

Processamento de Linguagens (3º ano de MIEI)

Trabalho Prático 2

Relatório de Desenvolvimento

António Silva
(A89558)

Diogo Araújo
(A89517)

Ivo Oliveira
(A84751)

30 de maio de 2021

Resumo

Neste segundo trabalho prático, no âmbito da unidade curricular de Processamento de Linguagens, foi elaborado um compilador para uma linguagem imperativa.

A linguagem imperativa é um linguagem desenhada e definida pelo grupo de forma a ser o mais completa possível e capaz de corresponder aos objetivos pedidos.

Os principais objetivos deste trabalho passam por: aumentar a experiência em Engenharia de Linguagens e programação generativa, desenvolver processadores de linguagens a partir de uma gramática tradutora, desenvolver um compilador gerando código para uma máquina de stack virtual específica e utilizar geradores de compiladores baseados em gramáticas tradutoras (Yacc).

A linguagem desenvolvida pelo grupo teve bastante influência da linguagem imperativa C, pelo que existem bastantes parecenças.

Conteúdo

1	Introdução	2
2	Análise e Especificação	3
2.1	Descrição informal do problema	3
2.2	Especificação do Requisitos	3
3	Concepção/desenho da Resolução	4
3.1	Gramática desenvolvida	4
3.2	Estruturas de Dados	6
3.3	Implementação	6
3.4	Alternativas, Decisões e Problemas de Implementação	14
3.5	Testes realizados e Resultados	14
4	Conclusão	22
A	Código do Programa	23

Capítulo 1

Introdução

Área: Processamento de Linguagens

Enquadramento Com o intuito de aumentar a experiência em engenharia de linguagens e em programação generativa (gramatical), foi proposta a criação de um compilador de uma linguagem imperativa.

Contexto Com a criação de uma linguagem de programação e o seu respectivo processador, espera-se que com este trabalho se desenvolva várias capacidades na área da Engenharia da Linguagem, sendo esta uma das grandes áreas dentro da Engenharia Informática. Também vários aspectos da Engenharia de linguagens podem ser aplicados a outros ramos da computação, fazendo deste, um trabalho bastante pedagógico.

Problema No enunciado são explícitos vários aspectos que a implementação da linguagem e do seu compilador têm de conseguir responder, assim como os testes específicos que têm de passar.

Objetivo Desenvolvimento de um compilador de uma linguagem desenvolvida, gerando código para uma Virtual Machine. Este compilador deve tirar partido de geradores de compiladores baseados em gramáticas tradutoras, concretamente o Yacc, versão PLY do Python, completado pelo gerador de analisadores léxicos Lex, também versão PLY do Python.

Resultados ou Contributos O grupo conseguiu com sucesso implementar o compilador proposto, assim aumentando a experiência em engenharia de linguagens e em programação generativa (gramatical), reforçando a capacidade de escrever gramáticas, quer independentes de contexto (GIC), quer tradutoras (GT);

Estrutura do documento No capítulo 2 é feita uma breve descrição do problema e da Especificação dos Requisitos. No capítulo 3 é feita a exposição da proposta de resolução. Por último, no capítulo 4, são apresentados testes.

No capítulo 2 aborda-se a descrição do problema proposto e os dados que fornecidos pelo mesmo. Capítulo 3 idealiza a forma e os algoritmos utilizados pelo grupo para dar respostas às perguntas requeridas. O capítulo 4 retrata as decisões tomadas e os testes efectuados para garantir que o programa está funcional. Por fim, o capítulo 5 reflete em que aspectos este trabalho foi importante para o grupo.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

O problema baseia-se no desenvolvimento de um linguagem de programação simples, a gosto do grupo, capaz de responder aos requisitos pedidos, bem como um requisito opcional. Todos estes estão descritos na próxima secção.

Assim, o projeto consiste numa linguagem algo que parecida à linguagem de programação **C**, tendo sido esta a principal inspiração para o seu desenvolvimento.

2.2 Especificação do Requisitos

O projeto teria que dar resposta com sucesso aos seguintes requisitos:

- Declarar variáveis atómicas do tipo inteiro, com as quais se podem realizar operações aritméticas, relacionais e lógicas;
- Efetuar instruções algorítmicas;
- Ler do *standard input* e escrever no *standard output*;
- Exercer instruções condicionais para controlo do fluxo de execução;
- Efetuar instruções cíclicas. Visto que o número do grupo é o **25**, o ciclo a implementar é o **repeat-until**;
- **(Opcional)** - Declarar e manusear variáveis do tipo **array** de 1 e 2 dimensões de inteiros;
- **(Opcional)** - Definir e invocar subprogramas sem parâmetros mas que possam retornar um resultado tipo inteiro.

O grupo optou por desenvolver o **primeiro** requisito opcional (arrays) e adicionalmente, desenvolver também o ciclo **for**.

Capítulo 3

Concepção/desenho da Resolução

3.1 Gramática desenvolvida

Na nossa gramática a **LExp** deriva em **Instrucoes**. Uma **Instrucao** pode ser de 3 tipos: **Atribuicao**, **Operacao** e **Funcao**.

```
LExp : Instrucoes
Instrucoes : Instrucoes Instrucao
           | Instrucao
Instrucao : Atribuicao
           | Operacao
           | Funcao
```

Uma **Atribuicao** consiste na manipulação das variáveis atômicas inteiras, como a declaração de uma variável ou a alteração do seu valor.

```
Atribuicao : int id
           | int id '[' Num ']'
           | int id '[' Num ',' Num ']'
           | int id '=' Operacao
           | int id '=' ReadInt '(' ' ')'
           | id '=' Operacao
           | id '=' ReadInt '(' ' ')'
           | id '[' Operacao ']' '=' Operacao
           | id '[' Operacao ']' '=' ReadInt '(' ' ')'
           | id '[' Operacao ',' Operacao ']' '=' Operacao
           | id '[' Operacao ',' Operacao ']' '=' ReadInt '(' ' ')'
           | id '+' '+'
           | id '-' '-'
           | id '+' '=' Operacao
           | id '-' '=' Operacao
```

Uma **Operacao** baseia-se na realização de operações aritméticas utilizando as derivações **Termo** e **Fator** para que estas tenham as suas propriedades corretas (dar prioridade aos parênteses, etc.) e para que seja possível a consulta do valor de uma variável.

```

Operacao : Operacao '+' Termo
          | Operacao '-' Termo
          | Termo
Termo : Termo '*' Fator
       | Termo '/' Fator
       | Termo '%' Fator
       | Fator
Fator : Num
       | Id
       | Id '[' Operacao ']'
       | Id '[' Operacao ',' Operacao ']'
       | '(' Operacao ')'

```

Uma **Funcao** é o que permite ler do *standard input* e escrever no *standard output*, efetuar instruções cíclicas e controlar o fluxo de execução, recorrendo à **Condicional** para tal.

```

Funcao : Write '(' String ')'
        | Write '(' Operacao ')'
        | ReadInt '(' ')'
        | Read '(' ')'
        | Repeat '(' Condicional ')' '{' Instrucoes '}'
        | If '(' Condicional ')' '{' Instrucoes '}' Else '{' Instrucoes '}'
        | If '(' Condicional ')' '{' Instrucoes '}'
        | For '(' Atribuicao ';' Condicional ';' Atribuicao ')' '{' Instrucoes '}'

```

Uma **Condicional** utiliza as derivações **Cond**, **Cond2** e **ExpRel** para a realização de operações lógicas.

```

Condicional : Condicional OR Cond
              | Cond
Cond : Cond AND Cond2
      | Cond2
Cond2 : NOT Cond
       | ExpRel
       | '(' Condicional ')'
ExpRel : Operacao '>' Operacao
        | Operacao '<' Operacao
        | Operacao '>' '=' Operacao
        | Operacao '<' '=' Operacao
        | Operacao '=' '=' Operacao
        | Operacao '!' '=' Operacao
        | Operacao

```

3.2 Estruturas de Dados

Para o controlo da posição 'atual' da Stack, aquando da leitura do programa, a variável **pos_stack** é incrementada e decrementada à medida que o compilador interpreta as diferentes instruções. Por exemplo, uma adição decrementa 1 ou a consulta de uma variável incrementa 1 na **pos_stack**.

Para o controlo das variáveis, foi necessário introduzir estruturas de dados que permitissem verificar se as variáveis eram declaradas corretamente, bem como em que posição da *Stack* ficariam guardadas. Para a implementação destas tabelas de símbolos foram usados três *Dictionaries*. Um que armazena a posição na *Stack* das variáveis do tipo inteiro (**ts**). Os outros dois são usados para armazenar a posição na *Stack* das variáveis do tipo *array* de inteiros bem como as dimensões destes (**ta** para arrays de 1 dimensão, **tm** para arrays de 2 dimensões).

Para evitar a criação de *Tags* repetidas é usada a variável **func_nr** que é incrementada a cada **Funcao** que é interpretada, tornando as etiquetas na VM únicas.

3.3 Implementação

O *lexer* desenvolvido tem os seguintes *literals* e *tokens*:

```
literals = ['(',')','+','-','*','/','=','>','<','!','{','}','[' ,']',';','%']
```

```
tokens = ["Num","If","Else","Id","Repeat","Int","Read","ReadInt","Write","String","For",  
"And","Or","Not"]
```

As ERs dos respetivos *tokens*, por ordem não relevante, são:

- Num: `r'\d+'`
- If: `r'if'`
- Else: `r'else'`
- Id: `r'\w+'`
- Repeat: `r'repeat_until'`
- Int: `r'int'`
- Read: `r'read'`
- ReadInt: `r'readInt'`
- Write: `r'write'`
- String: `r'\"[^\"]*\"'`
- For: `r'for'`
- And: `r'and'`
- Or: `r'or'`
- Not: `r'not'`

Traduzindo o descrito na secção 3.1, foram implementadas as seguintes regras no **Yacc**:

- LExp - apenas retorna o resultado de 'Instrucoes'.

```
def p_LExp(p):  
    "LExp : Instrucoes"  
    p[0] = p[1]
```

- Instrucoes - recursiva e apenas retorna o resultado de 'Instrucao'.

```
def p_Instrucoes_Instrucao(p):  
    "Instrucoes : Instrucoes Instrucao"  
    p[0] = p[1] + p[2]
```

```
def p_Instrucoes_Vazio(p):  
    "Instrucoes : Instrucao"  
    p[0] = p[1]
```

- Instrucao - retorna o resultado de 'Atribuicao', 'Operacao' e 'Funcao'.

```
def p_Instrucao_Atribuicao(p):  
    "Instrucao : Atribuicao"  
    p[0] = p[1]
```

```
def p_Instrucao_Operacao(p):  
    "Instrucao : Operacao"  
    p[0] = p[1]
```

```
def p_Instrucao_Funcao(p):  
    "Instrucao : Funcao"  
    p[0] = p[1]
```

- Atribuicao - retorna o código máquina correspondente à respetiva declaração de variável ou atribuição de valor.

```
def p_Atribuicao_Inc_Id(p):  
    "Atribuicao : Id '+' '+'"  
    if(p[1] in ts):  
        p[0] = "\npushg " + str(ts[p[1]])  
        + "\npushi 1\nadd\nstoreg " + str(ts[p[1]])
```

```
def p_Atribuicao_Dec_Id(p):  
    "Atribuicao : Id '-' '-'"  
    if(p[1] in ts):  
        p[0] = "\npushi 1\npushg " + str(ts[p[1]])  
        + "\nsub\nstoreg" + str(ts[p[1]])
```

```

def p_Atribuicao_Inc_Id_Op(p):
    "Atribuicao : Id '+' '=' Operacao"
    if(p[1] in ts):
        p[0] = "\npushg " + str(ts[p[1]]) + str(p[4])
        + "\nadd\nstoreg " + str(ts[p[1]])

def p_Atribuicao_Dec_Id_Op(p):
    "Atribuicao : Id '-' '=' Operacao"
    if(p[1] in ts):
        p[0] = str(p[4]) + "\npushg " + str(ts[p[1]])
        + "\nsub\nstoreg " + str(ts[p[1]])

def p_Atribuicao_Declaracao_Zero(p):
    "Atribuicao : Int Id"
    if(p[2] not in ts and p[2] not in ta and p[2] not in tm):
        global pos_stack
        ts[p[2]] = pos_stack
        p[0] = "\npushi 0"
        pos_stack+=1

def p_Atribuicao_Declaracao_Input(p):
    "Atribuicao : Int Id '=' ReadInt '(' ' ')"
    if(p[2] not in ts and p[2] not in ta and p[2] not in tm):
        global pos_stack
        ts[p[2]] = pos_stack
        p[0] = "\nread \natoi"
        pos_stack+=1

def p_Atribuicao_Declaracao(p):
    "Atribuicao : Int Id '=' Operacao"
    if(p[2] not in ts and p[2] not in ta and p[2] not in tm):
        global pos_stack
        ts[p[2]] = pos_stack-1
        p[0] = str(p[4])

def p_Atribuicao_Declaracao_Array(p):
    "Atribuicao : Int Id '[' Num ']"
    if(p[2] not in ts and p[2] not in ta and p[2] not in tm):
        global pos_stack
        ta[p[2]] = (pos_stack,int(p[4]))
        p[0] = "\npushn " + str(p[4])
        pos_stack += int(p[4])

def p_Atribuicao_Declaracao_Matriz(p):
    "Atribuicao : Int Id '[' Num ',' Num ']"
    if(p[2] not in ts and p[2] not in ta and p[2] not in tm):
        global pos_stack

```

```

        tm[p[2]] = (pos_stack,int(p[4]),int(p[6]))
        p[0] = "\npushn " + str(int(p[4])*int(p[6]))
        pos_stack += int(p[4])*int(p[6])

def p_Atribuicao_Alt(p):
    "Atribuicao : Id '=' Operacao"
    if(p[1] in ts):
        global pos_stack
        p[0] = str(p[3]) + "\nstoreg " + str(ts[p[1]])
        pos_stack-=1

def p_Atribuicao_Input(p):
    "Atribuicao : Id '=' ReadInt '(' ')"
    if(p[1] in ts):
        global pos_stack
        p[0] = "\nread \natoi \nstoreg " + str(ts[p[1]])
        pos_stack-=1

def p_Atribuicao_Array(p):
    "Atribuicao : Id '[' Operacao ']' '=' Operacao"
    if(p[1] in ta):
        global pos_stack
        p[0] = "\npushgp \npushi " + str(ta[p[1]][0]) + "\npadd"
        + p[3] + p[6] + "\nstoren"
        pos_stack-=2

def p_Atribuicao_Array_Input(p):
    "Atribuicao : Id '[' Operacao ']' '=' ReadInt '(' ')"
    if(p[1] in ta):
        global pos_stack
        p[0] = "\npushgp \npushi " + str(ta[p[1]][0]) + "\npadd"
        + p[3] + "\nread \natoi \nstoren"
        pos_stack-=1

def p_Atribuicao_Matriz(p):
    "Atribuicao : Id '[' Operacao ',' Operacao ']' '=' Operacao"
    if(p[1] in tm):
        global pos_stack
        p[0] = "\npushgp" + "\npushi " + str(tm[p[1]][0]) + "\npadd"
        + p[3] + "\npushi " + str(tm[p[1]][2]) + "\nmul" +
        p[5] + "\nadd" + p[8] + "\nstoren"
        pos_stack-=2

def p_Atribuicao_Matriz_Input(p):
    "Atribuicao : Id '[' Operacao ',' Operacao ']' '=' ReadInt '(' ')"
    if(p[1] in tm):
        global pos_stack
        p[0] = "\npushgp" + "\npushi " + str(ta[p[1]][0]) + "\npadd"

```

```

+ p[3] + "\npushi " + str(tm[p[1]][1]) + "\nmul" +
p[5] + "\nadd \nread \natoi \nstoren"
pos_stack-=1

```

- Operacao - recursiva e devolve o código máquina correspondente a operações aritméticas (adição e subtração).

```

def p_Operacao_Mais(p):
    "Operacao : Operacao '+' Termo"
    p[0] = str(p[1]) + str(p[3]) + "\nadd"
    global pos_stack
    pos_stack-=1

def p_Operacao_Menos(p):
    "Operacao : Operacao '-' Termo"
    p[0] = str(p[1]) + str(p[3]) + "\nsub"
    global pos_stack
    pos_stack-=1

def p_Operacao_Termo(p):
    "Operacao : Termo"
    p[0] = str(p[1])

```

- Termo - recursiva e devolve o código máquina correspondente a operações aritméticas (multiplicação, divisão e resto de divisão inteira).

```

def p_Termo_Mul(p):
    "Termo : Termo '*' Fator"
    p[0] = str(p[1]) + str(p[3]) + "\nmul"
    global pos_stack
    pos_stack-=1

def p_Termo_Div(p):
    "Termo : Termo '/' Fator"
    p[0] = str(p[1]) + str(p[3]) + "\ndiv"
    global pos_stack
    pos_stack-=1

def p_Termo_Mod(p):
    "Termo : Termo '%' Fator"
    p[0] = str(p[1]) + str(p[3]) + "\nmod"
    global pos_stack
    pos_stack-=1

def p_Termo_Fator(p):
    "Termo : Fator"

```

```
p[0] = str(p[1])
```

- Fator - devolver o código máquina correspondente à consulta de uma variável ou constante. Introduce também os parênteses nas operações aritméticas.

```
def p_Fator_Num(p):
    "Fator : Num"
    p[0] = "\npushi " + str(p[1])
    global pos_stack
    pos_stack+=1

def p_Fator_Id(p):
    "Fator : Id"
    p[0] = "\npushg " + str(ts[p[1]])
    global pos_stack
    pos_stack+=1

def p_Fator_Array(p):
    "Fator : Id '[' Operacao ']"
    global pos_stack
    p[0] = "\npushgp" + "\npushi " + str(ta[p[1]][0]) + "\npadd" + p[3] + "\nloadn"
    pos_stack-=1

def p_Fator_Matriz(p):
    "Fator : Id '[' Operacao ',' Operacao ']"
    global pos_stack
    p[0] = "\npushgp" + "\npushi " + str(tm[p[1]][0]) + "\npadd" + p[3] + p[5] + "\nmul\nloadn"
    pos_stack-=1
    func_nr+=1

def p_Fator_Operacao(p):
    "Fator : '(' Operacao '"
    p[0] = p[2]
```

- Funcao - devolve o código máquina para as funcionalidades de ler e escrever do *input/output* (**Write**, **Read** e **ReadInt**), controlo do fluxo de execução e ciclos.

```
def p_Funcao_For(p):
    "Funcao : For '(' Atribuicao ';' Condicional ';' Atribuicao ')' '{' Instrucoes '}"
    global func_nr
    p[0] = (str(p[3]) + "\nfor_" + str(func_nr) + ":\n" + str(p[5]) + "\njz fim_for_"
            + str(func_nr) + str(p[10]) + str(p[7]) + "\njump for_" + str(func_nr)
            + "\nfim_for_" + str(func_nr) + ":")
    func_nr += 1
```

```

def p_Funcao_Write_String(p):
    "Funcao : Write '(' String ')"
    p[3] = p[3][:-1] + "\\n\"
    p[0] = "\\npushs " + p[3] + "\\nwrites"

def p_Funcao_Write_Operacao(p):
    "Funcao : Write '(' Operacao ')"
    p[0] = p[3] + "\\nstri" + "\\nwrites \\npushs " + "\\n\\n\" + "\\nwrites"

def p_Funcao_ReadInt(p):
    "Funcao : ReadInt '(' ')"
    global pos_stack
    p[0] = "\\nread \\natoi"
    pos_stack+=1

def p_Funcao_Read(p):
    "Funcao : Read '(' ')"
    global pos_stack
    p[0] = "\\nread"
    pos_stack+=1

def p_Funcao_Repeat(p):
    "Funcao : Repeat '(' Condicional ')' '{' Instrucoes '}"
    global func_nr
    p[0] = ("\\nrepeat" + str(func_nr) + ":"
        + p[3] + "\\npushi 0\\nequal \\njz end" + str(func_nr)
        + p[6]
        + "\\njump repeat" + str(func_nr)
        + "\\nend" + str(func_nr) + ":"
        )
    func_nr+=1

def p_Funcao_IfElse(p):
    "Funcao : If '(' Condicional ')' '{' Instrucoes '}' Else '{' Instrucoes '}"
    global func_nr
    p[0] = (p[3] + "\\njz else" + str(func_nr)
        + p[6]
        + "\\njump end" + str(func_nr)
        + "\\nelse" + str(func_nr) + ":"
        + p[10]
        + "\\nend" + str(func_nr) + ":"
        )
    func_nr+=1

def p_Funcao_If(p):
    "Funcao : If '(' Condicional ')' '{' Instrucoes '}"
    global func_nr
    global pos_stack

```

```

p[0] = (p[3]+" \njz end" + str(func_nr)
      + p[6]
      + " \nend" + str(func_nr) + ":"
      )
func_nr+=1

```

- Condicional - recursiva e devolve o código máquina para operações lógicas (OR).

```

def p_Condicional_Or_Cond(p):
    "Condicional : Condicional Or Cond"
    p[0] = p[1] + p[3] + " \nadd" + p[1] + p[3] + " \nmul \nsub"

def p_Condicional_Cond(p):
    "Condicional : Cond"
    p[0] = p[1]

```

- Cond - recursiva e devolve o código máquina para operações lógicas (AND).

```

def p_Cond_And_Cond2(p):
    "Cond : Cond And Cond2"
    p[0] = p[1] + p[3] + " \nmul"

def p_Cond_Cond2(p):
    "Cond : Cond2"
    p[0] = p[1]

```

- Cond2 - devolve o código máquina para operações lógicas (NOT) e introduz os parênteses nas condições.

```

def p_Cond2_Not(p):
    "Cond2 : Not Condicional"
    p[0] = p[2] + " \npushi 0 \nequal"

def p_Cond2_ExpRel(p):
    "Cond2 : ExpRel"
    p[0] = p[1]

def p_Cond2_Condicional(p):
    "Cond2 : '(' Condicional ')'"
    p[0] = p[2]

```

3.4 Alternativas, Decisões e Problemas de Implementação

Para o desenvolvimento deste projeto o grupo teve que tomar decisões de como o abordar. Posto isto, a nossa linguagem tem as seguintes propriedades:

- Cada variável só pode ser declarada uma única vez;
- Para consultar o valor uma variável esta terá de ter sido declarada posteriormente;
- Para distinguir *strings* de inteiros no *input* fornecido pelo utilizador aquando da execução do programa, foram criadas duas funções diferentes: **read()** e **readInt()**, respetivamente;
- Na declaração do tamanho de um *array*, este terá de ser uma constante;

Quanto à gramática, o grupo tentou desenvolver as produções utilizando recursividade à esquerda em vez de recursividade à direita que, embora não a torna-se incorreta, tornaria a gramática menos eficiente.

A gramática apresenta ainda 6 conflitos *shift/reduce*, não sendo estes problemáticas pois os *lookheads* das produções que o originam são diferentes. Estes conflitos são gerados pelas produções relativas ao símbolo não terminal **ExpRel**.

3.5 Testes realizados e Resultados

De modo a cumprir o pedido no enunciado, foram criados 5 ficheiros que traduzem as seguintes funcionalidades:

- 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 o seu produto;
- Contar e imprimir os números ímpares de uma sequência de número naturais;
- Ler e armazenar N números num array e imprimir os valores por ordem inversa.

Na nossa linguagem, estas seriam possíveis maneiras de implementar estas funcionalidades:

Listing 3.1: Ler 4 números e dizer se podem ser os lados de um quadrado

```
1 int a = readInt()
2 int b = readInt()
3 int c = readInt()
4 int d = readInt()
5 if (a==b){
6     if (b==c){
7         if (c==d){
8             write("Sao lados de um quadrado")
9         } else {
10            write("Nao sao lados de um quadrado")
11        }
12    }
13 }
```



```

12     }else{
13         write("Nao sao lados de um quadrado")
14     }
15 }else{
16     write("Nao sao lados de um quadrado")
17 }

```

Listing 3.2: Ler um inteiro N, depois ler N números e escrever o menor deles

```

1 int n = readInt()
2 int menor = readInt()
3 n=n-1
4 int prox
5 repeat_until(n<=0){
6     prox = readInt()
7     if(prox<menor){
8         menor = prox
9     }
10    n = n-1
11 }
12 write("O menor numero e:")
13 write(menor)

```

Listing 3.3: Ler N (constante do programa) números e calcular o seu produto

```

1 int n = 10
2 int acc = 1
3 int aux
4 repeat_until(n<=0){
5     aux = readInt()
6     acc = acc*aux
7     n=n-1
8 }
9 write(acc)

```

Listing 3.4: Contar e imprimir os números ímpares de uma sequência de número naturais

```

1 int count = 0
2 int aux
3 write("Introduza um numero natural, se deseja concluir insira um numero menor que 0")
4 aux = readInt()
5 repeat_until(aux<0){
6     if(aux % 2 == 1){
7         count = count + 1
8         write("Numero impar introduzido")
9     }
10    aux = readInt()
11 }
12 write("Numeros impares contados: ")
13 write(count)

```

Listing 3.5: Ler e armazenar N números num array e imprimir os valores por ordem inversa

```

1 int n = 10
2 int a[10]

```

```

3 int i
4 int aux
5 repeat_until(i>=n){
6     aux = readInt()
7     a[i] = aux
8     i = i+1
9 }
10 repeat_until(n<=0){
11     write(a[n-1])
12     n=n-1
13 }
14 \paragraph{}

```

O código máquina resultante é:

Listing 3.6: Código máquina 1º exemplo

```

1 read
2 atoi
3 read
4 atoi
5 read
6 atoi
7 read
8 atoi
9 pushg 0
10 pushg 1
11 equal
12 jz else2
13 pushg 1
14 pushg 2
15 equal
16 jz else1
17 pushg 2
18 pushg 3
19 equal
20 jz else0
21 pushes "Sao lados de um quadrado\n"
22 writes
23 jump end0
24 else0:
25 pushes "Nao sao lados de um quadrado\n"
26 writes
27 end0:
28 jump end1
29 else1:
30 pushes "Nao sao lados de um quadrado\n"
31 writes
32 end1:
33 jump end2
34 else2:
35 pushes "Nao sao lados de um quadrado\n"
36 writes
37 end2:

```

Listing 3.7: Código máquina 2º exemplo

```
1 read
2 atoi
3 read
4 atoi
5 pushg 0
6 pushi 1
7 sub
8 storeg 0
9 pushi 0
10 repeat1:
11 pushg 0
12 pushi 0
13 infeq
14 pushi 0
15 equal
16 jz end1
17 read
18 atoi
19 storeg 2
20 pushg 2
21 pushg 1
22 inf
23 jz end0
24 pushg 2
25 storeg 1
26 end0:
27 pushg 0
28 pushi 1
29 sub
30 storeg 0
31 jump repeat1
32 end1:
33 pushs "O menor numero e:\n"
34 writes
35 pushg 1
36 stri
37 writes
38 pushs "\n"
39 writes
```

Listing 3.8: Código máquina 3º exemplo

```
1 pushi 10
2 pushi 1
3 pushi 0
4 repeat0:
5 pushg 0
6 pushi 0
7 infeq
8 pushi 0
9 equal
10 jz end0
11 read
```

```

12 atoi
13 storeg 2
14 pushg 1
15 pushg 2
16 mul
17 storeg 1
18 pushg 0
19 pushi 1
20 sub
21 storeg 0
22 jump repeat0
23 end0:
24 pushg 1
25 stri
26 writes
27 pushs "\n"
28 writes

```

Listing 3.9: Código máquina 4º exemplo

```

1
2 pushi 0
3 pushi 0
4 pushs "Introduza um numero natural, se deseja concluir insira um numero menor que 0\n
   "
5 writes
6 read
7 atoi
8 storeg 1
9 repeat1:
10 pushg 1
11 pushi 0
12 inf
13 pushi 0
14 equal
15 jz end1
16 pushg 1
17 pushi 2
18 mod
19 pushi 1
20 equal
21 jz end0
22 pushg 0
23 pushi 1
24 add
25 storeg 0
26 pushs "Numero impar introduzido\n"
27 writes
28 end0:
29 read
30 atoi
31 storeg 1
32 jump repeat1
33 end1:
34 pushs "Numeros impares contados: \n"

```

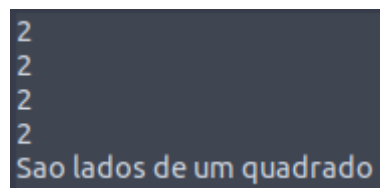
```
35 writes
36 pushg 0
37 stri
38 writes
39 pushs "\n"
40 writes
```

Listing 3.10: Código máquina 5º exemplo

```
1 pushi 10
2 pushn 10
3 pushi 0
4 pushi 0
5 repeat0:
6 pushg 11
7 pushg 0
8 supeq
9 pushi 0
10 equal
11 jz end0
12 read
13 atoi
14 storeg 12
15 pushgp
16 pushi 1
17 padd
18 pushg 11
19 pushg 12
20 storen
21 pushg 11
22 pushi 1
23 add
24 storeg 11
25 jump repeat0
26 end0:
27 repeat1:
28 pushg 0
29 pushi 0
30 infeq
31 pushi 0
32 equal
33 jz end1
34 pushgp
35 pushi 1
36 padd
37 pushg 0
38 pushi 1
39 sub
40 loadn
41 stri
42 writes
43 pushs "\n"
44 writes
45 pushg 0
46 pushi 1
```

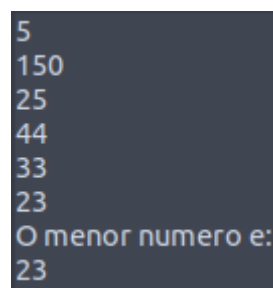
```
47 sub
48 storeg 0
49 jump repeat1
50 endl:
```

Input e Output resultante:



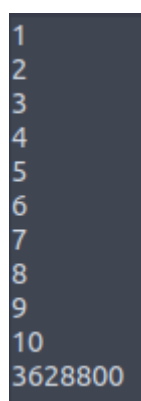
```
2
2
2
2
Sao lados de um quadrado
```

Figura 3.1: Input e Output 1º exemplo



```
5
150
25
44
33
23
O menor numero e:
23
```

Figura 3.2: Input e Output 2º exemplo



```
1
2
3
4
5
6
7
8
9
10
3628800
```

Figura 3.3: Input e Output 3º exemplo

```
Introduza um numero natural, se deseja concluir
25
Numero impar introduzido
23
Numero impar introduzido
22
21
Numero impar introduzido
404
34
1
Numero impar introduzido
-1
Numeros impares contados:
4
```

Figura 3.4: Input e Output 4º exemplo

```
1
2
3
2
3
4
5
6
7
8
8
7
6
5
4
3
2
3
2
1
```

Figura 3.5: Input e Output 5º exemplo

Capítulo 4

Conclusão

Finalizado este trabalho, conseguimos com sucesso criar a nossa linguagem imperativa, que consideramos ser algo idêntica à Linguagem de Programação C e o seu respectivo compilador.

A adaptação ao ambiente da Virtual Machine foi feito sem qualquer tipo de problemas, sendo no entanto necessário um estudo prévio da mesma estudando as várias instruções começando obviamente pelas mais básicas e seguindo as instruções dadas na aula prática onde a VM foi abordada.

O trabalho desenvolvido também incutiu em nós uma mais sensibilidade para o desenvolvimento de gramáticas que solucionam um determinado problema através do módulo YACC e LEX do Python, aumentando assim o nosso conhecimento na Engenharia de Linguagens.

Foram implementadas e testadas todas as funcionalidades pedidas de forma a corresponder ao pedido no enunciado, passando todos os testes a que foram submetidas.

A utilização de uma Linguagem de Programação mais moderna, como o Python para o desenvolvimento da aplicação e as suas respectivas bibliotecas demonstrou ser bastante pedagógico sendo que nenhuma das suas funcionalidades foi de difícil aprendizagem, retirando assim preocupação na utilização das ferramentas e voltando essa preocupação para a matéria afectiva a esta Unidade Curricular.

Apêndice A

Código do Programa

Listing A.1: Código lex.py

```
1 #coding:utf-8
2 import ply.lex as lex
3 import sys
4
5 literals = ['(', ')', ',', '+', '-', '*', '/', '=', '>', '<', '!', '{', '}', '[', ']', ';', '%']
6
7 tokens = ["Num", "If", "Else", "Id", "Repeat", "Int", "Read", "ReadInt", "Write", "String", "For", "And", "Or", "Not"]
8
9 def t_And(t):
10     r'and'
11     return(t)
12 def t_Or(t):
13     r'or'
14     return(t)
15 def t_Not(t):
16     r'not'
17     return(t)
18 def t_For(t):
19     r'for'
20     return(t)
21 def t_If(t):
22     r'if'
23     return(t)
24 def t_Num(t):
25     r'\d+'
26     return(t)
27 def t_Repeat(t):
28     r'repeat_until'
29     return(t)
30 def t_ReadInt(t):
31     r'readInt'
32     return(t)
33 def t_Read(t):
34     r'read'
35     return(t)
36 def t_Int(t):
```

```
37     r'int'
38     return(t)
39 def t_Write(t):
40     r'write'
41     return(t)
42 def t_Else(t):
43     r'else'
44     return(t)
45 def t_Id(t):
46     r'\w+'
47     return(t)
48 def t_String(t):
49     r'"[^"]*" '
50     return(t)
51
52 t_ignore = " \t\n"
53
54 def t_error(t):
55     print("Character ilegal " + t.value(0))
56     t.lexer.skip(1)
57
58 lexer = lex.lex()
```

```

1 #coding:utf-8
2 import ply.yacc as yacc
3 from lex import tokens
4 from lex import literals
5 import sys
6 from os import write
7
8 """
9 LExp : Instrucoes
10 Instrucoes : Instrucoes Instrucao
11             | Instrucao
12 Instrucao : Atribuicao
13            | Operacao
14            | Funcao
15 Atribuicao : int id
16            | int id '[' Num ']'
17            | int id '[' Num ',' Num ']'
18            | int id '=' Operacao
19            | int id '=' ReadInt '(' ')'
20            | id '=' Operacao
21            | id '=' ReadInt '(' ')'
22            | id '[' Operacao ']' '=' Operacao
23            | id '[' Operacao ']' '=' ReadInt '(' ')'
24            | id '[' Operacao ',' Operacao ']' '=' Operacao
25            | id '[' Operacao ',' Operacao ']' '=' ReadInt '(' ')'
26            | id '+' '+'
27            | id '-' '-'
28            | id '+' '=' Operacao
29            | id '-' '=' Operacao
30 Operacao : Operacao '+' Termo
31           | Operacao '-' Termo
32           | Termo
33 Termo : Termo '*' Fator
34        | Termo '/' Fator
35        | Termo '%' Fator
36        | Fator
37 Fator : Num
38        | Id
39        | Id '[' Operacao ']'
40        | Id '[' Operacao ',' Operacao ']'
41        | '(' Operacao ')'
42 Condicional : Condicional OR Cond
43              | Cond
44 Cond : Cond AND Cond2
45       | Cond2
46 Cond2 : NOT Cond
47        | ExpRel
48        | '(' Condicional ')'
49 ExpRel : Operacao '>' Operacao
50        | Operacao '<' Operacao
51        | Operacao '>' '=' Operacao
52        | Operacao '<' '=' Operacao
53        | Operacao '=' '=' Operacao

```

```

54         | Operacao '! ' '=' Operacao
55         | Operacao
56 Funcao : Write '(' String ')'
57         | Write '(' Operacao ')'
58         | ReadInt '(' ')'
59         | Read '(' ')'
60         | Repeat '(' Condicional ')' '{' Instrucoes '}'
61         | If '(' Condicional ')' '{' Instrucoes '}' Else '{' Instrucoes '}'
62         | If '(' Condicional ')' '{' Instrucoes '}'
63         | For '(' Atribuicao ';' Condicional ';' Atribuicao ')' '{' Instrucoes '}'
64 ""
65
66 # Tabela de Simbolos dict{variavel : pos_stack}
67 ts = dict({})
68 # Tabela de Arrays dict{variavel : (pos_stack,tamanho)}
69 ta = dict({})
70 # Tabela de Arrays2D dict{variavel : (pos_stack,tamanho_linha,tamnhho_coluna)}
71 tm = dict({})
72 #variavel que aponta para a posição atual da stack
73 pos_stack=0
74 #variavel para tornar as etiquetas unicas
75 func_nr = 0
76
77 def p_LExp(p):
78     "LExp : Instrucoes"
79     p[0] = p[1]
80
81 def p_Instrucoes_Instrucao(p):
82     "Instrucoes : Instrucoes Instrucao"
83     p[0] = p[1] + p[2]
84
85 def p_Instrucoes_Vazio(p):
86     "Instrucoes : Instrucao"
87     p[0] = p[1]
88
89 def p_Instrucao_Atribuicao(p):
90     "Instrucao : Atribuicao"
91     p[0] = p[1]
92
93 def p_Instrucao_Operacao(p):
94     "Instrucao : Operacao"
95     p[0] = p[1]
96
97 def p_Instrucao_Funcao(p):
98     "Instrucao : Funcao"
99     p[0] = p[1]
100
101
102
103 def p_Funcao_For(p):
104     "Funcao : For '(' Atribuicao ';' Condicional ';' Atribuicao ')' '{' Instrucoes
        '}'"
105     global func_nr
106     p[0] = (str(p[3]) + "\nfor_" + str(func_nr) + ":\n" + str(p[5]) + "\njz fim_for_"

```

```

107         + str(func_nr) + str(p[10]) + str(p[7]) + "\njump for_" + str(func_nr)
108         + "\nfin_for_" + str(func_nr) + ":" )
109     func_nr += 1
110
111 def p_Funcao_Write_String(p):
112     "Funcao : Write '(' String ')'"
113     p[3] = p[3][: -1] + "\\n\"
114     p[0] = "\npushs " + p[3] + "\nwrites"
115
116 def p_Funcao_Write_Operacao(p):
117     "Funcao : Write '(' Operacao ')'"
118     p[0] = p[3] + "\nstri" + "\nwrites \npushs " + "\"\\n\" + "\nwrites"
119
120 def p_Funcao_ReadInt(p):
121     "Funcao : ReadInt '(' ')'"
122     global pos_stack
123     p[0] = "\nread \natoi"
124     pos_stack+=1
125
126 def p_Funcao_Read(p):
127     "Funcao : Read '(' ')'"
128     global pos_stack
129     p[0] = "\nread"
130     pos_stack+=1
131
132 def p_Funcao_Repeat(p):
133     "Funcao : Repeat '(' Condicional ')' '{' Instrucoes '}'"
134     global func_nr
135     p[0] = ("\nrepeat" + str(func_nr) + ":"
136           + p[3] + "\npushi 0\nequal \njz end" + str(func_nr)
137           + p[6]
138           + "\njump repeat" + str(func_nr)
139           + "\nend" + str(func_nr) + ":"
140           )
141     func_nr+=1
142
143 def p_Funcao_IfElse(p):
144     "Funcao : If '(' Condicional ')' '{' Instrucoes '}' Else '{' Instrucoes '}'"
145     global func_nr
146     p[0] = (p[3] + "\njz else" + str(func_nr)
147           + p[6]
148           + "\njump end" + str(func_nr)
149           + "\nelse" + str(func_nr) + ":"
150           + p[10]
151           + "\nend" + str(func_nr) + ":"
152           )
153     func_nr+=1
154
155 def p_Funcao_If(p):
156     "Funcao : If '(' Condicional ')' '{' Instrucoes '}'"
157     global func_nr
158     global pos_stack
159     p[0] = (p[3] + "\njz end" + str(func_nr)
160           + p[6]

```

```

161         + "\nend" + str(func_nr) + ":"
162     )
163     func_nr+=1
164
165
166
167 def p_Atribuicao_Inc_Id(p):
168     "Atribuicao : Id '+' '+'"
169     if(p[1] in ts):
170         p[0] = "\npushg " + str(ts[p[1]]) + "\npushi 1\nadd\nstoreg " + str(ts[p[1]])
171
172 def p_Atribuicao_Dec_Id(p):
173     "Atribuicao : Id '-' '-' "
174     if(p[1] in ts):
175         p[0] = "\npushi 1\npushg " + str(ts[p[1]]) + "\nsb\nstoreg" + str(ts[p[1]])
176
177 def p_Atribuicao_Inc_Id_Op(p):
178     "Atribuicao : Id '+' '=' Operacao"
179     if(p[1] in ts):
180         global pos_stack
181         p[0] = "\npushg " + str(ts[p[1]]) + str(p[4]) + "\nadd\nstoreg " + str(ts[p
182             [1]])
183         pos_stack-=1
184
185 def p_Atribuicao_Dec_Id_Op(p):
186     "Atribuicao : Id '-' '=' Operacao"
187     if(p[1] in ts):
188         global pos_stack
189         p[0] = str(p[4]) + "\npushg " + str(ts[p[1]]) + "\nsb\nstoreg " + str(ts[p
190             [1]])
191         pos_stack-=1
192
193 def p_Atribuicao_Declaracao_Zero(p):
194     "Atribuicao : Int Id"
195     if(p[2] not in ts and p[2] not in ta and p[2] not in tm):
196         global pos_stack
197         ts[p[2]] = pos_stack
198         p[0] = "\npushi 0"
199         pos_stack+=1
200
201 def p_Atribuicao_Declaracao_Input(p):
202     "Atribuicao : Int Id '=' ReadInt '(' ' ')"
203     if(p[2] not in ts and p[2] not in ta and p[2] not in tm):
204         global pos_stack
205         ts[p[2]] = pos_stack
206         p[0] = "\nread \natoi"
207         pos_stack+=1
208
209 def p_Atribuicao_Declaracao(p):
210     "Atribuicao : Int Id '=' Operacao"
211     if(p[2] not in ts and p[2] not in ta and p[2] not in tm):
212         global pos_stack
213         ts[p[2]] = pos_stack-1
214         p[0] = str(p[4])

```

```

213
214 def p_Atribuicao_Declaracao_Array(p):
215     "Atribuicao : Int Id '[' Num ']' "
216     if(p[2] not in ts and p[2] not in ta and p[2] not in tm):
217         global pos_stack
218         ta[p[2]] = (pos_stack, int(p[4]))
219         p[0] = "\npushn " + str(p[4])
220         pos_stack += int(p[4])
221
222 def p_Atribuicao_Declaracao_Matriz(p):
223     "Atribuicao : Int Id '[' Num ', ' Num ']' "
224     if(p[2] not in ts and p[2] not in ta and p[2] not in tm):
225         global pos_stack
226         tm[p[2]] = (pos_stack, int(p[4]), int(p[6]))
227         p[0] = "\npushn " + str(int(p[4])*int(p[6]))
228         pos_stack += int(p[4])*int(p[6])
229
230 def p_Atribuicao_Alt(p):
231     "Atribuicao : Id '=' Operacao"
232     if(p[1] in ts):
233         global pos_stack
234         p[0] = str(p[3]) + "\nstoreg " + str(ts[p[1]])
235         pos_stack-=1
236
237 def p_Atribuicao_Input(p):
238     "Atribuicao : Id '=' ReadInt '(' ' )' "
239     if(p[1] in ts):
240         global pos_stack
241         p[0] = "\nread \natoi \nstoreg " + str(ts[p[1]])
242         pos_stack-=1
243
244 def p_Atribuicao_Array(p):
245     "Atribuicao : Id '[' Operacao ']' '=' Operacao"
246     if(p[1] in ta):
247         global pos_stack
248         p[0] = "\npushgp \npushi " + str(ta[p[1]][0]) + "\npadd" + p[3] + p[6] + "\nstoren"
249         pos_stack-=2
250
251 def p_Atribuicao_Array_Input(p):
252     "Atribuicao : Id '[' Operacao ']' '=' ReadInt '(' ' )' "
253     if(p[1] in ta):
254         global pos_stack
255         p[0] = "\npushgp \npushi " + str(ta[p[1]][0]) + "\npadd" + p[3] + "\nread \natoi \nstoren"
256         pos_stack-=1
257
258 def p_Atribuicao_Matriz(p):
259     "Atribuicao : Id '[' Operacao ', ' Operacao ']' '=' Operacao"
260     if(p[1] in tm):
261         global pos_stack
262         p[0] = "\npushgp " + "\npushi " + str(tm[p[1]][0]) + "\npadd" + p[3] + "\npushi " + str(tm[p[1]][2]) + "\nmul" + p[5] + "\nadd" + p[8] + "\nstoren"
263         pos_stack-=2

```

```

264
265 def p_Atribuicao_Matriz_Input(p):
266     "Atribuicao : Id '[' Operacao ',' Operacao ']' '=' ReadInt '(' ' ') '"
267     if(p[1] in tm):
268         global pos_stack
269         p[0] = "\npushgp" + "\npushi " + str(ta[p[1]][0]) + "\npadd" + p[3] + "\n\npushi " + str(tm[p[1]][1]) + "\nmul" + p[5] + "\nadd \nread \natoi \nstoren"
270         pos_stack-=1
271
272
273
274 def p_Operacao_Mais(p):
275     "Operacao : Operacao '+' Termo"
276     p[0] = str(p[1]) + str(p[3]) + "\nadd"
277     global pos_stack
278     pos_stack-=1
279
280 def p_Operacao_Menos(p):
281     "Operacao : Operacao '-' Termo"
282     p[0] = str(p[1]) + str(p[3]) + "\nsub"
283     global pos_stack
284     pos_stack-=1
285
286 def p_Operacao_Termo(p):
287     "Operacao : Termo"
288     p[0] = str(p[1])
289
290
291
292 def p_Termo_Mul(p):
293     "Termo : Termo '*' Fator"
294     p[0] = str(p[1]) + str(p[3]) + "\nmul"
295     global pos_stack
296     pos_stack-=1
297
298 def p_Termo_Div(p):
299     "Termo : Termo '/' Fator"
300     p[0] = str(p[1]) + str(p[3]) + "\ndiv"
301     global pos_stack
302     pos_stack-=1
303
304 def p_Termo_Mod(p):
305     "Termo : Termo '%" Fator"
306     p[0] = str(p[1]) + str(p[3]) + "\nmod"
307     global pos_stack
308     pos_stack-=1
309
310 def p_Termo_Fator(p):
311     "Termo : Fator"
312     p[0] = str(p[1])
313
314
315

```



```

316 def p_Fator_Num(p):
317     "Fator : Num"
318     p[0] = "\npushi " + str(p[1])
319     global pos_stack
320     pos_stack+=1
321
322 def p_Fator_Id(p):
323     "Fator : Id"
324     p[0] = "\npushg " + str(ts[p[1]])
325     global pos_stack
326     pos_stack+=1
327
328 def p_Fator_Array(p):
329     "Fator : Id '[' Operacao ']' "
330     global pos_stack
331     p[0] = "\npushgp" + "\npushi " + str(ta[p[1]][0]) + "\npadd" + p[3] + "\nloadn"
332     pos_stack-=1
333
334 def p_Fator_Matriz(p):
335     "Fator : Id '[' Operacao ',' Operacao ']' "
336     global pos_stack
337     p[0] = "\npushgp" + "\npushi " + str(tm[p[1]][0]) + "\npadd" + p[3] + p[5] + "\n\nmul\nloadn"
338     pos_stack-=1
339     func_nr+=1
340
341 def p_Fator_Operacao(p):
342     "Fator : '(' Operacao '"
343     p[0] = p[2]
344
345
346
347 def p_ExpRel_Maior(p):
348     "ExpRel : Operacao '>' Operacao"
349     p[0] = str(p[1]) + str(p[3]) + "\nsup"
350     global pos_stack
351     pos_stack-=2
352
353 def p_ExpRel_Menor(p):
354     "ExpRel : Operacao '<' Operacao"
355     p[0] = str(p[1]) + str(p[3]) + "\ninf"
356     global pos_stack
357     pos_stack-=2
358
359 def p_ExpRel_MaiorIgual(p):
360     "ExpRel : Operacao '>' '=' Operacao"
361     p[0] = str(p[1]) + str(p[4]) + "\nsupeq"
362     global pos_stack
363     pos_stack-=2
364
365 def p_ExpRel_MenorIgual(p):
366     "ExpRel : Operacao '<' '=' Operacao"
367     p[0] = str(p[1]) + str(p[4]) + "\ninfeq"
368     global pos_stack

```

```

369     pos_stack-=2
370
371 def p_ExpRel_Igual(p):
372     "ExpRel : Operacao '=' Operacao"
373     p[0] = str(p[1]) + str(p[4]) + "\nequal"
374     global pos_stack
375     pos_stack-=2
376
377 def p_ExpRel_Diferente(p):
378     "ExpRel : Operacao '!' Operacao"
379     p[0] = str(p[1]) + str(p[4]) + "\nequal \npushi 0\nequal"
380     global pos_stack
381     pos_stack-=2
382
383 def p_ExpRel_Exp(p):
384     "ExpRel : Operacao"
385     p[0] = str(p[1])
386
387
388
389 def p_Condicional_Or_Cond(p):
390     "Condicional : Condicional Or Cond"
391     p[0] = p[1] + p[3] + "\nadd" + p[1] + p[3] + "\nmul\nsub"
392
393 def p_Condicional_Cond(p):
394     "Condicional : Cond"
395     p[0] = p[1]
396
397 def p_Cond_And_Cond2(p):
398     "Cond : Cond And Cond2"
399     p[0] = p[1] + p[3] + "\nmul"
400
401 def p_Cond_Cond2(p):
402     "Cond : Cond2"
403     p[0] = p[1]
404
405 def p_Cond2_Not(p):
406     "Cond2 : Not Condicional"
407     p[0] = p[2] + "\npushi 0 \nequal"
408
409 def p_Cond2_ExpRel(p):
410     "Cond2 : ExpRel"
411     p[0] = p[1]
412
413 def p_Cond2_Condicional(p):
414     "Cond2 : '(' Condicional ')'"
415     p[0] = p[2]
416
417
418
419 def p_error(p):
420     print("Syntax error in input: ",p)
421
422 #build the parser

```

```
423 parser = yacc.yacc()
424
425 #reading input
426 linhas = ""
427 for line in sys.stdin:
428     linhas += line
429 #writing to file the result
430 f = open("codigo.vm", "a")
431 result = parser.parse(linhas)
432 f.write(result)
433 f.close()
```
