Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 2

Relatório de Desenvolvimento

Ricardo Gomes (a93785)

Hugo Pereira (a93752)

30 de maio de 2021

Resumo

O TP2 realizado pelo nosso grupo, na unidade curricular de *Processamento de Linguagens*, consistiu em selecionar um tipo de linguagem imperativa simples, à nossa escolha, no caso *C*, e criar um compilador capaz de traduzir o código de *C* para linguagem de uma *máquina de stack virtual*.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

A realização do presente trabalho foi-nos proposto no âmbito da unidade curricular de Processamento de Linguagens.

Definimos como objectivos, aumentar a capacidade de escrever gramáticas independentes de contexto e gramáticas tradutoras.

Relativamente ao projeto, foi desenvolvido um compilador capaz de gerar código para uma máquina de stack virtual a partir de uma gramática tradutora, usando o método da tradução dirigida pela sintaxe. Para o efeito utilizamos o *Yacc*, versão Ply do Python, completado pelo *Lex*, recorrendo a **C** como linguagem de programação imperativa simples,

Estrutura do Relatório

No decorrer deste relatório, podemos verificar que no capítulo 1 está presente a introdução referente a este trabalho, no capítulo 2 faz-se uma análise do problema proposto, no capítulo 3 detalhamos a resolução do problema e mostrámos a forma como definimos as regras de forma a responder às querys do enunciado, no capítulo 4 apresentamos os testes realizados, assim como os seus resultados, no capítulo 5 apresentamos uma conclusão e no Apêndice disponibilizamos o código.

Capítulo 2

Descrição do Problema

Neste trabalho pretendeu-se desnvolver um compilador para a linguagem imperativa **C** com base numa gramática independente de contexto utilizando os módulos *Yacc*, versão Ply do Python, e *Lex*.

Para resolver o trabalho, o grupo inicialmente teve de definir os *tokens* que iria utilizar, as *palavras reservadas* referentes à linguagem **C** e as *produções* que iriamos utilizar no decorrer do trabalho.

Capítulo 3

Concepção/desenho da Resolução

3.1 Lex

3.1.1 Tokens

No início do trabalho, o grupo decidiu definir como tokens os seguintes elementos, de forma a melhor conseguir traduzir de forma léxica a linguagem imperativa utilizada

```
1 tokens = ['NUM', 'REAL', 'ID', 'IGUAL', 'PV', 'VIR', 'ADD', 'SUB',
2           'MUL', 'DIV', 'PE', 'PD', 'ENDID', 'CE', 'CD', 'GT', 'GE', 'LT',
3           'LE', 'EQ', 'DIF', 'TEXT', 'RE', 'RD', 'MOD']
```

Listing 1: Tokens

De forma a utilizar os tokens definidos, tivemos que definir as regras refernte a cada token:

3.1.2 Palvras reservadas

Para além dos tokens, definimos também um grupo de palavras reservadas. As palavras reservadas definidas, são também elas palavras reservadas da linguagem imperativa escolhida, no caso **C**.

3.2 Yacc

Após a definição dos tokens a utilizar e das palavras reservadas o grupo passou à definição das produçoes

3.2.1 File

Regra de produção referente ao Axioma do ficheiro. Sendo o Inic referente à Inicialização e o BlocoInst refernte ao resto do programa, ou seja, as instruções.

3.2.2 Inic

A regra de produção Inic foi dividida em DeclVar, que devolve que variáveis foram inicializadas e na sua chamada Inic.

3.2.3 DeclVar

A regra de produção DeclVar pode devolver uma variável inteira *INT* ou uma variável float *FLOAT*

3.2.4 BlocoInst

Passando à regra de produção BlocoInst, esta está definida como podendo ser uma ou mais instruções, *Inst*

3.2.5 Inst

A regra de produção *Inst* foi definida como podendo ser uma atribuição, um print, uma instrução para ler, um if e um do while. Como exemplo de uma atribuição, podemos considerar **mul = mul * 3**, como exemplo de um print, **printf(...);**, de uma instrução de leitura, **scanf(...);**, um if, **if(...)...** e finalmente de um do while, **do... while (...)s**

3.2.6 Atribuicao

A regra de produção Atribuição pode ser uma expressão atribuída a uma variável, a uma posição de um array ou a uma posiçao de um array definida como variável. Foi também definida a regra de produção RestoAtrib, assim como outras, em que definimos a ordem pelo qual o programa deve realizar contas, como adição, multiplicação, módulo, etc. Estas regras estarão presentes no final do relatório em ***Apêndice***

3.2.7 Printf

A regra de produção Printf está definida para apanhar os *printf(..);*. O resto das regras associadas ao Printf encontram-se também no ***Apêndice***

3.2.8 Scanf

A regra de produção Scanf está definida para apanhar os *scanf(..);*. O resto das regras associadas ao Scanf encontram-se também no ***Apêndice***

3.2.9 If

A regra de produção If foi definida como tendo uma condição, *Cond*, e um bloco de instrução referentes ao If, *BlocoInstIf*.

3.2.10 Cond

A regra de produção Cond é onde se regista a condição para que o *If* ocorra. Para isso foi definida uma regra *Conta*, que se refere ao que vai ser comparado com expressão relacional, *ExpRel*, também estas, assim como outras, presentes no ***Apêndice***

3.2.11 BlocoInstIf

A regra de produção BlocoInstIf é onde se encontram as instruções a serem realizadas caso o *If* se verifique. É composto, também este, por instruções.

3.2.12 DoWhile

A regra de produção DoWhile é definida como tendo um bloco de instruções a realizar, *BlocoDoWhile*, enquanto uma condição, *CondDo*, se verificar

3.2.13 BlocoDoWhile

A regra de produção BlocoDoWhile é onde se encontram as instruções a serem executadas. É composto, também este, por instruções.

3.2.14 CondDo

A regra de produção CondDo é semelhante à regra de produção Cond. A diferença é que o resultado de *ExpRelDo* é diferente de *ExpRel*. Ambas estas regras estão presentes no ***Apêndice***

3.3 Estruturas de Dados

Para a elaboração do trabalho foram criados alguns dicionários e algumas variáveis soma. **Registos** era para guardar variáveis e a posição na stack onde elas se encontravam. **Tipos** era para guardar variáveis e o seu tipo de dados. **ArrayTam** era para guardar o tamanho associado a um array.

[H]

```
t_MOD = r'%'
t_NUM = r'\d+'
t_REAL = r'[+\-]?\d*\.\d+'
t_IGUAL = r'='
t_PV = r';'
t_VIR = r','
t_ADD = r'\+'
t_SUB = r'\-'
t_MUL = r'\*'
t_DIV = r'\/'
t_PE = r'\('
t_PD = r'\)'
t_ENDID = r'&\w+'
t_CE = r'\{'
t_CD = r'\}'
t_GT = r'>'
t_GE = r'>='
t_LT = r'<'
t_LE = r'<='
t_EQ = r'=='
t_DIF = r'!='
t_RE = r'\['
t_RD = r'\]'


def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = reservadas.get(t.value, 'ID')  # Check for reserved words
    return t


def t_TEXT(t):
    r'"[\w\s\\%:]+"'
    t.type = reservadas.get(t.value, 'TEXT')  # Check for reserved words
    return t


t_ignore = " \t\n"
```

Listing 2: Regras dos Tokens

```
reservadas = {
    "float": "FLOAT",
    "int": "INT",
    "printf": "PRINT",
    "scanf": "SCAN",
    "do": "DO",
    "while": "WHILE",
    "if": "IF",
}
```

Listing 3: Palavras Reservadas

```
    def p_File(p):
    "File : Inic BlocoInst"
    p[0] = p[1] + p[2]
```

Listing 4: Regra de produção File

```
def p_Inic(p):
    "Inic : DeclVar Inic"
    p[0] = p[1] + p[2]

def p_Inic_vazio(p):
    "Inic : "
    p[0] = ""
```

Listing 5: Regras de produção Inic

```
def p_DeclVar_int(p):
    "DeclVar : INT RestoDeclInt"
    p[0] = p[2]

def p_DeclVar_char(p):
    "DeclVar : FLOAT RestoDeclFloat"
    p[0] = p[2]
```

Listing 6: Regra de produçao DeclVar

```
def p_BlocoInst_inst(p):
    "BlocoInst : Inst BlocoInst"
    p[0] = "START\n" + p[1] + p[2]

def p_BlocoInst_vazio(p):
    "BlocoInst : "
    p[0] = ""
```

Listing 7: Regra de produção BlocoInst

```
def p_Inst_atribuicao(p):
    "Inst : Atribuicao"
    p[0] = p[1]

def p_Inst_print(p):
    "Inst : Printf"
    p[0] = p[1]

def p_Inst_ler(p):
    "Inst : Scanf"
    p[0] = p[1]

def p_Inst_if(p):
    "Inst : If"
    p[0] = p[1]

def p_Inst_dowhile(p):
    "Inst : DoWhile"
    p[0] = p[1]
```

Listing 8: Regra de produção Inst

```
def p_Atribuicao_id(p):
    "Atribuicao : ID IGUAL RestoAtrib"
    p[0] = p[3] + "STOREG " + str(p.parser.registos.get(p[1])) + "\n"

def p_Atribuicao_idarraynum(p):
    "Atribuicao : ID RE NUM RD IGUAL RestoAtrib"
    p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(p[1])) + "\nPADD\n" + "PUSHI " + p[3] +
    "\n" + p[5] + "\nSTOREN\n"

def p_Atribuicao_idarrayid(p):
    "Atribuicao : ID RE ID RD IGUAL RestoAtrib"
    p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(p[1])) + "\nPADD\n" + "PUSHG " +
    str(p.parser.registos.get(p[3])) + "\n"
```

Listing 9: Regra de produção Atribuicao

```
def p_Printf_print(p):
    "Printf : PRINT PE TEXT RestoPrintf"
    p[0] = "PUSHS " + p[3] + "\n" + "WRITES" + "\n" + p[4]

def p_RestoPrintf_pd(p):
    "RestoPrintf : PD PV"
    p[0] = ""

def p_RestoPrintf_vir(p):
    "RestoPrintf : VIR ContPrintf"
    p[0] = p[2]
```

Listing 10: Regra de produção Printf

```
def p_Scanf_scanf(p):
    "Scanf : SCAN PE TEXT VIR RestoScanf"
    p[0] = p[5]
```

Listing 11: Regra de produção Scanf

```
def p_If_if(p):
    "If : IF Cond CE BlocoInstIf CD"
    parser.somaIf += 1
    p[0] = p[2] + "\nJZ Endif" + str(parser.somaIf) + "\n" + p[4] + "\nEndif" + str(parser.somaIf) + ":\n"
```

Listing 12: Regra de produção If

```
def p_Cond_exp(p):
    "Cond : PE Conta ExpRel Conta PD"
    p[0] = p[2] + p[4] + p[3]

def p_Cond_conta(p):
    "Cond : Conta"
    p[0] = p[1]
```

Listing 13: Regra de produção Cond

```
def p_BlocoInstIf_inst(p):
    "BlocoInstIf : InstBlocoIf BlocoInstIf"
    p[0] = p[1] + p[2]
```

Listing 14: Regra de produção BlocoInstIf

```
def p_DoWhile_do(p):
    "DoWhile : DO CE BlocoDoWhile CD WHILE CondDo PV"
    parser.somaDoWhile += 1
    p[0] = "DoWhile" + str(parser.somaDoWhile) + ":\n" + p[3] + "\n" + p[6] + "\nJZ DoWhile" +
    str(parser.somaDoWhile) + "\n"
```

Listing 15: Regra de produção DoWhile

```
def p_BlocoDoWhile_inst(p):
    "BlocoDoWhile : InstBlocoDo BlocoDoWhile"
    p[0] =p[1] + p[2]

def p_BlocoDoWhile_vazio(p):
    "BlocoDoWhile : "
    p[0] = ""
```

Listing 16: Regra de produção BlocoDoWhile

```
def p_CondDo_exp(p):
    "CondDo : PE Conta ExpRelDo Conta PD"
    p[0] = p[2] + p[4] + p[3]

def p_CondDo_conta(p):
    "CondDo : Conta"
    p[0] = p[1]
```

Listing 17: Regra de produção CondDo

```
        parser.registos = {}
        parser.tipos = {}
        parser.arraysTam = {}
        parser.endArray = 0
        parser.soma = 0
        parser.somaIf = 0
        parser.somaDoWhile = 0
```

Listing 18: Estruturas de Dados

Capítulo 4

Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Durante a elaboração do trabalho o grupo apercebeu-se da existência de alguns problemas no trabalho. Um desses problemas foi o facto de não conseguirmos ler instruções com quebras de linha, apesar de termos definido o t_ignore

4.2 Testes realizados e Resultados

Mostram-se a seguir alguns resultados obtidos referentes às queries colocadas:

```c
int l1;
int l2;
int l3;
int l4;
int i = 0;

printf("Digite o lado 1: \n");
scanf("%i", &l1);

printf("Digite o lado 2: \n");
scanf("%i", &l2);

printf("Digite o lado 3: \n");
scanf("%i", &l3);

printf("Digite o lado 4: \n");
scanf("%i", &l4);

if (l1 == l2){ if (l1 == l3){ if (l1 == l4){printf("Sao os lados de um quadrado\n");i=1;}}}

if (i==0){printf("Nao sao os lados de um quadrado\n");}
```

Listing 19: Algoritmo da query 1

```c
int n;
int menor;
int ncomp;

printf("Insira um N: \n");
scanf("%d", &n);


printf("Insira os numeros: \n");
scanf("%d", &menor);

n = n - 1;

do { scanf("%d", &ncomp); if(ncomp < menor){ menor = ncomp; } n = n - 1;} while (n > 0);

printf("Menor: %d ", menor);
```

Listing 20: Algoritmo da query 2

```
PUSHI 0
PUSHI 0
PUSHI 0
PUSHI 0
PUSHI 0
START
PUSHS "Digite o lado 1: \n"
WRITES
START
READ
ATOI
STOREG 0
START
PUSHS "Digite o lado 2: \n"
WRITES
START
READ
ATOI
STOREG 1
START
PUSHS "Digite o lado 3: \n"
WRITES
START
READ
ATOI
STOREG 2
START
PUSHS "Digite o lado 4: \n"
WRITES
START
READ
ATOI
STOREG 3
START
PUSHG 0
PUSHG 1
EQUAL

JZ Endif3
PUSHG 0
PUSHG 2
EQUAL

JZ Endif2
PUSHG 0
PUSHG 3
EQUAL

JZ Endif1
PUSHS "Sao os lados de um quadrado\n"
WRITES
PUSHI 1
STOREG 4

Endif1:

Endif2:

Endif3:
START
PUSHG 4
PUSHI 0
EQUAL

JZ Endif4
PUSHS "Nao sao os lados de um quadrado\n"
WRITES

Endif4:
STOP
```

Figura 4.1: Resultado query 1

```
PUSHI 0
PUSHI 0
PUSHI 0
START
PUSHS "Insira um N: \n"
WRITES
START
READ
ATOI
STOREG 0
START
PUSHS "Insira os numeros: \n"
WRITES
START
READ
ATOI
STOREG 1
START
PUSHG 0
PUSHI 1
SUB
STOREG 0
START
DoWhile1:
READ
ATOI
STOREG 2
PUSHG 2
PUSHG 1
INF

JZ Endif1
PUSHG 2
STOREG 1

Endif1:
PUSHG 0
PUSHI 1
SUB
STOREG 0

PUSHG 0
PUSHI 0
INFEQ

JZ DoWhile1
START
PUSHS "Menor: %d "
WRITES
PUSHG 1
WRITEI
STOP
```

Figura 4.2: Resultado query 2

```c
int n;
int mul = 1;
int i = 0;
int valor;

printf("Insira um numero: \n");
scanf("%d", &n);

printf("Insira os numeros: \n");

do{scanf("%d", &valor); mul = mul * valor; i = i + 1; } while(i < n);

printf("Resultado: %d ", mul);
```

Listing 21: Algoritmo da query 3

```c
int n[6];
int impares = 0;
int valor;
int i = 0;


printf("Insira os valores: \n");

do{ scanf("%d", &n[i]); i = i + 1; } while (i < 6);


do{ i = i - 1; if((n[i] % 2) == 1){impares = impares + 1; printf("%d\n", n[i]) } } while(i > 0);


printf("impares: %d\n", impares);
```

Listing 22: Algoritmo da query 4

```c
int n[5];

int i = 0;

printf("Insira os valores: \n");

do{ scanf("%d", &n[i]); i = i + 1; } while (i < 5);

do{ i = i - 1; printf("valor: %d \n", n[i]); printf("\n"); } while(i > 0);
```

Listing 23: Algoritmo da query 5

```
PUSHI 0
PUSHI 1
PUSHI 0
PUSHI 0
START
PUSHS "Insira um numero: \n"
WRITES
START
READ
ATOI
STOREG 0
START
PUSHS "Insira os numeros: \n"
WRITES
START
DoWhile1:
READ
ATOI
STOREG 3
PUSHG 1
PUSHG 3
MUL
STOREG 1
PUSHG 2
PUSHI 1
ADD
STOREG 2

PUSHG 2
PUSHG 0
SUPEQ

JZ DoWhile1
START
PUSHS "Resultado: %d "
WRITES
PUSHG 1
WRITEI
STOP
```

Figura 4.3: Resultado query 3

```
PUSHN 6
PUSHI 0
PUSHI 0
PUSHI 0
START
PUSHS "Insira os valores: \n"
WRITES
START
DoWhile1:
PUSHGP
PUSHI 0
PADD
PUSHG 8
READ
ATOI
STOREN
PUSHG 8
PUSHI 1
ADD
STOREG 8

PUSHG 8
PUSHI 6
SUPEQ

JZ DoWhile1
START
PUSHS "impares: %d\n"
WRITES
PUSHG 6
WRITEI
STOP
```

Figura 4.4: Resultado query 4

```
PUSHN 5
PUSHI 0
START
PUSHS "Insira os valores: \n"
WRITES
START
DoWhile1:
PUSHGP
PUSHI 0
PADD
PUSHG 5
READ
ATOI
STOREN
PUSHG 5
PUSHI 1
ADD
STOREG 5

PUSHG 5
PUSHI 5
SUPEQ

JZ DoWhile1
START
DoWhile2:
PUSHG 5
PUSHI 1
SUB
STOREG 5
PUSHS "valor: %d \n"
WRITES
PUSHGP
PUSHI 0
PADD
PUSHG 5
LOADN
WRITEI
PUSHS "\n"
WRITES

PUSHG 5
PUSHI 0
INFEQ

JZ DoWhile2
STOP
```

Figura 4.5: Resultado query 5

Capítulo 5

Conclusão

Apesar do trabalho prático não estar totalmente concluído, podemos afirmar que na generalidade, os objetivos foram alcansados. Fomos capazes de responder a todas as questões menos à query 4, sendo importante realçar o interesse e empenho de ambos os elementos no decorrer do trabalho, para que tal fosse possível. Este trabalho serviu para que os elementos do grupo melhorassem as suas capacidades, relativamente à escrita de gramáticas.

Apêndice A

Código do Programa

Lista-se a seguir o código do programa que foi desenvolvido.

A.1 Lex

```
1 import ply.lex as lex
2
3 reservadas = {
4     "float": "FLOAT",
5     "int": "INT",
6     "printf": "PRINT",
7     "scanf": "SCAN",
8     "do": "DO",
9     "while": "WHILE",
10    "if": "IF",
11 }
12
13 tokens = ['NUM', 'REAL', 'ID', 'IGUAL', 'PV', 'VIR', 'ADD', 'SUB',
```

```
14              'MUL', 'DIV', 'PE', 'PD', 'ENDID', 'CE', 'CD', 'GT', 'GE', 'LT',
15              'LE', 'EQ', 'DIF', 'TEXT', 'RE', 'RD', 'MOD']
16
17 tokens += list(reservadas.values())
18
19 t_MOD = r'%'
20 t_NUM = r'\d+'
21 t_REAL = r'[+\-]?\d*\.\d+'
22 t_IGUAL = r'='
23 t_PV = r';'
24 t_VIR = r','
25 t_ADD = r'\+'
26 t_SUB = r'\-'
27 t_MUL = r'\*'
28 t_DIV = r'\/'
29 t_PE = r'\('
30 t_PD = r'\)'
31 t_ENDID = r'&\w+'
32 t_CE = r'\{'
33 t_CD = r'\}'
34 t_GT = r'>'
35 t_GE = r'>='
36 t_LT = r'<'
37 t_LE = r'<='
38 t_EQ = r'=='
39 t_DIF = r'!='
40 t_RE = r'\['
```

```
41 t_RD = r'\]'
42
43
44 def t_ID(t):
45     r'[a-zA-Z_][a-zA-Z_0-9]*'
46     t.type = reservadas.get(t.value, 'ID')  # Check for reserved words
47     return t
48
49
50 def t_TEXT(t):
51     r'"[\w\s\\%:]+"'
52     t.type = reservadas.get(t.value, 'TEXT')  # Check for reserved words
53     return t
54
55
56 t_ignore = " \t\n"
57
58
59 def t_error(t):
60     print("Car cter ilegal: ", t.value[0])
61     t.lexer.skip(1)
62
63
64 # build the lexer
65 lexer = lex.lex()
```

A.2 Yacc

```
1 import sys
2
3 import ply.yacc as yacc
4 import re
5
6 from trab2_lex import tokens
7
8
9 #Production rules
10 def p_File(p):
11     "File : Inic BlocoInst"
12     p[0] = p[1] + p[2]
13
14 def p_Inic(p):
15     "Inic : DeclVar Inic"
16     p[0] = p[1] + p[2]
17
18 def p_Inic_vazio(p):
19     "Inic : "
20     p[0] = ""
21
22 def p_DeclVar_int(p):
23     "DeclVar : INT RestoDeclInt"
24     p[0] = p[2]
25
26 def p_DeclVar_char(p):
```

```
27      "DeclVar : FLOAT RestoDeclFloat"
28      p[0] = p[2]
29
30
31  def p_RestoDeclInt_id(p):
32      "RestoDeclInt : ID OpcDeclInt"
33      p.parser.registos.update({p[1]: p.parser.soma})
34      p.parser.tipos.update({p[1]: 'int'})
35      p.parser.soma += 1
36      p[0] = "PUSHI " + p[2]
37
38  def p_RestoDeclInt_idarray(p):
39      "RestoDeclInt : ID RE NUM RD PV"
40      #nome = re.match(r'[a-zA-Z][a-zA-Z]*', p[1])
41      #tamanho = re.search(r'\d+', p[1])
42      #p.parser.registos.update({nome.group(): p.parser.soma})
43      #p.parser.tipos.update({nome.group(): 'int'})
44      #p.parser.soma += int(tamanho.group())
45      #p[0] = "PUSHN " + tamanho.group() + "\n" + p[2]
46      p.parser.registos.update({p[1]: p.parser.soma})
47      p.parser.tipos.update({p[1]: 'int'})
48      p.parser.soma += int(p[3])
49      p[0] = "PUSHN " + p[3] + "\n"
50
51  def p_RestoDeclInt_idarray_igual(p):
52      "RestoDeclInt : ID RE NUM RD IGUAL RE NUM SegueIgualArrayInt RD PV"
53      p.parser.registos.update({p[1]: p.parser.soma})
```

```
54      p.parser.tipos.update({p[1]: 'int'})
55      p.parser.soma += int(p[3])
56      p[0] = "PUSHN " + p[3] + "\n" + "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get
            (p[1])) + "\nPADD\n" + "PUSHI " + str(p.parser.endArray) + "\nPUSHI " + p[7] +
             "\nSTOREN\n" + p[8]
57
58
59  def p_SegueIgualArrayInt_vir(p):
60      "SegueIgualArrayInt : VIR NUM SegueIgualArrayInt"
61      p.parser.endArray += 1
62      p[0] = "\nPUSHI " + str(p.parser.endArray) + "\nPUSHI " + p[2] + "\nSTOREN\n" + p
            [3]
63
64  def p_SegueIgualArrayInt_vazio(p):
65      "SegueIgualArrayInt : "
66      p[0] = ""
67
68
69  def p_OpcDeclInt_igual(p):
70      "OpcDeclInt : IGUAL SegueIgual"
71      p[0] = p[2]
72
73  def p_OpcDeclInt_pv(p):
74      "OpcDeclInt : PV"
75      p[0] = "0\n"
76
77
```

```
78
79
80
81 def p_RestoDeclFloat_id(p):
82     "RestoDeclFloat : ID OpcDeclFloat"
83     p.parser.registos.update({p[1]: p.parser.soma})
84     p.parser.tipos.update({p[1]: 'float'})
85     p.parser.soma += 1
86     p[0] = "PUSHF " + p[2]
87
88 def p_RestoDeclFloat_idarray(p):
89     "RestoDeclFloat : ID RE NUM RD PV"
90     #nome = re.match(r'[a-zA-Z][a-zA-Z]*', p[1])
91     #tamanho = re.search(r'\d+', p[1])
92     #p.parser.registos.update({nome.group(): p.parser.soma})
93     #p.parser.tipos.update({nome.group(): 'int'})
94     #p.parser.soma += int(tamanho.group())
95     #p[0] = "PUSHN " + tamanho.group() + "\n" + p[2]
96     p.parser.registos.update({p[1]: p.parser.soma})
97     p.parser.tipos.update({p[1]: 'FLOAT'})
98     p.parser.soma += int(p[3])
99     p[0] = "PUSHN " + p[3] + "\n"
100
101 def p_RestoDeclFloat_idarray_igual(p):
102     "RestoDeclFloat : ID RE NUM RD IGUAL RE REAL SegueIgualArrayFloat RD PV"
103     p.parser.registos.update({p[1]: p.parser.soma})
104     p.parser.tipos.update({p[1]: 'float'})
```

```
105     p.parser.soma += int(p[3])
106     p[0] = "PUSHN " + p[3] + "\n" + "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get
            (p[1])) + "\nPADD\n" + "PUSHI " + str(p.parser.endArray) + "\nPUSHF " + p[7] +
             "\nSTOREN\n" + p[8]
107
108
109 def p_SegueIgualArrayFloat_vir(p):
110     "SegueIgualArrayFloat : VIR NUM SegueIgualArrayFloat"
111     p.parser.endArray += 1
112     p[0] = "\nPUSHI " + str(p.parser.endArray) + "\nPUSHF " + p[2] + "\nSTOREN\n" + p
            [3]
113
114 def p_SegueIgualArrayFloat_vir_vazio(p):
115     "SegueIgualArrayFloat : "
116     p[0] = ""
117
118
119
120
121
122
123
124
125
126
127 def p_OpcDeclFloat_igual(p):
128     "OpcDeclFloat : IGUAL SegueIgual"
```

```
129     p[0] = p[2]
130
131 def p_OpcDeclFloat_pv(p):
132     "OpcDeclFloat : PV"
133     p[0] = "0.0\n"
134
135 def p_SegueIgual_num(p):
136     "SegueIgual : NUM PV"
137     p[0] = p[1] + "\n"
138
139 def p_SegueIgual_real(p):
140     "SegueIgual : REAL PV"
141     p[0] = p[1] + "\n"
142
143
144
145
146
147 def p_BlocoInst_inst(p):
148     "BlocoInst : Inst BlocoInst"
149     p[0] = "START\n" + p[1] + p[2]
150
151 def p_BlocoInst_vazio(p):
152     "BlocoInst : "
153     p[0] = ""
154
155 def p_Inst_atribuicao(p):
```

```
156     "Inst : Atribuicao"
157     p[0] = p[1]
158
159 def p_Inst_print(p):
160     "Inst : Printf"
161     p[0] = p[1]
162
163 def p_Inst_ler(p):
164     "Inst : Scanf"
165     p[0] = p[1]
166
167 def p_Inst_if(p):
168     "Inst : If"
169     p[0] = p[1]
170
171 def p_Inst_dowhile(p):
172     "Inst : DoWhile"
173     p[0] = p[1]
174
175
176
177 def p_Atribuicao_id(p):
178     "Atribuicao : ID IGUAL RestoAtrib"
179     p[0] = p[3] + "STOREG " + str(p.parser.registos.get(p[1])) + "\n"
180
181 def p_Atribuicao_idarraynum(p):
182     "Atribuicao : ID RE NUM RD IGUAL RestoAtrib"
```

```
183     #nome = re.match(r'[a-zA-Z][a-zA-Z]*', p[1])
184     #tamanho = re.search(r'\d+', p[1])
185     #p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(nome.group())) + "\
            nPADD\n" + "PUSHI " + tamanho.group() + "\n" + p[3] + "\nSTOREN\n"
186     p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(p[1])) + "\nPADD\n" + "
            PUSHI " + p[3] + "\n" + p[5] + "\nSTOREN\n"
187
188 def p_Atribuicao_idarrayid(p):
189     "Atribuicao : ID RE ID RD IGUAL RestoAtrib"
190     #nome = re.match(r'[a-zA-Z][a-zA-Z]*', p[1])
191     #tamanho = re.search(r'\d+', p[1])
192     #p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(nome.group())) + "\
            nPADD\n" + "PUSHI " + tamanho.group() + "\n" + p[3] + "\nSTOREN\n"
193     p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(p[1])) + "\nPADD\n" + "
            PUSHG " + str(p.parser.registos.get(p[3])) + "\n" + p[5] + "\nSTOREN\n"
194
195 def p_RestoAtrib_add(p):
196     "RestoAtrib : Exp ADD Exp PV"
197     p[0] = p[1] + p[3] + "ADD\n"
198
199 def p_RestoAtrib_sub(p):
200     "RestoAtrib : Exp SUB Exp PV"
201     p[0] = p[1] + p[3] + "SUB\n"
202
203 def p_RestoAtrib_exp(p):
204     "RestoAtrib : Exp PV"
205     p[0] = p[1]
```

```
206
207
208 def p_Exp_mul(p):
209     "Exp : Exp2 MUL Exp2"
210     p[0] = p[1] + p[3] + "MUL\n"
211
212 def p_Exp_div(p):
213     "Exp : Exp2 DIV Exp2"
214     p[0] = p[1] + p[3] + "DIV\n"
215
216 def p_Exp_mod(p):
217     "Exp : Exp2 MOD Exp2"
218     p[0] = p[1] + p[3] + "MOD\n"
219
220 def p_Exp_exp2(p):
221     "Exp : Exp2"
222     p[0] = p[1]
223
224 def p_Exp2_id(p):
225     "Exp2 : ID"
226     p[0] = "PUSHG " + str(p.parser.registos.get(p[1])) + "\n"
227
228 def p_Exp2_num(p):
229     "Exp2 : NUM"
230     p[0] = "PUSHI " + str(p[1]) + "\n"
231
232 def p_Exp2_real(p):
```

```
233     "Exp2 : REAL"
234     p[0] = "PUSHF " + str(p[1]) + "\n"
235
236 def p_Exp2_idarray_num(p):
237     "Exp2 : ID RE_NUM RD"
238     p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(p[1])) + "\nPADD\n" + "
            PUSHG " + p[3] + "\nLOADN\n"
239
240 def p_Exp2_idarray_id(p):
241     "Exp2 : ID RE_ID RD"
242     p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(p[1])) + "\nPADD\n" + "
            PUSHG " + str(p.parser.registos.get(p[3])) + "\nLOADN\n"
243
244
245
246
247
248 def p_Printf_print(p):
249     "Printf : PRINT PE_TEXT RestoPrintf"
250     p[0] = "PUSHS " + p[3] + "\n" + "WRITES" + "\n" + p[4]
251
252 def p_RestoPrintf_pd(p):
253     "RestoPrintf : PD PV"
254     p[0] = ""
255
256 def p_RestoPrintf_vir(p):
257     "RestoPrintf : VIR ContPrintf"
```

```
258     p[0] = p[2]
259
260
261 def p_ContPrintf_id(p):
262     "ContPrintf : ID PD PV"
263     if(str(p.parser.tipos.get(p[1])) == "int"):
264         p[0] = "PUSHG " + str(p.parser.registos.get(p[1])) + "\n" + "WRITEI" + "\n"
265     else:
266         p[0] = "PUSHG " + str(p.parser.registos.get(p[1])) + "\n" + "WRITEF" + "\n"
267
268 def p_ContPrintf_idarraynum(p):
269     "ContPrintf : ID RE NUM RD PD PV"
270     #nome = re.match(r'[a-zA-Z][a-zA-Z]*', p[1])
271     #tamanho = re.search(r'\d+', p[1])
272     #if (str(p.parser.tipos.get(nome.group()))) == "int"):
273     #     p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(nome.group())) + "\
        nPADD\n" + "PUSHI " + tamanho.group() + "\n" + p[3] + "\nLOADN\n" + "WRITEI\n"
274     #else:
275     #     p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(nome.group())) + "\
        nPADD\n" + "PUSHI " + tamanho.group() + "\n" + p[3] + "\nLOADN\n" + "WRITEF\n"
276     if (str(p.parser.tipos.get(p[1])) == "int"):
277         p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(p[1])) + "\nPADD\n"
                + "PUSHI " + p[3] + "\nLOADN\n" + "WRITEI\n"
278     else:
279         p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(p[1])) + "\nPADD\n"
                + "PUSHI " + p[3] + "\nLOADN\n" + "WRITEF\n"
280
```

```
281 def p_ContPrintf_idarrayid(p):
282     "ContPrintf : ID RE ID RD PD PV"
283     if (str(p.parser.tipos.get(p[1])) == "int"):
284         p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(p[1])) + "\nPADD\n"
                + "PUSHG " + str(p.parser.registos.get(p[3])) + "\nLOADN\n" + "WRITEI\n"
285     else:
286         p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(p[1])) + "\nPADD\n"
                + "PUSHG " + str(p.parser.registos.get(p[3])) + "\nLOADN\n" + "WRITEF\n"
287
288
289
290
291 def p_Scanf_scanf(p):
292     "Scanf : SCAN PE TEXT VIR RestoScanf"
293     p[0] = p[5]
294
295 def p_RestoScanf_id(p):
296     "RestoScanf : ENDID PD PV"
297     nome = p[1]
298     if (str(p.parser.tipos.get(nome[1:])) == "int"):
299         p[0] = "READ\n" + "ATOI\n" + "STOREG " + str(p.parser.registos.get(nome[1:]))
                + "\n"
300     else:
301         p[0] = "READ\n" + "ATOF\n" + "STOREG " + str(p.parser.registos.get(nome[1:]))
                + "\n"
302
303 def p_RestoScanf_idarraynum(p):
```

```
304     "RestoScanf : ENDID RE NUM RD PD PV"
305     nome = p[1]
306     if (str(p.parser.tipos.get(nome[1:])) == "int"):
307         p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(nome[1:])) + "\nPADD
                \n" + "PUSHI " + p[3] + "\n" + "READ\n" + "ATOI\n" + "STOREN\n"
308     else:
309         p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(nome[1:])) + "\nPADD
                \n" + "PUSHI " + p[3] + "\n" + "READ\n" + "ATOF\n" + "STOREN\n"
310
311 def p_RestoScanf_idarrayid(p):
312     "RestoScanf : ENDID RE ID RD PD PV"
313     nome = p[1]
314     if (str(p.parser.tipos.get(nome[1:])) == "int"):
315         p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(nome[1:])) + "\nPADD
                \n" + "PUSHG " + str(p.parser.registos.get(p[3])) + "\n" + "READ\n" + "
                ATOI\n" +  "STOREN\n"
316     else:
317         p[0] = "PUSHGP\n" + "PUSHI " + str(p.parser.registos.get(nome[1:])) + "\nPADD
                \n" + "PUSHG " + str(p.parser.registos.get(p[3])) + "\n" + "READ\n" + "
                ATOF\n" +  "STOREN\n"
318
319
320
321
322
323
324 def p_If_if(p):
```

```
325     "If : IF Cond CE BlocoInstIf CD"
326     parser.somaIf += 1
327     p[0] = p[2] + "\nJZ Endif" + str(parser.somaIf) + "\n" + p[4] + "\nEndif" + str(
            parser.somaIf) + ":\n"
328
329 def p_Cond_exp(p):
330     "Cond : PE Conta ExpRel Conta PD"
331     p[0] = p[2] + p[4] + p[3]
332
333 def p_Cond_conta(p):
334     "Cond : Conta"
335     p[0] = p[1]
336
337 def p_ExpRel_gt(p):
338     "ExpRel : GT"
339     p[0] = "SUP\n"
340
341 def p_ExpRel_ge(p):
342     "ExpRel : GE"
343     p[0] = "SUPEQ\n"
344
345 def p_ExpRel_lt(p):
346     "ExpRel : LT"
347     p[0] = "INF\n"
348
349 def p_ExpRel_le(p):
350     "ExpRel : LE"
```

```
351     p[0] = "INFEQ\n"
352
353 def p_ExpRel_eq(p):
354     "ExpRel : EQ"
355     p[0] = "EQUAL\n"
356
357 def p_ExpRel_dif(p): ##########################################################
358     "ExpRel : DIF"
359     p[0] = "EQUAL\n" + "NOT\n"
360
361 def p_Conta_pe(p):
362     "Conta : PE Conta2 PD"
363     p[0] = p[2]
364
365 def p_Conta_conta2(p):
366     "Conta : Conta2"
367     p[0] = p[1]
368
369 def p_Conta2_sub(p):
370     "Conta2 : Exp SUB Exp"
371     p[0] = p[1] + p[3] + "SUB\n"
372
373 def p_Conta2_add(p):
374     "Conta2 : Exp ADD Exp"
375     p[0] = p[1] + p[3] + "ADD\n"
376
377 def p_Conta2_exp(p):
```

```
378     "Conta2 : Exp"
379     p[0] = p[1]
380
381 def p_BlocoInstIf_inst(p):
382     "BlocoInstIf : InstBlocoIf BlocoInstIf"
383     p[0] = p[1] + p[2]
384
385 def p_BlocoInstIf_vazio(p):
386     "BlocoInstIf : "
387     p[0] = ""
388
389 def p_InstBlocoIf_atr(p):
390     "InstBlocoIf : Atribuicao"
391     p[0] = p[1]
392
393 def p_InstBlocoIf_print(p):
394     "InstBlocoIf : Printf"
395     p[0] = p[1]
396
397 def p_InstBlocoIf_scan(p):
398     "InstBlocoIf : Scanf"
399     p[0] = p[1]
400
401 def p_InstBlocoIf_if(p):
402     "InstBlocoIf : If"
403     p[0] = p[1]
404
```

```
405
406
407
408
409
410
411 def p_DoWhile_do(p):
412     "DoWhile : DO CE BlocoDoWhile CD WHILE CondDo PV"
413     parser.somaDoWhile += 1
414     p[0] = "DoWhile" + str(parser.somaDoWhile) + ":\n" + p[3] + "\n" + p[6] + "\nJZ
            DoWhile" + str(parser.somaDoWhile) + "\n"
415
416 def p_BlocoDoWhile_inst(p):
417     "BlocoDoWhile : InstBlocoDo BlocoDoWhile"
418     p[0] =p[1] + p[2]
419
420 def p_BlocoDoWhile_vazio(p):
421     "BlocoDoWhile : "
422     p[0] = ""
423
424 def p_InstBlocoDo_atr(p):
425     "InstBlocoDo : Atribuicao"
426     p[0] = p[1]
427
428 def p_InstBlocoDo_print(p):
429     "InstBlocoDo : Printf"
430     p[0] = p[1]
```

```
431
432 def p_InstBlocoDo_scan(p):
433     "InstBlocoDo : Scanf"
434     p[0] = p[1]
435
436 def p_InstBlocoDo_if(p):
437     "InstBlocoDo : If"
438     p[0] = p[1]
439
440
441
442 def p_CondDo_exp(p):
443     "CondDo : PE Conta ExpRelDo Conta PD"
444     p[0] = p[2] + p[4] + p[3]
445
446 def p_CondDo_conta(p):
447     "CondDo : Conta"
448     p[0] = p[1]
449
450 def p_ExpRelDo_gt(p):
451     "ExpRelDo : GT"
452     p[0] = "INFEQ\n"
453
454 def p_ExpRelDo_ge(p):
455     "ExpRelDo : GE"
456     p[0] = "INF\n"
457
```

```
458 def p_ExpRelDo_lt(p):
459     "ExpRelDo : LT"
460     p[0] = "SUPEQ\n"
461
462 def p_ExpRelDo_le(p):
463     "ExpRelDo : LE"
464     p[0] = "SUP\n"
465
466 def p_ExpRelDo_eq(p):
467     "ExpRelDo : EQ"
468     p[0] = "EQUAL\n" + "NOT\n"
469
470 def p_ExpRelDo_dif(p): ##########################################################
471     "ExpRelDo : DIF"
472     p[0] = "EQUAL\n"
473
474
475
476
477 #error rule for syntax errors
478 def t_newline(t):
479     r'\n+'
480     t.lexer.lineno += len(t.value)
481
482 def p_error(p):
483     if p == None:
484         token = "end of file"
```

```
485     else:
486         token = f"{p.type}({p.value}) on line {p.lineno}"
487
488     print(f"Syntax error: Unexpected {token}")
489
490 #build the parser
491 parser = yacc.yacc()
492
493
494
495 print(
496     "Escolha uma opcao:\n    1:Ler 4 n meros e dizer se podem ser os lados de um
        quadrado.\n    2:Ler um inteiro N, depois ler N n meros e escrever o menor
        deles.\n    " +
497     "3:Ler N n meros e calcular e imprimir o seu produt rio.\n    4:Contar e
        imprimir os n meros impares de uma sequ ncia de n meros naturais.\n    " +
498     "5:Ler e armazenar N n meros num array; imprimir os valores por ordem inversa.\n
        \n0:Sair do Programa\n")
499 opcao = input()
500
501 while opcao != '0':
502     if opcao == '1':
503         parser.registos = {}
504         parser.tipos = {}
505         parser.arraysTam = {}
506         parser.endArray = 0
507         parser.soma = 0
```

```
508         parser.somaIf = 0
509         parser.somaDoWhile = 0
510         f = open("1.c", "r")
511         res = open("res1.txt", "w")
512         # reading input
513         for linha in f:
514             resultado = parser.parse(linha)
515             res.write(str(resultado))
516         res.write("STOP")
517         f.close()
518         res.close()
519         for elem in parser.registos:
520             print(elem + ": " + str(parser.registos.get(elem)))
521
522         for elem in parser.tipos:
523             print(elem + ": " + str(parser.tipos.get(elem)))
524
525         for elem in parser.arraysTam:
526             print(elem + ": " + str(parser.arraysTam.get(elem)))
527
528     if opcao == '2':
529         parser.registos = {}
530         parser.tipos = {}
531         parser.arraysTam = {}
532         parser.endArray = 0
533         parser.soma = 0
534         parser.somaIf = 0
```

```
535         parser.somaDoWhile = 0
536         f = open("2.c", "r")
537         res = open("res2.txt", "w")
538         # reading input
539         for linha in f:
540             resultado = parser.parse(linha)
541             res.write(str(resultado))
542         res.write("STOP")
543         f.close()
544         res.close()
545         for elem in parser.registos:
546             print(elem + ": " + str(parser.registos.get(elem)))
547 
548         for elem in parser.tipos:
549             print(elem + ": " + str(parser.tipos.get(elem)))
550 
551         for elem in parser.arraysTam:
552             print(elem + ": " + str(parser.arraysTam.get(elem)))
553 
554     if opcao == '3':
555         parser.registos = {}
556         parser.tipos = {}
557         parser.arraysTam = {}
558         parser.endArray = 0
559         parser.soma = 0
560         parser.somaIf = 0
561         parser.somaDoWhile = 0
```

```
562         f = open("3.c", "r")
563         res = open("res3.txt", "w")
564         # reading input
565         for linha in f:
566             resultado = parser.parse(linha)
567             res.write(str(resultado))
568         res.write("STOP")
569         f.close()
570         res.close()
571         for elem in parser.registos:
572             print(elem + ": " + str(parser.registos.get(elem)))
573 
574         for elem in parser.tipos:
575             print(elem + ": " + str(parser.tipos.get(elem)))
576 
577         for elem in parser.arraysTam:
578             print(elem + ": " + str(parser.arraysTam.get(elem)))
579 
580     if opcao == '4':
581         parser.registos = {}
582         parser.tipos = {}
583         parser.arraysTam = {}
584         parser.endArray = 0
585         parser.soma = 0
586         parser.somaIf = 0
587         parser.somaDoWhile = 0
588         f = open("4.c", "r")
```

```
589         res = open("res4.txt", "w")
590         # reading input
591         for linha in f:
592             resultado = parser.parse(linha)
593             res.write(str(resultado))
594         res.write("STOP")
595         f.close()
596         res.close()
597         for elem in parser.registos:
598             print(elem + ": " + str(parser.registos.get(elem)))
599 
600         for elem in parser.tipos:
601             print(elem + ": " + str(parser.tipos.get(elem)))
602 
603         for elem in parser.arraysTam:
604             print(elem + ": " + str(parser.arraysTam.get(elem)))
605 
606     if opcao == '5':
607         parser.registos = {}
608         parser.tipos = {}
609         parser.arraysTam = {}
610         parser.endArray = 0
611         parser.soma = 0
612         parser.somaIf = 0
613         parser.somaDoWhile = 0
614         f = open("5.c", "r")
615         res = open("res5.txt", "w")
```

```
616         # reading input
617         for linha in f:
618             resultado = parser.parse(linha)
619             res.write(str(resultado))
620         res.write("STOP")
621         f.close()
622         res.close()
623         for elem in parser.registos:
624             print(elem + ": " + str(parser.registos.get(elem)))
625 
626         for elem in parser.tipos:
627             print(elem + ": " + str(parser.tipos.get(elem)))
628 
629         for elem in parser.arraysTam:
630             print(elem + ": " + str(parser.arraysTam.get(elem)))
631 
632     print(
633         "Escolha uma opcao:\n   1:Ler 4 n meros e dizer se podem ser os lados de um
                quadrado.\n   2:Ler um inteiro N, depois ler N n meros e escrever o menor
                deles.\n   " +
634         "3:Ler N n meros e calcular e imprimir o seu produt rio.\n   4:Contar e
                imprimir os n meros impares de uma sequ ncia de n meros naturais.\n   "
                 +
635         "5:Ler e armazenar N n meros num array; imprimir os valores por ordem
                inversa.\n\n0:Sair do Programa\n")
636     opcao = input()
```