Processamento de Linguagens e Compiladores
(3ºano da Licenciatura em Ciências da
Computação)

Trabalho Prático 2

Relatório de Desenvolvimento

Bruno Guedes (A68707) Isac Meira (A70069)
Rafael Silva(A68685)

9 de Fevereiro de 2015

Resumo

Este relatório tem como fim, ilustrar todo o processo de desenvolvimento de um compilador de uma Linguagem de Programação Imperativa Simples, por nós definida, gerando pseudo-código assembly da Máquina Virtual, difundida pelo Professor Pedro Rangel, docente da Unidade Curricular.

Neste relatório, de escrita simples e clara, o leitor será levado a aprender e a conhecer as ideias por detrás das instruções desenvolvidas na solução, guiando-o através deste relatório para a solução final por nós desenvolvida.

Conteúdo

Capítulo 1

Introdução

No âmbito da Curricular de Processamento de Linguagens e Compiladores, enquadrada no 3º ano da Licenciatura em Ciências da Computação, fomos desafiados pelo Exmo. Professor Pedro Rangel Henriques, professor associado com agregação do Departamento de Informática da Universidade do Minho, no âmbito do 2º trabalho prático da disciplina, a realizar um compilador de uma Linguagem de Programação Imperativa que gerasse pseudo-código assembly da Virtual Machine, cuja a mesma e a sua documentação, foram por si divulgados.

O trabalho tem como principais objectivos, genericamente, aumentar a experiência dos alunos em engenharia de linguagens e em programação generativa e de forma específica, desenvolver capacidades na concepção de processadores de linguagens segundo o método da tradução dirigida pela sintaxe, suportado numa gramática tradutora, como também desenvolver um compilador gerando código para uma máquina de stack virtual e utilizar geradores de compiladores baseados em gramáticas tradutoras, sendo o AnTLR o escolhido pelo nosso grupo.

O Relatório de desenvolvimento começará, por uma análise dos requisitos, capítulo no qual tentámos descrever de forma sintética e assertiva aos requisitos que a nossa solução irá obedecer. De seguida, começaremos a abordar o nosso processo de desenvolvimento da Linguagem de Programação Imperativa, definindo inicialmente a gramática por nós utilizada e validada pelo docente da Unidade Curricular, focando-nos depois no código JAVA desenvolvido para a solução, mostrando uma vista global do mesmo e de seguida, sendo mais concretos e explicando cada uma das classes desenvolvidas e seus métodos mais importantes.

Após esta secção, explicaremos as acções semânticas por nós realizadas na gramática, sendo a sua análise feita através da explicação das declarações e de seguida das atribuições, dos condicionais e dos ciclos. Por fim, apresentaremos alguns testes e resultados da solução por nós realizados e falaremos em problemas de implementação que nos foram acontecendo no decorrer do trabalho e as decisões alternativas aos mesmos.

Capítulo 2

Análise dos Requisitos

No âmbito da Unidade Curricular de Processamento de Linguagens e Compiladores, foi-nos proposto pelo Exmo. Professor Pedro Rangel Henriques a definição de uma Linguagem de Programação Imperativa a nosso gosto, que deverá conseguir manusear tanto inteiros como *array's* de inteiros e realizar as operações básicas como atribuições de expressões a variáveis, ler do *standard input* e escrever no *standard output* bem como as instruções de controlo de fluxo condicional e cíclica. Deverão ainda estar disponíveis as operações relacionais, lógicas e aritméticas, como ainda as operações de indexação em *array's*.

A Linguagem de Programação Imperativa deverá apenas permitir as declarações de variáveis no ínicio do programa, como não deverá permitir re-declarações de variáveis nem o uso de variáveis sem declaração prévia.

Desta feita, será necessário desenvolver uma Gramática Independente de Contexto (GIC) e um compilador com base nessa GIC criada com o recurso ao Gerador AnTLR.

Assim, o compilador da Linguagem de Programação Imperativa deverá gerar pseudo-código Assembly da Máquina Virtual (VM) disponibilizada pelo Exmo. Professor Pedro Rangel Henriques.

Capítulo 3

Concepção da Linguagem de Programação Imperativa

Para a correcta concepção da Linguagem de Programação Imperativa, o nosso trabalho baseou-se nos seguintes pontos que serão abordados de seguida.

3.1 Definição da Gramática para a Linguagem de Programação Imperativa

Na concepção da nossa Linguagem de Programação Imperativa, primariamente definimos a nossa gramática para a mesma, a qual se segue:

```
grammar Gramatica;

lingImp     :  'BEGIN' decls statments 'END'
            ;
decls       :  ( decl )*
            ;
statments   : ( statment)*
            ;
statment    :  atrib
            |  ciclo
            |  condicional
            |  output
            |  input
            ;
atrib       :  nome '=' exp ';'
            | nome '[' exp ']' '=' exp
            ;
```

```
exp          :  termo ( opad termo )*
             ;
termo        :  factor ( opmul factor )*
             ;
factor       :  NUM
             |  bool
             |  nome
             |  nome '['exp ']'
             ;
opad         :  '+'
             |  '-'
             |  '||'
             ;
opmul        :  '*'
             |  '/'
             | '&&'
             ;
nome         :  PAL
             ;
ciclo        :  'WHILE' '(' condicao ')' '{' statments '}'
             ;
condicao     :  exp( simb exp ) *
             ;
bool         :  'TRUE'
             |  'FALSE'
             ;
simb         :  '<'
             |  '>'
             |  '>='
             |  '<='
             |  '&&'
             |  '||'
             |  '=='
             |  '!='
             ;
condicional  :  'IF' '(' condicao ')' 'THEN' '{' statments'}'
             ;
senao        :
             |  'ELSE' '{' statments '}'
             ;
decl         : 'INT' nome ';'
             | 'INT' nome '[' NUM ']' ';'
             ;
output       : 'PRINT' '(' TEXTO ',' exp ')' ';'
             ;
input        :   'SCAN' '(' nome ')' ';'
```

```
            ;

// ANALISADOR LEXICO.

TEXTO   :   '"' ~('"')* '"';
PAL  : (('a'..'z')|('A'..'Z'))+ ;
NUM  : [0-9]+;
WS   : ('\r'?'\n' | ' ' | '\t')+ -> skip;
```

Esta gramática foi validada pelo professor Pedro Rangel e tem como símbolos não terminais **lingImp, decls, statments, decl, statment, atrib, ciclo, condicional, output, input, atrib, exp, termo, factor, opad, opmul, bool, nome, condicao e simb**. E tem como símbolos não terminais os restantes símbolos da gramática.

3.2 Vista Global da Implementação das Classes em JAVA

Tendo este passo concluído, seguiu-se a elaboração de classes em JAVA a fim de criar uma Tabela que armazena os dados das variáveis declaradas no programa, segue-se uma vista global da sua óptica de concepção:

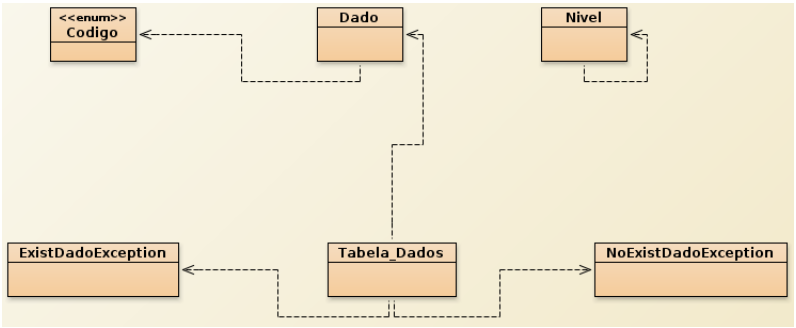

Figura 3.1: Vista Global das Classes elaboradas em JAVA.

3.3 Classe Enumeradora Codigo

A Classe Enumeradora Codigo, atribui um número a cada elemento da sua classe e possui métodos herdados da sua super-classe como ordinal(String) que nos dá o númeral da String armazenada.

```
public enum Codigo
{
    Verdadeiro, Falso, Soma, Sub, Ou, E, Vezes, Dividir, Menor, Maior, MenorIgual,
    MaiorIgual, Igual, Diferente, Inteiro, Array, Vazio
}
```

Figura 3.2: Classe Enum Codigo

3.4 Classe Dado

A classe Dado, tem como fim armazenar os dados das variáveis declaradas no programa. Esta classe guarda o nome da variável, o tipo da variável, que pode ser um Inteiro ou Array, o seu Endereço de Memória e a quantidade de células de memória ocupadas,caso seja um Inteiro o seu valor é 1, caso seja um Array o seu valor é o número de componentes do Array.

Esta classe tem como métodos os habituais métodos get's e set's das suas variáveis e os métodos clone() que permitem realizar *deep copys* dos dados o método equals() que permite comparar se dois dados são iguais e o método toString() que retorna uma String com os dados armazenados no objecto.

```
public class Dado
{
    // Variáveis de instância
    private Codigo tipo;
    private int endereco;
    private int quantidade;
    private String nome;
```

Figura 3.3: Classe Dado

3.5 Classe Tabela Dados

Esta classe implementa um mapeamento entre o nome de uma variável e os seus Dados. O nome da variável é nesta classe a chave do mapeamento. Esta classe usa um HashMap pois não é necessária ordem nos dados e esta implementação é mais eficiente computacionalmente.

```
import java.util.HashMap;
public class Tabela_Dados
{
    //Variavel de instância
    private HashMap<String, Dado> dcs; //Mapeia o nome da variavel nos seus dados (nome, quantidade, tipo, endereço)
```

Figura 3.4: Classe Tabela Dados

3.5.1 Métodos de instância da Classe Tabela Dados

Para além dos habituais métodos get's, set's, clone, equals e toString. A classe implementa os seguintes métodos.

```
/**
 * Elimina um Dado vindo de parâmetro do mapeamento caso este exista nele, caso contrário lança uma mensagem de erro.
 * @param Dado
 */
public void eliminaDado(Dado d)throws NoExistDadoException{
```

Figura 3.5: Método de Instância para eliminação de um Dado

```
/**
 * Testa a existência de um dado no mapeamento.
 * @param Dado
 * @return boolean
 */
public boolean existeDado(Dado d){
```

Figura 3.6: Método de Instância que testa a existência de um Dado no Mapeamento

```
/**
 * Dado um nome de uma variavel, testa se esse nome já existe como chave de mapeamento na tabela de dados.
 * @param String
 * @return Boolean
 */
public boolean existeChave(String chave){
```

Figura 3.7: Método de Instância que testa a existência de um chave no Mapeamento

```
/**
 * Dada o nome de uma variavel, testa se a mesma é chave no mapeamento, caso tal se verifique, retorna o Endereço de Memória que a variavel
 * ocupa na VM. Em caso contrário, lança uma mensagem de erro.
 * @param String
 * @return int
 */
public int getEndereco(String cod)throws NoExistDadoException{
```

Figura 3.8: Método de Instância que dada uma Chave do Mapeamento retorna o seu Endereço de Memória

```
/**
 * Insere um Dado vindo de parâmetro no mapeamento, caso este ainda não exista na tabela de dados, caso contrário lança uma mensagem de erro.
 * @param Dado
 */
public void insereDado(Dado d)throws ExistDadoException{
```

Figura 3.9: Método de Instância que Insere um Dado no Mapeamento

3.6 Classe Nível

Esta classe implementada com uma *stack* de inteiros, permite atribuir um nome único às *labels* usadas no pseudo-código *assembly* da VM.

```
public class Nivel
{
    private Stack<Integer> stk;
    private Integer i;
```

Figura 3.10: Classe Nivel

3.6.1 Métodos de instância da Classe Nivel

Para além dos habituais métodos get's, set's, clone, equals e toString. A classe implementa os seguintes métodos.

```
/**
 * coloca na stack a varíavel de instância i e incrementa essa mesma variável.
 */
public void incNivel(){
```

Figura 3.11: Método de Instância para a incrementação de Nível

```
/**
 * Retira da stack o inteiro do topo da mesma.
 */
public void decNivel(){
```

Figura 3.12: Método de Instância para a decrementação de Nível

```
/**
 * Retorna o inteiro que se encontra no topo da stack sem o remover da mesma.
 * @return Integer
 */
public int topNivel(){
```

Figura 3.13: Método de Instância para retornar o valor do Nível

3.7 Acções Semânticas Sobre a Gramática da Linguagem Imperativa

Nesta secção, explicar-se-á os pontos chave na concepção da solução. Começaremos por abordar as implementações fundamentais para criarmos uma tabela com os dados das variáveis declaradas e de seguida a forma como as atribuições, ciclos e condicionais foram implementados.

3.7.1 Declarações

As declarações de variáveis na nossa Linguagem de Programação Imperativa, apenas poderão ser declarações de variáveis inteiras ou variáveis na forma de um array.

Sempre que é declarada uma variável pretendemos guardar o seu nome, o endereço de memória que ocupará, se é um inteiro ou um array e a quantidade, isto é, caso seja um array a quantidade é o seu número de componentes, caso contrário a quantidade é 1. Pretendemos alojar estas declarações numa tabela, pois queremos que todas as variáveis usadas de seguida tenham sido previamente declaradas e que não permita re-declarações de variáveis, bem como quando estas são usadas em expressões possuir os dados necessários para as manipular na stack da VM.

Uma Linguagem de Programação Imperativa, poderá não ter declarações de variáveis como poderá ter várias declarações, pelo que como vimos anteriormente a gramática tem definido que decls (declarações) deriva em decl (declaração) [0 ou mais].

Como mencionado uma decl poderá ser apenas um inteiro ou um array pelo que as suas acções semânticas nestas produções serão: Após criarmos uma instância

```
decl returns[Dado d]
            : 'INT' nome ';' {  $decl.d = new Dado(Codigo.Inteiro, proxAdress++,
                                1, $nome.name); fich.println("PUSHI 0");}
            | 'INT' nome '[' n3 = NUM ']' ';'  {    $decl.d =
                new Dado(Codigo.Array, proxAdress, $n3.int, $nome.name);
                proxAdress+=$n3.int; fich.println("PUSHN " + $n3.int);}
            ;
```

Figura 3.14: Acções semânticas em decl

de Dado com os dados da varíavel que será retornado para decls, escrevemos no ficheiro que albergará o código assembley da VM a instrução PUSHI 0 que carrega para a *stack* a variável caso esta seja inteira, caso seja um array, realizamos a instrução PUSHN com o número de componentes do array, que carrega para a *stack* a declaração do array.

Desta feita, em decls criámos uma instância da Tabela de Dados que adiciona um mapeamento entre o nome da variável e os seus dados e que insere esses mesmos dados, caso estes ainda não existam no mapeamento, caso contrário,

emite uma mensagem de erro e aborta o programa, como se mostra de seguida:

```
decls   returns[Tabela_Dados dcs]
@init{$decls.dcs = new Tabela_Dados();}
            :  ( decl { try{
                           $decls.dcs.insereDado($decl.d);
                        }
                        catch(ExistDadoException e){
                           System.out.println(e.getMessage());
                           System.exit(1);
                        }
                      }
               )*
            ;
```

Figura 3.15: Acções Semânticas em Decls

3.7.2 Atribuição

A Tabela de Dados criada com as declarações será agora passada para os *statments* que tanto poderão ser atribuições, ciclos e condicionais, leitura do *standard input* e escrita do *standard output*.

Nesta secção abordaremos o caso da atribuição.

Na atribuição poderemos atribuir expressões a uma variável inteira, ou a um índice de um array.

Uma expressão consiste num termo e caso se possua mais termos um operador aditivo por cada termo. E cada termo consiste num factor e caso se possua mais factores um operador multiplicativo por cada factor. Desta forma, com estes níveis de abaixamento conseguimos manter a precedência entre os operadores multiplicativos e os operadores aditivos.

Nos operadores aditivos, assim como nos operadores multiplicativos, retornámos o seu Codigo que como já vimos um Codigo é uma classe enumeradora que atribui um numeral à lista de componentes que possuí.

Segue-se a forma simples como estas acções semânticas se procederam: Um

```
opmul returns [Codigo cod]
          :  '*'  {$opmul.cod = Codigo.Vezes;}
          |  '/'  {$opmul.cod = Codigo.Dividir;}
          | '&&' {$opmul.cod = Codigo.E;}
          ;
```

Figura 3.16: Acções Semânticas em Opmul

factor poderá ser tanto um número inteiro, como um booleano, uma variável inteira ou uma componente de um array.

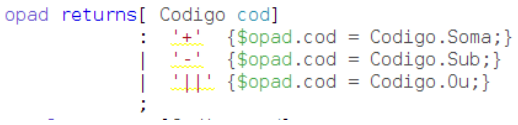

Figura 3.17: Acções Semânticas em Opad

Caso seja um número inteiro, apenas teremos que empilhar esse inteiro com a instrução PUSHI e o seu valor.

No caso de se tratar de um booleano, caso seja TRUE, escrevemos o seu valor como 1 e 0 como FALSE. No caso de se tratar de uma variável inteira, em-

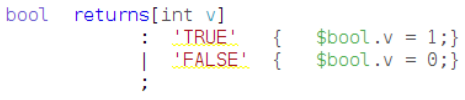

Figura 3.18: Acções Semânticas em Bool

pilhámos o valor da variável global, localizada no seu endereço de memória com PUSHG do endereço, caso a mesma tenha sido previamente declarada na devida secção, caso contrário emitirá uma mensagem de erro e parará o processo de compilação.

Caso se trate de uma componente de um array, teremos que empilhar o valor do endereço base do registo com PUSHGP, empilhar o endereço com PUSHI e o valor do endereço e de seguida fazer PADD que tem como fim retirar da pilha o inteiro n e o endereço a e empilhar o endereço a + n e por fim escrever no ficheiro LOADN que retira da pilha um inteiro n, um endereço a e empilha o valor na pilha ou no heap (dependendo do tipo de a) em a[n], isto se o array tiver sido declarado previamente, assim como no caso da variável inteira. Tal se demonstra na seguinte imagem: Desta feita, termo e exp terão uma codificação muito parecida respeitando os níveis de abaixamento em que caso haja operadores envolvidos, testa qual foi o operador utilizado e escreve o seu código assembly no ficheiro, como se pode ver: Por fim, teremos então que para termos uma atribuição, caso pretendamos passar uma expressão a uma variável inteira, teremos que primeiro testar se a variável foi declarada. De seguida, se tal condição se verificar, retirámos um valor da pilha e arquivamos na pilha no endereço da variável global com a instrução STOREG e o valor do Endereço.

No caso da expressão ser passada a um índice do array, após verificar que o mesmo foi declarado, empilhámos o valor do registo com PUSHGP, de seguida faz-se um PUSHI ao endereço base do array para o empilhar e realizámos a instrução PADD explicada anteriormente.

```
factor[Tabela_Dados dcs]
            :  NUM  {fich.println("PUSHI " + $NUM.int );}
            |  bool {fich.println($bool.v);}
            |  nome {   try{
                            Endr = $factor.dcs.getEndereco($nome.name);
                            fich.println("PUSHG " + Endr);
                        }
                        catch(NoExistDadoException e){
                           System.out.println(e.getMessage());
                           System.exit(1);
                        }
                    }
            | nome '['{ try{
                          Endr = $factor.dcs.getEndereco($nome.name);
                          fich.println("PUSHGP");
                          fich.println("PUSHI " + Endr);
                          fich.println("PADD");
                        }
                        catch(NoExistDadoException e){
                          System.out.println(e.getMessage());
                          System.exit(1);
                        }
                      }
                      exp[$factor.dcs] ']'{ fich.println("LOADN");}
            ;
```

Figura 3.19: Acções Semânticas em Factor

```
termo[Tabela_Dados dcs]
            :  factor[$termo.dcs] ( opmul factor[$termo.dcs]{ i =
                                                    $opmul.cod.ordinal();
                                                   switch(i){
                                                    case 5:
                                                     fich.println("AND");
                                                      break;
                                                    case 6:
                                                     fich.println("MUL");
                                                      break;
                                                    case 7:
                                                     fich.println("DIV");
                                                          break;
                                                    default:
                                                    System.out.println
                                                    ("Operador errado!");
                                                     System.exit(1);
                                                     break;
                                                            }
                                                          }
                                    )*
            ;
```

Figura 3.20: Acções Semânticas em Termo

3.7.3 Condicional

Um condicional é iniciado pela sua clausúla IF, seguido de uma condição que se se verificar, executa as instruções do bloco THEN e em caso contrário se

```
exp[Tabela_Dados dcs]
            :  termo[$exp.dcs] ( opad termo[$exp.dcs]{ i = $opad.cod.ordinal();
                                                       switch(i){
                                                          case 2:
                                                            fich.println("ADD");
                                                            break;
                                                          case 3:
                                                            fich.println("SUB");
                                                            break;
                                                          case 4:
                                                            fich.println("OR");
                                                            break;
                                                          default:
                                                            System.out.println
                                                           ("Operador errado!");
                                                            System.exit(1);
                                                            break;
                                                       }

                                                      }
                                )*
            ;
```

Figura 3.21: Acções Semânticas em exp

```
atrib[Tabela_Dados dcs]
            :  nome '=' exp[$atrib.dcs] ';'{try{
                                             Endr =
                                             $atrib.dcs.getEndereco($nome.name);
                                             fich.println("STOREG " + Endr);
                                            }
                                            catch(NoExistDadoException e){
                                             System.out.println(e.getMessage());
                                             System.exit(1);
                                            }
                                           }
            | nome '['{ try{
                          Endr = $atrib.dcs.getEndereco($nome.name);
                          fich.println("PUSHGP");
                          fich.println("PUSHI " + Endr);
                          fich.println("PADD");
                        }
                        catch(NoExistDadoException e){
                           System.out.println(e.getMessage());
                           System.exit(1);
                        }

                        }
                        e1=exp[$atrib.dcs] ']' '=' e2=exp[$atrib.dcs] ';'
                        {fich.println("STOREN");}
            ;
```

Figura 3.22: Acções Semânticas em atrib

a instrução ELSE estiver definida, dá origem à execução das instruções desse bloco.

Assim, como vimos anteriormente as codificações das exp vamos nos concentrar do que de novo esta zona trás.

No condicional, teremos após a sua condição um JUMP ON ZERO para a *label* senao. Após os statments do THEN fazemos um JUMP para a *label* fse.

Caso exista uma condição ELSE no fim do bloco de instruções da mesma, teremos uma *label* fse.

Uma condição irá derivar em expressões relacionadas com símbolos relacionais e lógicos. O símbolo usado, será passado para a condicao, onde avaliamos qual

```
simb    returns[Codigo cod]
            :  '<'  {$simb.cod = Codigo.Menor;}
            |  '>'  {$simb.cod = Codigo.Maior;}
            |  '>=' {$simb.cod = Codigo.MaiorIgual;}
            |  '<=' {$simb.cod = Codigo.MenorIgual;}
            |  '&&' {$simb.cod = Codigo.E;}
            |  '||' {$simb.cod = Codigo.Ou;}
            |  '==' {$simb.cod = Codigo.Igual;}
            |  '!=' {$simb.cod = Codigo.Diferente;}
            ;
```

Figura 3.23: Acções Semânticas em simb

o símbolo usado e o seu respectivo código *assembly*. Como se pode ver nesta figura. Pelo que desta forma, o condicional estará codificado da seguinte maneira, gerando as instruções *assembly* necessárias à sua correcta compilação.

3.8 Ciclo

Em ciclo, iremos ter o seu símbolo terminal WHILE, seguido de uma condição, que foi explicado a sua codificação na secção anterior. Depois do símbolo terminal WHILE e antes do da condição iremos ter uma *label* ciclo para a qual saltaremos por JZ antes do ínicio dos statments e para a qual saltaremos também após os statments onde definiremos também uma *label* para fim do ciclo, fciclo.

```
condicao[Tabela_Dados dcs]
            :  exp[$condicao.dcs] ( simb exp[$condicao.dcs] { //alterar
                                                              i = $simb.cod.ordinal();
                                                               switch(i){
                                                                 case 4:
                                                                     fich.println("OR");
                                                                     break;
                                                                 case 5:
                                                                     fich.println("AND");
                                                                     break;
                                                                 case 8:
                                                                     fich.println("INF");
                                                                      break;
                                                                 case 9:
                                                                     fich.println("SUP");
                                                                      break;
                                                                 case 10:
                                                                     fich.println("INFEQ");
                                                                      break;
                                                                 case 11:
                                                                     fich.println("SUPEQ");
                                                                      break;
                                                                 case 12:
                                                                     fich.println("EQUAL");
                                                                     fich.println("NOT");
                                                                      break;
                                                                 case 13:
                                                                     fich.println("EQUAL");
                                                                      break;
                                                                 default:
                                                                     System.out.println("Operador errado!")
                                                                     System.exit(1);
                                                                     break;
                                                                }
                                                              }
                                    ) *
            ;
```

Figura 3.24: Acções Semânticas em condicao

```
condicional[Tabela_Dados dcs]
            :  'IF' '(' condicao[$condicional.dcs] ')'
               {fich.println("JZ senao");}
               'THEN' '{' statments[$condicional.dcs] '}'
               {fich.println("JUMP fse");
               fich.println("senao: NOP");} senao[$condicional.dcs]
            ;
```

Figura 3.25: Acções Semânticas em condicional

```
ciclo[Tabela_Dados dcs]
            :  'WHILE' '('{fich.println("ciclo: NOP");} condicao[$ciclo.dcs] ')'
               {fich.println("JZ fciclo");} '{' statments[$ciclo.dcs]'}'
               {fich.println("JUMP ciclo"); fich.println("fciclo: NOP");}
            ;
```

Figura 3.26: Acções Semânticas em ciclo

Capítulo 4

Testes

4.1 Exemplo 1:

entrada de ficheiro vazio.

4.1.1 Resultados do Exemplo 1:

```
Arguments: [plc14tp2_isac, lingImp, -encoding, UTF-8, -tokens, -tree, -gui, /home/isac/Desktop/tplc/ex1.txt]
[@0,0:-1='<EOF>',<-1>,1:0]
line 1:0 mismatched input '<EOF>' expecting 'BEGIN'
lingImp
```

Figura 4.1: Mensagem de erro do exemplo 1 no ambiente AnTLR

4.2 Exemplo 2:

```
BEGIN
```

4.2.1 Resultados do Exemplo 2:

```
Arguments: [plc14tp2_isac, lingImp, -encoding, UTF-8, -tokens, -tree, -gui, /home/isac/Desktop/tplc/ex2.txt]
[@0,0:4='BEGIN',<29>,1:0]
[@1,5:4='<EOF>',<-1>,1:5]
line 1:5 missing 'END' at '<EOF>'
(lingImp BEGIN decls statments <missing 'END'>)
```

Figura 4.2: Mensagem de erro do exemplo 2 no ambiente AnTLR

4.3 Exemplo 3:

```
BEGIN
END
```

4.3.1 Resultados do Exemplo 3:

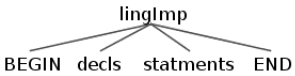

Figura 4.3: Árvore de Derivação do exemplo 3

Ficheiro Assembly gerado a partir do Código do Exemplo 3:

```
START
```

4.4 Exemplo 4:

```
BEGIN
INT a;
END
```

4.4.1 Resultado do Exemplo 4:

Ficheiro Assembly gerado a partir do Código do Exemplo 4:

```
PUSHI 0
START
```

4.5 Exemplo 5:

```
BEGIN
INT a;
```

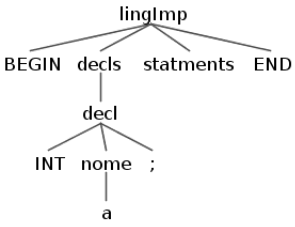

Figura 4.4: Árvore de Derivação do exemplo 4

```
INT a;
END
```

4.5.1 Resultado do Exemplo 5:

```
Arguments: [plc14tp2_isac, lingImp, -encoding, UTF-8, -tokens, -tree, -gui, /home/isac/Desktop/tplc/ex5.txt]
[@0,0:4='BEGIN',<29>,1:0]
[@1,6:8='INT',<31>,2:0]
[@2,10:10='a',<34>,2:4]
[@3,11:11=';',<23>,2:5]
[@4,13:15='INT',<31>,3:0]
[@5,17:17='a',<34>,3:4]
[@6,18:18=';',<23>,3:5]
[@7,20:22='END',<10>,4:0]
[@8,24:23='<EOF>',<-1>,5:0]
A variavel a já foi declarada anteriormente!
```

Figura 4.5: Mensagem de erro do exemplo 5 no ambiente AnTLR

4.6 Exemplo 6:

```
BEGIN
INT a;
INT b;

a = b + 2*b /100;
END
```

4.6.1 Resultado do Exemplo 6:

Ficheiro Assembly gerado a partir do Código do Exemplo 6:

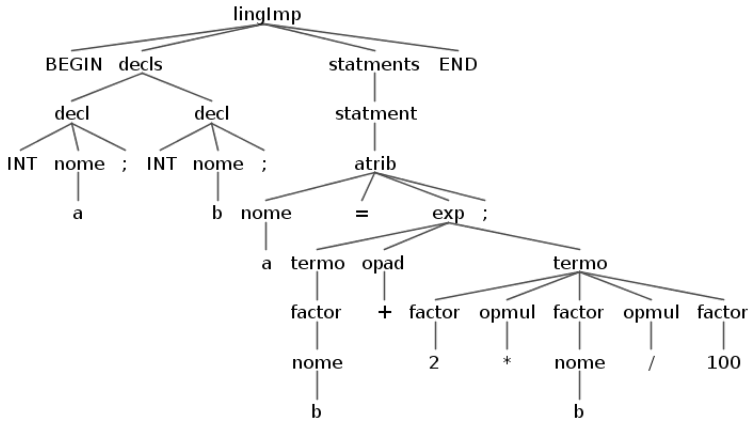

Figura 4.6: Árvore de Derivação do exemplo 6

```
PUSHI 0
PUSHI 0
START
PUSHG 1
PUSHI 2
PUSHG 1
MUL
PUSHI 100
DIV
ADD
STOREG 0
```

4.7 Exemplo 7:

```
BEGIN
INT a;

a = b + 2*b /100
END
```

4.7.1 Resultado do Exemplo 7:

4.8 Exemplo 8:

```
BEGIN
INT a;
INT b;
```

```
Arguments: [plc14tp2_isac, lingImp, -encoding, UTF-8, -tokens, -tree, -gui, /home/isac/Desktop/tplc/ex7.txt]
[@0,0:4='BEGIN',<29>,1:0]
[@1,6:8='INT',<31>,2:0]
[@2,10:10='a',<34>,2:4]
[@3,11:11=';',<23>,2:5]
[@4,14:14='a',<34>,4:0]
[@5,16:16='=',<22>,4:2]
[@6,18:18='b',<34>,4:4]
[@7,20:20='+',<20>,4:6]
[@8,22:22='2',<35>,4:8]
[@9,23:23='*',<7>,4:9]
[@10,24:24='b',<34>,4:10]
[@11,26:26='/',<28>,4:12]
[@12,27:29='100',<35>,4:13]
[@13,31:33='END',<10>,5:0]
[@14,34:33='<EOF>',<-1>,5:3]
A variavel b está a ser usada sem ter sido declarada!
```

Figura 4.7: Mensagem de erro do exemplo 7 no ambiente AnTLR

```
INT c;
INT d[100];

d[30] = a + b - c * d[10];
END
```

4.8.1 Resultado do Exemplo 8:

Ficheiro Assembly gerado a partir do Código do Exemplo 8:

```
PUSHI 0
PUSHI 0
PUSHI 0
PUSHN 100
START
PUSHGP
PUSHI 3
PADD
PUSHI 30
PUSHG 0
PUSHG 1
ADD
PUSHG 2
PUSHGP
PUSHI 3
PADD
PUSHI 10
LOADN
MUL
SUB
STOREN
```

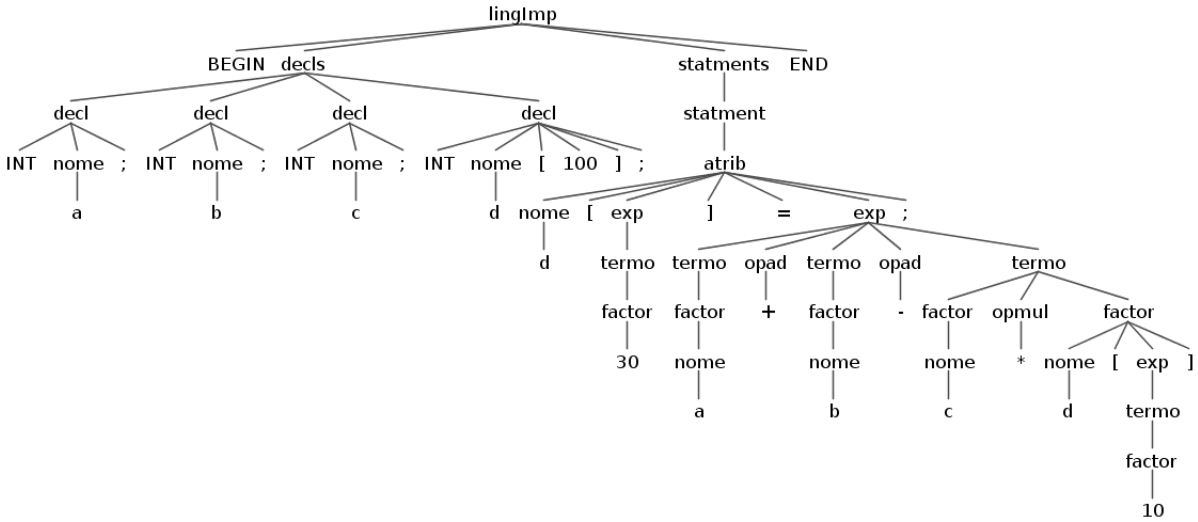

Figura 4.8: Árvore de Derivação do exemplo 8

4.9 Exemplo 9:

```
BEGIN
INT a;

a = 10;

IF (a <= 2*5)
THEN {
        a = 200 / 2;
```

```
}
ELSE{
        a = a + a;
}

a = 5;

IF (a <= 2*5)
THEN {
        a = 200 / 2;
}
ELSE{
        a = a + a;
}

END
```

4.9.1 Resultado do Exemplo 9:

Ficheiro Assembly gerado a partir do Código do Exemplo 9:

```
PUSHI 0
START
PUSHI 10
STOREG 0
PUSHG 0
PUSHI 2
PUSHI 5
MUL
INFEQ
JZ senao1
PUSHI 200
PUSHI 2
DIV
STOREG 0
JUMP fse1
senao1: NOP
PUSHG 0
PUSHG 0
ADD
STOREG 0
fse1: NOP
PUSHI 5
STOREG 0
PUSHG 0
PUSHI 2
```

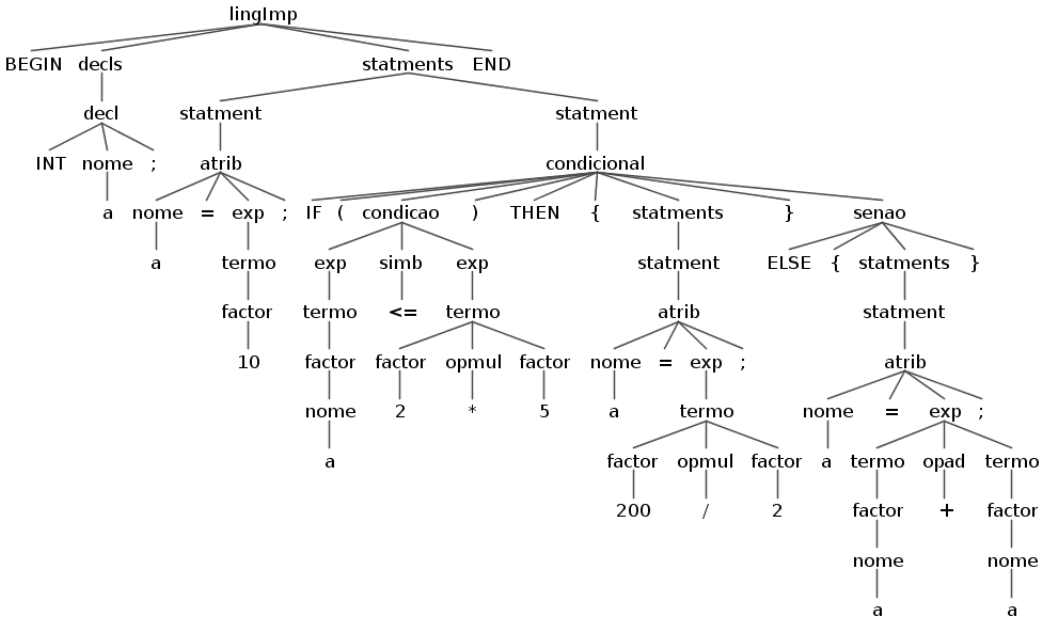

Figura 4.9: Árvore de Derivação do exemplo 9

```
PUSHI 5
MUL
INFEQ
JZ senao2
PUSHI 200
PUSHI 2
DIV
STOREG 0
JUMP fse2
senao2: NOP
```

```
PUSHG 0
PUSHG 0
ADD
STOREG 0
fse2: NOP
STOP
```

4.10 Exemplo 10:

```
BEGIN
INT a;
INT b;
a = 10;
b = 0;

IF (a <= 2*5)
THEN {
        a = 200 / 2;
}
ELSE{
        a = a + a;
}

WHILE(a < 200){
        b = b + 1;
}

END
```

4.10.1 Resultado do Exemplo 10:

Ficheiro Assembly gerado a partir do Código do Exemplo 10:

```
PUSHI 0
PUSHI 0
START
PUSHI 10
STOREG 0
PUSHI 0
STOREG 1
PUSHG 0
PUSHI 2
PUSHI 5
MUL
INFEQ
```

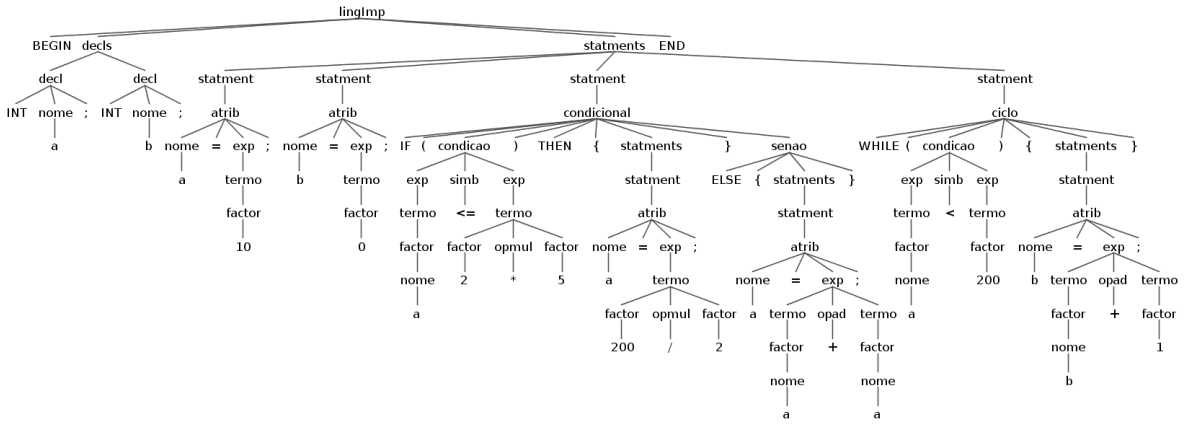

Figura 4.10: Árvore de Derivação do exemplo 10

```
JZ senao1
PUSHI 200
PUSHI 2
DIV
STOREG 0
JUMP fse1
senao1: NOP
PUSHG 0
PUSHG 0
ADD
STOREG 0
fse1: NOP
ciclo1: NOP
PUSHG 0
PUSHI 200
INF
JZ fciclo1
PUSHG 1
PUSHI 1
ADD
```

```
STOREG 1
JUMP ciclo1
fciclo1: NOP
STOP
```

4.11 Exemplo 11:

```
INT a;
INT b;
INT c;
INT arr[20];

a = 30;
b = 20;
c = 0;
WHILE(a <= 30 && a>=0  || b>=20 && b<=30){
        c = c+1;
}

INT d;

d = c;

END
```

4.11.1 Resultado do Exemplo 11:

```
line 1:0 missing 'BEGIN' at 'INT'
line 13:0 extraneous input 'INT' expecting {'SCAN', 'END', 'IF', 'WHILE', 'PRINT', PAL}
A variavel d está a ser usada sem ter sido declarada!
```

Figura 4.11: Mensagem de erro do exemplo 11 no ambiente AnTLR

4.12 Exemplo 12:

```
BEGIN

INT a;

SCAN(a);
PRINT("O Valor do Dobro da entrada Ã©", 2*a);

END
```

4.12.1 Resultado do Exemplo 12:

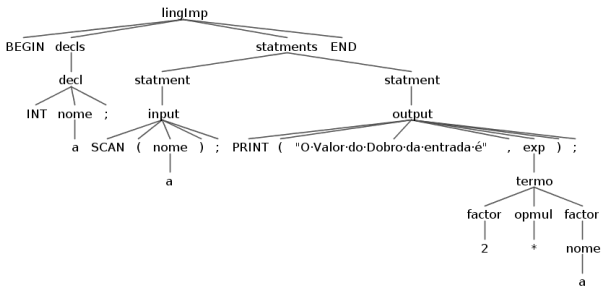

Figura 4.12: Árvore de Derivação do exemplo 12

Ficheiro Assembly gerado a partir do Código do Exemplo 12:

```
PUSHI 0
START
READ
ATOI
STOREG 0
PUSHI 2
PUSHG 0
MUL
PUSHS "O Valor do Dobro da entrada Ã©"
WRITES
WRITEI
STOP
```

4.13 Exemplo 13:

```
BEGIN

INT arr[100];
INT i;
INT j;

j = 100;
i = 0;

WHILE(i <=100){
        arr[i] = j;
        j = j-1;
        i = i+1;
}
```

```
WHILE(j <=100){
        PRINT("O valor do array na posiÃ§Ã£o Ã©", arr[j]);
}

END
```

4.13.1 Resultado do Exemplo 13:

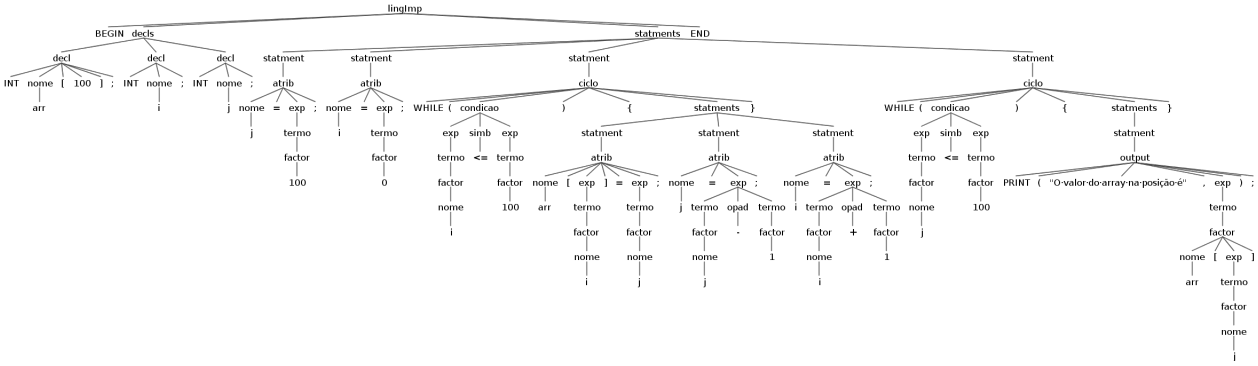

Figura 4.13: Árvore de Derivação do exemplo 13

Ficheiro Assembly gerado a partir do Código do Exemplo 13:

```
PUSHN 100
PUSHI 0
PUSHI 0
START
PUSHI 100
STOREG 101
PUSHI 0
STOREG 100
ciclo1: NOP
PUSHG 100
PUSHI 100
INFEQ
JZ fciclo1
PUSHGP
PUSHI 0
PADD
PUSHG 100
PUSHG 101
STOREN
PUSHG 101
```

```
PUSHI 1
SUB
STOREG 101
PUSHG 100
PUSHI 1
ADD
STOREG 100
JUMP ciclo1
fciclo1: NOP
ciclo2: NOP
PUSHG 101
PUSHI 100
INFEQ
JZ fciclo2
PUSHGP
PUSHI 0
PADD
PUSHG 101
LOADN
PUSHS "O valor do array na posiÃ§Ã£o Ã©"
WRITES
WRITEI
JUMP ciclo2
fciclo2: NOP
STOP
```

Capítulo 5

Alternativas, Decisões e Problemas de Implementação

No decorrer do trabalho seguiram-se vários problemas que levaram a implementações alternativas da solução.

O primeiro problema surgiu quando não se conseguiu realizar um *import* do *package* com a implementação das classes em JAVA no ambiente de desenvolvimento do AnTLR. Desta feita, a única alternativa viável para resolver este problema foi copiar o código da implementação das classes e colocar-lho antes do inicio das produções da gramática em "@members", perdendo-se assim, uma maior abstração do código produzido em JAVA diminuindo a concentração da parte central do trabalho, a gramática e as suas acções semânticas.

Outro problema de implementação, foi a tentativa frustrada de tornar a variável que atribui os Endereços de Memória aos objectos da Classe Dado uma variável de Classe em Dado, possuindo também os seus métodos de classe para darem o próximo Endereço de Memória disponível e a incrementação do valor do Endereço em uma ou mais unidades, que do ponto de vista conceptual e até por um caso de estilo e elegância tornariam a solução mais interessante do que declarar essa variável como global em "@members".

Por fim, tendo em conta o parco tempo disponível que o grupo possuía para a realização do trabalho prático, foi tomada a decisão de abandonar a implementação de declarações de funções e a sua invocação. Tornando a solução menos realista de uma Linguagem de Programação Imperativa útil à concepção de programas, mas que em contrapartida nos permitiu executar de forma elegante e robusta o restante trabalho.

Capítulo 6

Conclusão

Após toda a análise da solução por nós desenvolvida, começando na definição da gramática da Linguagem de Programação Imperativa, passando pela codificação das classes em JAVA e pelas respectivas acções semânticas na gramática, o leitor foi guiado a conhecer a nossa solução deste trabalho prático.

Como o leitor terá tido a oportunidade de visualizar, a definição da Gramática da Linguagem Imperativa, foi uma definição simples, onde não poderá ter o uso de , *float's string's*, declaração e invocação de funções, nem estruturas de dados, para além dos *array's*. Desta feita, seria interessante, futuramente acrescentar estas funcionalidades à Linguagem de Programação Imperativa, a fim de esta se tornar mais interessante e próxima de uma Linguagem de Programação Imperativa, usada de forma comum.

Tal situação se deve essencialmente ao pouco tempo disponível por parte dos redatores deste texto para a realização dessas mesmas funcionalidades, dentro do prazo de entrega definido, preferindo nós, possuir uma solução simples e arrojada que realiza de forma correcta tudo a que se predispôs.

Obrigado ao leitor, por acompanhar este trabalho, esperemos que tenha disfrutado da viagem e percebido os conceitos que pretendíamos transmitir com o mesmo.

Apêndice A

Código do Programa

Lista-se a seguir o código do par JAVA e AnTLR do programa que foi desenvolvido.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

grammar plc14tp2_isac;

@header{ import java.util.*;
         import java.io.*;
       }
@members{ int proxAdress = 0;
          int i;
          int Endr;
          PrintWriter fich;

         public enum Codigo
{
    Verdadeiro, Falso, Soma, Sub, Ou, E, Vezes, Dividir, Menor, Maior,
    MenorIgual, MaiorIgual, Igual, Diferente, Inteiro, Array, Vazio
}

public class Dado
{
    public Dado(Codigo tipo, int endereco, int quantidade, String nome){
        this.tipo = tipo;
        this.endereco = endereco;
        this.quantidade = quantidade;
```

```
    this.nome = nome;
}

public Dado(Dado d){
    this.tipo = d.getTipo();
    this.endereco = d.getEndereco();
    this.quantidade = d.getQuantidade();
    this.nome = d.getNome();
}

private Codigo tipo;
private int endereco;
private int quantidade;
private String nome;

public Codigo getTipo(){
    return this.tipo;
}

public int getEndereco(){
    return this.endereco;
}

public int getQuantidade(){
    return this.quantidade;
}

public String getNome(){
    return this.nome;
}

public void setTipo(Codigo tipo){
    this.tipo = tipo;
}

public void setEndereco(int endereco){
    this.endereco = endereco;
}

public void setQuantidade(int quantidade){
    this.quantidade = quantidade;
}

public void setNome(String nome){
    this.nome = nome;
}
```

```
    public boolean equals(Object obj){
        if(this == obj)
            return true;

        if(obj == null || this.getClass() != obj.getClass())
            return false;

        Dado d = (Dado) obj;

        return(this.tipo== d.getTipo() && this.endereco == d.getEndereco() &&
               this.quantidade == d.getQuantidade() &&
               this.nome.equals(d.getNome()));
    }

    public Dado clone(){
        return new Dado(this);
    }

    public String toString(){
        StringBuffer b = new StringBuffer();
        b.append("O nome Ã© " + this.nome + "\n");
        b.append("O tipo Ã© " + this.tipo.name() + "\n");
        b.append("A quantidade Ã© " + this.quantidade + "\n");
        b.append("O endereÃ§o Ã© " + this.endereco + "\n");

        return b.toString();
    }
}

public class Tabela_Dados
{
    private HashMap<String, Dado> dcs;

    public Tabela_Dados(){
        this.dcs = new HashMap<String, Dado>();
    }

    public Tabela_Dados(HashMap<String, Dado> dcs){
        dcs = new HashMap<String, Dado>();

        for(Dado d : dcs.values()){
            this.dcs.put(d.getNome(), d.clone());
        }

    }
```

```
public Tabela_Dados(Tabela_Dados tb){
    this.dcs = new HashMap<String, Dado>();

    for(Dado d : (tb.getDcs()).values()){
        this.dcs.put(d.getNome(), d.clone());
    }
}

public HashMap<String, Dado> getDcs(){
    HashMap<String, Dado> aux = new HashMap<String, Dado>();

    for(Dado d : this.dcs.values()){
        aux.put(d.getNome(), d.clone());
    }

    return aux;
}

public void setDcs(HashMap<String, Dado> dcs){
    this.dcs = new HashMap<String, Dado>();

    for(Dado d : this.dcs.values()){
        this.dcs.put(d.getNome(), d.clone());
    }
}

public void insereDado(Dado d)throws ExistDadoException{
    if(this.dcs.containsKey(d.getNome()))
        throw new ExistDadoException("A variavel " + d.getNome() +
                                     " jÃ¡ foi declarada anteriormente!");

    else
        this.dcs.put(d.getNome(), d.clone());

}

public void eliminaDado(Dado d)throws NoExistDadoException{
    if(!this.dcs.containsKey(d.getNome()))
        throw new NoExistDadoException("A variavel " + d.getNome() +
                                       " nÃ£o foi declarada!");

    else
        this.dcs.remove(d.getNome());

}
```

```
public boolean existeDado(Dado d){
    if(this.dcs.containsKey(d.getNome()))
        return true;
    else
        return false;
}

public boolean existeChave(String chave){
    if(this.dcs.containsKey(chave))
        return true;
    else
        return false;
}

public int getEndereco(String cod)throws NoExistDadoException{
    if(this.dcs.containsKey(cod)){
        Dado d = this.dcs.get(cod);
        return d.getEndereco();
    }
    else
        throw new NoExistDadoException("A variavel " + cod +
                                       " estÃ¡ a ser usada sem ter sido
                                       declarada!");
}

public boolean equals(Object obj){
    if(obj == this)
        return true;
    if(obj == null || obj.getClass() != this.getClass())
        return false;

    Tabela_Dados tb = (Tabela_Dados) obj;

    return(this.dcs.equals(tb.getDcs()));
}

public Tabela_Dados clone(){
    return new Tabela_Dados(this);
}

public String toString(){
    StringBuffer b = new StringBuffer();

    for(Dado d : this.dcs.values()){
        b.append("Chave: " + d.getNome());
```

```
            b.append("Valor: " + d.toString());
        }

        return b.toString();
    }
}

public class ExistDadoException extends Exception
{
    ExistDadoException(){
        super();
    }

    ExistDadoException(String e){
        super(e);
    }
}

public class NoExistDadoException extends Exception
{
    public NoExistDadoException(){
        super();
    }

    public NoExistDadoException(String e){
        super(e);
    }
}

public class Nivel
{
    private Stack<Integer> stk;
    private Integer i;

    public Nivel(){
        this.stk = new Stack<Integer>();
        this.i = new Integer(1);
    }

    public Nivel(Stack<Integer> stk, Integer i){
        this.stk = new Stack<Integer>();
        for(Integer j : stk)
            this.stk.push(j);
        this.i = new Integer(i.intValue());
    }
```

```
public Nivel(Nivel n){
    this.stk = new Stack<Integer>();
    for(Integer i : n.getStk())
        this.stk.push(i);

    this.i = n.getI();
}

public Stack<Integer> getStk(){
    Stack<Integer> aux = new Stack<Integer>();
    for(Integer i : this.stk)
        aux.push(i);
    return aux;
}

public Integer getI(){
    return this.i;
}

public void setStk(Stack<Integer> stk){
    this.stk = new Stack<Integer>();
    for(Integer i : stk)
        this.stk.push(i);
}

public void setI(Integer i){
    this.i = new Integer(i.intValue());
}

public void incNivel(){
    this.stk.push(this.i);
    this.i++;
}

public void decNivel(){
    this.stk.pop();
}

public int topNivel(){
    return this.stk.peek();
}

public Nivel clone(){
    return new Nivel(this);
}
```

```
    public boolean equals(Object obj){
        if(this == obj)
            return true;
        if(obj == null || this.getClass() != obj.getClass())
            return false;

        Nivel n = (Nivel) obj;

        return(this.stk.equals(n.getStk()));
    }

    public String toString(){
        StringBuffer s = new StringBuffer();
        for(Integer i : this.stk)
            s.append("1Â° nÃ°mero " + i);
        return s.toString();
    }
}

}

lingImp     :  'BEGIN' {    try{
                                fich = new PrintWriter(new FileWriter
                                ("Desktop/tplc/LingImp.asm"));
                            }
                            catch(IOException e){
                                System.out.println(e.getMessage());
                                System.exit(1);
                            }
                        }
                        decls    {   fich.println("START");}
                                     statments[$decls.dcs] 'END'
                                     {   fich.flush(); fich.close(); }
            ;
decls   returns[Tabela_Dados dcs]
@init{$decls.dcs = new Tabela_Dados();}
            :  ( decl { try{
                           $decls.dcs.insereDado($decl.d);
                        }
                        catch(ExistDadoException e){
                           System.out.println(e.getMessage());
                           System.exit(1);
                        }
                      }
               )*
            ;
```

```
statments[Tabela_Dados dcs]
            : ( statment[$statments.dcs] )*
            ;
statment[Tabela_Dados dcs]
            :  atrib[$statment.dcs]
            |  ciclo[$statment.dcs]
            |  condicional[$statment.dcs]
            |  output[$statment.dcs]
            |  input[$statment.dcs]
            ;
atrib[Tabela_Dados dcs]
            :  nome '=' exp[$atrib.dcs] ';'{try{
                                             Endr =
                                             $atrib.dcs.getEndereco($nome.name);
                                             fich.println("STOREG " + Endr);
                                            }
                                            catch(NoExistDadoException e){
                                             System.out.println(e.getMessage());
                                             System.exit(1);
                                            }
                                           }
            | nome '['{ try{
                            Endr = $atrib.dcs.getEndereco($nome.name);
                            fich.println("PUSHGP");
                            fich.println("PUSHI " + Endr);
                            fich.println("PADD");
                        }
                        catch(NoExistDadoException e){
                           System.out.println(e.getMessage());
                           System.exit(1);
                        }

                        }
                        e1=exp[$atrib.dcs] ']' '=' e2=exp[$atrib.dcs] ';'
                        {fich.println("STOREN");}
            ;
exp[Tabela_Dados dcs]
            :  termo[$exp.dcs] ( opad termo[$exp.dcs]{ i = $opad.cod.ordinal();
                                                       switch(i){
                                                          case 2:
                                                            fich.println("ADD");
                                                            break;
                                                          case 3:
                                                            fich.println("SUB");
                                                            break;
                                                          case 4:
```

```
                                                          fich.println("OR");
                                                          break;
                                                        default:
                                                          System.out.println
                                                         ("Operador errado!");
                                                          System.exit(1);
                                                          break;
                                                   }

                                                 }
                                )*
            ;
termo[Tabela_Dados dcs]
            :  factor[$termo.dcs] ( opmul factor[$termo.dcs]{ i =
                                                           $opmul.cod.ordinal();
                                                          switch(i){
                                                           case 5:
                                                            fich.println("AND");
                                                             break;
                                                           case 6:
                                                            fich.println("MUL");
                                                             break;
                                                           case 7:
                                                            fich.println("DIV");
                                                                 break;
                                                           default:
                                                           System.out.println
                                                           ("Operador errado!");
                                                            System.exit(1);
                                                            break;
                                                                    }
                                                                  }
                                   )*
            ;
factor[Tabela_Dados dcs]
            :  NUM  {fich.println("PUSHI " + $NUM.int );}
            |  bool {fich.println($bool.v);}
            |  nome {   try{
                             Endr = $factor.dcs.getEndereco($nome.name);
                             fich.println("PUSHG " + Endr);
                        }
                        catch(NoExistDadoException e){
                           System.out.println(e.getMessage());
                           System.exit(1);
                        }
                    }
```

```
            | nome '['{ try{
                          Endr = $factor.dcs.getEndereco($nome.name);
                          fich.println("PUSHGP");
                          fich.println("PUSHI " + Endr);
                          fich.println("PADD");
                        }
                        catch(NoExistDadoException e){
                          System.out.println(e.getMessage());
                          System.exit(1);
                        }
                      }
                      exp[$factor.dcs] ']'{ fich.println("LOADN");}
            ;
opad returns[ Codigo cod]
            :  '+'  {$opad.cod = Codigo.Soma;}
            |  '-'  {$opad.cod = Codigo.Sub;}
            |  '||' {$opad.cod = Codigo.Ou;}
            ;
opmul returns [Codigo cod]
            :  '*'  {$opmul.cod = Codigo.Vezes;}
            |  '/'  {$opmul.cod = Codigo.Dividir;}
            | '&&' {$opmul.cod = Codigo.E;}
            ;
nome returns [String name]
            :  n1=PAL { $nome.name = $n1.text;}
            ;
ciclo[Tabela_Dados dcs]
            :  'WHILE' '('{nCicl.incNivel(); fich.println("ciclo" +
                          nCicl.topNivel() + ": NOP");} condicao[$ciclo.dcs]
               ')' {fich.println("JZ fciclo" + nCicl.topNivel());}
               '{' statments[$ciclo.dcs]'}'{fich.println("JUMP ciclo" +
                nCicl.topNivel());
                fich.println("fciclo" + nCicl.topNivel() +
                ": NOP"); nCicl.decNivel();}
            ;
condicao[Tabela_Dados dcs]
            :  exp[$condicao.dcs] ( simb exp[$condicao.dcs] {
                                                    i = $simb.cod.ordinal();
                                                    switch(i){
                                                      case 4:
                                                       fich.println("OR");
                                                        break;
                                                      case 5:
                                                        fich.println("AND");
                                                        break;
                                                      case 8:
```

```
                                                                    fich.println("INF");
                                                                     break;
                                                                  case 9:
                                                                    fich.println("SUP");
                                                                     break;
                                                                 case 10:
                                                                  fich.println("INFEQ");
                                                                    break;
                                                                 case 11:
                                                                  fich.println("SUPEQ");
                                                                    break;
                                                                 case 12:
                                                                  fich.println("EQUAL");
                                                                  fich.println("NOT");
                                                                    break;
                                                                 case 13:
                                                                  fich.println("EQUAL");
                                                                    break;
                                                                  default:
                                                                   System.out.println
                                                                   ("Operador errado!");
                                                                   System.exit(1);
                                                                   break;
                                                                       }
                                                                     }) *
            ;
bool  returns[int v]
            :  'TRUE'   {   $bool.v = 1;}
            |  'FALSE'  {   $bool.v = 0;}
            ;
simb    returns[Codigo cod]
            :  '<'  {$simb.cod = Codigo.Menor;}
            |  '>'  {$simb.cod = Codigo.Maior;}
            |  '>=' {$simb.cod = Codigo.MaiorIgual;}
            |  '<=' {$simb.cod = Codigo.MenorIgual;}
            |  '&&' {$simb.cod = Codigo.E;}
            |  '||' {$simb.cod = Codigo.Ou;}
            |  '==' {$simb.cod = Codigo.Igual;}
            |  '!=' {$simb.cod = Codigo.Diferente;}
            ;
condicional[Tabela_Dados dcs]
            :  'IF' '(' condicao[$condicional.dcs] ')'
               {nCond.incNivel(); fich.println("JZ senao" + nCond.topNivel());}
               'THEN' '{' statments[$condicional.dcs] '}'
               {fich.println("JUMP fse" + nCond.topNivel());
               fich.println("senao" + nCond.topNivel() +": NOP");}
```

```
               senao[$condicional.dcs] { nCond.decNivel();}
            ;
senao[Tabela_Dados dcs]
            :
            |  'ELSE' '{' statments[$senao.dcs] '}'
               {fich.println("fse" + nCond.topNivel() + ": NOP");}
            ;
decl returns[Dado d]
            : 'INT' nome ';' {  $decl.d = new Dado(Codigo.Inteiro, proxAdress++,
                                1, $nome.name); fich.println("PUSHI 0");}
            | 'INT' nome '[' n3 = NUM ']' ';'  {    $decl.d =
                new Dado(Codigo.Array, proxAdress, $n3.int, $nome.name);
                proxAdress+=$n3.int; fich.println("PUSHN " + $n3.int);}
            ;
output[Tabela_Dados dcs]
            : 'PRINT' '(' TEXTO ',' exp[$output.dcs] ')' ';'
              { fich.println("PUSHS " + $TEXTO.text); fich.println("WRITES");
               fich.println("WRITEI"); }
            ;
input[Tabela_Dados dcs]
            :   'SCAN' '(' nome ')' ';'{  try{
                                             Endr =
                                             $input.dcs.getEndereco($nome.name);
                                                fich.println("READ");
                                                fich.println("ATOI");
                                                fich.println("STOREG " + Endr);
                                                }
                                          catch(NoExistDadoException e){
                                            System.out.println(e.getMessage());
                                            System.exit(1);
                                           }
                                       }
            ;


TEXTO   :   '"' ~('"')* '"';
PAL  : (('a'..'z')|('A'..'Z'))+ ;
NUM  : [0-9]+;
WS   : ('\r'?'\n' | ' ' | '\t')+ -> skip;
```