

Processamento de Linguagens e Compiladores

Trabalho Prático 2

Relatório de Desenvolvimento

Miguel Gonçalves
a90416

João Nogueira
a87973

Rui Baptista
a87989

16 de janeiro de 2022

Resumo

O seguinte relatório documenta, justifica, analisa e expõe todas as decisões tomadas ao longo do Trabalho Prático 2 realizado no âmbito da Unidade Curricular denominada Processamento de Linguagens e Compiladores no contexto do 3º ano do curso Licenciatura em Ciências da Computação. O seguinte trabalho tem como principais objetivos aumentar a experiência em engenharia de linguagens e em programação generativa, desenvolver processadores de linguagens segundo o método da tradução dirigida pela sintaxe, desenvolver um compilador gerando código para uma máquina de stack virtual e utilizar geradores de compiladores baseados em gramáticas tradutoras.

Conteúdo

1	Introdução	3
1.1	Compilador em Yacc	3
2	Análise do problema	4
2.1	Descrição informal do problema	4
3	Analizador léxico (Flex)	5
4	Concepção da Solução (Yacc)	7
4.1	Declaração de variáveis	7
4.2	Calculo de expressões	8
4.3	Standard input e output	9
4.4	Ciclos	9
4.4.1	Ciclo For	9
4.4.2	Ciclo If	10
4.4.3	Ciclo While	10
4.5	Arrays	11
5	Gerar código Assembly	12
6	Testes e Exemplos	13
6.1	Ler 4 números e dizer se podem ser os lados de um quadrado	13
6.1.1	Código	13
6.1.2	Assembly gerado	13
6.1.3	Máquina virtual VM	15
6.2	Ler um inteiro N, Depois ler N números e escrever o menor deles	16
6.2.1	Código	16
6.2.2	Assembly gerado	17
6.2.3	Máquina virtual VM	18
6.3	Ler N (constante do programa) números e calcular e imprimir o seu produtório	19
6.3.1	Código	19
6.3.2	Assembly gerado	19
6.3.3	Máquina virtual VM	20
6.4	Contar e imprimir os números impares de uma sequência de números naturais	21

6.4.1	Código	21
6.4.2	Assembly gerado	21
6.4.3	Máquina virtual VM	22
6.5	Ler e armazenar N números num array e imprimir os valores por ordem inversa	23
6.5.1	Código	23
6.5.2	Assembly gerado	23
6.5.3	Máquina virtual VM	25
7	Conclusão	26
A	Código Flex	27
B	Código Yacc	29

Capítulo 1

Introdução

1.1 Compilador em Yacc

Este trabalho tem como objetivo desenvolver um compilador para a nossa própria linguagem, fazendo a tradução entre o código da nossa linguagem, de acordo com as regras estipuladas para a mesma, para código Assembly da VM. De forma a desenvolver uma linguagem imperativa simples, foram tomadas decisões à cerca de como apresentar e declarar as variáveis inteiras, como definir as habituais operações aritméticas, relacionais e lógicas, e ainda outras “regras” essenciais para o iminente funcionamento da linguagem criada. Todas estas decisões foram devidamente discutidas e tomadas em grupo, com unanimidade. Para melhor compreensão de como funciona a linguagem desenvolvida, serão apresentados alguns exemplos da própria em funcionamento.

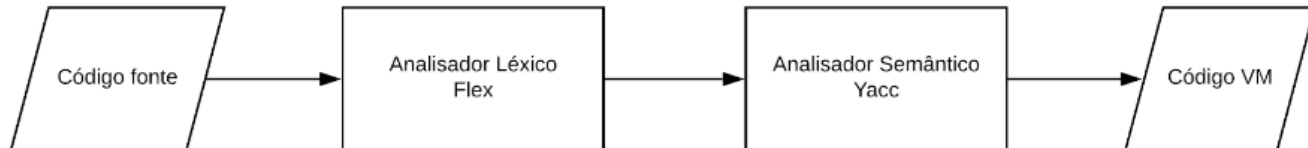


Figura 1. Fluxograma que ilustra o processo de tradução código fonte para código VM

Capítulo 2

Análise do problema

2.1 Descrição informal do problema

No enunciado era pedido para criar uma linguagem imperativa a nosso gosto, mas que devia permitir fazer o seguinte:

1. Declarar variáveis atômicas do tipo inteiro, com os quais se podem realizar as habituais operações aritméticas.
2. Efetuar instruções algorítmicas básicas como a atribuição do valor de expressões numéricas a variáveis
3. Ler do stdin e escrever no stdout
4. Efetuar instruções cíclicas para controlo do fluxo de execução, permitindo o seu aninhamento (Implementar pelo menos o ciclo while-do, repeat-until ou for-do).

Adicionalmente deve ainda suportar, à sua escolha, uma das duas funcionalidades seguintes:

5. Declarar e manusear variáveis estruturadas do tipo array (a 1 ou 2 dimensões) de inteiros.
6. Definir e invocar subprogramas sem parâmetros mas que possam retornar um resultado do tipo inteiro.

Relativamente à escolha das 2 funcionalidades extra, decidimos escolher declarar e manusear variáveis estruturadas do tipo array.

Capítulo 3

Analizador léxico (Flex)

Inicialmente foi criado um analisador léxico, utilizando a ferramenta flex, que permite processar o código fonte, contendo um programa escrito na nossa linguagem. Durante este processamento, são retornados diferentes tokens, correspondentes a expressões regulares que identificam padrões do nosso código.

Palavras Reservadas

Ao longo do código, existem expressões utilizadas diretamente pela linguagem, designadas por "palavras reservadas".

```
reserved = {
    'se' : 'IF',
    'else' : 'ELSE',
    'enquanto' : 'WHILE',
    'num' : 'DCLN',
    'string' : 'DCLS',
    'desde' : 'FOR',
    'ate' : 'UNTIL',
    'imprime' : 'PRINT',
    'le' : 'SCAN',
    'devolve' : 'RETURN',
    'lista' : 'ARRAY'
}

tokens = [
    'LPAREN',
    'RPAREN',
    'VIRG',
    'ID',
    'BOOL',
    'NUM',
    'SOMA',
    'SUBTRACAO',
    'MULTIPLICACAO',
    'DIVISAO',
    'IGUAL',
    'CONJUNCAO',
    'DISJUNCAO',
    'DOISPONTOS',
    'MAIOR',
```

```

'MENOR' ,
'ASPAS' ,
'REAL' ,
'PRE' ,
'PRD' ,
'MOD' ,

```

```
]
```

Quando estas são identificadas, os seus tokens são devolvidos.

```

t_MOD      = r'% '
t_PRE      = r'\['
t_PRD      = r'\]'
t_ASPAS    = r'" '
t_DOISPONTOS = r'\:'
t_IGUAL     = r'\='
t_LPAREN    = r'\('
t_HPAREN   = r'\)'
t_VIRG     = r','
t_CONJUNCAO = r'\&'
t_DISJUNCAO = r'\| '
t_SOMA      = r'\+'
t_SUBTRACAO = r'\-'
t_MULTIPLICACAO = r'\*'
t_DIVISAO   = r'\/'
t_MAIOR     = r'\>'
t_MENOR     = r'\<'

def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    return t

def t_REAL(t):
    r'([1-9]+\.[0-9]+)|(0\.[0-9]+)'
    return t

def t_NUM(t):
    r'([0-9]+)'
    return t

def t_BOOL(t):
    r'True|False'
    return t

```


Capítulo 4

Concepção da Solução (Yacc)

A nossa linguagem é baseada na grande maioria em pseudocódigo, acreditamos que, a nosso ver, torna certas funcionalidades mais simpáticas e simples de serem escritas.

4.1 Declaração de variáveis

A linguagem permite a declaração de variáveis do tipo número, string e arrays. Em Yacc definimos que podemos atribuir um valor a estas variáveis ou sempre que é declarada sem ser feita essa atribuição, é-lhe atribuído o valor 0.

Vamos ver agora um exemplo de uma declaração de variável do tipo número sem ser feito qualquer atribuição. Esta implementação está escrita em Yacc da seguinte forma:

```
def p_declararNumSolo(p):  
    'declara : DCLN ID'
```

Assim temos:

```
num i
```

Podemos também atribuir um valor a essa variável adicionando "IGUAL" e "expression" à implementação em Yacc.

```
def p_declararNum(p):  
    'declara : DCLN ID IGUAL expression'
```

Desta forma, temos:

```
num i = 1
```

4.2 Cálculo de expressões

A nossa linguagem permite as habituais operações aritméticas, relacionais e lógicas.

1. Típicas operações de verificação de igualdades/desigualdades (" $==$ ", (" $<$ ", (" $>$ ", (" \leq ") e (" \geq ").
2. Multiplicação (" $*$ "), divisão (" $/$ ") e módulo (" $\%$ ").
3. Soma (" $+$ ") e subtração (" $-$ ").
4. Operações booleanas de conjunção (" $\&$ ") e disjunção (" $|$ ").

Por exemplo, para soma, subtração e divisão a implementação em Yacc foi feita da seguinte forma:

```
def p_soma(p):
    'expression : expression SOMA term'

def p_somaId(p):
    'expression : expression SOMA ID'

def p_subtracao(p):
    'expression : expression SUBTRACAO term'

def p_subtracaoId(p):
    'expression : expression SUBTRACAO ID'

def p_divisao(p):
    'expression : expression DIVISAO term'
def p_divisaoId(p):
    'expression : expression DIVISAO ID'
```

No Apêndice B encontra-se o restante código Yacc para o resto das operações,

A título de exemplo, uma atribuição a uma variável pode ser feita da seguinte forma:

```
num x = 6
num y = x%2

se y==0:
    (num c = c+1)
```

4.3 Standard input e output

Tal como é pedido no enunciado do trabalho, a nossa linguagem permite ler do standard input e escrever no standard output. Seguindo o objetivo da linguagem, que é ser simpática e simples de escrever, no caso de ler do standard input, definimos da seguinte forma em Yacc:

```
def p_scan(p):  
    'scan : SCAN LPAREN ID RPAREN'
```

Ou seja, se quisermos ler, por exemplo, um tipo inteiro podemos fazer da seguinte forma:

```
num n  
le (n)
```

Para escrever no standard output, temos várias opções na linguagem. Podemos escrever um termo, uma expressão, um id ou fazer um composto que contém um id e expressão. Em Yacc estão definidos da seguinte forma:

```
def p_printId(p):  
    'print : PRINT LPAREN ID RPAREN'  
  
def p_print(p):  
    'print : PRINT LPAREN ASPAS ID ASPAS RPAREN'  
  
def p_printN(p):  
    'print : PRINT LPAREN ASPAS term ASPAS RPAREN'  
  
def p_printComposto(p):  
    'print : PRINT LPAREN ASPAS ID ASPAS VIRG expression RPAREN'  
  
def p_printExpressao(p):  
    'print : PRINT LPAREN expression RPAREN'
```

Desta forma, permite-nos fazer os seguintes exemplos:

```
num x = 5  
imprime(x)
```

```
imprime("Texto aqui")
```

4.4 Ciclos

Falando agora de instruções cíclicas, no enunciado do problema pedia para implementar pelo menos o ciclo while-do, repeat-until ou for-do. O nosso grupo decidiu implementar os 3 tipos de ciclos.

4.4.1 Ciclo For

Queremos que os ciclos sejam simpáticos de escrever e fáceis de entender. Sendo assim, o ciclo for foi implementado em Yacc da seguinte maneira:

```
def p_cicloFor(p):
    'for : FOR ID UNTIL comparacoes VIRG sinal NUM DOISPONTOS LPAREN statements RPAREN'
```

Como exemplo desta implementação temos o seguinte código:

```
num i
desde i ate i<6,+1: (imprime(i))
```

Ou seja, desde i até $i < 6$ em incrementos de 1.

Ao adicionar aquele "sinal" à implementação do Yacc significa que o nosso ciclo for consegue também andar para trás, sendo possível fazer algo assim:

```
num i = 5
desde i ate i>0,-1: (imprime(i))
```

Como podemos verificar, a linguagem permite a fácil interpretação do código, um dos nossos objetivos principais.

4.4.2 Ciclo If

Para a implementação do ciclo If temos 2 opções, um If que contém depois um Else e outro que tem apenas If. Ambos estão escritos em Yacc da seguinte forma:

```
def p_cicloIf(p):
    'if : IF tipos DOISPONTOS LPAREN statements RPAREN ELSE LPAREN statements RPAREN'

def p_cicloIfSolo(p):
    'if : IF tipos DOISPONTOS LPAREN statements RPAREN'
```

Para exemplo do ciclo If, temos o seguinte código:

```
num x = 0
num y

se x==1: (y=2) else (y=x)

devolve(y)
```

4.4.3 Ciclo While

O ciclo While está escrito em Yacc da seguinte forma:

```
def p_cicloWhile(p):
    'while : WHILE tipos DOISPONTOS LPAREN statements RPAREN'
```

Como exemplo temos o seguinte código:

```
num i

enquanto i < 2: (i = i + 1)

devolve(i)
```

4.5 Arrays

Como já tínhamos falado no capítulo de Declaração de variáveis, a nossa linguagem permite também declarar e manusear variáveis estruturadas do tipo array. A declaração de um array está escrita em Yacc da seguinte maneira:

```
def p_declararArray(p):
    'declara : ARRAY PRE NUM PRD ID'
```

Permitindo declarações assim:

```
lista [10] v
```

De forma a permitir manusear estas variáveis, através de atribuições, implementamos em Yacc as seguintes definições:

```
def p_atribuicaoArray(p):
    'atribuicao : ID PRE NUM PRD IGUAL expression'
```

```
def p_atribuicaoArrayID(p):
    'atribuicao : ID PRE ID PRD IGUAL expression'
```

```
def p_atribuicaoIdArray(p):
    'atribuicao : ID IGUAL ID PRE ID PRD'
```

```
def p_atribuicaoNumArray(p):
    'atribuicao : ID IGUAL ID PRE NUM PRD'
```

Desta forma, a linguagem permite fazer atribuições do seguinte tipo:

```
lista [10] v
num x = 5
num i = 1

v[i] = x

x = v[i]
```

Capítulo 5

Gerar código Assembly

Um ponto importante do trabalho é o compilador gerar automaticamente código Assembly, para depois ser usado pela VM. Para tal, aplicamos os conhecimentos adquiridos nas aulas e ao longo do código Yacc fomos adicionando os respetivos comandos assembly, como é visível no apêndice B.

Para provar o bom funcionamento do programa, temos o seguinte exemplo onde vamos declarar duas variáveis e realizar um ciclo, o compilador vai então gerar o seguinte código assembly:

```
num x = 0
num y
se x==1: (y=2) else (y=x)
START
PUSHI 0

PUSHI 0
PUSHG 0

PUSHI 1
EQUAL
JZ L0
PUSHI 2
STOREG 1

JUMP L1
L0:
PUSHG 0
STOREG 1

L1:
STOP
```

Figura 2. Exemplo código assembly

Para tal, o ciclo If está implementado em Yacc da seguinte forma:

```
def p_cicloIf(p):
    'if : IF tipos DOISPONTOS LPAREN statements RPAREN ELSE LPAREN statements RPAREN'
    global nroIf
    aux = nroIf + 1
    p[0] = p[2] + r'JZ L' + str(nroIf) + '\n' + p[5] + '\nJUMP L' + str(aux) + '\nL' +
    str(nroIf) + ':\n' + p[9] + '\nL' + str(aux) + ':\n'
```

Capítulo 6

Testes e Exemplos

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

6.1.1 Código

Para este problema temos o seguinte código escrito na nossa linguagem:.

```
num x
num y
num z
num w
imprime(" variavel")
le(x)
imprime(" variavel")
le(y)
imprime(" variavel")
le(z)
imprime(" variavel")
le(w)
num r = 0
num soma = 0
se x==y:(soma = soma +1)
se y==z:(soma = soma +1)
se z==w:(soma = soma +1)
se soma==3:(r=1)
devolve(r)
```

6.1.2 Assembly gerado

A linguagem vai gerar o seguinte código Assembly:

```
START
PUSHI 0
PUSHI 0
PUSHI 0
PUSHI 0
PUSHI 0
PUSHS " variavel"
WRITES
```

```
READ
ATOI
STOREG 0
PUSHS "variavel"
WRITES
READ
ATOI
STOREG 1
PUSHS "variavel"
WRITES
READ
ATOI
STOREG 2
PUSHS "variavel"
WRITES
READ
ATOI
STOREG 3
PUSHI 0

PUSHI 0

PUSHG 0

PUSHG 1
EQUAL
JZ L0
PUSHG 5

PUSHI 1

ADD
STOREG 5

JUMP L0
L0:
PUSHG 1

PUSHG 2
EQUAL
JZ L2
PUSHG 5

PUSHI 1

ADD
STOREG 5

JUMP L2
L2:
PUSHG 2

PUSHG 3
EQUAL
```



```

JZ L4
PUSHG 5

PUSHI 1

ADD
STOREG 5

JUMP L4
L4:
PUSHG 5

PUSHI 3
EQUAL
JZ L6
PUSHI 1
STOREG 4

JUMP L6
L6:
PUSHG 4

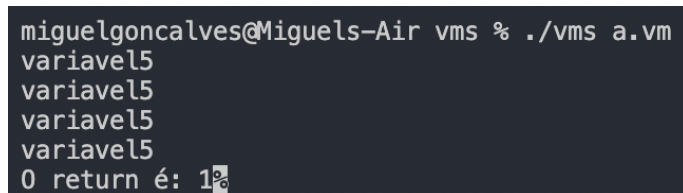
PUSHS "O return : "
WRITES
WRITEI
STOP

```

6.1.3 Máquina virtual VM

Vamos agora passar o ficheiro com o código assembly gerado à VM e ver o seu comportamento. A VM vai pedir para introduzir 4 valores e dá return 1 se forem todos iguais (lados de um quadrado) e 0 senão. Para correr a VM em modo de texto fazemos:

```
> ./vms [nome do ficheiro]
```

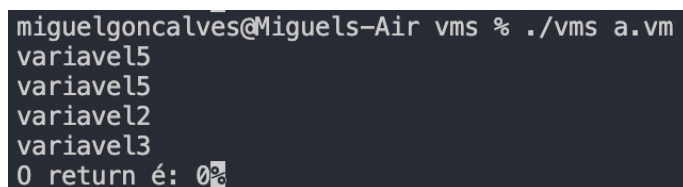


```

miguelgoncalves@Miguels-Air vms % ./vms a.vm
variavel5
variavel5
variavel5
variavel5
0 return é: 1%

```

Figura 3. Exemplo do programa 6.1 na VM - Modo texto



```

miguelgoncalves@Miguels-Air vms % ./vms a.vm
variavel5
variavel5
variavel2
variavel3
0 return é: 0%

```

Figura 4. Exemplo do programa na 6.1 VM - Modo texto

Podemos também correr a maquina em modo gráfico com:

```
> ./vms -g [nome do ficheiro]
```

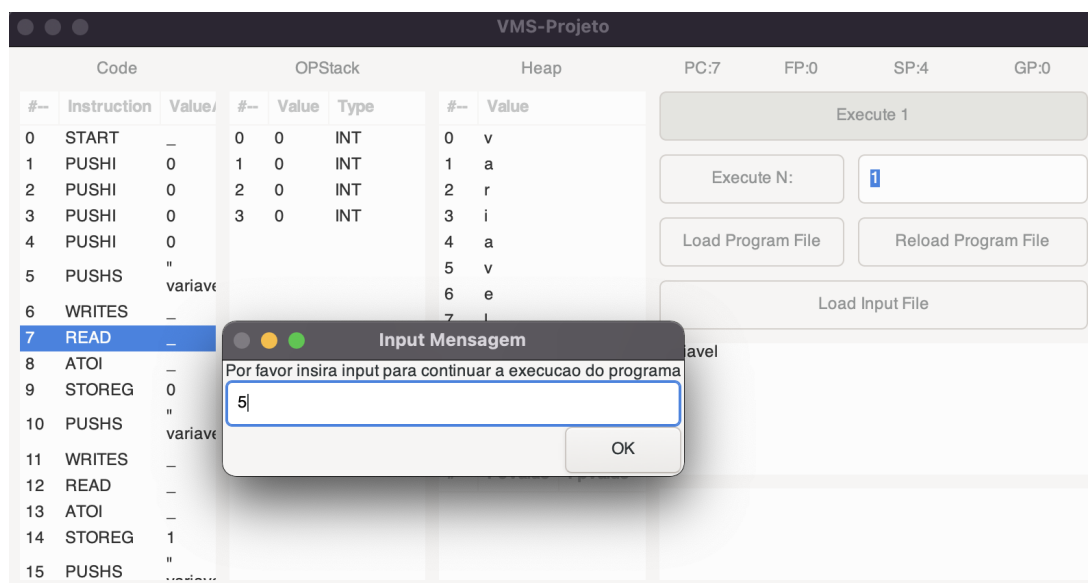


Figura 5. Exemplo do programa 6.1 na VM - Modo Gráfico

6.2 Ler um inteiro N, Depois ler N números e escrever o menor deles

6.2.1 Código

Para este problema temos o seguinte código escrito na nossa linguagem:

```
num n
le(n)
num y = 0
num m
num x
num i = 1
desde i ate i<n+1,+1:
(
  le(x)
  se y==0: (
    m = x
    y = 1
  )
  else (
    se x<m: (
      m = x
    )
  )
)
)
devolve(m)
```

6.2.2 Assembly gerado

A linguagem vai gerar o seguinte código Assembly:

```
START
PUSHI 0
READ
ATOI
STOREG 0
PUSHI 0

PUSHI 0
PUSHI 0
PUSHI 1

CICLO0:
PUSHG 4

PUSHG 0

PUSHI 1

ADD
INF
JZ FIM0
READ
ATOI
STOREG 3
PUSHG 1

PUSHI 0
EQUAL
JZ L2
PUSHG 3
STOREG 2
PUSHI 1
STOREG 1

JUMP L3
L2:
PUSHG 3

PUSHG 2
INF
JZ L0
PUSHG 3
STOREG 2

JUMP L0
L0:

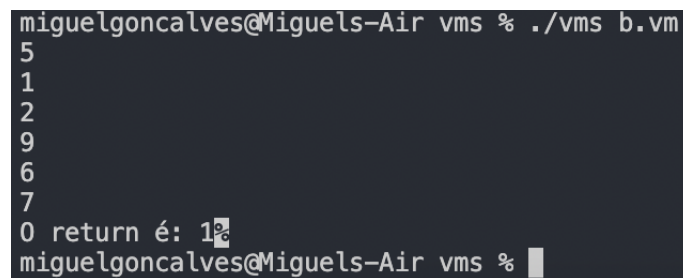
L3:
PUSHG 4
PUSHI 1
ADD
```

```
STOREG 4
JUMP CICLO0
FIM0:
PUSHG 2

PUSHS "O return : "
WRITES
WRITEI
STOP
```

6.2.3 Máquina virtual VM

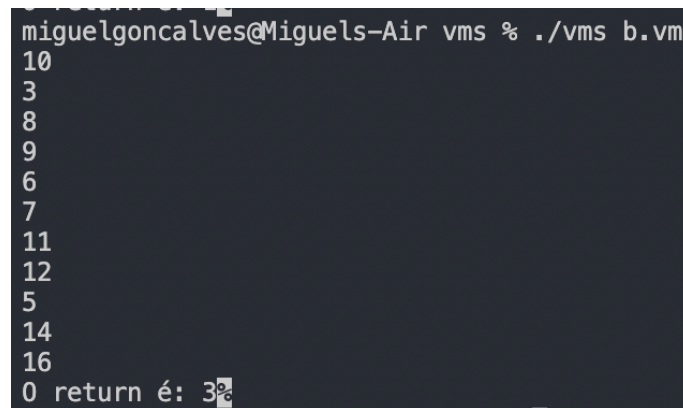
Como pedido, introduzimos primeiramente um inteiro N, neste caso 5, e depois introduzimos 5 números (1,2,9,6 e 7) e vai return ao menor, neste caso é o 1.



```
miguelgoncalves@Miguels-Air vms % ./vms b.vm
5
1
2
9
6
7
O return é: 1
miguelgoncalves@Miguels-Air vms %
```

Figura 6. Exemplo do programa 6.2 na VM - Modo texto

Outro exemplo, agora sendo o $N = 10$ e os numeros introduzidos 3,8,9,6,7,11,12,5,14,16.



```
miguelgoncalves@Miguels-Air vms % ./vms b.vm
10
3
8
9
6
7
11
12
5
14
16
O return é: 3
miguelgoncalves@Miguels-Air vms %
```

Figura 7. Exemplo do programa 6.2 na VM - Modo texto

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

6.3.1 Código

Para este problema temos o seguinte código escrito na nossa linguagem:

```
num n
le (n)
num i = 1
num x
num r = 1

desde i ate i<n+1,+1: (
    le (x)
    num r = r * x
)

devolve(r)
```

6.3.2 Assembly gerado

A linguagem vai gerar o seguinte código Assembly:

```
START
PUSHI 0
READ
ATOI
STOREG 0
PUSHI 1

PUSHI 0
PUSHI 1

CICLO0:
PUSHG 1

PUSHG 0

PUSHI 1

ADD
INF
JZ FIM0
READ
ATOI
STOREG 2
PUSHG 3

PUSHG 2
MUL
```

```

STOREG 3
PUSHG 1
PUSHI 1
ADD

STOREG 1
JUMP CICLO0
FIM0:
PUSHG 3

PUSHS "O return      : "
WRITES
WRITEI
STOP

```

6.3.3 Máquina virtual VM

Começamos então por introduzir um N, neste exemplo será 5, e depois 5 vezes o número 3. Como podemos verificar $3 \times 3 \times 3 \times 3 \times 3 = 243$.

```

miguelgoncalves@Miguels-Air vms % ./vms c.vm
5
3
3
3
3
3
3
0 return é: 243%

```

Figura 8. Exemplo do programa 6.3 na VM - Modo texto

Sendo agora $N = 3$ e os números introduzidos 3,2,1. Logo $3 \times 2 \times 1 = 6$.

```

miguelgoncalves@Miguels-Air vms % ./vms c.vm
3
3
2
1
0 return é: 6%

```

Figura 9. Exemplo do programa 6.3 na VM - Modo texto

6.4 Contar e imprimir os números ímpares de uma sequência de números naturais

6.4.1 Código

Para este problema temos o seguinte código escrito na nossa linguagem:

```
num n
le(n)
num c
num x
num y
num i

desde i ate i<n,+1:(
  le(x)
  num y = x%2
  se y==0:
    (imprime(" par"))
  else (
    num c = c+1
    imprime(" impar")
  )
)
devolve(c)
```

6.4.2 Assembly gerado

```
START
PUSHI 0
READ
ATOI
STOREG 0
PUSHI 0
PUSHI 0
PUSHI 0
PUSHI 0
CICLO0:
PUSHG 4

PUSHG 0
INF
JZ FIM0
READ
ATOI
STOREG 2
PUSHG 2
PUSHI 2
MOD

STOREG 3
PUSHG 3
```

```

PUSHI 0
EQUAL
JZ L0

JUMP L1
L0:
PUSHG 1

PUSHI 1

ADD

STOREG 1
PUSHS "impar"
WRITES

L1:
PUSHG 4
PUSHI 1
ADD

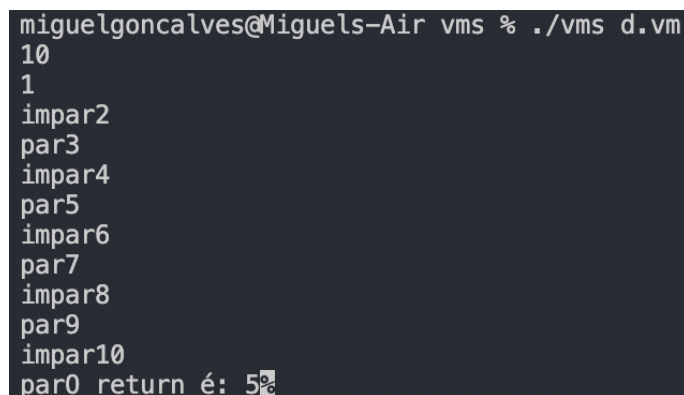
STOREG 4
JUMP CICLO0
FIM0:
PUSHG 1

PUSHS "O return : "
WRITES
WRITEI
STOP

```

6.4.3 Máquina virtual VM

Começamos por introduzir um N, neste exemplo será 10, e depois os números de 1 até 10. Como sabemos existe 5 ímpares (1,3,5,6,9) e é esse número que o programa devolve. Ao longo da execução vamos dando print se o número é par ou ímpar.



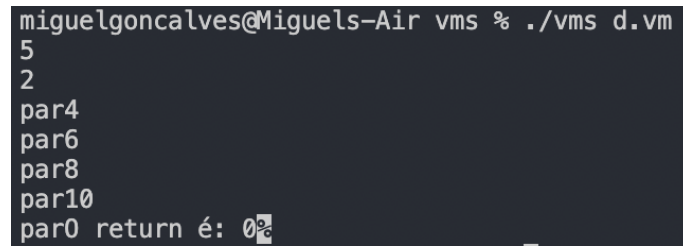
```

miguelgoncalves@miguels-Air vms % ./vms d.vm
10
1
impar2
par3
impar4
par5
impar6
par7
impar8
par9
impar10
par0 return é: 5%

```

Figura 10. Exemplo do programa 6.4 na VM - Modo texto

Sendo agora $N = 5$, vamos apenas introduzir números pares para ver se o programa retorna 0.



```
miguelgoncalves@Miguels-Air vms % ./vms d.vm
5
2
par4
par6
par8
par10
par0 return é: 0%
```

Figura 11. Exemplo do programa 6.4 na VM - Modo texto

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

6.5.1 Código

Para este problema temos o seguinte código escrito na nossa linguagem:

```
num n
le(n)
num i
num x
lista [10] v

desde i ate i<n,+1: (
    le(x)
    v[i] = x
)

i = n-1
desde i ate i>=0,-1:(
    x=v[i]
    imprime(x)
    se i==0: ()
    else (
        imprime(,)
    )
)
```

6.5.2 Assembly gerado

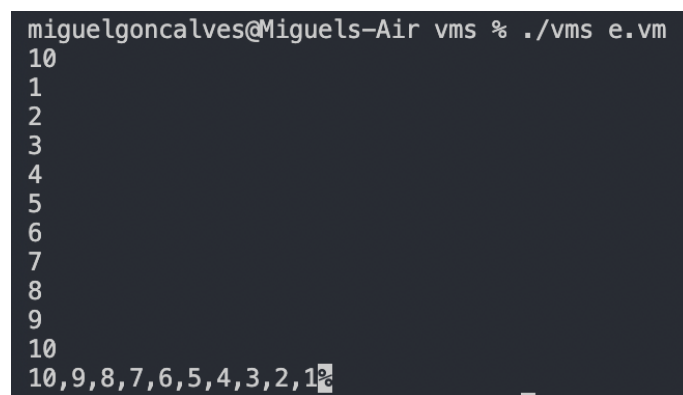
```
START
PUSHI 0
READ
ATOI
STOREG 0
PUSHI 0
PUSHI 0
PUSHN 10
CICLO0:
```

PUSHG 1
PUSHG 0
INF
JZ FIM0
READ
ATOI
STOREG 2
PUSHGP
PUSHI 3
PADD
PUSHG 1
PUSHG 2
STOREN
PUSHG 1
PUSHI 1
ADD
STOREG 1
JUMP CICLO0
FIM0:
PUSHG 0
PUSHI 1
SUB
STOREG 1
CICLO2:
PUSHG 1
PUSHI 0
SUPEQ
JZ FIM2
PUSHGP
PUSHI 3
PADD
PUSHG 1
LOADN
STOREG 2
PUSHG 2
STRI
WRITES
PUSHG 1
PUSHI 0
EQUAL
JZ L0
JUMP L1
L0:
PUSHS " ,"
WRITES

```
L1:  
PUSHG 1  
PUSHI 1  
SUB  
  
STOREG 1  
JUMP CICLO2  
FIM2:  
STOP
```

6.5.3 Máquina virtual VM

Sendo então $N = 10$, inserimos os números de 1 a 10 e verificamos que o programa dá print aos valores corretamente por ordem inversa.



```
miguelgoncalves@Miguels-Air vms % ./vms e.vm  
10  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
10,9,8,7,6,5,4,3,2,1
```

Figura 12. Exemplo do programa 6.5 na VM - Modo texto

Capítulo 7

Conclusão

A oportunidade de desenvolver a nossa própria linguagem de programação foi algo inédito na nossa carreira acadêmica enquanto alunos de Ciências da Computação.

Foi um processo desafiante, com atenção ao detalhe de forma a garantir o funcionamento de cada por menor da linguagem, para haver a certeza do funcionamento da linguagem como um todo. Um ponto bastante importante quando se utiliza uma linguagem é a forma como esta se conecta com o seu utilizador, ou seja, a forma como reage perante o mesmo. De acordo com a nossa experiência anteriormente adquirida na utilização de diferentes linguagens, tentámos tornar esta em particular bastante user friendly, de forma que seja bastante simples de usar e entender.

A dificuldade de certos programas, em alguns casos, não está na sua resolução, ou pelo menos não no raciocínio da resolução, mas sim na dificuldade em aplicar essa resolução à gramática em questão. Embora a nossa linguagem não seja de todo complexa, o processo da sua criação permitiu-nos adquirir conhecimentos acerca de várias técnicas como parsing, a tradução de código para VM e a compilação do próprio código.

Podemos assim concluir que apesar da baixa complexidade da linguagem gerada, todas as etapas até à criação da mesma foram bastante frutíferas no que toca a adquirir conhecimento sobre o seu funcionamento, as suas limitações e pontos fortes.

Poder efetuar operações numa linguagem escrita por nós é sem dúvida um sentimento único, de reconhecimento e orgulho no nosso trabalho.

Apêndice A

Código Flex

```
1 import ply.lex as lex
2 import sys
3
4 reserved = {
5     'se' : 'IF',
6     'else' : 'ELSE',
7     'enquanto' : 'WHILE',
8     'num' : 'DCLN',
9     'string' : 'DCLS',
10    'desde' : 'FOR',
11    'ate' : 'UNTIL',
12    'imprime' : 'PRINT',
13    'le' : 'SCAN',
14    'devolve' : 'RETURN',
15    'lista' : 'ARRAY'
16 }
17
18 tokens = [
19     'LPAREN',
20     'RPAREN',
21     'VIRG',
22     'ID',
23     'BOOL',
24     'NUM',
25     'SOMA',
26     'SUBTRACAO',
27     'MULTIPLICACAO',
28     'DIVISAO',
29     'IGUAL',
30     'CONJUNCAO',
31     'DISJUNCAO',
32     'DOISPONTOS',
33     'MAIOR',
34     'MENOR',
35     'ASPAS',
36     'REAL',
37     'PRE',
38     'PRD',
39     'MOD',
```

```

40 ] + list(reserved.values())
41
42 t_MOD          = r'%'
43 t_PRE          = r'\['
44 t_PRD          = r'\]'
45 t_ASPIAS       = r'"'
46 t_DOISPONTOS   = r'\:'
47 t_IGUAL        = r'\='
48 t_LPAREN       = r'\('
49 t_RPAREN       = r'\)'
50 t_VIRG         = r','
51 t_CONJUNCAO    = r'\&'
52 t_DISJUNCAO    = r'\|'
53 t_SOMA         = r'\+'
54 t_SUBTRACAO    = r'\-'
55 t_MULTIPLICACAO = r'\*'
56 t_DIVISAO      = r'\/'
57 t_MAIOR        = r'>'
58 t_MENOR        = r'<'
59
60
61 def t_ID(t):
62     r'[a-zA-Z_][a-zA-Z_0-9]*'
63     t.type = reserved.get(t.value, 'ID')    # Check for reserved words
64     return t
65
66 def t_REAL(t):
67     r'([0-9]+\.[0-9]+)|(0\.[0-9]+)'
68     return t
69
70 def t_NUM(t):
71     r'([0-9]+)'
72     return t
73
74 def t_BOOL(t):
75     r'True|False'
76     return t
77
78 t_ignore = ' \r\n\t'
79 def t_error(t):
80     print('Illegal character: ' + t.value[0])
81     t.lexer.skip()
82     t.lexer.skip(1)
83
84
85 lexer = lex.lex() # cria um AnaLex especifico a partir da especifica o acima
                    usando o gerador 'lex' do objeto 'lex'

```

Apêndice B

Código Yacc

```
1 import ply.yacc as yacc
2 import sys
3 from myLanguageLex import tokens
4
5 global store
6 global nroIf
7 global nroWhile
8 global nroFor
9 variaveis = {}
10 array = {}
11
12 nroWhile = 0
13 nroIf = 0
14 nroFor = 0
15 store = 0
16
17 def p_final(p):
18     'final : reconhece '
19     p[0] = r'START' + '\n' + p[1] + r'STOP ' + '\n'
20     print(p[0])
21
22 def p_reconhece(p):
23     '''reconhece : tipos
24                   | declara
25                   | ciclos
26                   | print
27                   | scan
28                   | devolve
29                   | atribuicao
30     '''
31     p[0] = p[1]
32
33 def p_reconheceComposto(p):
34     '''reconhece : tipos reconhece
35                   | declara reconhece
36                   | ciclos reconhece
37                   | print reconhece
38                   | scan reconhece
39                   | atribuicao reconhece
```

```

40     ''
41     p[0] = p[1] + p[2]
42
43 def p_multiplicacao(p):
44     'expression : expression MULTIPLICACAO term'
45     p[0] = p[1] + '\n' + p[3] + '\n' + r'MUL' + '\n'
46
47 def p_multiplicacaoId(p):
48     'expression : expression MULTIPLICACAO ID'
49     if p[3] in variaveis:
50         p[0] = p[1] + '\n' + r'PUSHG ' + str(variaveis[p[3]]) + '\n' + r'MUL' + '\n'
51     else:
52         p[0] = r'PUSHS "Erro: Variavel desconhecida :(" ' + '\n' + r'ERR' + '\n'
53
54 def p_divisao(p):
55     'expression : expression DIVISAO term'
56     p[0] = p[1] + '\n' + p[3] + '\n' + r'DIV' + '\n'
57
58 def p_divisaoId(p):
59     'expression : expression DIVISAO ID'
60     if p[3] in variaveis:
61         p[0] = p[1] + '\n' + r'PUSHG ' + str(variaveis[p[3]]) + '\n' + r'DIV' + '\n'
62     else:
63         p[0] = r'PUSHS "Erro: Variavel desconhecida :(" ' + '\n' + r'ERR' + '\n'
64
65 def p_somaId(p):
66     'expression : expression SOMA ID'
67     if p[3] in variaveis:
68         p[0] = p[1] + '\n' + r'PUSHG ' + str(variaveis[p[3]]) + '\n' + r'ADD' + '\n'
69     else:
70         p[0] = r'PUSHS "Erro: Variavel desconhecida :(" ' + '\n' + r'ERR' + '\n'
71
72 def p_soma(p):
73     'expression : expression SOMA term'
74     p[0] = p[1] + '\n' + p[3] + '\n' + r'ADD' + '\n'
75
76 def p_subtracao(p):
77     'expression : expression SUBTRACAO term'
78     p[0] = p[1] + '\n' + p[3] + '\n' + r'SUB' + '\n'
79
80 def p_subtracaoId(p):
81     'expression : expression SUBTRACAO ID'
82     if p[3] in variaveis:
83         p[0] = p[1] + '\n' + r'PUSHG ' + str(variaveis[p[3]]) + '\n' + r'SUB' + '\n'
84     else:
85         p[0] = r'PUSHS "Erro: Variavel desconhecida :(" ' + '\n' + r'ERR' + '\n'
86
87 def p_modulo(p):
88     'expression : NUM MOD NUM'
89     p[0] = r'PUSHI ' + str(int(p[1])) + '\n' + r'PUSHI ' + str(int(p[3])) + '\nMOD\n'
90
91 def p_moduloId(p):
92     'expression : ID MOD NUM'
93     if p[1] in variaveis:

```



```

94         p[0] = r'PUSHG ' + str(variaveis[p[1]]) + '\n' + r'PUSHI ' + str(int(p[3])) +
          '\nMOD\n'
95     else:
96         p[0] = r'PUSHS "Erro: Variavel desconhecida :(" ' + '\n' + r'ERR' + '\n'
97
98 def p_moduloNUMId(p):
99     'expression : NUM MOD ID'
100     if p[3] in variaveis:
101         p[0] = r'PUSHI ' + str(int(p[1])) + '\n' + r'PUSHG ' + str(variaveis[p[3]]) +
          '\nMOD\n'
102     else:
103         p[0] = r'PUSHS "Erro: Variavel desconhecida :(" ' + '\n' + r'ERR' + '\n'
104
105 def p_moduloIdID(p):
106     'expression : ID MOD ID'
107     if p[1] in variaveis and p[3] in variaveis:
108         p[0] = r'PUSHG ' + str(variaveis[p[1]]) + r'PUSHG ' + str(variaveis[p[3]]) +
          '\nMOD\n'
109     else:
110         p[0] = r'PUSHS "Erro: Variavel desconhecida :(" ' + '\n' + r'ERR' + '\n'
111
112 def p_fator(p):
113     'factor : NUM'
114     p[0] = r'PUSHI ' + str(int(p[1])) + '\n'
115
116 def p_fatorReal(p):
117     'factor : REAL'
118     p[0] = r'PUSHI ' + str(int(p[1])) + '\n'
119
120 def p_termo(p):
121     'term : factor '
122     p[0] = p[1]
123
124 def p_expressao(p):
125     'expression : term '
126     p[0] = p[1]
127
128 def p_expressaoId(p):
129     'expression : ID'
130     if (p[1] in variaveis):
131         p[0] = r'PUSHG ' + str(variaveis[p[1]]) + '\n'
132     else:
133         p[0] = r'PUSHS "Erro: Variavel desconhecida :(" ' + '\n' + r'ERR' + '\n'
134
135 def p_elementosC(p):
136     '''els : comparacoes
137         | term
138     '''
139     p[0] = p[1]
140
141 def p_elementosUnicos(p):
142     '''els : ID
143         | BOOL
144     '''

```

```

145     if p[1] == 'True':
146         p[0] = r'PUSHI 1' + '\n'
147     elif p[1] == 'False':
148         p[0] = r'PUSHI 0' + '\n'
149     elif isinstance(p[1], str):
150         if (p[1] in variaveis):
151             p[0] = r'PUSHG ' + str(variaveis[p[1]]) + '\n'
152         else:
153             p[0] = r'PUSHS "Erro: Variavel desconhecida :(" ' + '\n' + r'ERR' + '\n'
154
155 def p_atribuicaoArray(p):
156     'atribuicao : ID PRE NUM PRD IGUAL expression '
157     if p[1] in variaveis:
158         p[0] = r'PUSHGP' + '\nPUSHI 0\n' + 'PADD\n' + 'PUSHI ' + str(variaveis[p[1]] +
159             int(p[3])) + '\n' + p[6] + '\nSTOREN\n'
160     else:
161         p[0] = r'PUSHS "Erro: Variavel nao definida :(" ' + '\n' + r'ERR' + '\n'
162
163 def p_atribuicaoArrayID(p):
164     'atribuicao : ID PRE ID PRD IGUAL expression '
165     if p[1] in variaveis and p[3] in variaveis:
166         p[0] = r'PUSHGP' + '\nPUSHI ' + str(store_array[p[1]]) + '\nPADD\n' + 'PUSHG
167             ' + str(variaveis[p[3]]) + '\n' + p[6] + '\nSTOREN\n'
168     else:
169         p[0] = r'PUSHS "Erro: Variavel nao definida :(" ' + '\n' + r'ERR' + '\n'
170
171 def p_atribuicaoIdArray(p):
172     'atribuicao : ID IGUAL ID PRE ID PRD'
173     if p[1] in variaveis and p[3] in variaveis and p[5] in variaveis:
174         p[0] = r'PUSHGP' + '\nPUSHI ' + str(store_array[p[3]]) + '\nPADD\n' + 'PUSHG
175             ' + str(variaveis[p[5]]) + '\nLOADN\n' + 'STOREG ' + str(variaveis[p[1]])
176             + '\n'
177     else:
178         p[0] = r'PUSHS "Erro: Variavel nao definida :(" ' + '\n' + r'ERR' + '\n'
179
180 def p_atribuicaoNumArray(p):
181     'atribuicao : ID IGUAL ID PRE NUM PRD'
182     if p[1] in variaveis and p[3] in variaveis and p[5] in variaveis:
183         p[0] = r'PUSHGP' + '\nPUSHI ' + str(store_array[p[3]]) + '\nPADD\n' + 'PUSHG
184             ' + str(p[5]) + '\nLOADN\n' + 'STOREG ' + str(variaveis[p[1]]) + '\n'
185     else:
186         p[0] = r'PUSHS "Erro: Variavel nao definida :(" ' + '\n' + r'ERR' + '\n'
187
188 def p_declararArray(p):
189     'declara : ARRAY PRE NUM PRD ID'
190     global store
191     if p[5] not in variaveis:
192         p[0] = r'PUSHN ' + str(p[3]) + '\n'
193         variaveis[p[5]] = store
194         store += int(p[3])
195         array[p[5]] = int(p[3])
196     else:
197         p[0] = r'PUSHS "Erro: Variavel ja declarada :(" ' + '\n' + r'ERR' + '\n'

```

```

194
195 def p_declararNum(p):
196     'declara : DCLN ID IGUAL expression '
197     global store
198     if p[2] not in variaveis:
199         p[0] = p[4] + '\n'
200         variaveis[p[2]] = store
201         store += 1
202     else:
203         p[0] = p[4] + '\n' + r'STOREG ' + str(variaveis[p[2]]) + '\n'
204
205 def p_declararNumSolo(p):
206     'declara : DCLN ID'
207     global store
208     if p[2] not in variaveis:
209         p[0] = r'PUSHI 0' + '\n'
210         variaveis[p[2]] = store
211         store += 1
212     else:
213         p[0] = r'PUSHI 0' + '\n'
214
215 def p_declararString(p):
216     'declara : DCLS ID'
217     p[0] = r'PUSHS ' + '"' + '\n'
218     global store
219     if p[2] not in variaveis:
220         p[0] = r'PUSHS ' + '"' + '\n'
221         variaveis[p[2]] = store
222         store += 1
223     else:
224         p[0] = r'PUSHS ' + '"' + '\n'
225
226 def p_declararStringSolo(p):
227     'declara : DCLS ID IGUAL ASPAS ID ASPAS'
228     global store
229     if p[2] not in variaveis:
230         p[0] = r'PUSHS ' + r''' + p[5] + r''' + '\n'
231         variaveis[p[2]] = store
232         store += 1
233     else:
234         p[0] = r'PUSHS ' + r''' + p[4] + r''' + '\n'
235
236 def p_conjuncao(p):
237     '''conjuncao : els CONJUNCAO els
238                  | els CONJUNCAO aux
239     '''
240     p[0] = p[1] + p[3]
241
242 def p_disjuncao(p):
243     '''disjuncao : els DISJUNCAO els
244                  | els DISJUNCAO aux
245     '''
246     p[0] = p[1] + p[3]
247

```

```

248 def p_auxiliar(p):
249     '''aux : conjuncao
250         | disjuncao
251     '''
252     p[0] = p[1]
253
254 def p_comparacoes(p):
255     '''comparacoes : expression IGUAL IGUAL expression
256                     | aux IGUAL IGUAL aux
257                     | aux MAIOR aux
258                     | aux MENOR aux
259                     | expression MAIOR expression
260                     | expression MENOR expression
261                     | term MAIOR term
262                     | term MENOR term
263                     | term IGUAL IGUAL term
264     '''
265     if p[2] == '=':
266         p[0] = p[1] + ' \n' + p[4] + r'EQUAL' + '\n'
267     elif p[2] == '<':
268         p[0] = p[1] + ' \n' + p[3] + r'INF' + '\n'
269     elif p[2] == '>':
270         p[0] = p[1] + ' \n' + p[3] + r'SUP' + '\n'
271
272
273 def p_comparacoesComIgualdade(p):
274     '''comparacoes : aux MAIOR IGUAL aux
275                     | aux MENOR IGUAL aux
276                     | expression MAIOR IGUAL expression
277                     | expression MENOR IGUAL expression
278                     | term MAIOR IGUAL term
279                     | term MENOR IGUAL term
280     '''
281     if p[2] == '<':
282         p[0] = p[1] + ' \n' + p[4] + r'INFEQ' + '\n'
283     elif p[2] == '>':
284         p[0] = p[1] + ' \n' + p[4] + r'SUPEQ' + '\n'
285
286
287 def p_tiposExpressao(p):
288     '''tipos : aux
289               | expression
290               | comparacoes
291               | print
292               | devolve
293               | scan
294               | declara
295     '''
296     p[0] = p[1]
297
298 def p_statements(p):
299     'statements : stat'
300     p[0] = p[1]
301

```

```

302 def p_statementsComposto(p):
303     'statements : stat statements'
304     p[0] = p[1] + p[2]
305
306 def p_stat(p):
307     '''stat : atribuicao
308         | tipos
309         | ciclos
310     '''
311     p[0] = p[1]
312
313 def p_statVazia(p):
314     'stat : '
315     p[0] = '\n'
316
317 def p_atribuicao(p):
318     'atribuicao : ID IGUAL expression'
319     if p[1] in variaveis:
320         p[0] = p[3] + r'STOREG ' + str(variaveis[p[1]]) + '\n'
321     else:
322         p[0] = r'PUSHS "Erro: Variavel nao definida :(" + '\n' + r'ERR' + '\n'
323
324 def p_cicloIf(p):
325     'if : IF tipos DOISPONTOS LPAREN statements RPAREN ELSE LPAREN statements RPAREN'
326     global nroIf
327     aux = nroIf + 1
328     p[0] = p[2] + r'JZ L' + str(nroIf) + '\n' + p[5] + '\nJUMP L' + str(aux) + '\nL'
329         + str(nroIf) + ':\n' + p[9] + '\nL' + str(aux) + ':\n'
330
331 def p_cicloIfSolo(p):
332     'if : IF tipos DOISPONTOS LPAREN statements RPAREN'
333     global nroIf
334     p[0] = p[2] + r'JZ L' + str(nroIf) + '\n' + p[5] + '\nJUMP L' + str(nroIf) + '\nL'
335         + str(nroIf) + ':\n'
336     nroIf += 1
337
338 def p_sinal(p):
339     '''sinal : SOMA
340         | SUBTRACAO
341         | MULTIPLICACAO
342         | DIVISAO
343     '''
344     if p[1] == '+' :
345         p[0] = '\nADD\n'
346     elif p[1] == '-' :
347         p[0] = '\nSUB\n'
348     elif p[1] == '*' :
349         p[0] = '\nMUL\n'
350     elif p[1] == '/' :
351         p[0] = '\nDIV\n'
352
353 def p_cicloFor(p):
354     'for : FOR ID UNTIL comparacoes VIRG sinal NUM DOISPONTOS LPAREN statements
355         RPAREN'

```

```

353     global nroFor
354     if p[2] in variaveis:
355         p[0] = r'CICLO' + str(nroFor) + ':\n' + p[4] + r'JZ FIM' + str(nroFor) + '\n'
356             + p[10] + r'PUSHG ' + str(variaveis[p[2]]) + '\n' + r'PUSHI ' + p[7] + p
357             [6] + '\nSTOREG ' + str(variaveis[p[2]]) + '\n' + r'JUMP CICLO' + str(
358                 nroFor) + '\nFIM' + str(nroFor) + ':\n'
359         nroFor+=1
360
361 def p_cicloWhile(p):
362     'while : WHILE tipos DOISPONTOS LPAREN statements RPAREN'
363     global nroWhile
364     aux = nroWhile +1
365     p[0] = r'L' + str(nroWhile) + ':\n' + p[2] + '\n' + r'JZ L' + str(aux) + '\n' + p
366         [5] + '\n' + r'JUMP L' + str(nroWhile) + '\n' + r'L' + str(aux) + ':\n'
367     nroWhile += 1
368
369 def p_cicloSIf(p):
370     'ciclos : if '
371     p[0] = p[1]
372     global nroIf
373     nroIf+=1
374
375 def p_cicloSFor(p):
376     'ciclos : for '
377     p[0] = p[1]
378     global nroFor
379     nroFor+=1
380
381 def p_cicloSWhile(p):
382     'ciclos : while '
383     p[0] = p[1]
384     global nroWhile
385     nroWhile+=1
386
387 def p_printId(p):
388     'print : PRINT LPAREN ID RPAREN'
389     if p[3] in variaveis:
390         p[0] = r'PUSHG ' + str(variaveis[p[3]]) + '\nSTRI\n' + r'WRITES' + '\n'
391     else:
392         p[0] = r'PUSHS ' + r'" Erro: Variavel desconhecida :(" ' + '\n' + r'ERR' + '\n'
393
394 def p_print(p):
395     '''print : PRINT LPAREN ASPAS ID ASPAS RPAREN'''
396     p[0] = r'PUSHS ' + r'" ' + p[4] + r'" ' + '\n' + r'WRITES' + '\n'
397
398 def p_printN(p):
399     '''print : PRINT LPAREN ASPAS term ASPAS RPAREN'''
400     p[0] = r'PUSHG ' + r'" ' + p[4] + r'" ' + '\n' + r'WRITEI' + '\n'
401
402 def p_printComposto(p):
403     '''print : PRINT LPAREN ASPAS ID ASPAS VIRG expression RPAREN'''

```

```

402     '''
403     p[0] = r'PUSHS ' + r''' + p[4] + r''' + '\n' + r'WRITES' + '\n' + r'PUSHG ' + p
         [7] + r'WRITEI' + '\n'
404
405 def p_printExpressao(p):
406     'print : PRINT LPAREN expression RPAREN'
407     p[0] = p[3] + r'WRITEI' + '\n'
408
409 def p_scan(p):
410     '''scan : SCAN LPAREN ID RPAREN
411     '''
412     global store
413     if p[3] in variaveis:
414         p[0] = r'READ' + '\n' + r'ATOI' + '\n' + r'STOREG ' + str(variaveis[p[3]]) +
            '\n'
415     else:
416         p[0] = r'PUSHS "Erro: Variavel desconhecida :(" ' + '\n' + r'ERR' + '\n'
417
418 def p_return(p):
419     'devolve : RETURN LPAREN expression RPAREN'
420     p[0] = p[3] + '\n' + r'PUSHS "O return : "' + '\nWRITES\n' + 'WRITEI\n'
421
422 def p_printVirgula(p):
423     'print : PRINT LPAREN VIRG RPAREN'
424     p[0] = 'PUSHS ","\n' + r'WRITES' + '\n'
425
426 def p_error(p):
427     parser.success = False
428     print('Syntax error!')
429
430 ###inicio do parsing
431 parser = yacc.yacc()
432 parser.success = True
433
434 fonte = ""
435 for line in sys.stdin:
436     fonte += line
437 parser.parse(fonte)

```
