Processamento de Linguagens
(3º ano de Engenharia Informática)

Trabalho Prático nº 2
Compilador LogoLISS

Relatório de Desenvolvimento

José Pedro Vieira Costa e Silva (49423)
Emanuel José Vieira Gonçalves (49336)

13 de Junho de 2010

Resumo

Este relatório resume, analisa e explana os objectivos deste segundo trabalho prático da unidade curricular Processamento de Linguagens, as decisões tomadas ao longo do seu desenvolvimento, tecnologias usadas e objectivos obtidos.
Consideramos um trabalho interessante e exemplar do quão útil pode ser a utilização das ferramentas Flex e Yacc.

Conteúdo

Capítulo 1

Introdução

Este trabalho é realizado no âmbito da UC de Processamento de Linguagens que tem como base o uso do analisador léxico Flex e do analisador sintáctico Yacc, com os quais se pretende desenvolver um compilador capaz de analisar um programa em LogoLISS e gerar o respectivo código máquina para uma máquina virtual.

LogoLISS é a simbiose da linguagem Logo, uma linguagem de programação principalmente dirigida a crianças, que permite o movimentação de uma tartaruga, e de uma versão simplificada da linguagem LISS, que permite manusear escalares e vectores, quer estes sejam constantes, quer sejam variáveis.

Neste relatório serão abordadas as decisões tomadas pelo grupo, as dificuldades sentidas, o conteúdo final do compilador criado, do código assembly gerado para máquina virtual e discutir-se-á o resultado final obitdo.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Como objectivo deste trabalho prático pretendemos construir um compilador para a linguagem LogoLISS, utilizando como recursos o analisador léxico FLEX e o analisador sintáctico e semântico YACC.
Começou-se por analisar a gramática fornecida e determinou-se todos os simbolos terminais sendo estes analisados e retornados no FLEX para o YACC. Posteriormente no YACC faz-se a análise sintática de cada produção e são gerados os comportamentos respectivos.

2.2 Especificação de Requisitos

2.2.1 Dados

No levantamento dos dados necessários à resolução do trabalho proposto tivemos que utilizar a gramática abaixo, que foi fornecida pela equipa docente:

```
1  /********************** Program
2          Liss ––> "PROGRAM" identifier "{" Body "}"
3          Body ––> "DECLARATIONS" Declarations
4          "STATEMENTS" Statements
5
6  /********************** Declarations
7          Declarations ––> Declaration
8                                  | Declarations Declaration
9          Declaration ––> Variable_Declaration
10
11 /********************** Declarations: Variables
12         Variable_Declaration ––> Vars "–>" Type ";"
13         Vars ––>  Var
```

```
                        | Vars "," Var
        Var --> identifier Value_Var
        Value_Var -->
                                | "=" Inic_Var
        Type --> "INTEGER"
                        | "BOOLEAN"
                        | "ARRAY" "SIZE" number
        Inic_Var --> Constant
                                | Array_Definition
        Constant --> Sign number
                                | "TRUE"
                                | "FALSE"
        Sign -->
                        | "+"
                        | "-"
/********************** Declarations: Variables: Array_Definition
Array_Definition --> "[" Array_Initialization "]"
Array_Initialization --> Elem
| Array_Initialization "," Elem
Elem --> Sign number

/************************ Statements
        Statements --> Statement
                                | Statements Statement
        Statement --> Turtle_Commands
                                | Assignment
                                | Conditional_Statement
                                | Iterative_Statement

/************************ Turtle Statement
        Turtle_Commands --> Step
                                        | Rotate
                                        | Mode
                                        | Dialogue
                                        | Location
        Step --> "FORWARD" Expression
                        | "BACKWARD" Expression
        Rotate --> "RRIGHT"
                        | "RLEFT"
        Mode --> "PEN" "UP"
                        | "PEN" "DOWN"
        Dialogue --> Say_Statement
                        | Ask_Statement
        Location --> "GOTO" number "," number
                        | "WHERE" "?"

/************************ Assignment Statement
Assignment --> Variable "=" Expression

Variable --> identifier Array_Acess
```

```
64
65  Array_Acess -->
66                          | "[" Single_Expression "]"
67
68  /********************** Expression
69          Expression --> Single_Expression
70                          | Expression Rel_Oper Single_Expression
71
72  /******** Single_Expression
73          Single_Expression --> Term
74                          | Single_Expression Add_Op Term
75
76  /******** Term
77          Term --> Factor
78                          | Term Mul_Op Factor
79
80  /******** Factor
81          Factor --> Constant
82                          | Variable
83                          | SuccOrPred
84                          | "!" Expression
85                          | "+" Expression
86                          | "-" Expression
87                          | "(" Expression ")"
88
89  /******** Operators
90          Add_Op --> "+"
91                          | "-"
92                          | "||"
93          Mul_Op --> "*"
94                          | "/"
95                          | "&&"
96                          | "**"
97          Rel_Op --> "=="
98                          | "!="
99                          | "<"
100                         | ">"
101                         | "<="
102                         | ">="
103                         | "in"
104
105 /******** SuccOrPredd
106         SuccOrPred --> SuccPred identifier
107         SuccPred -->  "SUCC"
108                                 | "PRED"
109
110 /******************** IO Statements
111         Say_Statement --> "SAY" "(" Expression ")"
112         Ask_Statement --> "ASK" "(" string "," Variable ")"
113
```

```
/********************* Conditional & Iterative Statements
        Conditional_Statement --> IfThenElse_Stat
        Iterative_Statement --> While_Stat

/******** IfThenElse_Stat
        IfThenElse_Stat -->  "IF" Expression
                               "THEN" "{" Statements "}"
                               Else_Expression
        Else_Expression -->
                               | "ELSE" "{" Statements "}"

/******** While_Stat
        While_Stat --> "WHILE" "(" Expression ")" "{" Statements "}"
```

No final deste trabalho usando a gramática acima referida deveremos ser capazes de compilar o exemplo abaixo mencionado (desenho de um quadrado de lado 100):

```
PROGRAM logolissExemplo
{
  DECLARATIONS
    x = (100) , y -> Integer ;
    z -> Boolean;
    w = TRUE -> Boolean ;
  STATEMENTS
    FORWARD x
    RRIGHT
    y = (100)
    FORWARD y
    RRIGHT
    x = x - (100)
    FORWARD x + (100)
    RRIGHT
    FORWARD (100)
}
```

2.2.2 Pedidos

No final deste trabalho prático pretende-se ter um compilador que seja capaz de ler programas em LogoLISS, detectar incorrecções sintácticas e que gere código em assembly para a máquina virtual.

Capítulo 3

Concepção/desenho da Resolução

3.1 Estruturas de Dados

Durante o desenvolvimento do compilador tivemos a necessidade de recorrer a estruturas de dados, tanto para analizar a informação lida como para guardar-la, as estrututras abaixo referidas encontram-se declaradas no ficheiro yacc.y. A estrutura tVarString, serve para guardar todas as declarações de variáveis de uma linha antes de ser adicionada na tabelas de identificadores e é constituida por um nome, valor e tem um apontador para a próxima variável se existir:

```
1  typedef struct Vars {
2          char* nome;
3          char* valor;
4          struct Vars* next;
5  }tVarString;
```

Definimos também uma struct ParIdVL que é utilizada quando se está a fazer um declaração ou uma atribuição, para serem guardados o nome e o valor da variável.

```
1  typedef struct ParIdentValor {
2    char* nome;
3    char* valor;
4  }ParIdVl;
```

Em último, definimos uma estrutura com um inteiro chamdo valorInt e um char* valor, que serve para guardar um valor, que será um inteiro ou um boolean, no primeiro caso é guardado na variável valorInt, no segundo será guardado na variável valor.

```
1  typedef struct {
```

```
2       int valorInt;
3     char * valorString;
4   } Parconst;
```

Recorremos a uma hashtable para representar a nossa tabela de identifcadores e definimos uma estrutura que de adpta-se às nossas necessidades.

```
1   typedef struct _node{
2           char *name;
3           int tipo;
4           int address;
5           int stackgl;
6           struct _node *next;
7   }node;
```

Nesta estrutura guardamos o nome da variável na variável name, o seu tipo que pode ser zero se for um integer ou 1 se for um boolean, a variável address é utilizada para guardar o posição de memória da variável na stack usada pela máquina virtual, a variável stackgl indica se a variável é global (stackgl igual a zero) ou não (stackgl igual a um), por fim vem a variável _node que aponta para o próximo elemento, caso exista, com a mesma chave de hash.

3.2 Algoritmos

Primeiramente criamos o ficheiro lex.l que contém padrões a ser detectatos no ficheiro que tem um programa em LogoLISS, depois esse ficheiro .l é passado ao flex para gerar um analisador léxico, posteriormente no ficheiro yacc.y, criado por nós, contém um include do ficheiro lex.yy.c que é gerado pelo flex. Seguidamente passa-se o ficheiro yacc.y ao YACC que desta forma utilizará o analisador léxico gerado pelo FLEX e fará um analisador sintáctico. Por fim compila-se usando o GCC o ficheiro yy.tab.c, que é gerado pelo YACC, e resulta o executável final do compilador. Utiliza-se o compilador dando como argumento um ficheiro em LogoLISS e será gerado um ficheiro com código assembly para ser usado na máquina virtual.

Funcionamento do compilador em alguns casos.

Para demontrar o funcionamemto do compilador, ao encontrar as diferentes linhas de código passíveis de serem analisadas e transformades em assembly, vamos usar alguns exemplos.

Para começar vamos abordar o exemplo de uma atriubuição.

```
1   x = (25) + y;
```

sendo x e y varíaveis do tipo inteiro, previamente declaradas, e inicializadas. Para começar é colocado o valor de 25 na stack, depois, vamos buscar a posição

de memória da variável y à tabela de identificadores, e fazemos o push da mesma para a stack,fazemos add, e no fim vamos buscar a poição de memória de x, e fazemos store do valor que está no topo da stack, na posição de memória de x.
Agora o exemplo de um turte command.

```
1 FORWARD SUCC x * 25
```

Para começar colocamos os actuais valores da posição da tartaruga no topo da stack, de modo a serem o ponto de partida para o movimento da tartaruga. Depois colocamos o valor da posição em x, ou em y, dependendo da direcção para qual a tartaruga está a olhar, caso seja para cima ou para baixo, fazemos push da posição y, caso seja para a esquerda ou para a direita fazemos pusha da posição x. Depois, sendo x uma variável do tipo inteiro guardada a stack, começamos por fazer push do valor que está na posição de memória de x, depois fazemos push de 1, e de seguida add, assim já temos o sucessor de x. De seguida colocamos o inteiro 25 na stack, e a instrução mul, para ser efectuada a multiplicação. Estando estas expressões tratadas, voltamos a ter em atenção a direcção da tartaruga, para saber se vamos adicionar ou subtrair, última à posição em x/y que está na stack. Depois de verificarmos e tomarmos a opção correcta, ja temos o novo valor x ou y da tartaruga. Consoante a direcção da tartaruga vamos fazer store da nova posição, na posição de memória da componente x ou y da posição da tartaruga. Por fim usamos instrução drawline, que apanha os últimos 4 valores na stack, para desenhar a linha que parte do primeiro ponto (dois primeiros valores que correspondem à posição antiga), e o último (dois últimos valores, que correspondem à nova posição), e refresh para actualziar a janela.

Utilizamos também uma hashtable para servir de nossa tabela de identificadores, recorremos a uma biblioteca já definida, contudo alteramos a estrutura de dados para a uma que se adapta-se ao nosso problema.

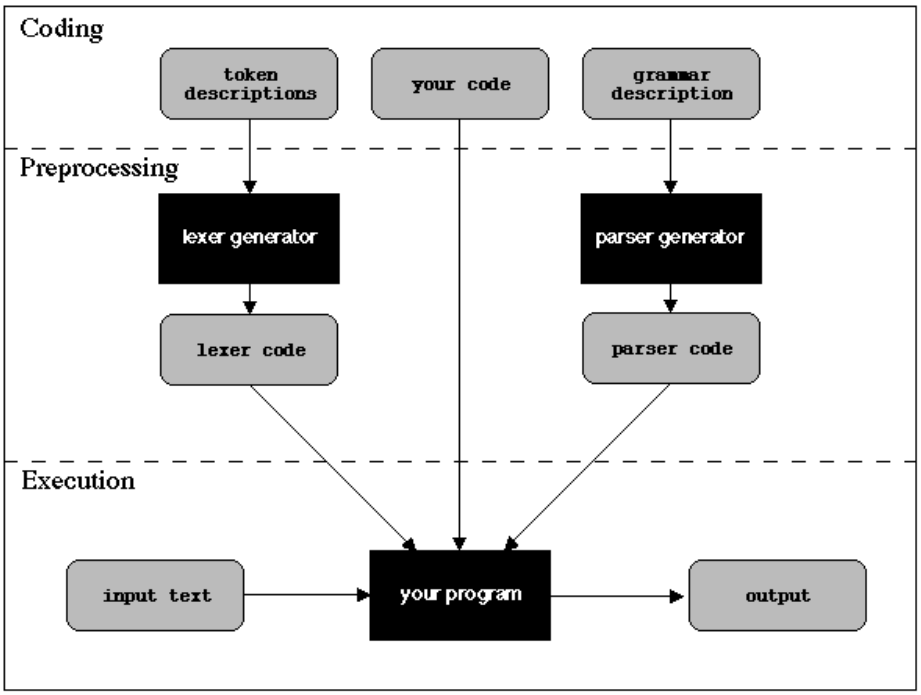

Figura 3.1: Máquina virtual e resultado final da execução do código do exemplo nº 2

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

No fim de passar a gramática para o ficheiro .y decidiu-se por forma a retirar conflitos e simplificar o tratamento da linguagem, fazer algumas alterações à gramática. Começou-se por colocar os números entre parênteses, para evitar conflitos do tipo *shift reduce*, depois decidimos também impossibilitar o simbolo não terminal *Constant* de poder ser uma string, isto leva a que não seja possivel o nosso compilador tratar variáveis do tipo string.
Com o desenvolvimento do trabalho e com o acompanhar das aulas da UC considerou-se que o nosso compilador não irá ser capaz de tratar *Arrays*.
Para ser possível controlar as posições actuais do cursor no programa, depois das *declarations* e antes de se começar os *statments* carregamos para a stack uma variável com a posição de X e outra com a posição de Y, estas ficam no fim da memória reservada para as variáveis globais na stack. Definimos também os simbolos terminais TRUE e FALSE do tipo *vals* que está definido na *union* como tipo char *, declaramos que os simbolos MENORIGUAL, MAIORIGUAL, IGUAL, E, MULTMULT, DIF, OU, +, -, ¡, ¿, * e / têm associatividade à esquerda. Consideramos, além do mais, que os simbolos Sign, Constant,Value_Var e Inic_Var também do tipo *vals*, Var do tipo *par* que é uma estrutura de dados já acima mencionada, concluindo, o simbolo Type ficou definido como sendo um *valn*, que também se encontra declarado na union com sendo um int.
O compilador gerado pelas ferramentas lex e yacc, é capaz de detectar erros sintácticos, ou seja, situações que não respeitem a gramática. Por outro lado é também capaz de detectar erros não previstos pela gramática, tal como, atribuição de um boolean a uma variável do tipo integer. Em qualquer uma destas situações o compilador avisa que encontrou o erro, fornece o tipo do erro, e termina.

4.2 Testes realizados e Resultados

Programa exemplo nº1:

```
PROGRAM logolissExemplo
{
  DECLARATIONS
    x = (100) , y -> Integer ;
    z -> Boolean ;
    w = TRUE -> Boolean ;
  STATEMENTS
    FORWARD x
    RRIGHT
    y = (100)
    FORWARD y
    RRIGHT
    x = x - (100)
    FORWARD x + (100)
    RRIGHT
    FORWARD (100)
}
```

Resultado em Assembly:

```
pushi 0
pushi 10
pushi 1
pushi 1
pushi 200
pushi 450
start
pushi 800
pushi 600
opendrawingarea
pushg 4
pushg 5
pushg 5
pushg 1
sub
storeg 5
pushg 4
pushg 5
drawline
refresh
pushi 10
storeg 0
pushg 4
pushg 5
pushg 4
pushg 0
```

```
27 add
28 storeg 4
29 pushg 4
30 pushg 5
31 drawline
32 refresh
33 pushg 1
34 pushi 10
35 sub
36 storeg 1
37 pushg 4
38 pushg 5
39 pushg 5
40 pushg 1
41 pushi 10
42 add
43 add
44 storeg 5
45 pushg 4
46 pushg 5
47 drawline
48 refresh
49 pushg 4
50 pushg 5
51 pushg 4
52 pushi 10
53 sub
54 storeg 4
55 pushg 4
56 pushg 5
57 drawline
58 refresh
59 stop
```

Resultado depois de executado o ficheiro com o coídgo assembly na máquina virtual:

Programa exemplo nº2:

```
1 program IntegerTest {
2   Declarations
3         intA = (4), intB, intC = (6) -> integer;
4         flag = TRUE -> boolean;
5   Statements
6         intA = (-3) + intC * (7)
7 }
```

Resultado em Assembly:

```
1 pushi 6
2 pushi 0
```

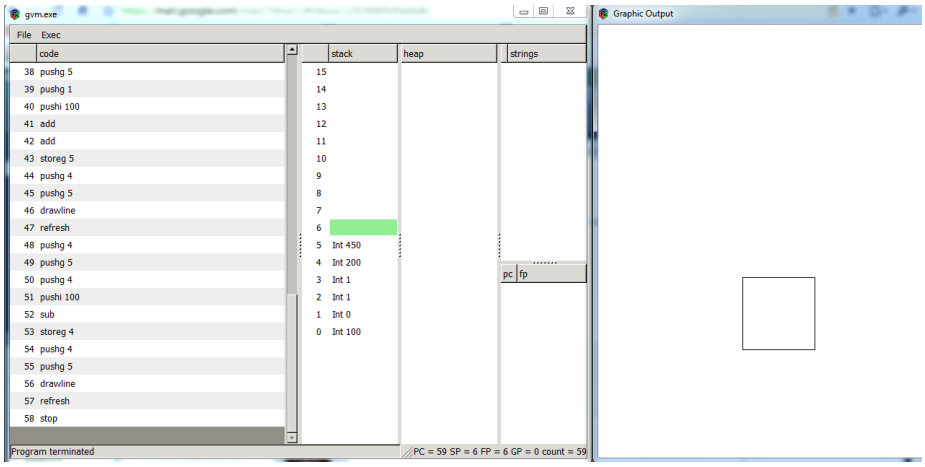

Figura 4.1: Máquina virtual e resultado final da execução do código do exemplo nº 2

```
3  pushi 4
4  pushi 1
5  pushi 200
6  pushi 450
7  start
8  pushi 800
9  pushi 600
10 opendrawingarea
11 pushi -3
12 pushg 0
13 pushi 7
14 mul
15 add
16 storeg 2
17 stop
```

Capítulo 5

Conclusão

Englobando todo o esforço dispendido e os objectivos alcançados relativamente aos que eram propostos, estamos de acordo que conseguimos obter um grau de satisfação bom e que a forma como desenvolvemos o nosso trabalho demonstra que estamos à vontade com a matéria que foi lecionada na UC. Infelizmente, pensamos que poderiamos poder ter conseguido acrescentar bastante mais ao trabalho, contudo devido ao prazo de entrega se aproximar muito das datas de exames e de não haver uma semana de interregno, condicionou bastante a nossa disponibilidade e entrega ao trabalho, desta forma ficaram algumas funcionalidades da linguagem por abranger. Em suma, foi um trabalho motivador e interessante que nos deu uma ideia bastante completa de como podemos interegir o FLEX com o YACC e como gerar código Assembly para a máquina virtual sugerida.

Bibliografia

[1] Site de Processamento de Linguagens, com documentação sobre a Máquina Virtual

[2] Site do projecto que disponibiliza a Máquina Virtual utilizada

[3] Sítio da unidade curricular

[4] Página com documentação do Latex

[5] Guia sobre Lex e Yacc

[6] Imagem que representa a interacção entre o flex, o yacc e o código do programa

Apêndice A

Código do Programa

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

A.1 yacc.y

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

  #define CIMA 0
  #define DIREITA 1
  #define BAIXO 2
  #define ESQUERDA 3

  int stackL=0;
  int stackG=0;

  int dir = CIMA;

  int pousada = 1;

  typedef struct Vars {
    char* nome;
    char* valor;
    struct Vars* next;
  }tVarString;

  tVarString* nodo;
```

```
28
29      typedef struct ParIdentValor {
30        char* nome;
31        char* valor;
32      }ParIdVl;
33
34      typedef struct {
35        int valorInt;
36        char* valorString;
37      }Parconst;
38    %}
39
40
41    %token DECLARATIONS
42    %token STATEMENTS
43    %token <vals>SUCC
44    %token <vals>PRED
45    %token IF
46    %token ELSE
47    %token THEN
48    %token WHILE
49    %token INTEGER
50    %token BOOLEAN
51    %token ARRAY
52    %token SIZE
53    %token <vals>TRUE
54    %token <vals>FALSE
55    %token FORWARD
56    %token BACKWARD
57    %token RRIGHT
58    %token RLEFT
59    %token PEN
60    %token UP
61    %token DOWN
62    %token GOTO
63    %token WHERE
64    %token SAY
65    %token ASK
66    %token OU
67    %token E
68    %token MULTMULT
69    %token IGUAL
70    %token DIF
71    %token MENORIGUAL
72    %token MAIORIGUAL
73    %token IN
74    %token SETA
75    %token PROGRAM
76    %token <valn>NUM
77    %token <vals>STRING
```

```
78  %token <vals>IDENT
79
80  %left '<' '>' MENORIGUAL MAIORIGUAL IGUAL E MULTMULT DIF OU
81  %left <vals>'+' '-'
82  %left '*' '/'
83
84  %type <par>Var
85  %type <parConst>Constant
86  %type <vals>Sign
87  %type <parConst>Value_Var
88  %type <parConst>Inic_Var
89  %type <valn>Type
90  %type <parConst>Factor
91  %type <vals>SuccPred
92  %type <valn>SuccOrPred
93  %type <parConst>Term
94  %type <parConst>Single_Expression
95  %type <parConst>Expression
96  %type <valn>Add_Op
97  %type <valn>Mul_Op
98  %type <vals>Array_Acess
99  %type <vals>Variable
100
101
102 %union{
103   int valn;
104   char *vals;
105   float valr;
106   ParIdVl par;
107   Parconst parConst;
108 }
109
110
111 %start LISS
112
113 %%
114
115 LISS   : PROGRAM IDENT '{' Body '}' {printf("stop\n");}
116          ;
117 Body   : DECLARATIONS Declarations
118                 {
119                   if(insere("posX",0,stackG,0))
120                   {
121                     stackG++;
122                     printf("pushi 200\n");
123                   }
124                   else
125                   {
126                     yyerror("Impossível adicionar Var posX");
127                     exit(0);
```

```
128                     }
129                     if(insere("posY",0,stackG,0))
130                     {
131                       stackG++;
132                       printf("pushi 450\n");
133                     }
134                     else
135                     {
136                       yyerror("Impossível adicionar Var posY");
137                       exit(0);
138                     }
139
140                     printf("start\n");
141                     printf("pushi 800\n");
142                     printf("pushi 600\n");
143                     printf("opendrawingarea\n");
144                   }
145                   STATEMENTS Statements
146           ;
147   Declarations  : Declaration
148                 | Declarations Declaration
149                 ;
150   Declaration   : Variable_Declaration
151           ;
152   Variable_Declaration  : Vars SETA Type ';'
153                   {
154                 tVarString* aux = nodo;
155                 while(aux!=NULL)
156                 {
157                   //printf("%s %d %d %s\n",aux->nome,$3,stackG, aux->valor);
158                   if(lookup(aux->nome) != NULL)
159                   {
160                     yyerror("VAR Duplicadas");
161                     exit(0);
162                   }
163                   else
164                   {
165                     if(!insere(aux->nome,$3,stackG,0))
166                     {
167                       yyerror("SEGMENTATION FAULT");
168                       exit(0);
169                     }
170                     switch($3)
171                     {
172                       case 0 :
173                           if((!strcmp("TRUE",aux->valor)) || (!strcmp("FALSE",aux->
                                valor)))
174                             {
175                               yyerror("Tipo Integer incorrecto");
176                               exit(0);
```

```
177                         }
178                       if(!strcmp(aux->valor,"vazio"))
179                         printf("pushi 0\n");
180                       else
181                         printf("pushi %d\n",atoi(aux->valor));
182                       break;
183                   case 1 :
184                       if(strcmp("TRUE",aux->valor)==0 || !strcmp(aux->valor,"vazio
                              ") )
185                       {
186                         printf("pushi 1\n");
187                       }
188                       else
189                         if(strcmp("FALSE",aux->valor)==0)
190                           printf("pushi 0\n");
191                         else
192                         {
193                           yyerror("Tipo boolean incorrecto");
194                           exit(0);
195                         }
196
197                       break;
198                   }
199                   stackG++;
200                 }
201                 aux = aux->next;
202               }
203
204               nodo = NULL;
205         }
206         ;
207   Vars  : Var
208               {
209           tVarString* aux = (tVarString*)malloc(sizeof(tVarString));
210           aux->nome = $1.nome;
211           aux->valor = $1.valor;
212           aux->next = NULL;
213           nodo=aux;
214         }
215         | Vars ',' Var
216         {
217           tVarString* aux = (tVarString*)malloc(sizeof(tVarString));
218           aux->nome = $3.nome;
219           aux->valor = $3.valor;
220           aux->next = nodo;
221           nodo = aux;
222         }
223         ;
224   Var   : IDENT Value_Var
225               {
```

```
226             $$.nome = $1;
227             if(!strcmp($2.valorString,"vazio"))
228             {
229               $$.valor="vazio";
230             }
231             else
232             {
233               if($2.valorInt==INT_MIN)
234                 $$.valor = $2.valorString;
235               else
236               {
237                 $$.valor = (char*)malloc(sizeof($2.valorInt));
238                 sprintf($$.valor,"%d",$2.valorInt);
239               }
240             }
241           }
242           ;
243   Value_Var :
244             {
245               $$.valorString = "vazio";
246               $$.valorInt = INT_MIN;
247             }
248           | '=' Inic_Var
249             {
250               $$ = $2;
251             }
252           ;
253   Type  : INTEGER {$$=0;}
254           | BOOLEAN {$$=1;}
255           | ARRAY SIZE NUM {$$=2;}
256           ;
257   Inic_Var : Constant
258             {
259             $$ = $1;
260           }
261           | Array_Definition
262           {
263             $$.valorString = "";
264             $$.valorInt = INT_MIN;
265           }
266           ;
267   Constant  : '(' Sign NUM ')'
268             {
269             char *res = (char*)malloc(sizeof($3));
270             if(strcmp($2,"-")==0)
271             {
272               sprintf(res,"-%d",$3);
273               $$.valorInt = atoi(res);
274               $$.valorString ="";
275             }
```

```
276             else
277             {
278               $$.valorInt=$3;
279               $$.valorString ="";
280             }
281           }
282           | TRUE {$$.valorString = $1; $$.valorInt = INT_MIN; }
283           | FALSE {$$.valorString = $1; $$.valorInt = INT_MIN; }
284           ;
285   Sign  : {$$ = "";}
286           | '+' {$$ = $1;}
287           | '-' {$$ = $1;}
288           ;
289   Array_Definition  : '[' Array_Initialization ']'
290                     ;
291   Array_Initialization  : Elem
292                         | Array_Initialization ',' Elem
293                         ;
294   Elem  : Sign NUM
295           ;
296   Statements  : Statement
297               | Statements Statement
298             ;
299   Statement : Turtle_Commands
300             | Assignment
301           | Conditional_Statement
302           | Iterative_Statement
303           ;
304   Turtle_Commands  : Step
305                    | Rotate
306                    | Mode
307                    | Dialogue
308                    | Location
309                    ;
310   Step  : FORWARD
311                   {
312                     node * auxiliar = lookup("posX");
313                     if(auxiliar)
314                       printf("pushg %d\n",auxiliar->address);
315                     auxiliar = lookup("posY");
316                     if(auxiliar)
317                       printf("pushg %d\n",auxiliar->address);
318
319                     node* posXX = lookup("posX");
320                     node* posYY = lookup("posY");
321                     switch(dir){
322                       case(0):printf("pushg %d\n",posYY->address);break;
323                       case(1):printf("pushg %d\n",posXX->address);break;
324                       case(2):printf("pushg %d\n",posYY->address);break;
325                       case(3):printf("pushg %d\n",posXX->address);break;
```

```
326                 default:yyerror("Direcção desconhecida");exit(0);
327               }
328             }
329         Expression
330               {
331                 if(!strcmp($3.valorString,"TRUE") || !strcmp($3.valorString,"FALSE"))
332                 {
333                   yyerror("forward true/false");
334                   exit(0);
335                 }
336                 else
337                 {
338                     node* posXX = lookup("posX");
339                     node* posYY = lookup("posY");
340                     if( !(posXX && posYY) )
341                     {
342                       yyerror("Impossível aceder a posXX ou posYY");
343                       exit(0);
344                     }
345                     switch(dir){
346                       case(0):printf("sub\n");printf("storeg %d\n",posYY->address);
                              break;
347                       case(1):printf("add\n");printf("storeg %d\n",posXX->address)
                              ;break;
348                       case(2):printf("add\n");printf("storeg %d\n",posYY->address)
                              ;break;
349                       case(3):printf("sub\n");printf("storeg %d\n",posXX->address);
                              break;
350                       default:yyerror("Direcção desconhecida");exit(0);
351                     }
352                                     if(pousada){
353                       printf("pushg %d\n",posXX->address);
354                       printf("pushg %d\n",posYY->address);
355                                       printf("drawline\n");
356                       printf("refresh\n");
357                                     }
358                     else
359                     {
360                       printf("pop 2\n");
361                       /*printf("pushg %d\n",posXX->address);
362                       printf("pushg %d\n",posYY->address);
363                       printf("drawpoint\n");
364                       printf("refresh\n");*/
365                     }
366                 }
367               }
368           | BACKWARD
369               {
370                 node * auxiliar = lookup("posX");
371                 if(auxiliar)
```

```
372                 printf("pushg %d\n",auxiliar->address);
373               auxiliar = lookup("posY");
374               if(auxiliar)
375                 printf("pushg %d\n",auxiliar->address);
376
377               node* posXX = lookup("posX");
378               node* posYY = lookup("posY");
379               switch(dir){
380                 case(0):printf("pushg %d\n",posYY->address);break;
381                 case(1):printf("pushg %d\n",posXX->address);break;
382                 case(2):printf("pushg %d\n",posYY->address);break;
383                 case(3):printf("pushg %d\n",posXX->address);break;
384                 default:yyerror("Direcção desconhecida");exit(0);
385               }
386             }
387 Expression
388             {
389               if(!strcmp($3.valorString,"TRUE") || !strcmp($3.valorString,"FALSE"))
390               {
391                 yyerror("forward true/false");
392                 exit(0);
393               }
394               else
395               {
396                 if(pousada)
397                 {
398                   node* posXX = lookup("posX");
399                   node* posYY = lookup("posY");
400                   switch(dir){
401                     case(0):printf("add\n");printf("storeg %d\n",posYY->address)
                            ;break;
402                     case(1):printf("sub\n");printf("storeg %d\n",posXX->address);
                            break;
403                     case(2):printf("sub\n");printf("storeg %d\n",posYY->address);
                            break;
404                     case(3):printf("add\n");printf("storeg %d\n",posXX->address)
                            ;break;
405                     default:break;
406                   }
407                                     if(pousada){
408                   printf("pushg %d\n",posXX->address);
409                   printf("pushg %d\n",posYY->address);
410                                       printf("drawline\n");
411                   printf("refresh\n");
412                                     }
413                   else
414                   {
415                     printf("pop 2\n");
416                     /*printf("pushg %d\n",posXX->address);
417                     printf("pushg %d\n",posYY->address);
```

```
418                         printf("drawpoint\n");
419                         printf("refresh\n");*/
420                       }
421                     }
422                   }
423                 }
424       ;
425 Rotate  : RRIGHT { //dir = (dir++)%4;
426               if((dir+1)>3)
427                 dir=0;
428               else
429                 dir++;
430             }
431       | RLEFT
432           {
433             if((dir-1)<0)
434               dir = 3;
435             else
436               dir--;
437           }
438       ;
439 Mode    : PEN UP {pousada=0;}
440         | PEN DOWN {pousada=1;}
441         ;
442 Dialogue  : Say_Statement
443           | Ask_Statement
444         ;
445 Location  : GOTO NUM ',' NUM
446               {
447                 node* posXX = lookup("posX");
448                 node* posYY = lookup("posY");
449                 printf("pushi %d\n",$2);
450                 printf("storeg %d\n",posXX->address);
451                 printf("pushi %d\n",$4);
452                 printf("storeg %d\n",posYY->address);
453               }
454           | WHERE '?'
455               {
456                 node* posXX = lookup("posX");
457                 node* posYY = lookup("posY");
458                 printf("pushg %d\n",posXX->address);
459                 printf("writei\n");
460                 printf("pushg %d\n",posYY->address);
461                 printf("writei\n");
462               }
463         ;
464 Assignment  : Variable '=' Expression
465               {
466                 node* var = lookup($1);
467                 if(!var)
```

```
468                     {
469                       yyerror("Variável desconhecida");
470                       exit(0);
471                     }
472
473                     if(!strcmp($3.valorString,""))
474                       if(! (($3.valorInt != INT_MIN) && (var->tipo==0)) )
475                       {
476                         yyerror("Conflito de tipos");
477                         exit(0);
478                       }
479
480                     if($3.valorInt==INT_MIN)
481                     {
482                     if(! (((!strcmp($3.valorString,"TRUE")) || (!strcmp($3.valorString,"
                            FALSE"))) && (var->tipo==1)) )
483                     {
484                       node* var2 = lookup($3.valorString);
485                       if(! ((!strcmp($3.valorString,"TRUE")) || (!strcmp($3.valorString,"
                              FALSE"))) )
486                       {
487                         if(!var2)
488                         {
489                           //char* nome = (char*)malloc(sizeof(strlen(var2->name)+40))
                                  ;
490                           //sprintf(nome,"Variável desconhecida: %s",var2->name);
491                           yyerror("Variável desconhecida");
492                           exit(0);
493                         }
494                         if(!(var2->tipo == var->tipo))
495                         {
496                           yyerror("Conflito de tipos");
497                           exit(0);
498                         }
499                       }
500                       else
501                       {
502                         yyerror("Conflito de tipos");
503                         exit(0);
504                       }
505                     }
506                     }
507
508                     if(var->stackgl)
509                       printf("storel %d\n",var->address);
510                     else
511                       printf("storeg %d\n",var->address);
512                   }
513               ;
514   Variable  : IDENT Array_Acess
```

```
515                 {
516                   if(!strcmp($2,""))
517                   {
518                     yyerror("Single_Expression Variable");
519                     exit(0);
520                   }
521                   $$=$1;
522                 }
523             ;
524 Array_Acess  :       { $$="vazia";}
525         | '[' Single_Expression ']' { $$="";}
526         ;
527 Expression  : Single_Expression {$$=$1;}
528             | Expression Rel_Op Single_Expression
529         ;
530 Single_Expression : Term {$$=$1;}
531               | Single_Expression Add_Op Term
532             {
533               switch($2)
534               {
535                 case 0 : printf("add\n");break;
536                 case 1 : printf("sub\n");break;
537                 case 2 : yyerror("OU add_op");exit(0);
538                 default: yyerror("Operador indefinido add_op");exit(0);
539               }
540             }
541             ;
542 Term   : Factor {$$=$1;}
543               | Term Mul_Op Factor
544               {
545                 switch($2)
546                 {
547                   case 0 : printf("mul\n");break;
548                   case 549 : printf("div\n");break;
549                   case 2 : yyerror("E Mul_Op");exit(0);
550                   case 3 : {
551                         printf("pop 1\n");
552                         int aux=$3.valorInt-1;
553                         while(aux>0)
554                         {
555                           printf("dup 1\n");
556                           aux--;
557                         }
558                         aux=($3.valorInt)-1;
559                         while(aux>0)
560                         {
561                           printf("mul\n");
562                           aux--;
563                         }
564                         break;
```

```
565                         }
566                     default: yyerror("Operador indefinido add_op");exit(0);
567                   }
568                 }
569           ;
570   Factor  : Constant {$$ = $1;
571                     if(strcmp($1.valorString,"TRUE")==0)
572                       printf("pushi 1\n");
573                     else
574                       if(strcmp($1.valorString,"FALSE")==0)
575                         printf("pushi 0\n");
576                       else
577                         printf("pushi %d\n",$1.valorInt);
578                   }
579         | Variable
580             {
581                   $$.valorInt = INT_MIN;
582                   $$.valorString =$1;
583                   node* var = lookup($1);
584                   if(!var)
585                   {
586                     yyerror("Variável desconhecida");
587                     exit(0);
588                   }
589                   if(var->stackgl)
590                     printf("pushl %d\n",var->address);
591                   else
592                     printf("pushg %d\n",var->address);
593                 }
594         | SuccOrPred {
595                     $$.valorInt = INT_MIN;
596                     $$.valorString="";
597                   }
598         | '(' '!' Expression ')' {$$.valorInt = 1;$$.valorString ="";}
599         | '(' '+' Expression ')' {$$.valorInt = 1;$$.valorString ="";}
600         | '(' '-' Expression ')' {$$.valorInt = 1;$$.valorString ="";}
601         | '(' Expression ')' {$$.valorInt = 1;$$.valorString ="";}
602         ;
603   Add_Op  : '+' { $$=0;}
604         | '-' { $$=1;}
605         | OU { $$=2;}
606         ;
607   Mul_Op  : '*'   { $$=0;}
608         | '/'   { $$=1;}
609         | E   { $$=2;}
610         | MULTMULT { $$=3;}
611         ;
612   Rel_Op  : IGUAL
613         | DIF
614         | '<'
```

```
615         | '>'
616         | MENORIGUAL
617         | MAIORIGUAL
618         | IN
619         ;
620 SuccOrPred  : SuccPred IDENT
621           {
622             node* x = lookup($2);
623             if(x==NULL)
624             {
625               yyerror("Variavél não existente.");
626               exit(0);
627             }
628             else
629             {
630
631               if(strcmp($1,"PRED")==0)
632               {
633                 if(x->stackgl==0)
634                   printf("pushg %d\n", x->address);
635                 else
636                   printf("pushl %d\n", x->address);
637                 printf("pushi 1\n");
638                 printf("sub");
639
640               }
641               else
642               {
643                 if(x->stackgl==0)
644                   printf("pushg %d\n", x->address);
645                 else
646                   printf("pushl %d\n", x->address);
647                 printf("pushi 1\n");
648                 printf("add\n");
649               }
650             }
651           }
652             ;
653 SuccPred  : SUCC {$$=$1;}
654           | PRED {$$=$1;}
655           ;
656 Say_Statement : SAY '(' Expression ')'
657               ;
658 Ask_Statement : ASK '(' STRING ',' Variable ')'
659               ;
660 Conditional_Statement : IfThenElse_Stat
661                       ;
662 Iterative_Statement : While_Stat
663               ;
664 IfThenElse_Stat  : IF Expression THEN '{' Statements '}' Else_Expression
```

```
665             ;
666 Else_Expression  :
667             | ELSE '{' Statements '}'
668             ;
669 While_Stat  : WHILE '(' Expression ')' '{' Statements '}'
670             ;
671 %%
672
673 #include "lex.yy.c"
674
675
676 int yyerror(char *s)
677 {
678   fprintf(stderr,"ERRO: %s \n",s);
679   return 0;
680 }
681
682
683 int main()
684 {
685   nodo = NULL;
686   inithashtab();
687   yyparse();
688   return(0);
689 }
```

A.2 lex.l

```
1  PROGRAM  [Pp][Rr][Oo][Gg][Rr][Aa][Mm]
2  DECLARATIONS  [Dd][Ee][Cc][Ll][Aa][Rr][Aa][Tt][Ii][Oo][Nn][Ss]
3  STATEMENTS  [Ss][Tt][Aa][Tt][Ee][Mm][Ee][Nn][Tt][Ss]
4  SUCC  [Ss][Uu][Cc][Cc]
5  PRED  [Pp][Rr][Ee][Dd]
6  IF  [Ii][Ff]
7  ELSE  [Ee][Ll][Ss][Ee]
8  THEN  [Tt][Hh][Ee][Nn]
9  WHILE  [Ww][Hh][Ii][Ll][Ee]
10 INTEGER  [Ii][Nn][Tt][Ee][Gg][Ee][Rr]
11 BOOLEAN  [Bb][Oo][Oo][Ll][Ee][Aa][Nn]
12 ARRAY  [Aa][Rr][Rr][Aa][Yy]
13 SIZE  [Ss][Ii][Zz][Ee]
14 TRUE  [T][R][U][E]
15 FALSE  [F][A][L][S][E]
16 FORWARD  [Ff][Oo][Rr][Ww][Aa][Rr][Dd]
17 BACKWARD [Bb][Aa][Cc][Kk][Ww][Aa][Rr][Dd]
18 RRIGHT  [Rr][Rr][Ii][Gg][Hh][Tt]
19 RLEFT  [Rr][Ll][Ee][Ff][Tt]
```

```
20  PEN  [Pp][Ee][Nn]
21  UP  [Uu][Pp]
22  DOWN  [Dd][Oo][Ww][Nn]
23  GOTO  [Gg][Oo][Tt][Oo]
24  WHERE  [Ww][Hh][Ee][Rr][Ee]
25  SAY  [Ss][Aa][Yy]
26  ASK  [Aa][Ss][Kk]
27  OU  "||"
28  E  "&&"
29  MULTMULT "**"
30  IGUAL  "=="
31  DIF  "!="
32  MENORIGUAL "<="
33  MAIORIGUAL ">="
34  IN  [Ii][Nn]
35  SETA  "->"
36  NUM  [0-9]+
37  LETRA  [a-zA-Z]
38  STRING  \"[^"]+\"
39  IDENT  [a-zA-Z][a-zA-Z0-9]*
40
41  %%
42
43  {PROGRAM}  {return(PROGRAM);}
44  {DECLARATIONS}  {return(DECLARATIONS);}
45  {STATEMENTS}  {return(STATEMENTS);}
46  {SUCC}  {yylval.vals=(char*)strdup(yytext);return(SUCC);}
47  {PRED}  {yylval.vals=(char*)strdup(yytext);return(PRED);}
48  {IF}  {return(IF);}
49  {ELSE}  {return(ELSE);}
50  {THEN}  {return(THEN);}
51  {WHILE}  {return(WHILE);}
52  {INTEGER}  {return(INTEGER);}
53  {BOOLEAN}  {return(BOOLEAN);}
54  {ARRAY}  {return(ARRAY);}
55  {SIZE}  {return(SIZE);}
56  {TRUE}  {yylval.vals=(char*)strdup(yytext);return(TRUE);}
57  {FALSE}  {yylval.vals=(char*)strdup(yytext);return(FALSE);}
58  {FORWARD}  {return(FORWARD);}
59  {BACKWARD}  {return(BACKWARD);}
60  {RRIGHT}  {return(RRIGHT);}
61  {RLEFT}  {return(RLEFT);}
62  {PEN}  {return(PEN);}
63  {UP}  {return(UP);}
64  {DOWN}  {return(DOWN);}
65  {GOTO}  {return(GOTO);}
66  {WHERE}  {return(WHERE);}
67  {SAY}  {return(SAY);}
68  {ASK}  {return(ASK);}
69  {OU}  {return(OU);}
```

```
70  {E}   {return(E);}
71  {MULTMULT}  {return(MULTMULT);}
72  {IGUAL}   {return(IGUAL);}
73  {DIF}   {return(DIF);}
74  {MENORIGUAL}  {return(MENORIGUAL);}
75  {MAIORIGUAL}  {return(MAIORIGUAL);}
76  {IN}   {return(IN);}
77  {SETA}   {return(SETA);}
78  [+−]   {yylval.vals=(char∗)strdup(yytext);return(yytext[0]);}
79  [,;<>]   {return(yytext[0]);}
80  [?\[\]()/!∗]  {return(yytext[0]);}
81  [{]   {return(yytext[0]);}
82  [}]   {return(yytext[0]);}
83  [=]   {return(yytext[0]);}
84  {NUM}   {yylval.valn=atoi(yytext);return(NUM);}
85  {STRING}  {yylval.vals=(char∗)strdup(yytext);return(STRING);}
86  {IDENT}   {yylval.vals=(char∗)strdup(yytext);return(IDENT);}
87  " "|\n|\t  {;}
88
89
90  %%
91
92  int yywrap()
93  { return(1); }
```

A.3 hashtable.c

```
1   #include <string.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include "hashtable.h"
5
6   static node∗ hashtab[HASHSIZE];
7
8   void inithashtab(){
9     int i;
10    for(i=0;i<HASHSIZE;i++)
11      hashtab[i]=NULL;
12  }
13
14  unsigned int hash(char ∗s){
15    unsigned int h=0;
16    for(;∗s;s++)
17      h=∗s+h∗31;
18    return h%HASHSIZE;
19  }
20
```

```
21  node* lookup(char *n){
22    unsigned int hi=hash(n);
23    node* np=hashtab[hi];
24    for(;np!=NULL;np=np->next){
25      if(!strcmp(np->name,n))
26        return np;
27    }
28
29    return NULL;
30  }
31
32  char* m_strdup(char *o){
33    int l=strlen(o)+1;
34    char *ns=(char*)malloc(l*sizeof(char));
35    strcpy(ns,o);
36    if(ns==NULL)
37      return NULL;
38    else
39      return ns;
40  }
41
42  int getTipo(char* name){
43    node* n=lookup(name);
44    if(n==NULL)
45      return -1;
46    else
47      return n->tipo;
48  }
49
50
51  int getAddress(char* name){
52    node* n=lookup(name);
53    if(n==NULL)
54      return -1;
55    else
56      return n->address;
57  }
58
59  int insere(char* name,int tipo,int address,int stackgl){
60    unsigned int hi;
61    node* np;
62    if((np=lookup(name))==NULL){
63      hi=hash(name);
64      np=(node*)malloc(sizeof(node));
65      if(np==NULL)
66        return 0;
67      np->name=m_strdup(name);
68      if(np->name==NULL) return 0;
69      np->tipo = tipo;
70      np->address = address;
```

```
71        np->next=hashtab[hi];
72        np->stackgl=stackgl;
73        hashtab[hi]=np;
74      }
75      else
76      np->tipo=tipo;
77      np->address=address;
78      np->stackgl=stackgl;
79      if(np->tipo==-1) return 0;
80
81      return 1;
82    }
83
84    void cleanup(){
85      int i;
86      node *np,*t;
87      for(i=0;i<HASHSIZE;i++){
88        if(hashtab[i]!=NULL){
89          np=hashtab[i];
90          while(np!=NULL){
91            t=np->next;
92            free(np->name);
93            free(np);
94            np=t;
95          }
96        }
97      }
98    }
```

A.4 hashtable.h

```
1   typedef struct _node{
2     char *name;
3     int tipo;
4     int address;
5     int stackgl;
6     struct _node *next;
7   }node;
8
9   #define HASHSIZE 101
10
11  void inithashtab();
12  unsigned int hash(char *s);
13  node* lookup(char *n);
14  char* m_strdup(char *o);
15  int getTipo(char* name);
16  int getAddress(char* name);
```

```
17 int insere(char* name,int tipo,int address,int stackgl);
18 void cleanup();
```