Processamento de Linguagens e Compiladores

Trabalho Prático 2

Relatório de Desenvolvimento

Alunos:
Gabriel Antunes nº a101101
Guilherme Pinho nº a105533
Oliver Teixeira nº a102506
Grupo 20

janeiro
2025

Conteúdo

Resumo

O Segundo projeto proposto, no âmbito da UC Processamento de Linguagens e Compiladores, propõe a criação de uma linguagem imperativa simples assim como a criação de um compilador recorrendo aos módulos de gramáticas tradutoras do Python.

Adicionalmente este compilador deve gerar pseudo-código Assembly da máquina virtual VM.

Este relatório tem então o propósito de ilustrar aquele que foi o nosso processo de desenvolvimento do projeto bem como esclarecer as decisões por nós tomadas.

Introdução

No âmbito da UC Processamento de Linguagens e Compiladores foi-nos proposto este projeto que tem como principal objetivo desenvolver um compilador utilizando gramáticas independentes de contexto (GIC) e gramáticas tradutoras (GT), baseado na técnica de tradução dirigida pela sintaxe. A linguagem desenvolvida para este projeto deve permitir a declaração de variáveis, operações aritméticas, lógicas e relacionais, além de controlar o fluxo de execução por meio de instruções de seleção e repetição. O compilador gerado visa transformar o código-fonte escrito nesta linguagem em código Assembly para uma Máquina Virtual (VM), facilitando a simulação e execução do programa.

Para nos ajudar na resolução deste problema recorremos aos módulos "Yacc/Lex" do "PLY/Python".

Problema proposto

Pretende-se que comece por definir uma linguagem de programação imperativa simples, a seu gosto. Apenas deve ter em consideração que essa linguagem terá de permitir:

• declarar variáveis atómicas do tipo inteiro, com os quais se podem realizar as habituais operações aritméticas, relacionais e lógicas.

• efetuar instruções algorítmicas básicas como a atribuição do valor de expressões numéricas a variáveis.

• ler do standard input e escrever no standard output.

• efetuar instruções de seleção para controlo do fluxo de execução.

• efetuar instruções de repetição (cíclicas) para controlo do fluxo de execução, permitindo o seu aninhamento. Note que deve implementar pelo menos o ciclo **while-do**, **repeat-until** ou **for-do**.

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

• declarar e manusear variáveis estruturadas do tipo array (a 1 ou 2 dimensões) de inteiros, em relação aos quais é apenas permitida a operação de indexação (índice inteiro).

• definir e invocar subprogramas sem parâmetros, mas que possam retornar um resultado do tipo inteiro.

Estrutura

Vamos dividir o nosso relatório em quatro secções:

1. Construção da Gramática Independente de Contexto (**GIC**) que descreve a estrutura sintática da linguagem
2. Construção do Analisador Léxico, **Lexer.**
3. Construção do Analisador Sintático, **Parser.**
4. Tradução das instruções para código **Assembly** da **VM.**

A implementação de todas estas secções será enviada anexada a este documento.

1 Gramática Independente de Contexto (GIC)

```
Program : Decls
        | Decls Body
        | Body

Decls : Declaration Decls
      | Declaration

Body : Assignment Body
     | Statement Body
     | Declaration Body
     | Assignment
     | Statement
     | Declaration

Declaration : INT ID
            | INT ID KEEPS Expression
            | ARRAY ID
            | ARRAY ID LPAREN Num RPAREN
            | ARRAY ID KEEPS LBRACKET List RBRACKET
            | ARRAY ID KEEPS ID
            | MATRIX ID
            | MATRIX ID LPAREN Num COMMA Num RPAREN
            | MATRIX ID KEEPS LBRACKET Matrix RBRACKET
            | MATRIX ID KEEPS ID

Assignment : ID KEEPS Expression
           | ID KEEPS LBRACKET List RBRACKET
           | ID LPAREN Expression RPAREN KEEPS Expression
           | ID KEEPS LBRACKET Matrix RBRACKET
           | ID LPAREN Expression COMMA Expression RPAREN KEEPS Expression
           | ID LPAREN Expression RPAREN KEEPS LBRACKET List RBRACKET
           | ID PLUS PLUS
           | ID MINUS MINUS

           | ID LPAREN Expression RPAREN SWAP ID LPAREN Expression RPAREN
           | ID LPAREN Expression COMMA Expression RPAREN SWAP
        ID LPAREN Expression COMMA Expression RPAREN
           | ID LPAREN Expression RPAREN SWAP
        ID LPAREN Expression COMMA Expression RPAREN
           | ID LPAREN Expression COMMA Expression RPAREN SWAP
        ID LPAREN Expression RPAREN
```

```
List : Num COMMA List
     | Num
     |

Matrix : LBRACKET List RBRACKET COMMA Matrix
       | LBRACKET List RBRACKET
       |

Num : NUM
    | NEGATIVE NUM

Expression : Num
           | ID
           | INPUT
           | Operation
           | SEARCH ID LPAREN Expression RPAREN
           | SEARCH ID LPAREN Expression COMMA Expression RPAREN

Operation : Expression PLUS Expression
          | Expression MINUS Expression
          | Expression TIMES Expression
          | Expression DIVIDEDBY Expression
          | Expression REMAINDER Expression

Statement : If
          | While_Do
          | Repeat_Until
          | For_Do
          | Output

If : IF Comparison LBRACE Body RBRACE END
   | IF Comparison LBRACE Body RBRACE ELSE LBRACE Body RBRACE END


While_Do : WHILE Comparison DO LBRACE Body RBRACE END

Repeat_Until : REPEAT LBRACE Body RBRACE UNTIL Comparison END

For_Do : FOR LPAREN Assignment SEMICOLON Comparison SEMICOLON
    Assignment RPAREN DO LBRACE Body RBRACE END

Output : OUTPUT TEXT
       | OUTPUT ID
       | OUTPUT Num
       | OUTPUT LBRACKET List RBRACKET
       | OUTPUT LBRACKET Matrix RBRACKET
```

```
Comparison : NOT Comparison
           | LPAREN Expression EQUAL Expression RPAREN
           | LPAREN Expression NOT_EQUAL Expression RPAREN
           | LPAREN Expression GREATER Expression RPAREN
           | LPAREN Expression GREATER_EQUAL Expression RPAREN
           | LPAREN Expression LOWER Expression RPAREN
           | LPAREN Expression LOWER_EQUAL Expression RPAREN
           | LPAREN Comparison AND Comparison RPAREN
           | LPAREN Comparison OR Comparison RPAREN
```

2 Lexer

O analisador léxico será responsável por capturar os **tokens** da nossa linguagem, estes podem ter uma definição literal, ou podem apresentar alguma flexibilidade e nesse caso são encontrados com a utilização de expressões regulares. Conseguimos reparar que os **tokens** serão os símbolos terminais que constam na Gramática Independente de Contexto que pode ser encontrada na secção anterior.

Para implementar o analisador léxico utilizamos o módulo "**Lex**" do "**PLY/PYTHON**".

Seguem se os tokens utilizados juntamente com as respetivas expressões regulares

```
'INT': 'INT',
'ARRAY': 'ARRAY',
'MATRIX': 'MATRIX',

'ID': r'[a-zA-Z_][\w_]*'

'NUM' : r'\d+'
'NEGATIVE' : r'-'
'TEXT' : r'\".*\"'

'KEEPS':'KEEPS',
'SWAP': 'SWAP',

'LPAREN' : r'\('
'RPAREN' : r'\)'
'LBRACE' : r'\{'
'RBRACE' : r'\}'
'LBRACKET' : r'\[
'RBRACKET' : r'\]
'COMMA' : r','
'SEMICOLON' : r';'




'PLUS': 'PLUS',
'MINUS': 'MINUS',
'TIMES': 'TIMES',
'DIVIDEDBY': 'DIVIDED_BY',
'REMAINDER': 'REMAINDER',

'INPUT': 'INPUT',
'OUTPUT': 'OUTPUT',
```

```
'SEARCH': 'SEARCH',

'IF': 'IF',
'ELSE': 'ELSE',
'WHILE': 'WHILE',
'DO': 'DO',
'REPEAT': 'REPEAT',
'UNTIL': 'UNTIL',
'FOR': 'FOR',
'END': 'END',

'NOT': 'NOT',
'EQUAL': 'EQUAL',
'NOTEQUAL': 'NOT_EQUAL',
'GREATER': 'GREATER',
'GREATEREQUAL': 'GREATER_EQUAL',
'LOWER': 'LOWER',
'LOWEREQUAL': 'LOWER_EQUAL',
'AND':'AND',
'OR':'OR'
```

3 Parser

Contrariamente ao analisador léxico que analisa cada termo de forma individual, o analisador sintático verifica se a forma como os termos estão organizados respeita a Gramática por nós definida. Como já foi referido esta etapa vai utilizar a GIC definida na primeira secção para fazer a correspondência dos termos, seguem alguns exemplos:

```
def p_program_decls(p):
    '''Program : Decls'''
    parser.assembly = f"{p[1]}"

def p_program_declsBody(p):
    '''Program : Decls Body'''
    parser.assembly = f"{p[1]}\nSTART\n{p[2]}STOP"

def p_program_body(p):
    '''Program : Body'''
    parser.assembly = f"START\n{p[1]}STOP"
```

Este segmento diz respeito à fase inicial da interpretação do código na nossa linguagem. Colocando assim é fácil perceber a forma como o programa vai decidir o que está a ler naquele momento e assim definir logo de inicio como será a estrutura do código em assembly. Neste caso o parser diferencia se o nosso código é composto apenas por declarações, por declarações seguidas do corpo do código ou se não apresenta declarações à cabeça e começa logo com o corpo do código.

```
def p_declAss_recCall(p):
    '''Decls : Declaration Decls'''
    p[0] = f"{p[1]}{p[2]}"

def p_declAss_term(p):
    '''Decls : Declaration'''
    p[0] = p[1]
```

Este segmento ilustra a forma como o parser reconhece multiplos elemento do mesmo tipo. Como foi referido no excerto anterior, o corpo do código pode ser antecedido por um conjunto de declarações, o parser interpreta este conjunto através de uma chamada recursiva que termina assim que seja colocada a ultima declaração.

```
def p_declaration_int(p):
    '''Declaration : INT ID'''
    nameVar = p[2]
    if nameVar not in parser.vars:
        parser.vars[nameVar] = (parser.stackPointer, None)
        p[0] = "PUSHI 0\n"
        parser.stackPointer +=1
    else:
        parser.success = False
        parser.error += f"\n>> The Variable {nameVar} already exists\n"
        parser.error += f"{currLinetxt}{parser.currLine}\n"
    parser.currLine +=1
```

Neste excerto podemos ver a forma como o parser reconhece uma declaração de um número inteiro, neste caso vazio, isto é, igual a zero. Podemos também ver a forma como é feita o tratamento de erros, em que o próprio programa verifica se a variável já estava registada e se for o caso emite uma mensagem de erro.

```
def p_assignment_id(p):
    '''Assignment : ID KEEPS Expression'''
    nameVar1 = p[1]
    value = p[3]
    if nameVar1 not in parser.vars:
        parser.error += f"\n{nameVar1} has not been declared\n"
        parser.error += f"{currLinetxt}{parser.currLine}\n"
        parser.success = False
    else:
        temp = f"{value}"
        if len(parser.vars[nameVar1]) == 3:
            lins = parser.vars[nameVar1][1]
            cols = parser.vars[nameVar1][2]
            size = lins * cols
            for s in reversed(range(size)):
                temp += f"STOREG {parser.vars[nameVar1][0] + s}\n"
            p[0] = temp
        elif parser.vars[nameVar1][1]:
            size = parser.vars[nameVar1][1]
            for c in reversed(range(size)):
                temp += f"STOREG {parser.vars[nameVar1][0] + c}\n"
            p[0] = temp

        else:
            p[0] = f"{value}STOREG {parser.vars[nameVar1][0]}\n"

    parser.currLine +=1
```

Este é um bom exemplo para demonstrar a forma como a partir do dicionário com as variáveis registadas, mesmo quando o tipo do ID não está especificado o programa consegue recolher essa informação e agir em conformidade ao tipo em questão.

Repare-se também mais uma vez a forma como ele verifica se a variável já se encontra registada ou não e emite o alerta.

```
def p_operation(p):
    '''Operation : Expression PLUS Expression
                 | Expression MINUS Expression
                 | Expression TIMES Expression
                 | Expression DIVIDED_BY Expression
                 | Expression REMAINDER Expression'''
    if p[2] == 'PLUS':
        p[0] = f"{p[1]}{p[3]}ADD\n"
    elif p[2] == 'MINUS':
        p[0] = f"{p[1]}{p[3]}SUB\n"
    elif p[2] == 'TIMES':
        p[0] = f"{p[1]}{p[3]}MUL\n"
    elif p[2] == 'DIVIDEDBY':
        p[0] = f"{p[1]}{p[3]}DIV\n"
    else:
        p[0] = f"{p[1]}{p[3]}MOD\n"
```

Aqui podemos ver a forma como o programa efetua operações aritméticas simples

```
def p_comparison_not(p):
    '''Comparison  : NOT Comparison'''
    p[0] = f"{p[2]}PUSHI 0\nEQUALS\n"

def p_comparison(p):
    '''Comparison  : LPAREN Expression EQUAL Expression RPAREN
                   | LPAREN Expression NOT_EQUAL Expression RPAREN
                   | LPAREN Expression GREATER Expression RPAREN
                   | LPAREN Expression GREATER_EQUAL Expression RPAREN
                   | LPAREN Expression LOWER Expression RPAREN
                   | LPAREN Expression LOWER_EQUAL Expression RPAREN'''
    if p[3] == 'EQUAL':
        p[0] = f"{p[2]}{p[4]}EQUAL\n"
    elif p[3] == "NOT_EQUAL":
        p[0] = f"{p[2]}{p[4]}EQUAL\nPUSHI 0\nEQUAL\n"
    elif p[3] == 'GREATER':
        p[0] = f"{p[2]}{p[4]}SUP\n"
    elif p[3] == 'GREATER_EQUAL':
        p[0] = f"{p[2]}{p[4]}SUPEQ\n"
    elif p[3] == 'LOWER':
        p[0] = f"{p[2]}{p[4]}INF\n"
    else:
        p[0] = f"{p[2]}{p[4]}INFEQ\n"

def p_comparison_aO(p):
    '''Comparison  : LPAREN Comparison AND Comparison RPAREN
                   | LPAREN Comparison OR Comparison RPAREN'''
    if p[3] == 'AND':
        p[0] = f"{p[2]}{p[4]}AND\n"
    else:
        p[0] = f"{p[2]}{p[4]}OR\n"
```

Neste excerto estão presentes todas as funções responsaveis pelas operações lógicas e relacionais entre variáveis. Todas funcionam de forma simples e semelhantes traduzindo automáticamente para a os comandos correspondentes em assembly.

```
def p_expression_input(p):
    '''Expression : INPUT
                  | Operation'''
    if(p[1] == 'INPUT'):
        p[0] = f"READ\nATOI\n"
    else:
        p[0] = p[1]
```

É importante referir que a nossa linguagem apenas é capaz de ler do standard input números inteiros, a conversão faz-se de forma muito simples.

```
def p_output_id(p):
    '''Output : OUTPUT ID'''
    nameVar = p[2]
    temp = ""
    if nameVar in parser.vars:
        sPointer = parser.vars[nameVar][0]
        if parser.vars[nameVar][1] == None:         # INTEIROS
            temp = f"\nPUSHG {sPointer}\nWRITEI\n"
        elif len(parser.vars[nameVar]) == 2:        # ARRAYS
            size = parser.vars[nameVar][1]
            temp += "\nPUSHS \"[\"\nWRITES"
            for i in range(size):
                temp += f"\nPUSHGP\nPUSHI {sPointer}\nPADD\nPUSHI {i}\nLOADN\nWRITEI\n"
                if i != size -1:
                    temp += "\nPUSHS \", \"\nWRITES\n"
            temp += "\nPUSHS \"]\"\nWRITES\n"
        else:                                       # MATRIXS
            lins = parser.vars[nameVar][1]
            cols = parser.vars[nameVar][2]
            temp += "\nPUSHS \"[\"\nWRITES\n"
            for l in range(lins):
                temp += "\nPUSHS \"[\"\nWRITES\n"
                for c in range(cols):
                    temp += f"PUSHGP\nPUSHI {sPointer}\nPADD\nPUSHI {l * cols + c}\nLOADN\nWRITEI\n"
                    if c != cols -1:
                        temp += "PUSHS \", \"\nWRITES\n"
                temp += "PUSHS \"]\"\nWRITES\n"
                if l != lins -1:
                    temp += "\nPUSHS \", \"\nWRITES\n"
            temp += "\nPUSHS \"]\"\nWRITES\n"
    else:
        parser.error += f"\n>> {nameVar} has not been declared\n"
        parser.error += f"{currLinetxt}{parser.currLine}\n"
        parser.success = False
    p[0] = temp
    parser.currLine += 1

def p_output_num(p):
    '''Output : OUTPUT Num'''
    p[0] = f"\nPUSHI {p[2]}\nWRITEI\n"
    parser.currLine += 1
```

Comparando com a função do INPUT, as funções que lidam com o OUTPUT além de terem mais opções possiveis são relativamente mais complexas. No caso de escrever uma variável no standard output, o programa age dde forma diferente dependendo do tipo de variável para garantir que todo o seu conteúdo é escrito.

```
def p_if(p):
    '''If : IF Comparison LBRACE Body RBRACE END'''
    p[0] = f"\n{p[2]}\nJZ cPI{parser.checkPoint}\n{p[4]}cPI{parser.checkPoint}: NOP\n"
    parser.checkPoint += 1
    parser.currLine +=1

def p_ifElse(p):
    '''If : IF Comparison LBRACE Body RBRACE ELSE LBRACE Body RBRACE END'''
    temp = f"\n{p[2]}\nJZ cPE{parser.checkPoint}\n{p[4]}\nJUMP cPI{parser.checkPoint}\ncPE{parser.checkPoint}: NOP\n"
    temp += f"{p[8]}cPI{parser.checkPoint}: NOP\n"
    p[0] = temp
    parser.checkPoint += 1
    parser.currLine += 1
```

Estas são as funções responsáveis pelas operações de seleção para controlo do fluxo da execução

```
def p_whileDo(p):
    '''While_Do : WHILE Comparison DO LBRACE Body RBRACE END'''
    p[0] = f"\ncPW{parser.checkPoint}: NOP\n{p[2]}JZ cPEW{parser.checkPoint}\n{p[5]}JUMP cPW{parser.checkPoint}\ncPEW{parser.checkPoint}: NOP"
    parser.checkPoint +=1
    parser.currLine += 1

# ------------------------------------------------------------
#
# Repeat_Until : REPEAT LBRACE Body RBRACE UNTIL Comparison END
#
# ------------------------------------------------------------

def p_repeatUntil(p):
    '''Repeat_Until : REPEAT LBRACE Body RBRACE UNTIL Comparison END'''
    p[0] = f"\ncPR{parser.checkPoint}: NOP\n{p[3]}{p[6]}PUSHI 0\nEQUAL\nJZ cPU{parser.checkPoint}\nJUMP cPR{parser.checkPoint}\ncPU{parser.checkPoint}: NOP\n"
    parser.checkPoint += 1
    parser.currLine += 1

# ------------------------------------------------------------------------------------------------------------
#
# For_Do : FOR LPAREN Assignment SEMICOLON Comparison SEMICOLON Assignment RPAREN DO LBRACE Body RBRACE END
#
# ------------------------------------------------------------------------------------------------------------

def p_forDo(p):
    '''For_Do : FOR LPAREN Assignment SEMICOLON Comparison SEMICOLON Assignment RPAREN DO LBRACE Body RBRACE END'''
    temp = f"\n{p[3]}"
    temp += f"\ncPF{parser.checkPoint}: NOP\n{p[5]}JZ cPEF{parser.checkPoint}\n{p[11]}{p[7]}JUMP cPF{parser.checkPoint}\ncPEF{parser.checkPoint}: NOP\n"
    p[0] = temp
    parser.checkPoint += 1
    parser.currLine += 1
```

Neste excerto estão as operações de repetição para controlo de fluxo da execução que implementamos nas nossa linguagem. À exceção do FOR que requer que algumas operações sejam efetuadas a cada iteração, todos estes ciclos funcionam de forma muito similar e só variam na ordem em que são executadas as verificações

```
def p_assignment_swap1D(p):
    '''Assignment : ID LPAREN Expression RPAREN SWAP ID LPAREN Expression RPAREN'''
    nameVar1 = p[1]
    nameVar2 = p[6]
    temp = "\n"
    if nameVar1 not in parser.vars or nameVar2 not in parser.vars:
        if nameVar1 not in parser.vars:
            parser.error += f"\n>> {nameVar1} has not been declared\n"
            parser.error += f"{currLinetxt}{parser.currLine}\n"
        if nameVar2 not in parser.vars:
            parser.error += f"\n>> {nameVar2} has not been declared\n"
            parser.error += f"{currLinetxt}{parser.currLine}\n"
        parser.success = False
    else:
        sPointer1 = parser.vars[nameVar1][0]
        sPointer2 = parser.vars[nameVar2][0]
        if len(parser.vars[nameVar1]) == 2 and len(parser.vars[nameVar2]) == 2 and parser.vars[nameVar1][1] and parser.vars[nameVar2][1]:   # Para ARRAYS
            temp += f"PUSHGP\nPUSHI {sPointer1}\nPADD\n{p[3]}PUSHGP\nPUSHI {sPointer2}\nPADD\n{p[8]}LOADN\n"
            temp += f"PUSHGP\nPUSHI {sPointer2}\nPADD\n{p[8]}PUSHGP\nPUSHI {sPointer1}\nPADD\n{p[3]}LOADN\n"
            temp += f"\nSTOREN\nSTOREN\n"
        elif len(parser.vars[nameVar1]) == 3 and len(parser.vars[nameVar2]) == 3:   # Para MATRIXS
            cols1 = parser.vars[nameVar1][2]
            cols2 = parser.vars[nameVar2][2]
            if cols1 != cols2:
                parser.error += f"\n>> To execute SWAP both lines must be of the same size\n"
                parser.error += f"{currLinetxt}{parser.currLine}\n"
                parser.success = False
            else:
                for c in range(cols1):
                    temp += f"PUSHGP\nPUSHI {sPointer1}\nPADD\n{p[3]}PUSHI {cols1}\nMUL\nPUSHI {c}\nADD\n"
                    temp += f"PUSHGP\nPUSHI {sPointer2}\nPADD\n{p[8]}PUSHI {cols2}\nMUL\nPUSHI {c}\nADD\nLOADN\n"
                    temp += f"PUSHGP\nPUSHI {sPointer2}\nPADD\n{p[8]}PUSHI {cols2}\nMUL\nPUSHI {c}\nADD\n"
                    temp += f"PUSHGP\nPUSHI {sPointer1}\nPADD\n{p[3]}PUSHI {cols1}\nMUL\nPUSHI {c}\nADD\nLOADN\n"
                    temp += f"\nSTOREN\nSTOREN\n"
                    if c != cols1 - 1:
                        temp += "\n"
        else:
            parser.error += f"\n>> SWAP only works with ARRAYS and MATRIXS \n"

            if not parser.vars[nameVar1][1]:
                parser.error += f">> {nameVar1} is an INT\n"
            if not parser.vars[nameVar2][1]:
                parser.error += f">> {nameVar2} is as INT\n"
            parser.error += f"{currLinetxt}{parser.currLine}\n"
            parser.success = False
    p[0] = temp
    parser.currLine += 1
```

Aqui está apresentado um exemplo um pouco mais complexo. Este excerto explicita a forma como o parser atua sobre a operação Swap entre ARRAYS e entre MATRIXS. Como esta é uma operação que envolve mais condições é possível observar que existe a possibilidade de ocorrerem bastantes erros diferentes, quer de variaveis não declaradas, incorrespondência nos valores a serem trocados, ou incapacidade de reconhecer uma indexação, neste caso em variáveis do tipo INT. Mais uma vez o programa é capaz de fazer distinção na forma como vai operar dependendo o tipo das variáveis que recebe.

```
def p_expression_searchArrayID(p):
    '''Expression : SEARCH ID LPAREN Expression RPAREN'''
    nameVar = p[2]
    if nameVar not in parser.vars:
        parser.error += f"\n>> The Variable {nameVar} has not been declared\n"
        parser.error += f"{currLinetxt}{parser.currLine}\n"
        parser.success = False
    else:
        if len(parser.vars[nameVar]) == 3:
            temp = "\n"
            cols = parser.vars[nameVar][2]
            for c in range(cols):
                temp += f"PUSHGP\nPUSHI {parser.vars[nameVar][0]}\nPADD\n{p[4]}PUSHI {cols}\nMUL\nPUSHI {c}\nADD\nLOADN\n"
            p[0] = temp
        elif parser.vars[nameVar][1]:
            p[0] = f"\nPUSHGP\nPUSHI {parser.vars[nameVar][0]}\nPADD\n{p[4]}LOADN\n"
        else:
            parser.error += f"\n>> {nameVar} must be an MATRIX or an ARRAY\n"
            parser.error += f"{currLinetxt}{parser.currLine}\n"
            parser.success = False
            p[0] = ""

    parser.currLine += 1
```

Para terminar as operações de indexação sobre ARRAYS e MATRIXS temos aqui uma das funções que lida com o Search, função que retorna o valor numa determinada posição de um ARRAY ou MATRIX. Neste exemplo em que apenas definimos uma dimensão, podemos ver que no caso das MATRIX o parser irá retornar o código que irá colocar no topo da stack, a linha completa no índice definido, por outro lado no caso dos ARRAYS apenas vai colocar no topo da stack o valor na posição indicada.

4 Conversão para assembly

4.1 Interpretação do input

Antes de passar para a demonstração dos resultados vamos mostrar as diferentes opções que temos para utilizar o nosso programa para correr o código.

Para correr o nosso código temos três inputs possíveis, podemos chamar a função com:

1 argumento.

No caso do programa ser chamado com 2 argumentos este deve seguir a seguinte estrutura:

```
    python3 yacc.py
```

Neste caso, o programa vai interpretar o código que for introduzido no terminal naquele momento

```
>>Press 'Enter' to Finish

>> INT a
>> INT b KEEPS 5
>> INT c KEEPS 5 PLUS b
>> OUTPUT c
>>
>> Do you want the save the generated code? [Y/n]

>> Insert File Name:
demo1arg
>> File saved successfully as demo1arg.vm
```

Podemos ver neste exemplo que o programa irá registar o código até receber uma linha vazia, se o código estiver correto sintáticamente o programa irá fazer a sua conversão e registar a mesma no ficheiro escolhido na pasta "Testes"que deverá estar na mesma pasta que o código fonte

2 argumentos.

No caso do programa ser chamado com 2 argumentos este deve seguir a seguinte estrutura:

```
    python3 yacc.py ./Testes/<Ficheiro Input>.ggo
```

Segue um exemplo de como seria a execução do programa com um input deste género

```
>> Registed variables: {'a': (0, None)}
>> Do you want the save the generated code? [Y/n]

>> The generated code will be saved in this file: <Ficheiro Input>.vm

>> Do you want to change the file name? [y/N]

>> <Ficheiro Input>.vm already exists, do you wish to replace it? [y/N]

>> File successfully saved as <Ficheiro Input>(1).vm
```

Se o código escrito na nossa linguagem estiver sintáticamente correto, o programa vai sugerir registar o código num ficheiro com o mesmo nome mas terminado na extensão ".vm"em vez de ".ggo".

Além disso, caso o ficheiro já exista irá nos ser colocada a opção de substituir o ficheiro existente por um novo ou guardar o novo conteúdo num ficheiro com o mesmo nome seguido do número da cópia.

3 argumentos.

No caso do programa ser chamado com 3 argumentos este deve seguir a seguinte estrutura:

```
    python3 yacc.py ./Testes/<Ficheiro Input>.ggo ./Testes/<Ficheiro Output>.vm
```

Nesta situação, é definido não só o Ficheiro de Input mas também o Ficheiro de Output onde será registada a conversão do código na nossa linguagem para pseudo-código assembly

Tanto para 3 como para 2 argumentos, os ficheiros devem respeitar as expansões dos ficheiros, caso tal não aconteça o programa não irá funcionar e vai emitir o seguinte alerta:

```
python3 yacc.py ./Testes/<Exemplo Errado>.py ./Testes/<Exemplo Errado>.ggo
```

```
>> Invalid file extension

>> Files must be .ggo and .vm
```

Caso o input esteja no formato esperado o programa decorrerá da seguinte forma:

```
>> <Ficheiro Output>.vm already as content, do you wish to Rewrite it? [y/N]

>> Try again with another file
>> Execution Finished
```

No caso do ficheiro selecionado para escrever o output já tenha conteúdo o programa irá questionar o utilizador se ele deseja reescrever o conteúdo presente no mesmo. Caso o utilizador assinale que sim, então o conteúdo nesse ficheiro será substituído pelo novo conteúdo, caso contrário o programa não irá registar o conteúdo e em vez disso sugere ao utilizador que tente novamente desta vez com um novo ficheiro

Nenhum do anteriores

Caso o programa seja chamado com mais do que 3 argumentos será emitida a seguinte mensagem:

```
>> ERROR
>> Invalid Format try one of the following:
>> python3 yacc.py
>> python3 yacc.py <Input File>.ggo
>> python3 yacc.py <Input File>.ggo <Output File>.vm
```

4.2 Conversão

Vamos então mostrar alguns exemplos que demonstram as principais funcionalidades que a nossa linguagem suporta, a sua conversão para pseudo-código assembly e os respetivos resultados

demoDeclare.ggo

```
INT a
INT b KEEPS 5
ARRAY v(5)
MATRIX m KEEPS [[-1,2],[3,4]]

OUTPUT "a: "
OUTPUT a
OUTPUT "\nb: "
OUTPUT b
OUTPUT "\nv: "
OUTPUT v
OUTPUT "\nm: "
OUTPUT m
```

demoDeclare.vm

```
PUSHI 0
PUSHI 5
PUSHN 5

PUSHN 4

PUSHGP
PUSHI 7
PADD
PUSHI 0
PUSHI -1
STOREN

PUSHGP
PUSHI 7
PADD
PUSHI 1
PUSHI 2
STOREN

PUSHGP
PUSHI 7
```

```
PADD
PUSHI 2
PUSHI 3
STOREN

PUSHGP
PUSHI 7
PADD
PUSHI 3
PUSHI 4
STOREN

START
PUSHS "a: "
WRITES

PUSHG 0
WRITEI
PUSHS "\nb: "
WRITES

PUSHG 1
WRITEI
PUSHS "\nv: "
WRITES

PUSHS "["
WRITES
PUSHGP
PUSHI 2
PADD
PUSHI 0
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 2
PADD
PUSHI 1
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 2
PADD
```

```
74 PUSHI 2
75 LOADN
76 WRITEI
77
78 PUSHS ", "
79 WRITES
80
81 PUSHGP
82 PUSHI 2
83 PADD
84 PUSHI 3
85 LOADN
86 WRITEI
87
88 PUSHS ", "
89 WRITES
90
91 PUSHGP
92 PUSHI 2
93 PADD
94 PUSHI 4
95 LOADN
96 WRITEI
97
98 PUSHS "]"
99 WRITES
100 PUSHS "\nm: "
101 WRITES
102
103 PUSHS "["
104 WRITES
105
106 PUSHS "["
107 WRITES
108 PUSHGP
109 PUSHI 7
110 PADD
111 PUSHI 0
112 LOADN
113 WRITEI
114 PUSHS ", "
115 WRITES
116 PUSHGP
117 PUSHI 7
118 PADD
119 PUSHI 1
120 LOADN
121 WRITEI
122 PUSHS "]"
123 WRITES
124
```

```
PUSHS ", "
WRITES

PUSHS "["
WRITES
PUSHGP
PUSHI 7
PADD
PUSHI 2
LOADN
WRITEI
PUSHS ", "
WRITES
PUSHGP
PUSHI 7
PADD
PUSHI 3
LOADN
WRITEI
PUSHS "]"
WRITES

PUSHS "]"
WRITES
STOP
```

demoDeclare output

```
a: 0
b: 5
v: [0, 0, 0, 0, 0]
m: [[-1, 2], [3, 4]]
```

Este exemplo demonstra as funcionalidades de declaração de variáveis, com este exemplo podemos ver a forma como o programa aborda vários tipos de declarações diferentes, por exemplo quando um INT é declarado e não for especificado que valor é que deverá ficar registado nesta variável o programa assume que a mesma será 0. Podemos através do comando KEEPS definir que valor será atribuído a uma variável.

Para ARRAYS e MATRIXS podemos não querer introduzir valores mas devemos indicar o tamanho da variável para que o programa reserve esse espaço na memória, para utilizar esta abordagem com MATRIXS devemos indicar o número de linhas e o número de colunas respetivamente. Também para ARRAYS e MATRIXS como podemos ver no caso da variável m, podemos usar o KEEPS para definir diretamente os valores que serão guardados, lembrando que como isto se trata de uma declaração o tamanho dessas variáveis ficarão fixos sendo iguais ao da lista ou matriz que lhe foi atribuída.

demoAtribExpressions.ggo

```
INT a KEEPS 5
INT b KEEPS 10
INT c
INT d
INT e

c KEEPS b
d KEEPS 15 PLUS 15
e KEEPS d TIMES 2

OUTPUT "c: "
OUTPUT c
OUTPUT "\nd: "
OUTPUT d
OUTPUT "\ne: "
OUTPUT e
```

demoAtribExpressions.vm

```
PUSHI 5
PUSHI 10
PUSHI 0
PUSHI 0
PUSHI 0

START
PUSHG 1
STOREG 2
PUSHI 15
PUSHI 15
ADD
STOREG 3
PUSHG 3
PUSHI 2
MUL
STOREG 4
PUSHS "c: "
WRITES

PUSHG 2
WRITEI
PUSHS "\nd: "
WRITES

PUSHG 3
WRITEI
```

```
PUSHS "\ne: "
WRITES

PUSHG 4
WRITEI
STOP
```

demoAtribExpressions output

```
c: 10
d: 30
e: 60
```

Neste exemplo vemos as funcionalidades de atribuição para variáveis INT, para atribuirmos um valor a uma variável, esta deverá ter sido declarada anteriormente no início do programa.

Podemos ver vários tipos de atribuicões diferentes, é possível atribuir valores numéricos simples, é também possível atribuir a uma variável o valor associado a outra variável do tipo INT, podemos ainda atribuir a uma variável o valor resultante de uma operação aritmética, estas operações podem inclusive incluir o uso de valores registados em variáveis tal como se pode observar no exemplo

demoInputOutput.ggo

```
INT a
INT b

OUTPUT "Insira um numero para se seja calculado o seu quadrado\n"
a KEEPS INPUT

OUTPUT "Input: "
OUTPUT a
OUTPUT "\n"

b KEEPS a TIMES a

OUTPUT a
OUTPUT " ao quadrado e igual a "
OUTPUT b
```

demoInputOutput.vm

```
PUSHI 0
PUSHI 0

START
PUSHS "Insira um numero para se seja calculado o seu quadrado\n"
WRITES
READ
ATOI
STOREG 0
PUSHS "Input: "
WRITES

PUSHG 0
WRITEI
PUSHS "\n"
WRITES
PUSHG 0
PUSHG 0
MUL
STOREG 1

PUSHG 0
WRITEI
PUSHS " ao quadrado e igual a "
WRITES

PUSHG 1
WRITEI
STOP
```

demoInputOutput output

```
Insira um numero para se seja calculado o seu quadrado
Input: 5
5 ao quadrado e igual a 25
```

Como se pode ver o programa é capaz de ler inteiros do standard input e associar os mesmos a uma variável para posteriormente utilizar esse valor em diversas operações, no caso do exemplo foi lido um "5". Quanto a escrever no standard output, o programa é capaz de traduzir para pseudo-código assembly comandos para a VM escreva tanto textos, como valores numéricos e até os valores que estejam registados em variáveis quer estas sejam do tipo INT, ARRAY ou MATRIX

demoDecision.ggo

```
INT a

OUTPUT "Introduza um numero para verificar a sua paridade\n"

a KEEPS INPUT

OUTPUT "Input: "
OUTPUT a
OUTPUT "\n"

IF (a REMAINDER 2 EQUAL 0) {
    OUTPUT a
    OUTPUT " e par\n"

    IF ((a GREATER -1) AND (a LOWEREQUAL 0)) {
        OUTPUT "Pois e, muita gente nao sabe mas o 0 tambem e par\n
           "
    } END

} ELSE {
    OUTPUT a
    OUTPUT " e impar\n"
} END
```

demoDecision.vm

```
PUSHI 0

START
PUSHS "Introduza um numero para verificar a sua paridade\n"
WRITES
READ
ATOI
STOREG 0
PUSHS "Input: "
WRITES

PUSHG 0
WRITEI
PUSHS "\n"
WRITES

PUSHG 0
PUSHI 2
MOD
PUSHI 0
```

```
EQUAL

JZ cPE1

PUSHG 0
WRITEI
PUSHS " e par\n"
WRITES

PUSHG 0
PUSHI -1
SUP
PUSHG 0
PUSHI 0
INFEQ
AND

JZ cPI0
PUSHS "Pois e, muita gente nao sabe mas o 0 tambem e par\n"
WRITES
cPI0: NOP

JUMP cPI1
cPE1: NOP

PUSHG 0
WRITEI
PUSHS " e impar\n"
WRITES
cPI1: NOP
STOP
```

demoDecision output

```
Introduza um nemero para verificar a sua paridade
Input: 0
0 e par
Pois e, muita gente nao sabe mas o 0 tambem e par
```

Este é um simples exemplo que demonstra a capacidade da nossa linguagem efetuar instruções de seleção para controlo de fluxo

demoCycles.ggo

```
INT a
INT b

OUTPUT "Introduza um valor entre 1 e 10\n"
a KEEPS INPUT
OUTPUT "Input: "
OUTPUT a
OUTPUT "\n"

WHILE ((a GREATER 10) OR (a LOWER 1)) DO {
    OUTPUT "O valor introduzido e invalido, introduza outro\n"
    a KEEPS INPUT
    OUTPUT "Input: "
    OUTPUT a
    OUTPUT "\n"
} END

b KEEPS a

OUTPUT "o fatorial de "
OUTPUT b
OUTPUT " e: "

REPEAT {

    a MINUS MINUS
    b KEEPS b TIMES a


} UNTIL (a LOWEREQUAL 1) END

OUTPUT b
```

demoCycles.vm

```
PUSHI 0
PUSHI 0

START
PUSHS "Introduza um valor entre 1 e 10\n"
WRITES
READ
ATOI
STOREG 0
PUSHS "Input: "
WRITES
```

```
12
13 PUSHG 0
14 WRITEI
15 PUSHS "\n"
16 WRITES
17
18 CPW0: NOP
19 PUSHG 0
20 PUSHI 10
21 SUP
22 PUSHG 0
23 PUSHI 1
24 INF
25 OR
26 JZ cPEW0
27 PUSHS "O valor introduzido e invalido, introduza outro\n"
28 WRITES
29 READ
30 ATOI
31 STOREG 0
32 PUSHS "Input: "
33 WRITES
34
35 PUSHG 0
36 WRITEI
37 PUSHS "\n"
38 WRITES
39 JUMP cPW0
40 cPEW0: NOPPUSHG 0
41 STOREG 1
42 PUSHS "o fatorial de "
43 WRITES
44
45 PUSHG 1
46 WRITEI
47 PUSHS " e: "
48 WRITES
49
50 cPR1: NOP
51
52 PUSHG 0
53 PUSHI 1
54 SUB
55 STOREG 0
56 PUSHG 1
57 PUSHG 0
58 MUL
59 STOREG 1
60 PUSHG 0
61 PUSHI 1
62 INFEQ
```

```
63 PUSHI 0
64 EQUAL
65 JZ cPU1
66 JUMP cPR1
67 cPU1: NOP
68
69 PUSHG 1
70 WRITEI
71 STOP
```

demoCycles output

```
1 Introduza um valor entre 1 e 10
2 Input: 17
3 O valor introduzido e invalido, introduza outro
4 Input: -3
5 O valor introduzido e invalido, introduza outro
6 Input: 6
7 o fatorial de 6 e: 720
```

A nossa linguagem suporta três tipos de instruções de repetição para controlo de fluxo. Neste exemplo são utilizados duas destas estruturas, o ciclo WHILE DO e o ciclo REPEAT UNTIL, como se pode ver o programa pede ao utilizador para introduzir um valor até este estar entre 1 e 10 e quando encontra um valor válido entra no ciclo REPEAT UNTIL para calcular o fatorial do do valor introduzido.

Demos também atenção às operações relacionais e lógicas utilizadas nas condições destas estruturas.

Implementamos também a estrutura FOR DO que explicaremos melhor quando ela aparecer nos próximos exemplos.

demoArrays.ggo

```
ARRAY a KEEPS [1,2,3]
ARRAY b KEEPS a
MATRIX c KEEPS [[2,4],[6,8]]
MATRIX d(3,2)

OUTPUT "a: "
OUTPUT a
OUTPUT "\nb: "
OUTPUT b
OUTPUT "\nc: "
OUTPUT c
OUTPUT "\nd: "
OUTPUT d
```

demoArrays.vm

```

PUSHN 3
PUSHGP
PUSHI 0
PADD
PUSHI 0
PUSHI 1
STOREN
PUSHGP
PUSHI 0
PADD
PUSHI 1
PUSHI 2
STOREN
PUSHGP
PUSHI 0
PADD
PUSHI 2
PUSHI 3
STOREN

PUSHN 3

PUSHG 0
STOREG 3

PUSHG 1
STOREG 4

PUSHG 2
```

```
STOREG 5

PUSHN 4

PUSHGP
PUSHI 6
PADD
PUSHI 0
PUSHI 2
STOREN

PUSHGP
PUSHI 6
PADD
PUSHI 1
PUSHI 4
STOREN

PUSHGP
PUSHI 6
PADD
PUSHI 2
PUSHI 6
STOREN

PUSHGP
PUSHI 6
PADD
PUSHI 3
PUSHI 8
STOREN
PUSHN 6

START
PUSHS "a: "
WRITES

PUSHS "["
WRITES
PUSHGP
PUSHI 0
PADD
PUSHI 0
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 0
```

```
PADD
PUSHI 1
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 0
PADD
PUSHI 2
LOADN
WRITEI

PUSHS "]"
WRITES
PUSHS "\nb: "
WRITES

PUSHS "["
WRITES
PUSHGP
PUSHI 3
PADD
PUSHI 0
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 3
PADD
PUSHI 1
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 3
PADD
PUSHI 2
LOADN
WRITEI

PUSHS "]"
WRITES
```

```
133 PUSHS "\nc: "
134 WRITES
135
136 PUSHS "["
137 WRITES
138
139 PUSHS "["
140 WRITES
141 PUSHGP
142 PUSHI 6
143 PADD
144 PUSHI 0
145 LOADN
146 WRITEI
147 PUSHS ", "
148 WRITES
149 PUSHGP
150 PUSHI 6
151 PADD
152 PUSHI 1
153 LOADN
154 WRITEI
155 PUSHS "]"
156 WRITES
157
158 PUSHS ", "
159 WRITES
160
161 PUSHS "["
162 WRITES
163 PUSHGP
164 PUSHI 6
165 PADD
166 PUSHI 2
167 LOADN
168 WRITEI
169 PUSHS ", "
170 WRITES
171 PUSHGP
172 PUSHI 6
173 PADD
174 PUSHI 3
175 LOADN
176 WRITEI
177 PUSHS "]"
178 WRITES
179
180 PUSHS "]"
181 WRITES
182 PUSHS "\nd: "
183 WRITES
```

```
184
185 PUSHS "["
186 WRITES
187
188 PUSHS "["
189 WRITES
190 PUSHGP
191 PUSHI 10
192 PADD
193 PUSHI 0
194 LOADN
195 WRITEI
196 PUSHS ", "
197 WRITES
198 PUSHGP
199 PUSHI 10
200 PADD
201 PUSHI 1
202 LOADN
203 WRITEI
204 PUSHS "]"
205 WRITES
206
207 PUSHS ", "
208 WRITES
209
210 PUSHS "["
211 WRITES
212 PUSHGP
213 PUSHI 10
214 PADD
215 PUSHI 2
216 LOADN
217 WRITEI
218 PUSHS ", "
219 WRITES
220 PUSHGP
221 PUSHI 10
222 PADD
223 PUSHI 3
224 LOADN
225 WRITEI
226 PUSHS "]"
227 WRITES
228
229 PUSHS ", "
230 WRITES
231
232 PUSHS "["
233 WRITES
234 PUSHGP
```

```
PUSHI 10
PADD
PUSHI 4
LOADN
WRITEI
PUSHS ", "
WRITES
PUSHGP
PUSHI 10
PADD
PUSHI 5
LOADN
WRITEI
PUSHS "]"
WRITES

PUSHS "]"
WRITES
STOP
```

demoArrays output

```
a: [1, 2, 3]
b: [1, 2, 3]
c: [[2, 4], [6, 8]]
d: [[0, 0], [0, 0], [0, 0]]
```

Neste exemplo ficam mais claras as formas como podemos declarar variáveis do tipo ARRAY e do tipo MATRIX

demoIndexingSearch.ggo

```
MATRIX a KEEPS [[1,2],[3,4],[5,6]]
MATRIX b KEEPS a

ARRAY c KEEPS [25,50,75,100]
ARRAY d(2)

INT i
INT temp

FOR (i KEEPS 0; (i LOWER 4); i PLUS PLUS) DO {
    temp KEEPS SEARCH c(i)
    OUTPUT "Elemento no indice "
    OUTPUT i
    OUTPUT " da lista c: "
    OUTPUT temp
    OUTPUT "\n"
} END

d KEEPS SEARCH a(1)

OUTPUT "d: "
OUTPUT d
```

demoIndexingSearch.vm

```

PUSHN 6

PUSHGP
PUSHI 0
PADD
PUSHI 0
PUSHI 1
STOREN

PUSHGP
PUSHI 0
PADD
PUSHI 1
PUSHI 2
STOREN

PUSHGP
PUSHI 0
PADD
PUSHI 2
```

```
PUSHI 3
STOREN

PUSHGP
PUSHI 0
PADD
PUSHI 3
PUSHI 4
STOREN

PUSHGP
PUSHI 0
PADD
PUSHI 4
PUSHI 5
STOREN

PUSHGP
PUSHI 0
PADD
PUSHI 5
PUSHI 6
STOREN

PUSHN 6

PUSHG 0
STOREG 6

PUSHG 1
STOREG 7

PUSHG 2
STOREG 8

PUSHG 3
STOREG 9

PUSHG 4
STOREG 10

PUSHG 5
STOREG 11

PUSHN 4
PUSHGP
PUSHI 12
PADD
PUSHI 0
PUSHI 25
STOREN
```

```
PUSHGP
PUSHI 12
PADD
PUSHI 1
PUSHI 50
STOREN
PUSHGP
PUSHI 12
PADD
PUSHI 2
PUSHI 75
STOREN
PUSHGP
PUSHI 12
PADD
PUSHI 3
PUSHI 100
STOREN
PUSHN 2
PUSHI 0
PUSHI 0

START

PUSHI 0
STOREG 18

cPF0: NOP
PUSHG 18
PUSHI 4
INF
JZ cPEF0

PUSHGP
PUSHI 12
PADD
PUSHG 18
LOADN
STOREG 19
PUSHS "Elemento no indice "
WRITES

PUSHG 18
WRITEI
PUSHS " da lista c: "
WRITES

PUSHG 19
WRITEI
PUSHS "\n"
WRITES
```

```
124
125 PUSHG 18
126 PUSHI 1
127 ADD
128 STOREG 18
129 JUMP cPF0
130 cPEF0: NOP
131
132 PUSHGP
133 PUSHI 0
134 PADD
135 PUSHI 1
136 PUSHI 2
137 MUL
138 PUSHI 0
139 ADD
140 LOADN
141 PUSHGP
142 PUSHI 0
143 PADD
144 PUSHI 1
145 PUSHI 2
146 MUL
147 PUSHI 1
148 ADD
149 LOADN
150 STOREG 17
151 STOREG 16
152 PUSHS "d: "
153 WRITES
154
155 PUSHS "["
156 WRITES
157 PUSHGP
158 PUSHI 16
159 PADD
160 PUSHI 0
161 LOADN
162 WRITEI
163
164 PUSHS ", "
165 WRITES
166
167 PUSHGP
168 PUSHI 16
169 PADD
170 PUSHI 1
171 LOADN
172 WRITEI
173
174 PUSHS "]"
```

```
175 WRITES
176 STOP
```

demoIndexingSearch output

```
1 Elemento no indice 0 da lista c: 25
2 Elemento no indice 1 da lista c: 50
3 Elemento no indice 2 da lista c: 75
4 Elemento no indice 3 da lista c: 100
5 d: [3, 4]
```

Este exemplo mostra a capacidade da nossa linguagem de ir buscar um valor colocado num ARRAY ou MATRIX procurando-o pelo seu índice com a ajuda da função SEARCH, podemos utilizar esta função para retirar valores inteiros simples ou como podemos ver no exemplo pode se retirar toda uma linha de uma matriz. O índice pode ser definido por uma expressão númerica, por valores númericos simples ou por variáveis.

Chamamos também à atenção ao uso do ciclo For Do, que opera de forma semelhante aos outros dois ciclos mencionados nos exemplos anteriores, com a diferença que este cumpre sempre uma instrução de atribuição a cada iteração, no caso do exemplo "i PLUS PLUS"

demoIndexingSwap.ggo

```
1  MATRIX a KEEPS [[1,2],[3,4]]
2  MATRIX b KEEPS [[10,20],[30,40]]
3
4  ARRAY c KEEPS [25,50,75,100]
5  ARRAY d KEEPS [-1,-2,-3,-4]
6
7  OUTPUT "MATRIXS a e b antes do SWAP\n"
8  OUTPUT "a: "
9  OUTPUT a
10 OUTPUT "\nb: "
11 OUTPUT b
12
13 a(0,0) SWAP b(0,1)
14 b(1) SWAP a(1)
15
16 OUTPUT "\nMATRIXS a e b depois do SWAP\n"
17 OUTPUT "a: "
18 OUTPUT a
19 OUTPUT "\nb: "
```

```
OUTPUT b

OUTPUT "\nARRAY d e MATRIX b antes do SWAP\n"
OUTPUT "d: "
OUTPUT d
OUTPUT "\nb: "
OUTPUT b

d(0) SWAP b(0,0)

OUTPUT "\nARRAY d e MATRIX b depois do SWAP\n"
OUTPUT "d: "
OUTPUT d
OUTPUT "\nb: "
OUTPUT b

OUTPUT "\nARRAY c antes do SWAP\n"
OUTPUT "c: "
OUTPUT c

c(1) SWAP c(2)

OUTPUT "\nARRAY c antes do SWAP\n"
OUTPUT "c: "
OUTPUT c
```

demoIndexingSwap.vm

```

PUSHN 4

PUSHGP
PUSHI 0
PADD
PUSHI 0
PUSHI 1
STOREN

PUSHGP
PUSHI 0
PADD
PUSHI 1
PUSHI 2
STOREN

PUSHGP
PUSHI 0
PADD
PUSHI 2
```

```
PUSHI 3
STOREN

PUSHGP
PUSHI 0
PADD
PUSHI 3
PUSHI 4
STOREN

PUSHN 4

PUSHGP
PUSHI 4
PADD
PUSHI 0
PUSHI 10
STOREN

PUSHGP
PUSHI 4
PADD
PUSHI 1
PUSHI 20
STOREN

PUSHGP
PUSHI 4
PADD
PUSHI 2
PUSHI 30
STOREN

PUSHGP
PUSHI 4
PADD
PUSHI 3
PUSHI 40
STOREN

PUSHN 4
PUSHGP
PUSHI 8
PADD
PUSHI 0
PUSHI 25
STOREN
PUSHGP
PUSHI 8
PADD
PUSHI 1
```

```
PUSHI 50
STOREN
PUSHGP
PUSHI 8
PADD
PUSHI 2
PUSHI 75
STOREN
PUSHGP
PUSHI 8
PADD
PUSHI 3
PUSHI 100
STOREN

PUSHN 4
PUSHGP
PUSHI 12
PADD
PUSHI 0
PUSHI -1
STOREN
PUSHGP
PUSHI 12
PADD
PUSHI 1
PUSHI -2
STOREN
PUSHGP
PUSHI 12
PADD
PUSHI 2
PUSHI -3
STOREN
PUSHGP
PUSHI 12
PADD
PUSHI 3
PUSHI -4
STOREN

START
PUSHS "MATRIXS a e b antes do SWAP\n"
WRITES
PUSHS "a: "
WRITES

PUSHS "["
WRITES

PUSHS "["
```

```
124 WRITES
125 PUSHGP
126 PUSHI 0
127 PADD
128 PUSHI 0
129 LOADN
130 WRITEI
131 PUSHS ", "
132 WRITES
133 PUSHGP
134 PUSHI 0
135 PADD
136 PUSHI 1
137 LOADN
138 WRITEI
139 PUSHS "]"
140 WRITES
141
142 PUSHS ", "
143 WRITES
144
145 PUSHS "["
146 WRITES
147 PUSHGP
148 PUSHI 0
149 PADD
150 PUSHI 2
151 LOADN
152 WRITEI
153 PUSHS ", "
154 WRITES
155 PUSHGP
156 PUSHI 0
157 PADD
158 PUSHI 3
159 LOADN
160 WRITEI
161 PUSHS "]"
162 WRITES
163
164 PUSHS "]"
165 WRITES
166 PUSHS "\nb: "
167 WRITES
168
169 PUSHS "["
170 WRITES
171
172 PUSHS "["
173 WRITES
174 PUSHGP
```

```
175 PUSHI 4
176 PADD
177 PUSHI 0
178 LOADN
179 WRITEI
180 PUSHS ", "
181 WRITES
182 PUSHGP
183 PUSHI 4
184 PADD
185 PUSHI 1
186 LOADN
187 WRITEI
188 PUSHS "]"
189 WRITES
190
191 PUSHS ", "
192 WRITES
193
194 PUSHS "["
195 WRITES
196 PUSHGP
197 PUSHI 4
198 PADD
199 PUSHI 2
200 LOADN
201 WRITEI
202 PUSHS ", "
203 WRITES
204 PUSHGP
205 PUSHI 4
206 PADD
207 PUSHI 3
208 LOADN
209 WRITEI
210 PUSHS "]"
211 WRITES
212
213 PUSHS "]"
214 WRITES
215
216 PUSHGP
217 PUSHI 0
218 PADD
219 PUSHI 0
220 PUSHI 2
221 MUL
222 PUSHI 0
223 ADD
224 PUSHGP
225 PUSHI 4
```

```
PADD
PUSHI 0
PUSHI 2
MUL
PUSHI 1
ADD
LOADN
PUSHGP
PUSHI 4
PADD
PUSHI 0
PUSHI 2
MUL
PUSHI 1
ADD
PUSHGP
PUSHI 0
PADD
PUSHI 0
PUSHI 2
MUL
PUSHI 0
ADD
LOADN

STOREN
STOREN

PUSHGP
PUSHI 4
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 0
ADD
PUSHGP
PUSHI 0
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 0
ADD
LOADN
PUSHGP
PUSHI 0
PADD
PUSHI 1
PUSHI 2
MUL
```

```
PUSHI 0
ADD
PUSHGP
PUSHI 4
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 0
ADD
LOADN

STOREN
STOREN

PUSHGP
PUSHI 4
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 1
ADD
PUSHGP
PUSHI 0
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 1
ADD
LOADN
PUSHGP
PUSHI 0
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 1
ADD
PUSHGP
PUSHI 4
PADD
PUSHI 1
PUSHI 2
MUL
PUSHI 1
ADD
LOADN

STOREN
```

```
328 STOREN
329 PUSHS "\nMATRIXS a e b depois do SWAP\n"
330 WRITES
331 PUSHS "a: "
332 WRITES
333
334 PUSHS "["
335 WRITES
336
337 PUSHS "["
338 WRITES
339 PUSHGP
340 PUSHI 0
341 PADD
342 PUSHI 0
343 LOADN
344 WRITEI
345 PUSHS ", "
346 WRITES
347 PUSHGP
348 PUSHI 0
349 PADD
350 PUSHI 1
351 LOADN
352 WRITEI
353 PUSHS "]"
354 WRITES
355
356 PUSHS ", "
357 WRITES
358
359 PUSHS "["
360 WRITES
361 PUSHGP
362 PUSHI 0
363 PADD
364 PUSHI 2
365 LOADN
366 WRITEI
367 PUSHS ", "
368 WRITES
369 PUSHGP
370 PUSHI 0
371 PADD
372 PUSHI 3
373 LOADN
374 WRITEI
375 PUSHS "]"
376 WRITES
377
378 PUSHS "]"
```

```
379 WRITES
380 PUSHS "\nb: "
381 WRITES
382
383 PUSHS "["
384 WRITES
385
386 PUSHS "["
387 WRITES
388 PUSHGP
389 PUSHI 4
390 PADD
391 PUSHI 0
392 LOADN
393 WRITEI
394 PUSHS ", "
395 WRITES
396 PUSHGP
397 PUSHI 4
398 PADD
399 PUSHI 1
400 LOADN
401 WRITEI
402 PUSHS "]"
403 WRITES
404
405 PUSHS ", "
406 WRITES
407
408 PUSHS "["
409 WRITES
410 PUSHGP
411 PUSHI 4
412 PADD
413 PUSHI 2
414 LOADN
415 WRITEI
416 PUSHS ", "
417 WRITES
418 PUSHGP
419 PUSHI 4
420 PADD
421 PUSHI 3
422 LOADN
423 WRITEI
424 PUSHS "]"
425 WRITES
426
427 PUSHS "]"
428 WRITES
429 PUSHS "\nARRAY d e MATRIX b antes do SWAP\n"
```

```
WRITES
PUSHS "d: "
WRITES

PUSHS "["
WRITES
PUSHGP
PUSHI 12
PADD
PUSHI 0
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 12
PADD
PUSHI 1
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 12
PADD
PUSHI 2
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 12
PADD
PUSHI 3
LOADN
WRITEI

PUSHS "]"
WRITES
PUSHS "\nb: "
WRITES

PUSHS "["
WRITES

```

```
PUSHS "["
WRITES
PUSHGP
PUSHI 4
PADD
PUSHI 0
LOADN
WRITEI
PUSHS ", "
WRITES
PUSHGP
PUSHI 4
PADD
PUSHI 1
LOADN
WRITEI
PUSHS "]"
WRITES

PUSHS ", "
WRITES

PUSHS "["
WRITES
PUSHGP
PUSHI 4
PADD
PUSHI 2
LOADN
WRITEI
PUSHS ", "
WRITES
PUSHGP
PUSHI 4
PADD
PUSHI 3
LOADN
WRITEI
PUSHS "]"
WRITES

PUSHS "]"
WRITES

PUSHGP
PUSHI 12
PADD
PUSHI 0
PUSHGP
PUSHI 4
PADD
```

```
PUSHI 0
PUSHI 2
MUL
PUSHI 0
ADD
LOADN
PUSHGP
PUSHI 4
PADD
PUSHI 0
PUSHI 2
MUL
PUSHI 0
ADD
PUSHGP
PUSHI 12
PADD
PUSHI 0
LOADN

STOREN
STOREN
PUSHS "\nARRAY d e MATRIX b depois do SWAP\n"
WRITES
PUSHS "d: "
WRITES

PUSHS "["
WRITES
PUSHGP
PUSHI 12
PADD
PUSHI 0
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 12
PADD
PUSHI 1
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 12
```

```
583 PADD
584 PUSHI 2
585 LOADN
586 WRITEI
587
588 PUSHS ", "
589 WRITES
590
591 PUSHGP
592 PUSHI 12
593 PADD
594 PUSHI 3
595 LOADN
596 WRITEI
597
598 PUSHS "]"
599 WRITES
600 PUSHS "\nb: "
601 WRITES
602
603 PUSHS "["
604 WRITES
605
606 PUSHS "["
607 WRITES
608 PUSHGP
609 PUSHI 4
610 PADD
611 PUSHI 0
612 LOADN
613 WRITEI
614 PUSHS ", "
615 WRITES
616 PUSHGP
617 PUSHI 4
618 PADD
619 PUSHI 1
620 LOADN
621 WRITEI
622 PUSHS "]"
623 WRITES
624
625 PUSHS ", "
626 WRITES
627
628 PUSHS "["
629 WRITES
630 PUSHGP
631 PUSHI 4
632 PADD
633 PUSHI 2
```

```
LOADN
WRITEI
PUSHS ", "
WRITES
PUSHGP
PUSHI 4
PADD
PUSHI 3
LOADN
WRITEI
PUSHS "]"
WRITES

PUSHS "]"
WRITES
PUSHS "\nARRAY c antes do SWAP\n"
WRITES
PUSHS "c: "
WRITES

PUSHS "["
WRITES
PUSHGP
PUSHI 8
PADD
PUSHI 0
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 8
PADD
PUSHI 1
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 8
PADD
PUSHI 2
LOADN
WRITEI

PUSHS ", "
WRITES
```

```
685
686 PUSHGP
687 PUSHI 8
688 PADD
689 PUSHI 3
690 LOADN
691 WRITEI
692
693 PUSHS "]"
694 WRITES
695
696 PUSHGP
697 PUSHI 8
698 PADD
699 PUSHI 1
700 PUSHGP
701 PUSHI 8
702 PADD
703 PUSHI 2
704 LOADN
705 PUSHGP
706 PUSHI 8
707 PADD
708 PUSHI 2
709 PUSHGP
710 PUSHI 8
711 PADD
712 PUSHI 1
713 LOADN
714
715 STOREN
716 STOREN
717 PUSHS "\nARRAY c antes do SWAP\n"
718 WRITES
719 PUSHS "c: "
720 WRITES
721
722 PUSHS "["
723 WRITES
724 PUSHGP
725 PUSHI 8
726 PADD
727 PUSHI 0
728 LOADN
729 WRITEI
730
731 PUSHS ", "
732 WRITES
733
734 PUSHGP
735 PUSHI 8
```

```
PADD
PUSHI 1
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 8
PADD
PUSHI 2
LOADN
WRITEI

PUSHS ", "
WRITES

PUSHGP
PUSHI 8
PADD
PUSHI 3
LOADN
WRITEI

PUSHS "]"
WRITES
STOP
```

demoIndexingSwap output

```
MATRIXS a e b antes do SWAP
a: [[1, 2], [3, 4]]
b: [[10, 20], [30, 40]]
MATRIXS a e b depois do SWAP
a: [[20, 2], [30, 40]]
b: [[10, 1], [3, 4]]
ARRAY d e MATRIX b antes do SWAP
d: [-1, -2, -3, -4]
b: [[10, 1], [3, 4]]
ARRAY d e MATRIX b depois do SWAP
d: [10, -2, -3, -4]
b: [[-1, 1], [3, 4]]
ARRAY c antes do SWAP
c: [25, 50, 75, 100]
ARRAY c antes do SWAP
c: [25, 75, 50, 100]
```

A outra operação de indexação que implementamos nas nossa linguagem foi o comando SWAP, este comando irá trocar os valores registados numa determinada posição de um ARRAY ou MATRIX e vai trocar a posição dos mesmos pelo seu índice.

No caso das MATRIXS podemos inclusive trocar linhas completas desde que elas sejam do mesmo tamanho, é também possível trocar elementos atómicos entre MATRIXS e ARRAYS. Além disto, podemos utilizar esta operação para trocar a posição de elementos numa mesma variável

4.3 Deteção de erros

Na secção anterior mostramos as capacidades que a nossa linguagem inclui, no entanto é também importante demonstrar como o nosso programa está preparado para detetar possíveis erros que ocorram ao utilizar a nossa linguagem de forma incorreta.

errorDeclAss.ggo

```
INT a KEEPS 5
INT a KEEPS 10

b KEEPS [1,2,3]
```

errorDeclAss output

```
>> ERROR

>> The Variable a already exists
>> Error found at line 2

>> b has not been declared
>> Error found at line 3
```

Como se pode ver o nosso programa não permite a declaração de variáveis já declaradas, nem permite a utilização de variáveis não declaradas. Além disso indica a linha do código em que ocorreu o erro.

errorInvalidSize.ggo

```
ARRAY a(-6)
MATRIX b (-1,3)

MATRIX c KEEPS [[1,2],[3],[4,5,6]]
```

errorInvalidSize output

```
>> ERROR

>> Array size must be greater or equal than 0
>> Error found at line 1

>> Lines and cols' size must be greater or equal than 0
>> Error found at line 2

>> Invalid Matrix: All lines must be of the same size and cannot be
    0
>> Error found at line 3
```

Este exemplo volta a reforçar as retrições nas declarações de valores, neste caso de variáveis do tipo ARRAY e MATRIX. Onde o tamanho definido deve ser um tamanho válido.

No caso das MATRIXS é também importante destacar que a MATRIX atribuída deve ser válida, caso contrário o programa não irá permitir a sua atribuição.

errorAssignArrMat.ggo

```
MATRIX a KEEPS [[1,2],[3,4]]
ARRAY b KEEPS [1,2,3]

a KEEPS [[1],[2]]
a KEEPS [1,2]

b KEEPS [1,2,3,4,5]
b KEEPS [[1,2,3],[4,5,6]]
```

errorAssignArrMat output

```
>> ERROR

>> Invalid Matrix: a has 2 lines and 2 cols
>> Error found at line 3

>> a is not an ARRAY
```

```
>> Error found at line 4

>> List size must be equal to b size: 3
>> Error found at line 5

>> b is not a MATRIX
>> Error found at line 6
```

Neste exemplo fica claro que o nosso programa é bem exigente quanto à atribuição de valores a variáveis do tipo ARRAY e MATRIX, onde não é possível atribuir novos valores se estes não forem do mesmo tamanho que os originais, isto é, quando estes foram declarados. É capaz de distinguir se o utilizador está a tentar atribuir um ARRAY a uma MATRIX e vice-versa.

O programa dá ainda informação detalhada ao utilizador sobre a causa do erro.

errorSwap.ggo

```
MATRIX a KEEPS [[1,2,3],[4,5,6],[7,8,9]]
MATRIX b KEEPS [[1,2],[3,4]]

INT c

ARRAY d KEEPS [1,2,3]

a(1) SWAP b(0)
c(1) SWAP d(1)
a(0) SWAP a(2,2)
```

errorSwap output

```
>> ERROR

>> To execute SWAP both lines must be of the same size
>> Error found at line 5

>> SWAP only works with ARRAYS and MATRIXS
>> c is an INT
>> Error found at line 6

>> The values you are trying to SWAP have different sizes
>> Error found at line 7
```

Por fim este é um exemplo importante que nos demonstra que existem bastantes restrições para a utilização do comando SWAP por nós criado.

É importante perceber que não se podem trocar elementos de tamanhos diferentes nem é possível trocar com variáveis do tipo INT uma vez que estas não contêm índices

Conclusão

No decorrer de todo o projeto, fizemos por utilizar todo o suporte que adquirimos nas aulas de modo a ter sempre uma maior solidez nas várias secções do mesmo, de certo modo isso contribuiu imenso para consolidar a matéria lecionada visto que fomos capazes de interagir de uma forma mais prática com esses conteúdos.

Acreditamos que os objetivos propostos foram todos satisfeitos ao cumprir os mesmo obtivemos uma maior capacidade para escrever gramáticas e desenvolver compiladores de linguagens. Com ele também fomos capazes de aprofundar conhecimentos no que diz respeito à máquina virtual e da escrita em assembly e de compreender como são definidas as linguagens de programação no geral.

Em suma, a realização deste projeto foi bastante útil para consolidar as nossas bases e deixar-nos mais à vontade ao abordar certas temáticas da UC que poderão surgir num futuro profissional.