Universidade do Minho

Processamento de Linguagens e Compiladores

Trabalho Prático nº2:

Compilador em Yacc

Grupo nº8

Carlos Ferreira
a87953

Daniel Ribeiro
a87994

Braga, Portugal
21 de janeiro de 2021

Resumo

O seguinte relatório documenta, justifica, analisa e expõe todas as decisões tomadas ao longo do projeto "Compilador em YACC"realizado no âmbito da Unidade Curricular denominada Processamento de Linguagens e Compiladores no contexto do 3º ano do curso universitário Licenciatura em Ciências da Computação.

O seguinte trabalho consistia em usar-mos o *Flex* (*Fast Lexical Analyzer Generator*) e o *Yacc* (*Yet Another Compiler Compiler*) para criar um compilador de uma linguagem de programação simples criada por nós, compilando a nossa linguagem para *Assembly* que seria executada numa máquina virtual.

Ao longo deste relatório vamos apresentar a linguagem por nós criada assim como as formas de filtrar o código e transformar-lo em *Assembly*.

Apesar de algumas dificuldades iniciais, conseguimos resolver o problema com elevado grau de sucesso e estamos bastante satisfeitos com o resultado.

Conteúdo

1 Introdução **3**
1.1 Compilador em Yacc . . . 3

2 Análise do problema **4**
2.1 Requisitos do Enunciado . . . 4
2.2 Estratégia inicial adoptada . . . 5

3 Documentação da Linguagem **6**
3.1 Instruções . . . 6
3.1.1 Declaração e atribuição a variáveis . . . 6
3.1.2 operações aritméticas,relacionais e lógicas . . . 8
3.1.3 Ler do standard input e escrever no standard output . . . 9
3.1.4 Instruções condicionais . . . 10
3.1.5 Instruções cíclicas . . . 11
3.1.6 Manipulação de arrays . . . 12

4 Soluções criadas **14**
4.1 Estruturas de dados . . . 14
4.2 Filtros de texto . . . 14
4.3 Gramáticas . . . 15

5 Exemplos e Utilização **16**
5.1 Exemplo do funcionamento . . . 16
5.1.1 Quadrado . . . 16
5.1.2 Mínimo . . . 16
5.1.3 Produtório . . . 17
5.1.4 Ímpares . . . 17
5.1.5 Inverter array . . . 18
5.2 Utilização do Programa . . . 19

6 Conclusão **20**

A Código do filtro de texto (tp2.l) **21**

B Código da gramática (tp2.y) **22**

Capítulo 1

Introdução

1.1 Compilador em Yacc

Vamos agora enquadrar e contextualizar o trabalho.

No capítulo 2 fazemos uma breve introdução ao enunciado, assim como os pontos importantes a retirar desse mesmo como os requisitos da nossa linguagem e instruções que deverão de ser implementadas.

No capítulo 3 é documentada a nossa linguagem, assim como alguns programas exemplos.

No capítulo 4 são explicados os procedimentos usados para filtrar, armazenar e imprimir para ficheiro em *Assembly* a nossa linguagem.

No capítulo 5 mostramos programas exemplo da nossa linguagem e também os scripts criados para a sua compilação e execução na máquina virtual

No capítulo 6 concluímos o trabalho fazendo umas considerações finais á cerca do mesmo.

Capítulo 2

Análise do problema

2.1 Requisitos do Enunciado

No enunciado era nos informado que devíamos criar uma linguagem imperativa a nosso gosto, mas que devia permitir fazer o seguinte:

- Declarar variáveis atómicas do tipo inteiro, com os quais se podem realizar as habituais operações aritméticas.
- efetuar instruções algorítmicas básicas como a atribuiçao do valor de expressões numéricas a variáveis.
- ler do standard input e escrever no standard output.
- efetuar instruções condicionais para controlo do fluxo de execução.
- efetuar instruções cíclicas para controlo do fluxo de execução, permitindo o seu aninhamento. (dependendo do módulo no número do grupo, o nosso calhou-nos o "for-do")

E ainda uma das seguintes á escolha:

- 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 parˆametros mas que possam retornar um resultado do tipo inteiro.

Nós escolhemos a declaração e implementação de arrays de 1 dimensão pois parecia de maior utilidade para a nossa linguagem.
Adicionalmente tinha-mos de implementar programas de teste a utilizar a nossa linguagem, sendo eles:

- ler 4 números e dizer se podem ser os lados de um quadrado
- ler um inteiro N, depois ler N números e escrever o menor deles.
- ler N (constante do programa) números e calcular e imprimir o seu produtório.
- contar e imprimir os números impares de uma sequˆencia de números naturais.

E ainda um dos seguintes á escolha:

- ler e armazenar N números num array; imprimir os valores por ordem inversa.
- invocar e usar num programa seu uma função 'potencia()', que começa por ler do input a base B e o expoente E e retorna o valor B^E.

Como escolhemos implementar arrays, a nossa escolha de programa opcional foi a de ler e armazenar números e imprimir de ordem inversa.

Também nos foi fornecida a máquina virtual, assim como a documentação da mesma, que foi estudada intensamente por nós a fim de entender-mos correctamente suas instruções.

A partir deste conjunto de requisitos começamos a modelar a nossa linguagem, com base em outras linguagens imperativas já conhecidas por nós.

2.2 Estratégia inicial adoptada

Face a este problema decidimos começar baseando-nos num exemplo duma calculadora que o responsável da U.C. nos forneceu, essa calculadora fazia as operações em C, então começamos a converter essas operações para assembly imprimindo num ficheiro, começando pelas mais básicas para as mais complexas e depois adicionamos novas funcionalidades como as operações condicionais, os ciclos e os arrays.

Capítulo 3

Documentação da Linguagem

3.1 Instruções

Agora serão explicadas as instruções presentes na nossa linguagem

3.1.1 Declaração e atribuição a variáveis

Para declarar variáveis apenas temos de declarar da forma "int nome da variável = valor da variável;"ou "int nome da variável;"onde o nome apenas pode ser uma letra e o valor tem de ser um inteiro, as variáveis da nossa linguagem são assim todas inteiras, como era pedido no enunciado, o segundo caso irá iniciar a variável com valor 0; Nós usamos um contador e um array onde guardamos as posições das variáveis na stack.
Exemplo

```
Inicialmente temos:

contador = 0
TabId['a' - 'a'] = INT_MAX
TabId['x' - 'a'] = INT_MAX

Depois de ler-mos:

int a = 1;
int x;

Ficamos com:                                E na stack ficamos com:

contador = 2                                POS
TabId['a' - 'a'] = 0                         0    PUSHI 1
TabId['x' - 'a'] = 1                         1    PUSHI 0
```

Para atribuir um valor a uma váriavél já declarada usamos a forma "nome da variável = valor da variável".
Exemplo , tendo em conta o estado anterior se fizer-mos:

```
x = 3;

Como sabemos que x está na posição 1 da stack escrevemos no ficheiro:
PUSHI 3
STOREG 1

E internamente na stack, o valor 3 vai ser alocado na posição 1.
 POS                                POS
 0    PUSHI 1                       0    PUSHI 1
 1    PUSHI 0          ---->        1    PUSHI 3
 2    PUSHI 3
 3    STOREG 1
```

Uma variável caso seja redeclarada provocará um erro e o programar irá parar a sua execução.
As variáveis também podem ser inicializadas através de expressões como vamos ver no a seguir.

3.1.2 operações aritméticas,relacionais e lógicas

A nossa linguagem suporta operações aritméticas, relacionais e lógicas, assim como os parênteses para aumento de prioridade.
Sendo assim para escrevemos uma expressão da seguinte forma:

```
{constante/Variável} {operador} {constante/Variável} {Expressão Adicional}
```

Onde a Expressão Adicional pode ser vazia ou ser outro operador com constante/variável:

```
{}
ou
{operador} {constante/Variável}
```

Os operadores possíveis na nossa linguagem são os seguintes:
Aritméticos:

- $+ -> $ Adição
- $- -> $ Subtracção
- $* -> $ Multiplicação
- $/ -> $ Divisão
- $\% -> $ Módulo

Relacionais:

- $== -> $ Igual
- $! = -> $ Diferente
- $> -> $ Maior
- $< -> $ Menor
- $>= -> $ Maior ou igual
- $<= -> $ Menor ou igual

Lógicos:

- And $->$ E
- Or $->$ Ou

Prioridade

- ({expressão}) $->$ Dar prioridade á expressão dentro

É de notar que usamos a convenção "Bodmas"para a escolha da prioridade:

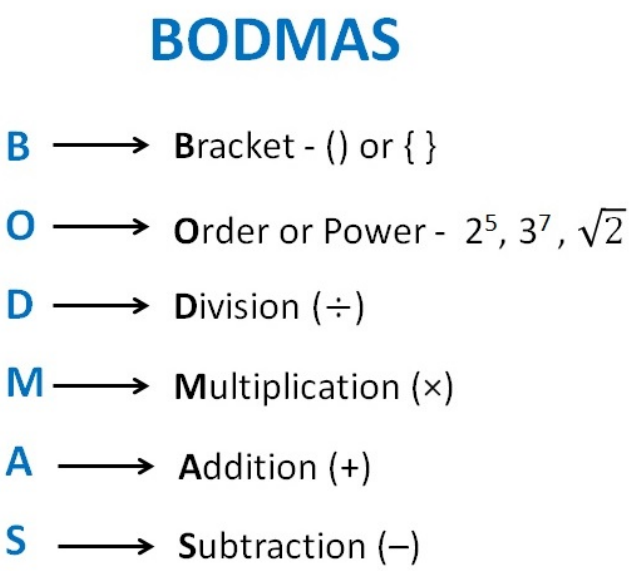

Figura 3.1: *Regra Bodmas*

Também podemos atribuir a qualquer variável o valor de uma expressão.
Assim, qualquer uma das seguintes Expressões são válidas na nossa linguagem:

```
int x = 7;
int y = 4;
y = 7 - 4;
int z = y * 3 + ( x >= y) ;

int x = 1;
int y = x > 7;
int z = x OR (y AND 1);
```

3.1.3 Ler do standard input e escrever no standard output

Para ler um inteiro do standard input usamos a função "READ", esta deve ser precedida por uma variável aonde fica guardado o inteiro lido, da seguinde forma:

```
int x = READ;
```

```
Que no ficheiro ficará:

READ
ATOI
```

Para escrever no standard output temos duas funções á escolha a "WRITE"e a "WRITES", a primeira devemos escrever como argumento uma variável para ser imprimida, já a segunda damos uma string para ser imprimida, como exemplo:

```
int x = Read;
Writes "O valor lido foi";
write x;

Na ficheiro ficará:
START
READ
ATOI
PUSHS "O valor lido foi"
WRITES
PUSHG 0
WRITEI
```

Além disso ainda temos a função return, invocada da forma "return Expressão"que imprime no standard output o valor de return da Expressão.

3.1.4 Instruções condicionais

Para as instruções condicionais implementamos o "IF ... THEN {...}"e o "IF ... THEN {...} ELSE {...}"
Cuja sintaxe é a seguinte: IF Expressão THEN {Comandos} e IF Expressão THEN {Comandos} ELSE {Comandos}
A expressão pode ser qualquer uma com operações aritméticas, lógicas ou relacionais, e os comandos pode ser qualquer expressão da linguagem, função de input ou output, outro if encadilhado ou até mesmo um ciclo.Resumidamente começamos por colocar a expressão condicional a ser testada antes de um JZ , caso seja válida o jump não é efetuado e a execução vai continuar normalmente (código do then) , caso a expressão testada seja inválida o JZ leva-nos para a etiqueta 'ELSE:' (código do else).

```
Expressão a ser verificada
JZ ELSE
Codigo do then
JUMP FIM
ELSE:
Codigo do else
FIM:
```

O seguinte é válido na nossa linguagem:

```
int x;
int k;
IF 1 < 5 Then {x = read; x = x + 1;} Else {k = 1;}

Que ficará no ficheiro :

START
PUSHI 0
PUSHI 0
PUSHI 1
PUSHI 5
INF
JZ ELSE0
READ
ATOI
STOREG 0
PUSHG 0
PUSHI 1
ADD
STOREG 0
JUMP FIM0
ELSE0:
PUSHI 1
STOREG 1
FIM0:
```

3.1.5 Instruções cíclicas

Devido aos requisitos do enunciado, o nosso grupo teve de implementar a instrução cíclica "for-do"
A instrução "for-do"é invocada com a seguinte sintaxe:

```
for(valor_inicial; condição_final; valor_incremento) do
{
    instruções;
}
```

Onde o valor inicial é o valor com que iniciamos a variável do ciclo, a condiçãofinal é a condição para finalização do ciclo e o valorincremento é o incremento da variável no final de cada iteração, que pode ser positivo ou negativo. Basicamente começamos por testar a condição com o JZ FIM , caso a condição se verifique entramos no codigo a ser executado e no fim tem um JUMP CICLO que nos volta a colocar outra vez na posição inicial para voltar a testar a condição.

```
CICLO:
Expressão a ser testada
JZ FIM
Codigo a ser executado
Codigo da incrementação da variavel
JUMP CICLO
FIMO:
```

Exemplo de funcionamento:

```
int i;
for(i = 0; i < 10; i = i + 1) do
{
    write i;
}

No ficheiro ficará:

START
PUSHI 0
PUSHI 0
STOREG 0
CICLO0:
PUSHG 0
PUSHI 10
INF
JZ FIM0
PUSHG 0
WRITEI
PUSHG 0
PUSHI 1
ADD
STOREG 0
JUMP CICLO0
FIM0:
```

3.1.6 Manipulação de arrays

A nossa linguagem permite a utilização de arrays de 1 e de 2 dimensões, a sintaxe para a declaração dos mesmos é a seguinte:

```
INT ARRAY [NUM] ID;

ou

INT ARRAY [NUM][NUM] ID;
```

Para arrays de 1 ou 2 dimensões respetivamente, o ID pode ser uma letra qualquer á escolha e o NUM é o tamanho do array que terá de ser um número positivo.

Para aceder ás diversas posições do array apenas temos de usar ID[Num], com isto podemos dar um valor á posição Num ou ver qual o valor lá guardado.

Nós guardamos o espaço para o array com PUSHN , e acede-mos a um posição do array com PUSHGP PUSHI (pos do array nao stack) PADD e dps por exemplo PUSHI 0 LOADN (da-nos primeira posição do array).

Como exemplo:

```
INT ARRAY [4] v;
v[0] = 2;

No ficheiro ficará:

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

O programa anterior é possível na nossa linguagem, o array v ficará assim com o valor 2 na posição 0.

Capítulo 4

Soluções criadas

Após a introdução á nossa linguagem no capítulo anterior, vamos mostrar como implementamos o nosso parser da linguagem

No início o nosso programa vai abrir o ficheiro de leitura e de escrita guardando os seus apontadores após isso é chamado yyparse() para fazer o parsing do ficheiro de entrada.

4.1 Estruturas de dados

Para auxiliar o nosso compilador, usamos um array de 26 posições chamado TabId onde guardamos a referência de cada variável na posição correspondente ao seu código ASCI - o código ASCI do carácter 'A', assim conseguimos converter para assembly sempre que precisar-mos da mesma.

4.2 Filtros de texto

Os nossos filtros de texto servem para auxiliar a nossa gramática, desta forma colocamos lá a filtragem de todos os tokens necessário, assim como strings e números.

4.3 Gramáticas

O axioma da nossa gramática chama-se "Codigo"é nele onde a impressão para ficheiro é feita, todas as outras transformações concatenam código assembly no acumulador ($$) para depois ser imprimido no axioma.

Após o "Codigo"temos os "Comandos"que são uma ou várias transformações "Comando"onde cada Comando pode ser uma condicional "if .. then ..", ou "if .. then .. else ..", um ciclo "for-do", uma declaração, uma atribuição, uma escrita para ficheiro "write s"ou "write"ou um return. Em cada um destes casos é então concatenado o respectivo código assembly necessário.

Ainda mais abaixo vamos ter outras transformações como a "Declaracao"que têm os vários tipos de declarações (variável, array de 1 ou 2 dimensões) ou a "Atrib"que tem os mesmos tipos que a declaração mas vai atribuir um novo valor em vez de o declarar.

Também temos as "ExprR"que são uma ou mais "Expr"e pode levar parênteses, as "Expr"são expressões mais básicas com os operadores lógicos, relacionais e aritméticos, sendo que estas também contém o "Termo"onde tem a multiplicação, divisação e módulo para evitar dupla recursividade no "Expr"e estes termos têm o "Fator"que é o mais básico da nossa linguagem que pode ser um número positivo ou negativo, um id de uma variável ou o id de um array de 1 ou 2 dimensões seguido da posição que queremos aceder ao valor.

Capítulo 5

Exemplos e Utilização

Neste capítulo será mostrada a utilização da nossa linguagem para resolver 5 problemas presentes no enunciado do trabalho

5.1 Exemplo do funcionamento

5.1.1 Quadrado

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

```
int x = READ;
int y = READ;
int z = READ;
int w = READ;
int r;

if ((x == y) and (y == z) and (z == w)) then {r = 1; writes "E um quadrado\n";}
                                        else {r = 0;writes "Nao e um quadrado\n";}
return r;
```

5.1.2 Mínimo

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

```
int n = READ;
int i;
int y = 0;
int m;
int x;

for ( i = 1; i < (n + 1) ; i = i + 1 ) do {
     x = READ;
```

```
     if (y == 0) then { m = x ; y = 1;} else { if (x < m) then {m = x;}}

     writes "apos ";
     write i;
     writes " iteracoes o minimo e ";
     write m;
     writes "\n";

     }

return m;
```

5.1.3 Produtório

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

```
int n = READ;
int i;
int x;
int r = 1;

for ( i = 1; i < (n + 1) ; i = i + 1 ) do {

     x = READ;
     r = r * x;

     writes "apos ";
     write i;
     writes " iteracoes o produtorio e ";
     write r;
     writes "\n";

     }

return r;
```

5.1.4 Ímpares

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

```
int n = READ;
int i;
```

```
int c = 0;
int x;
int y;

for ( i = 0; i < n ; i = i + 1 ) do {

     x = READ;
     y = x%2;
     if (   y != 0 ) then { c = c + 1; writes "Numero impar = " ; write x ; writes "\n"; } }


return c;
```

5.1.5 Inverter array

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

```
int n = READ;
int i;
int x;
int array [10] v;

for ( i = 0; i < n ; i = i + 1 ) do {

     x = READ;
     v[i] = x;

     }

for ( i = n-1; i > -1 ; i = i - 1 ) do {

     x = v[i];

     writes "v[";
     write i;
     writes "] = ";
     write x;
     writes "\n\n";

     }

return 0;
```

5.2 Utilização do Programa

Todos estes exemplos podem ser executados seguidos utilizando o nosso script "ExecutaTestes", abrindo o terminal na pasta "vms"e usando o código:

```
$ ./ExecutaTestes.sh
```

Para usar qualquer outro ficheiro de código, apenas tem de escrever um código para um ficheiro .txt e passsar o mesmo para a pasta "vms", após isso abrir a consola nessa pasta e digitar:

```
$ ./tp2 [nome do ficheiro txt] [nome do output]
```

Para compilar o código para assembly e após isso:

```
$ ./vms -g [nome do output]
```

Para executar o código assembly na máquina virtual.

Capítulo 6

Conclusão

Este trabalho teve um elevado grau de importância pois ajudou-nos bastante a consolidar a matéria lecionada nas aulas sobre *FLex e Yacc*, com um trabalho complexo e complicado mas enriquecedor e divertido.

Apesar de nos deparar-mos com diversas dificuldades, o grupo esteve á altura para tais, como por exemplo os operadores "E"e "OU"que tiverão de ser implementados em assembly, as prioridades das várias expressões e a implementação de arrays.

Finalizamos este relatório com certeza de termos acrescentado conhecimentos úteis e importantes para o nosso futuro, não só como filtragem em *FLex* mas também de gramáticas em *Yacc*, trabalho e organização em grupo assim como escrita de documentos em *Latex*, algo que não tinhamos muita experiência prévia.

Apêndice A

Código do filtro de texto (tp2.l)

```
%option noyywrap

%%
[+\-*\/()\r=\<\>\!\:\;\{\}\[\]\%]      {return yytext[0];}
(?i:RETURN)     {return(RETURN);}
(?i:AND)        {return(AND);}
(?i:OR)         {return(OR);}
(?i:IF)         {return(IF);}
(?i:THEN)       {return(THEN);}
(?i:ELSE)       {return(ELSE);}
(?i:FOR)        {return(FOR);}
(?i:DO)         {return(DO);}
(?i:READ)       {return(READ);}
(?i:WRITE)      {return(WRITE);}
(?i:WRITES)     {return(WRITES);}
(?i:INT)        {return(INT);}
(?i:ARRAY)      {return(ARRAY);}
[0-9]+          {yylval.valN = atof(yytext); return(NUM);}
[a-zA-Z]        {yylval.valC = toupper(yytext[0]); return (ID);}
\"[a-zA-Z\[\]0-9 \\n\=\,]+\"     {yylval.valS = strdup(yytext); return (STRING);}
.|\n            {;}
%%
```

Apêndice B

Código da gramática (tp2.y)

```
%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h>

int TabId[26];
int error = 0;
extern FILE *yyin,*yyout;
int c = 0 ;
int conta = 0;

%}

%union{ int valN; char valC; char* valS; }
%token <valN> NUM
%token <valC> ID
%token <valS> STRING
%token AND OR READ WRITE WRITES INT ARRAY
%token RETURN IF THEN ELSE FOR DO
%type <valS> Expr Termo ExprR Fator Comando Atrib Comandos Declaracao

%%


Codigo: Comandos                { fprintf(yyout,"START\n%s" , $1);}
      ;


Comandos : Comando              { if (error != 1 ) asprintf( &$$ , "%s" , $1);  }
         | Comandos Comando     { if (error != 1 ) asprintf( &$$ , "%s%s" , $1 , $2);  }
         ;
```

```
Comando : IF ExprR THEN '{' Comandos '}' ELSE '{' Comandos '}'
{ asprintf ( &$$ , "%sJZ ELSE%d\n%sJUMP FIM%d\nELSE%d:\n%sFIM%d:\n" , $2,c,$5, c,c,$9,c);c++;}
        | IF ExprR THEN '{' Comandos '}'
{ asprintf ( &$$ , "%sJZ FIM%d\n%sFIM%d:\n" , $2,c,$5,c);c++;}
        | FOR '(' Atrib ';' Expr ';' Atrib ')' DO '{' Comandos '}'
{ asprintf ( &$$ , "%sCICLO%d:\n%sJZ FIM%d\n%s%sJUMP CICLO%d\nFIM%d:\n",$3,c,$5,c,$11,$7,c,c);c+
        | Declaracao ';'                              { $$ = strdup($1); }
        | Atrib ';'                                   { $$ = strdup($1); }
        | WRITES STRING ';' { asprintf( &$$ , "PUSHS %s\n
                              WRITES\n",$2);}
        | WRITE  ID ';' { asprintf(&$$,"PUSHG %d\n
                        WRITEI\n" , TabId[$2-'A'] );}
        | RETURN ExprR  ';' { asprintf(&$$,"%sPUSHS \
                              "O return e \"\nWRITES\nWRITEI\nSTOP\n" , $2);}
      ;

Declaracao: INT Atrib                { $$ = strdup($2); }
          | INT ID                   { if (TabId[$2-'A'] == INT_MAX) {
                                               TabId[$2 - 'A'] =  conta ;
                                               conta++;
                                               asprintf( &$$ , "PUSHI 0\n" ); }
                                       else { error = 1;
                                              $$ = strdup("");
               fprintf(yyout,"ERR \"Redeclaracao de variavel\"\nSTOP\n");}  }

          | INT ARRAY '[' NUM ']' ID { if (TabId[$6-'A'] == INT_MAX) {
                                               TabId[$6 - 'A'] =  conta ;
                                               conta = conta + $4;
                                               asprintf( &$$ , "PUSHN %d\n" , $4);}

                                       else { error = 1;
                                              $$ = strdup("");
              fprintf(yyout,"ERR \"Redeclaração de variavel\"\nSTOP\n");}}

          | INT ARRAY '[' NUM ']' '[' NUM ']' ID { if (TabId[$9-'A'] == INT_MAX) {
                                               TabId[$9 - 'A'] =  conta ;
                                               conta = conta + ($4*$7);
                                               asprintf( &$$ , "PUSHN %d\n" , ($4*$7));}

                                                   else { error = 1;
                                                     $$ = strdup("");
            fprintf(yyout,"ERR \"Redeclaração de variavel\"\nSTOP\n");}}
          ;

Atrib : ID '=' ExprR                 { if (TabId[$1-'A'] == INT_MAX) {
                                               TabId[$1 - 'A'] =  conta ;
                                               asprintf( &$$ , "%s" ,$3);
                                               conta++;}
```

```
                else {asprintf( &$$ , "%sSTOREG %d\n" ,$3 , TabId[$1-'A']);}}

      | ID '[' Fator ']'  '=' ExprR    { if (TabId[$1-'A'] == INT_MAX) {
                                            error = 1;
                                            $$ = strdup("");
                fprintf(yyout,"ERR \"Array não declarado\"\nSTOP\n");}

                else{ asprintf( &$$ , "PUSHGP\nPUSHI %d\nPADD\n%s%sSTOREN\n" ,
                TabId[$1-'A'] , $3 , $6 ); }}

      | ID NUM '[' Fator ']''[' Fator ']'  '=' ExprR    { if (TabId[$1-'A'] == INT_MAX) {
                                                            error = 1;
                                                            $$ = strdup("");
                     fprintf(yyout,"ERR \"Array não declarado\"\nSTOP\n");}

                     else{ asprintf( &$$ , "PUSHGP\nPUSHI %d\nPADD\n%sPUSHI
                     %d\nMUL\n%sADD\n%sSTOREN\n" ,TabId[$1-'A'] , $4 , $2 , $7 ,$10 ); }}
      ;

ExprR : Expr                    { $$ = strdup($1); }
      | READ                    { asprintf( &$$ , "READ\nATOI\n");}
      | ExprR AND Expr          { asprintf ( &$$ ,
                        "%sPUSHI 0\nEQUAL\n%sPUSHI 0\nEQUAL\nADD\nPUSHI 0\nEQUAL\n" , $1,$3 );}
      | ExprR OR Expr           { asprintf ( &$$ ,
                        "%sPUSHI 0\nEQUAL\n%sPUSHI 0\nEQUAL\nADD\nPUSHI 2\nINF\n" , $1,$3 );}
      | '(' ExprR ')'           { $$ = strdup($2);}
      ;

Expr  : Termo                   { $$ = strdup($1); }
      | Expr '+' Termo          { asprintf( &$$ , "%s%sADD\n" , $1 ,$3 );}
      | Expr '-' Termo          { asprintf( &$$ , "%s%sSUB\n" , $1 ,$3 );}
      | Expr '=''=' Termo       { asprintf( &$$ , "%s%sEQUAL\n" , $1 ,$4 );}
      | Expr '!''=' Termo       { asprintf( &$$ , "%s%sEQUAL\nNOT\n" , $1 ,$4 ); }
      | Expr '<' Termo          { asprintf( &$$ , "%s%sINF\n" , $1 ,$3 ); }
      | Expr '<''=' Termo       { asprintf( &$$ , "%s%sINFEQ\n" , $1 ,$4 ); }
      | Expr '>' Termo          { asprintf( &$$ , "%s%sSUP\n" , $1 ,$3 ); }
      | Expr '>''=' Termo       { asprintf( &$$ , "%s%sSUPEQ\n" , $1 ,$4 ); }
      ;




Termo : Fator                   { $$ = strdup($1);}
      | Termo '*' Fator         { asprintf( &$$ , "%s%sMUL\n" , $1 ,$3 );}
      | Termo '/' Fator         {
                        asprintf( &$$ , "%sPUSHI 0\nEQUAL\nNOT\nJZ
                        ELSE%d\n%s%sDIV\nJUMP FIM%d\nELSE%d:
                        \nERR \"Divisão por zero\"\nSTOP\nFIM%d:\n" ,
```

```
                        $3,c,$1,$3, c,c,c);c++;}
      | Termo '%' Fator          { asprintf( &$$ , "%s%sMOD\n" , $1 ,$3 );}
      ;


Fator : NUM                        { asprintf( &$$ , "PUSHI %d\n" , $1 ); }
| '-' NUM                  { asprintf( &$$ , "PUSHI -%d\n" , $2 );}
| ID                       { if ( TabId[ $1-'A' ] != INT_MAX ) {
asprintf( &$$ , "PUSHG %d\n" , TabId[ $1-'A' ] ); }
            else {  fprintf(yyout,"ERR \"Variável não declarada\"\nSTOP\n");
                                          $$ = strdup("");
                                          error=1; } }
      | ID '[' Fator ']'          { if ( TabId[ $1-'A' ] != INT_MAX ) {
      asprintf( &$$ , "PUSHGP\nPUSHI %d\nPADD\n%sLOADN\n" , TabId[$1-'A'] , $3); }
                else {  fprintf(yyout,"ERR \"Array não declarado\"\nSTOP\n");
                           $$ = strdup("");
                           error=1; } }
      | ID NUM '[' Fator ']' '[' Fator ']'  { if ( TabId[ $1-'A' ] != INT_MAX ) {
      asprintf( &$$ , "PUSHGP\nPUSHI %d\nPADD\n%sPUSHI %d\nMUL\n%sADD\nLOADN\n" ,
      TabId[$1-'A'] , $4 , $2 , $7); }
                else {  fprintf(yyout,"ERR \"Array não declarado\"\nSTOP\n");
                          $$ = strdup("");
                          error=1; } }

| '(' Expr ')'              { $$ = strdup($2);}
;




%%

#include "lex.yy.c"


int yyerror(char * s){
printf("Frase invalida:%s\n" , s);

}


int main(int argc, char *argv[]){


         int i;
         for (i=0; i<26; i++) { TabId[i] = INT_MAX; }
```

```
         if ((yyin = fopen(argv[1],"r") )== NULL ) {
        printf("Não consegui ler '%s'\n", argv[1]);
                   return 0;
}



           if ((yyout = fopen(strcat(argv[2],".vm"),"w")) == NULL ) {
                 printf("Não consegui escrever '%s'\n", argv[2]);
            return 0;
}

                        printf("Inicio do parsing\n");


            yyparse();




                        printf("Fim do parsing\n");

            return 0;

}
```

Bibliografia

[1] Enunciado do trabalho prático,
https://elearning.uminho.pt/bbcswebdav/pid-1057968-dt-content-rid-4109395_1/courses/2021.850507_1/plc20tp2.pdf

[2] Template de um relatório em Latex fornecido pelo docente responsável pela U.C.,
https://elearning.uminho.pt/bbcswebdav/pid-1021529-dt-content-rid-3863379_1/courses/ 2021.850507_1/relprojLayout.pdf

[3] Regras de prioridade das operações,
https://www.teachoo.com/7479/2286/BODMAS/category/Definitions/