maiori (x, y)	x >= y
menori (x, y)	x <= y
nigual (x, y)	x != y

- Operações lógicas:

Comando	Função
neg (x)	Negar "x"
e (x, y)	Conjunção de "x"com "y"
ou (x, y)	Disjunção de "x"e "y"

3.1.3 Atribuição

Para a atribuição a uma variável "x"utilizamos a notação `x <- y`, onde "y"pode ser um inteiro, uma variável ou uma expressão.

3.1.4 IO

Para as operações de *input/output* criamos os seguintes comandos.

Comando	Função
escrever x	Escrever o valor de "x"no *standard output*
escrevera x	Escrever os valores guardados num array ou matriz "x"no *standard output*
ler	Ler um inteiro do *standard input*

3.1.5 Instruções condicionais

Para efetuar instruções condicionais para controlo do fluxo de execução, implementamos as seguintes estruturas.

- Executar algo se a expressão "x"for verdadeira.

```
se x entao
...
fim
```

- Executar algo se a expressão "x"for verdadeira ou outra coisa de "x"for falsa.

```
se x entao
...
senao
...
fim
```

3.1.6 Instruções cíclicas

Para efetuar instruções cíclicas para controlo do fluxo de execução, implementamos o ciclo `while-do` da seguinte forma, para uma condição "x"e um conjunto de instruções "...".

```
enquanto x faz
...
fim
```

3.1.7 Indexação

- Para aceder ao índice "i"do array "a"utilizamos a seguinte notação.

  ```
  a[i]
  ```

- Para aceder à posição "l,c"da matriz "m"utilizamos a seguinte notação.

  ```
  m[l,c]
  ```

3.2 Desenho da gramática

Os símbolos terminais utilizados na gramática independente de contexto implementada para reconhecer a linguagem foram: 'NUM', 'NOME', 'INT', 'ARRAY', 'MATRIZ', 'SOMA', 'SUB', 'MULT', 'DIV', 'MOD', 'MAIOR', 'MENOR', 'IGUAL', 'MAIORI', 'MENORI', 'NIGUAL', 'NEG', 'E', 'OU', 'ATR', 'LER', 'ESCREVER', 'SE', 'ENTAO', 'SENAO', 'ENQUANTO', 'FAZ', 'FIM', 'PCABRIR', 'PCFECHAR', 'PRABRIR', 'PRFECHAR', 'VIRG', 'ESCREVERA'.(A sua implementação está descrita no Apêndice A.)

A gramática implementada utiliza o método recursivo à esquerda e é constituída pelas seguintes regras de derivação.

```
Programa : Corpo
         | Decls Corpo
Decls    : Decl
         | Decls Decl
Decl     : INT NOME
         | INT NOME ATR NUM
         | ARRAY NOME NUM
         | MATRIZ NOME NUM NUM
Corpo    : Proc
         | Corpo Proc
Proc     : Atrib
         | Se
         | Escrever
         | Enquanto
Se       : SE Cond ENTAO Corpo FIM
         | SE Cond ENTAO Corpo SENAO Corpo FIM
Enquanto : ENQUANTO Cond FAZ Corpo FIM
Atrib    : NOME ATR Expr
         | NOME PRABRIR Expr PRFECHAR ATR Expr
```

```
          | NOME PRABRIR Expr VIRG Expr PRFECHAR ATR Expr
          | NOME PRABRIR Expr PRFECHAR ATR LER
          | NOME PRABRIR Expr VIRG Expr PRFECHAR ATR LER
          | NOME ATR LER
Escrever : ESCREVERA NOME
          | ESCREVER Expr
Expr      : PCABRIR Expr PCFECHAR
          | Var
          | Num
          | SOMA PCABRIR Expr VIRG Expr PCFECHAR
          | SUB PCABRIR Expr VIRG Expr PCFECHAR
          | MULT PCABRIR Expr VIRG Expr PCFECHAR
          | DIV PCABRIR Expr VIRG Expr PCFECHAR
          | MOD PCABRIR Expr VIRG Expr PCFECHAR
          | Cond
Cond      : PCABRIR Cond PCFECHAR
          | MAIOR PCABRIR Expr VIRG Expr PCFECHAR
          | MENOR PCABRIR Expr VIRG Expr PCFECHAR
          | MAIORI PCABRIR Expr VIRG Expr PCFECHAR
          | MENORI PCABRIR Expr VIRG Expr PCFECHAR
          | IGUAL PCABRIR Expr VIRG Expr PCFECHAR
          | NIGUAL PCABRIR Expr VIRG Expr PCFECHAR
          | E PCABRIR Cond VIRG Cond PCFECHAR
          | OU PCABRIR Cond VIRG Cond PCFECHAR
          | NEG PCABRIR Cond PCFECHAR
Var       : NOME PRABRIR Expr VIRG Expr PRFECHAR
          | NOME PRABRIR Expr PRFECHAR
          | NOME
```

Capítulo 4

Regras de tradução para Assembly

As regras de tradução para Assembly estão apresentadas no Apêndice B.

Capítulo 5

Exemplos de utilização

5.1 Ler 4 números e dizer se podem ser os lados de um quadrado

- Programa-fonte

```
int a
int b
int c
int d

a <- ler
b <- ler
c <- ler
d <- ler
se maior (a, 0) entao
    se e (igual (a,b), e (igual (b,c), igual (c,d))) entao
        escrever 1
    senao
        escrever 0
    fim
senao
    escrever 2
fim
```

- Código produzido

```
PUSHI 0          READ             STOREG 3         PUSHG 1
PUSHI 0          ATOI             PUSHG 0          PUSHG 2
PUSHI 0          STOREG 1         PUSHI 0          EQUAL
PUSHI 0          READ             SUP              PUSHG 2
START            ATOI             JZ l1            PUSHG 3
READ             STOREG 2         PUSHG 0          EQUAL
ATOI             READ             PUSHG 1          ADD
STOREG 0         ATOI             EQUAL            PUSHI 2
```

```
                WRITEI          PUSHS "\n"      PUSHS "\n"
EQUAL           PUSHS "\n"      WRITES          WRITES
ADD             WRITES          l0f: NOP        l1f: NOP
PUSHI 2         JUMP l0f        JUMP l1f        STOP
EQUAL           l0: NOP         l1: NOP
JZ l0           PUSHI 0         PUSHI 2
PUSHI 1         WRITEI          WRITEI
```

5.2 Ler um inteiro N, depois ler N números e escrever o menor deles

- Programa-fonte

```
int n
int i
int x
int y

n <- ler
enquanto nigual (n, i) faz
    y <- ler
    se ou (menor (y, x), igual (i,0)) entao
        x <- y
    fim
    i <- soma (i, 1)
fim
escrever x
```

- Código produzido

```
PUSHI 0         EQUAL           EQUAL           STOREG 1
PUSHI 0         NOT             ADD             JUMP l1c
PUSHI 0         JZ l1f          PUSHI 1         l1f: NOP
PUSHI 0         READ            SUPEQ           PUSHG 2
START           ATOI            JZ l0           WRITEI
READ            STOREG 3        PUSHG 3         PUSHS "\n"
ATOI            PUSHG 3         STOREG 2        WRITES
STOREG 0        PUSHG 2         l0: NOP         STOP
l1c: NOP        INF             PUSHG 1
PUSHG 0         PUSHG 1         PUSHI 1
PUSHG 1         PUSHI 0         ADD
```

5.3 Ler N (constante do programa) números e calcular e imprimir o seu produtório

- Programa-fonte

```
int n <- 5
```

```
int res <- 1
int x

enquanto nigual (n, 0) faz
    x <- ler
    res <- mult (res, x)
    n <- sub (n, 1)
fim
escrever res
```

- Código produzido

```
PUSHI 5          NOT              STOREG 1         WRITEI
PUSHI 1          JZ l0f           PUSHG 0          PUSHS "\n"
PUSHI 0          READ             PUSHI 1          WRITES
START            ATOI             SUB              STOP
l0c: NOP         STOREG 2         STOREG 0
PUSHG 0          PUSHG 1          JUMP l0c
PUSHI 0          PUSHG 2          l0f: NOP
EQUAL            MUL              PUSHG 1
```

5.4 Contar e imprimir os números impares de uma sequência de números naturais

- Programa-fonte

```
int inf1
int sup
int i
int som

inf1 <- ler
sup <- ler
i <- inf1
enquanto nigual (i, sup) faz
    se igual (mod (i, 2), 1) entao
        escrever i
        som <- soma (som, 1)
    fim
    i <- soma (i, 1)
fim
escrever som
```

- Código produzido

```
PUSHI 0          START            READ             STOREG 2
PUSHI 0          READ             ATOI             l1c: NOP
PUSHI 0          ATOI             STOREG 1         PUSHG 2
PUSHI 0          STOREG 0         PUSHG 0          PUSHG 1
```

```
                EQUAL           ADD             l1f: NOP
EQUAL           JZ l0           STOREG 3        PUSHG 3
NOT             PUSHG 2         l0: nop         WRITEI
JZ l1f          WRITEI          PUSHG 2         PUSHS "\n"
PUSHG 2         PUSHS "\n"      PUSHI 1         WRITES
PUSHI 2         WRITES          ADD             STOP
MOD             PUSHG 3         STOREG 2
PUSHI 1         PUSHI 1         JUMP l1c
```

5.5 Ler e armazenar N números num array; imprimir os valores por ordem inversa

- Programa-fonte

```
int n <- 10
array a 10
int i <- 0
int temp

escrever n
enquanto nigual (i, n) faz
    a[i] <- ler
    i <- soma (i, 1)
fim

i <- 0
enquanto nigual (i, (div (n,2))) faz
    temp <- a[i]
    a[i] <- a[sub (sub(n,1), i)]
    a[sub (sub(n,1), i)] <- temp
    i <- soma (i, 1)
fim

escrevera a
```

- Código produzido

```
PUSHI 10        NOT             JUMP l0c        PUSHI 1
PUSHN 10        JZ l0f          l0f: NOP        PADD
PUSHI 0         PUSHGP          PUSHI 0         PUSHG 11
PUSHI 0         PUSHI 1         STOREG 11       LOADN
START           PADD            l1c: NOP        STOREG 12
PUSHG 0         PUSHG 11        PUSHG 11        PUSHGP
WRITEI          READ            PUSHG 0         PUSHI 1
PUSHS "\n"      ATOI            PUSHI 2         PADD
WRITES          STOREN          DIV             PUSHG 11
l0c: NOP        PUSHG 11        EQUAL           PUSHGP
PUSHG 11        PUSHI 1         NOT             PUSHI 1
PUSHG 0         ADD             JZ l1f          PADD
EQUAL           STOREG 11       PUSHGP          PUSHG 0
```

```
                LOADN           WRITES          PADD
PUSHI 1         WRITEI          PUSHGP          PUSHI 7
SUB             PUSHS " "       PUSHI 1         LOADN
PUSHG 11        WRITES          PADD            WRITEI
SUB             PUSHGP          PUSHI 4         PUSHS " "
LOADN           PUSHI 1         LOADN           WRITES
STOREN          PADD            WRITEI          PUSHGP
PUSHGP          PUSHI 1         PUSHS " "       PUSHI 1
PUSHI 1         LOADN           WRITES          PADD
PADD            WRITEI          PUSHGP          PUSHI 8
PUSHG 0         PUSHS " "       PUSHI 1         LOADN
PUSHI 1         WRITES          PADD            WRITEI
SUB             PUSHGP          PUSHI 5         PUSHS " "
PUSHG 11        PUSHI 1         LOADN           WRITES
SUB             PADD            WRITEI          PUSHGP
PUSHG 12        PUSHI 2         PUSHS " "       PUSHI 1
STOREN          LOADN           WRITES          PADD
PUSHG 11        WRITEI          PUSHGP          PUSHI 9
PUSHI 1         PUSHS " "       PUSHI 1         LOADN
ADD             WRITES          PADD            WRITEI
STOREG 11       PUSHGP          PUSHI 6         PUSHS " "
JUMP l1c        PUSHI 1         LOADN           WRITES
l1f: NOP        PADD            WRITEI          PUSHS "\n"
PUSHGP          PUSHI 3         PUSHS " "       WRITES
PUSHI 1         LOADN           WRITES          STOP
PADD            WRITEI          PUSHGP
PUSHI 0         PUSHS " "       PUSHI 1
```

5.6 Ler e armazenar N números num array; ordenar o array; imprimir os valores ordenados

- Programa-fonte

```
array a 10
int i
int j
int temp

enquanto (nigual (i, 10)) faz
    a[i] <- ler
    i <- soma (i, 1)
fim

i <- 0
enquanto (nigual (i, 10)) faz
    j <- soma (i,1)
    enquanto (nigual (j, 10)) faz
        se (maior (a[i], a[j])) entao
            temp <- a[i]
            a[i] <- a[j]
            a[j] <- temp
        fim
```

```
        j <- soma (j,1)
    fim
    i <- soma (i, 1)
fim

escrevera a
```

- Código produzido

```
PUSHN 10
PUSHI 0
PUSHI 0
PUSHI 0
START
l0c: NOP
PUSHG 10
PUSHI 10
EQUAL
NOT
JZ l0f
PUSHGP
PUSHI 0
PADD
PUSHG 10
READ
ATOI
STOREN
PUSHG 10
PUSHI 1
ADD
STOREG 10
JUMP l0c
l0f: NOP
PUSHI 0
STOREG 10
l3c: NOP
PUSHG 10
PUSHI 10
EQUAL
NOT
JZ l3f
PUSHG 10
PUSHI 1
ADD
STOREG 11
l2c: NOP
PUSHG 11
PUSHI 10
EQUAL
NOT
JZ l2f
PUSHGP
PUSHI 0
PADD
PUSHG 10
PUSHGP
PUSHI 0
PADD
PUSHG 11
LOADN
SUP
JZ l1
PUSHGP
PUSHI 0
PADD
PUSHG 10
LOADN
STOREG 12
PUSHGP
PUSHI 0
PADD
PUSHG 10
PUSHGP
PUSHI 0
PADD
PUSHG 11
LOADN
STOREN
PUSHGP
PUSHI 0
PADD
PUSHG 11
PUSHG 12
STOREN
l1: NOP
PUSHG 11
PUSHI 1
ADD
STOREG 11
JUMP l2c
l2f: NOP
PUSHG 10
PUSHI 1
ADD
STOREG 10
JUMP l3c
l3f: NOP
PUSHGP
PUSHI 0
PADD
PUSHI 0
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 1
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 2
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 3
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 4
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 5
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 6
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 7
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 8
LOADN
WRITEI
PUSHS " "
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 9
LOADN
WRITEI
PUSHS " "
WRITES
PUSHS "\n"
WRITES
STOP
```

5.7 Ler e armazenar N números numa matriz; imprimir a soma de cada linha

- Programa-fonte

```
matriz m 5 3
int i
int j
int som

enquanto menor (i, 5) faz
    j <- 0
    enquanto menor (j, 3) faz
        m[i,j] <- ler
        j <- soma (j, 1)
    fim
    i <- soma (i, 1)
fim

i <- 0
enquanto menor (i, 5) faz
    j <- 0
    som <- 0
    enquanto menor (j, 3) faz
        som <- soma (som, m[i,j])
        j <- soma (j, 1)
    fim
    escrever som
    i <- soma (i, 1)
fim
```

- Código produzido

```
PUSHN 15
PUSHI 0
PUSHI 0
PUSHI 0
START
l1c: NOP
PUSHG 15
PUSHI 5
INF
JZ l1f
PUSHI 0
STOREG 16
l0c: NOP
PUSHG 16
PUSHI 3
INF
JZ l0f
PUSHGP
PUSHI 0
PADD
PUSHG 15
PUSHI 3
MUL
PUSHG 16
ADD
READ
ATOI
STOREN
PUSHG 16
PUSHI 1
ADD
STOREG 16
JUMP l0c
l0f: NOP
PUSHG 15
PUSHI 1
ADD
STOREG 15
JUMP l1c
l1f: NOP
PUSHI 0
STOREG 15
l3c: NOP
PUSHG 15
PUSHI 5
INF
JZ l3f
PUSHI 0
STOREG 16
PUSHI 0
STOREG 17
l2c: NOP
PUSHG 16
PUSHI 3
INF
JZ l2f
PUSHG 17
PUSHGP
PUSHI 0
PADD
PUSHG 15
PUSHI 3
MUL
PUSHG 16
ADD
LOADN
ADD
STOREG 17
PUSHG 16
PUSHI 1
ADD
STOREG 16
JUMP l2c
l2f: NOP
PUSHG 17
WRITEI
```

```
                    PUSHG 15            STOREG 15           STOP
PUSHS "\n"          PUSHI 1             JUMP l3c
WRITES              ADD                 l3f: NOP
```

5.8 Tratamento de erros

Este exemplo mostra alguns erros que são identificados pelo compilador, nomeadamente a utilização de variáveis com o tipo errado.

```
int x
int y
array a 5
matriz m 2 2

escrevera x

escrever x[0]
x[0] <- ler
x[1] <- 1
y <- soma (x[0], x[1])

escrever x[0,0]
x[0,1] <- ler
x[1,0] <- 1
y <- soma (x[0,1], x[1,0])

escrever a[0]
a[0] <- ler
a[1] <- 1
escrevera a

escrever m[0,0]
m[0,1] <- ler
m[1,0] <- 1
escrevera m
```

```
carricossauro@carricossauro:~/Documents/LCC 3º ano/TP PLC/TP2$ python3 yacc.py testes/teste.txt
Erro: Variável não é um array.
Erro: Variável x não é um array.
Erro: Variável x não é um array.
Erro: Variável x não é um array.
Erro: Variável x não é um array.
Erro: Variável x não é um array.
Erro: Variável x não é uma matriz.
Erro: Variável x não é uma matriz.
Erro: Variável x não é uma matriz.
Erro: Variável x não é uma matriz.
Erro: Variável x não é uma matriz.
--------------------------------------
Erro ao compilar.
--------------------------------------
```

Capítulo 6

Conclusão

Durante a realização deste trabalho aplicamos todos os conhecimentos adquiridos durante as aulas sobre GIC's, YACC (gerador de compiladores baseado em gramáticas tradutoras) e Lex (gerador de analisadores léxicos) e dessa forma aprofundamos e consolidamos esses conhecimentos.

Consideramos que conseguimos alcançar os objetivos esperados, sendo que nos sentimos mais experientes e prontos no que toca a programação generativa (gramatical) e na aptidão para a escrita de gramáticas.

Este trabalho levou também a um maior conhecimento no que diz respeito á máquina virtual e a uma maior compreensão de escrita em `Assembly`.

Por fim, podemos garantir que este trabalho fundamentou as nossas bases para que futuramente consigamos tirar um maior proveito do que foi lecionado durante toda esta unidade curricular.

Apêndice A

Analisador Léxico

```
import ply.lex as lex
import sys

tokens = (
    'NUM',
    'NOME',
    'INT',
    'ARRAY',
    'MATRIZ',
    'SOMA',
    'SUB',
    'MULT',
    'DIV',
    'MOD',
    'MAIOR',
    'MENOR',
    'IGUAL',
    'MAIORI',
    'MENORI',
    'NIGUAL',
    'NEG',
    'E',
    'OU',
    'ATR',
    'LER',
    'ESCREVER',
    'SE',
    'ENTAO',
    'SENAO',
    'ENQUANTO',
    'FAZ',
    'FIM',
    'PCABRIR',
    'PCFECHAR',
    'PRABRIR',
    'PRFECHAR',
    'VIRG',
    'ESCREVERA'
)
```

```
t_PCABRIR = r'\('
t_PCFECHAR = r'\)'
t_PRABRIR = r'\['
t_PRFECHAR = r'\]'
t_VIRG   = r','

def t_NUM(t):
    r'\d+'
    return t

def t_INT(t):
    r'(?i:int)'
    return t

def t_ARRAY(t):
    r'(?i:array)'
    return t

def t_MATRIZ(t):
    r'(?i:matriz)'
    return t

def t_SOMA(t):
    r'(?i:soma)'
    return t

def t_SUB(t):
    r'(?i:sub)'
    return t

def t_MULT(t):
    r'(?i:mult)'
    return t

def t_DIV(t):
    r'(?i:div)'
    return t

def t_MOD(t):
    r'(?i:mod)'
    return t

def t_MAIORI(t):
    r'(?i:maiori)'
    return t

def t_MAIOR(t):
    r'(?i:maior)'
    return t

def t_MENORI(t):
    r'(?i:menori)'
    return t

def t_MENOR(t):
    r'(?i:menor)'
    return t
```

```
def t_IGUAL(t):
    r'(?i:igual)'
    return t

def t_NIGUAL(t):
    r'(?i:nigual)'
    return t

def t_NEG (t):
    r'(?i:neg)'
    return t

def t_OU (t):
    r'(?i:ou)'
    return t

def t_ATR (t):
    r'<-'
    return t

def t_LER (t):
    r'(?i:ler)'
    return t

def t_ESCREVERA(t):
    r'(?i:escrevera)'
    return t

def t_ESCREVER (t):
    r'(?i:escrever)'
    return t

def t_ENTAO(t):
    r'(?i:entao)'
    return t

def t_SENAO(t):
    r'(?i:senao)'
    return t

def t_SE (t):
    r'(?i:se)'
    return t

def t_ENQUANTO(t):
    r'(?i:enquanto)'
    return t

def t_FAZ(t):
    r'(?i:faz)'
    return t

def t_FIM(t):
    r'(?i:fim)'
    return t

def t_E (t):
    r'(?i:e)'
```

```
    return t

def t_NOME(t):
    r'[a-z]\w*'
    return t

t_ignore = ' \r\t\n'

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


lexer = lex.lex()
```

Apêndice B

Compilador

```
import ply.yacc as yacc
import sys

from lex import tokens

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

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

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

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

def p_Decl_Int(p):
    "Decl     : INT NOME"
    if p[2] not in p.parser.registers:
        p.parser.registers.update({p[2] : p.parser.gp})
        p[0] = f'PUSHI 0\n'
        p.parser.ints.append(p[2])
        p.parser.gp += 1
    else:
        print("Erro: Variável já existe.")
        parser.success = False

def p_Decl_Int_Atr(p):
    "Decl     : INT NOME ATR NUM"
    if p[2] not in p.parser.registers:
        p.parser.registers.update({p[2] : p.parser.gp})
        p[0] = f'PUSHI {p[4]}\n'
        p.parser.ints.append(p[2])
        p.parser.gp += 1
    else:
```

```
        print("Erro: Variável já existe.")
        parser.success = False

def p_Decl_Array(p):
    "Decl     : ARRAY NOME NUM"
    if p[2] not in p.parser.registers:
        p.parser.registers.update({p[2] : (p.parser.gp, int(p[3]))})
        p[0] = f'PUSHN {p[3]}\n'
        p.parser.gp += int(p[3])
    else:
        print("Erro: Variável já existe.")
        parser.success = False

def p_Decl_Matriz(p):
    "Decl     : MATRIZ NOME NUM NUM"
    if p[2] not in p.parser.registers:
        p.parser.registers.update({p[2] : (p.parser.gp, int(p[3]), int(p[4]))})
        size = int(p[3])*int(p[4])
        p[0] = f'PUSHN {str(size)}\n'
        p.parser.gp += size
    else:
        print("Erro: Variável já existe.")
        parser.success = False

def p_Corpo(p):
    "Corpo    : Proc"
    p[0] = p[1]

def p_Corpo_Recursiva(p):
    "Corpo    : Corpo Proc"
    p[0] = f'{p[1]}{p[2]}'

def p_Proc_Atrib(p):
    "Proc     : Atrib"
    p[0] = p[1]

def p_Proc_Escrever(p):
    "Proc     : Escrever"
    p[0] = p[1]

def p_Proc_Se(p):
    "Proc     : Se"
    p[0] = p[1]

def p_Proc_Enquanto(p):
    "Proc     : Enquanto"
    p[0] = p[1]

def p_Se(p):
    "Se       : SE Cond ENTAO Corpo FIM"
    p[0] = f'{p[2]}JZ l{p.parser.labels}\n{p[4]}l{p.parser.labels}: NOP\n'
    p.parser.labels += 1

def p_Se_Senao(p):
    "Se       : SE Cond ENTAO Corpo SENAO Corpo FIM"
    p[0] = f'{p[2]}JZ l{p.parser.labels}\n{p[4]}JUMP l{p.parser.labels}f\nl{p.parser.labels}:
    NOP\n{p[6]}l{p.parser.labels}f: NOP\n'
    p.parser.labels += 1
```

```
def p_Enquanto(p):
    "Enquanto : ENQUANTO Cond FAZ Corpo FIM"
    p[0] = f'l{p.parser.labels}c: NOP\n{p[2]}JZ l{p.parser.labels}f\n{p[4]}JUMP l{p.parser.labels}
            c\nl{p.parser.labels}f: NOP\n'
    p.parser.labels += 1

def p_Atrib_expr_Int(p):
    "Atrib    : NOME ATR Expr"
    if p[1] in p.parser.registers:
        if p[1] in p.parser.ints:
            p[0] = f'{p[3]}STOREG {p.parser.registers.get(p[1])}\n'
        else:
            print("Erro: Variável não é de tipo inteiro.")
            parser.success = False

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

def p_Atrib_expr_Array(p):
    "Atrib    : NOME PRABRIR Expr PRFECHAR ATR Expr"
    if p[1] in p.parser.registers:
        if p[1] not in p.parser.ints and len(p.parser.registers.get(p[1])) == 2:
            p[0] = f'PUSHGP\nPUSHI {p.parser.registers.get(p[1])[0]}\nPADD\n{p[3]}{p[6]}STOREN\n'
        else:
            print(f"Erro: Variável {p[1]} não é um array.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_Atrib_expr_Matriz(p):
    "Atrib    : NOME PRABRIR Expr VIRG Expr PRFECHAR ATR Expr"
    if p[1] in p.parser.registers:
        if p[1] not in p.parser.ints and len(p.parser.registers.get(p[1])) == 3:
            c = p.parser.registers.get(p[1])[2]
            p[0] = f'PUSHGP\nPUSHI {p.parser.registers.get(p[1])[0]}\nPADD\n{p[3]}PUSHI {c}\n
                    MUL\n{p[5]}ADD\n{p[8]}STOREN\n'
        else:
            print(f"Erro: Variável {p[1]} não é uma matriz.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_Atrib_Ler_Array(p):
    "Atrib    : NOME PRABRIR Expr PRFECHAR ATR LER"
    if p[1] in p.parser.registers:
        if p[1] not in p.parser.ints and len(p.parser.registers.get(p[1])) == 2:
            p[0] = f'PUSHGP\nPUSHI {p.parser.registers.get(p[1])[0]}\nPADD\n{p[3]}READ\nATOI\nSTOREN\n'
        else:
            print(f"Erro: Variável {p[1]} não é um array.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False
```

```
def p_Atrib_Ler_Matriz(p):
    "Atrib    : NOME PRABRIR Expr VIRG Expr PRFECHAR ATR LER"
    if p[1] in p.parser.registers:
        if p[1] not in p.parser.ints and len(p.parser.registers.get(p[1])) == 3:
            c = p.parser.registers.get(p[1])[2]
            p[0] = f'PUSHGP\nPUSHI {p.parser.registers.get(p[1])[0]}\nPADD\n{p[3]}PUSHI {c}\n
                    MUL\n{p[5]}ADD\nREAD\nATOI\nSTOREN\n'
        else:
            print(f"Erro: Variável {p[1]} não é uma matriz.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_Atrib_Ler(p):
    "Atrib    : NOME ATR LER"
    if p[1] in p.parser.registers:
        p[0] = f'READ\nATOI\nSTOREG {p.parser.registers.get(p[1])}\n'
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_Escrever_a(p):
    "Escrever : ESCREVERA NOME"
    if p[2] in p.parser.registers:
        if p[2] not in p.parser.ints:
            if len(p.parser.registers.get(p[2])) == 2:
                array = ""
                for i in range(p.parser.registers.get(p[2])[1]):
                    array += f'PUSHGP\nPUSHI {p.parser.registers.get(p[2])[0]}\nPADD\nPUSHI {i}\n
                                LOADN\nWRITEI\nPUSHS " "\nWRITES\n'
                p[0] = array + 'PUSHS "\\n"\nWRITES\n'
            else:
                matriz = ""
                for l in range(p.parser.registers.get(p[2])[1]):
                    for c in range(p.parser.registers.get(p[2])[2]):
                        matriz += f'PUSHGP\nPUSHI {p.parser.registers.get(p[2])[0]}\nPADD\n
                                    PUSHI {p.parser.registers.get(p[2])[2] * l + c}\nLOADN\n
                                    WRITEI\nPUSHS " "\nWRITES\n'
                    matriz += 'PUSHS "\\n"\nWRITES\n'
                p[0] = matriz

        else:
            print("Erro: Variável não é um array.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

def p_Escrever(p):
    "Escrever : ESCREVER Expr"
    p[0] = f'{p[2]}WRITEI\nPUSHS "\\n"\nWRITES\n'

def p_Expr_P(p):
    "Expr     : PCABRIR Expr PCFECHAR"
    p[0] = p[2]

def p_Expr_Var(p):
```

```
    "Expr     : Var"
    p[0] = p[1]

def p_Expr_Num(p):
    "Expr     : NUM"
    p[0] = f'PUSHI {p[1]}\n'

def p_Expr_Soma(p):
    "Expr     : SOMA PCABRIR Expr VIRG Expr PCFECHAR"
    p[0] = f'{p[3]}{p[5]}ADD\n'

def p_Expr_Sub(p):
    "Expr     : SUB PCABRIR Expr VIRG Expr PCFECHAR"
    p[0] = f'{p[3]}{p[5]}SUB\n'

def p_Expr_Mult(p):
    "Expr     : MULT PCABRIR Expr VIRG Expr PCFECHAR"
    p[0] = f'{p[3]}{p[5]}MUL\n'

def p_Expr_Div(p):
    "Expr     : DIV PCABRIR Expr VIRG Expr PCFECHAR"
    p[0] = f'{p[3]}{p[5]}DIV\n'

def p_Expr_Mod(p):
    "Expr     : MOD PCABRIR Expr VIRG Expr PCFECHAR"
    p[0] = f'{p[3]}{p[5]}MOD\n'

def p_Expr_Cond(p):
    "Expr     : Cond"
    p[0] = p[1]

def p_Cond_P(p):
    "Cond     : PCABRIR Cond PCFECHAR"
    p[0] = p[2]

def p_Cond_Maior(p):
    "Cond     : MAIOR PCABRIR Expr VIRG Expr PCFECHAR"
    p[0] = f'{p[3]}{p[5]}SUP\n'

def p_Cond_Menor(p):
    "Cond     : MENOR PCABRIR Expr VIRG Expr PCFECHAR"
    p[0] = f'{p[3]}{p[5]}INF\n'

def p_Cond_Maiori(p):
    "Cond     : MAIORI PCABRIR Expr VIRG Expr PCFECHAR"
    p[0] = f'{p[3]}{p[5]}SUPEQ\n'

def p_Cond_Menori(p):
    "Cond     : MENORI PCABRIR Expr VIRG Expr PCFECHAR"
    p[0] = f'{p[3]}{p[5]}INFEQ\n'

def p_Cond_Igual(p):
    "Cond     : IGUAL PCABRIR Expr VIRG Expr PCFECHAR"
    p[0] = f'{p[3]}{p[5]}EQUAL\n'

def p_Cond_Nigual(p):
    "Cond     : NIGUAL PCABRIR Expr VIRG Expr PCFECHAR"
    p[0] = f'{p[3]}{p[5]}EQUAL\nNOT\n'
```

```
def p_Cond_E(p):
    "Cond     : E PCABRIR Cond VIRG Cond PCFECHAR"
    p[0] = f'{p[3]}{p[5]}ADD\nPUSHI 2\nEQUAL\n'

def p_Cond_Ou(p):
    "Cond     : OU PCABRIR Cond VIRG Cond PCFECHAR"
    p[0] = f'{p[3]}{p[5]}ADD\nPUSHI 1\nSUPEQ\n'

def p_Cond_Neg(p):
    "Cond     : NEG PCABRIR Cond PCFECHAR"
    p[0] = f'{p[3]}NOT\n'

def p_Var_Matriz(p):
    "Var      : NOME PRABRIR Expr VIRG Expr PRFECHAR"
    if p[1] in p.parser.registers:
        if p[1] not in p.parser.ints and len(p.parser.registers.get(p[1])) == 3:
            c = p.parser.registers.get(p[1])[2]
            p[0] = f'PUSHGP\nPUSHI {p.parser.registers.get(p[1])[0]}\nPADD\n{p[3]}PUSHI {c}\n
                    MUL\n{p[5]}ADD\nLOADN\n'
        else:
            print(f"Erro: Variável {p[1]} não é uma matriz.")
            parser.success = False
    else:
        print("Erro: Variável não definida.")
        parser.success = False

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

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

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

#---------------------------------------
#inicio do Parser
```

```
parser = yacc.yacc()

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

try:
    if len(sys.argv) > 1:
        with open(sys.argv[1],'r') as file:
            inp = file.read()
            parser.parse(inp)
            if parser.success:
                if len(sys.argv) > 2:
                    with open(sys.argv[2], 'w') as output:
                        output.write(parser.assembly)
                        print("---------------------------------------")
                        print(f"Ficheiro {sys.argv[1]} compilado com sucesso.\n
                                Output guardado em {sys.argv[2]}.")
                        print("---------------------------------------")
                else:
                    print(parser.assembly)
            else:
                print("---------------------------------------")
                print("Erro ao compilar.")
                print("---------------------------------------")
    else:
        for line in sys.stdin:
            parser.success = True
            parser.registers = {}
            parser.labels = 0
            parser.gp = 0
            parser.ints = []
            parser.assembly = ""
            parser.parse(line)
            if parser.success:
                print("---------------------------------------")
                print(parser.assembly)
                print("---------------------------------------")
            else:
                print("---------------------------------------")
                print("Erro ao compilar.")
                print("---------------------------------------")
except KeyboardInterrupt:
    print()
```