Processamento de Linguagens

Compilador para a Linguagem LogoLISS

54776 - Rafael Abreu
54789 - Bruno Azevedo
54821 - Vítor Costa

5 de Junho de 2011

Conteúdo

1 Introdução

Na disciplina de Processamento de Linguagens foi-nos proposto a realização de um trabalho prático utilizando a ferramenta Yacc. Tem como objectivo a prática de manipulação de uma linguagem a conhecendo a sua gramática. Neste relatório está explicado todo o procedimento para o seu desenvolvimento e o resultado final.

2 Escolha do Enunciado

Neste segundo trabalho, apenas tínhamos um enunciado disponível, não havendo opções de escolha. Este enunciado propunha-nos a criação de um interpretador léxico para a linguagem LogoLISS, e disponibilizava-nos a sua gramática na íntegra. Mais tarde, foi-nos disponibilizado um segundo enunciado, em que a linguagem que tratava era bastante mais simples que o LogoLISS, mas, para equilibrar o nível de dificuldade, não nos disponibilizava o gramática.

Optámos pelo primeiro enunciado pela simples razão de já estarmos dentro do problema e mais ambientados com ele do que com o enunciado alternativo.

3 Desenvolvimento do Programa

A primeira grande dificuldade com que nos deparamos (e que se foi revelando ocasionalmente durante o desenvolvimento do trabalho) foi: "O que é o LogoLISS? O que é que é suposto determinado comando fazer? Como é que vou conseguir guardar esta informação de modo a que a VM consiga aceder e trabalhar com bons resultados?". Foi este tipo de obstáculos que tivemos de superar, ambientando-nos no mundo do LogoLISS e da máquina virtual (VM), principalmente à sua parte gráfica (GVM).

Para que conseguissemos enfrentar tal barreiras e ultrapassa-las com sucesso, decidimos implementar o seguinte:

3.1 O Ficheiro FLEX e o ficheiro YACC

Para dar início ao trabalho prático, começamos por implementar a sua base: os ficheiro FLEX e YACC. De seguida, encontra-se o principal dessa base:

```
{PROGRAM}               { return PROGRAM; }
{DECLARATIONS}          { return DECLARATIONS; }
{STATEMENTS}            { return STATEMENTS; }
"->"                    { return ARROW; }
{INTEGER}               { return INTEGER; }
{BOOLEAN}               { return BOOLEAN; }
{STRING}                { return STRING; }
{ARRAY}                 { /*return ARRAY;*/ }
{SIZE}                  { /*return SIZE;*/ }
{TRUE}                  { yylval.stringvalue=(char*)strdup(yytext);
                          return TRUE; }
{FALSE}                 { yylval.stringvalue=(char*)strdup(yytext);
                          return FALSE; }
{FORWARD}               { return FORWARD; }
{BACKWARD}              { return BACKWARD; }
{RRIGHT}                { return RRIGHT; }
{RLEFT}                 { return RLEFT; }
{PEN}                   { return PEN; }
{UP}                    { return UP; }
{DOWN}                  { return DOWN; }
{GOTO}                  { return GOTO; }
{WHERE}                 { return WHERE; }
```

```
"||"                        { return OR; }
"&&"                        { return AND; }
"**"                        { return POW; }
"=="                        { return EQUAL; }
"!="                        { return DIF; }
"<"                         { return MINOR; }
">"                         { return MAJOR; }
"<="                        { return MINOREQUAL; }
">="                        { return MAJOREQUAL; }
{IN}                        { return IN; }
[=\-,;!?/()\[\]\{\}*+]      {return(yytext[0]);}
{SUCC}                      { yylval.stringvalue=(char*)strdup(yytext);
                              return SUCC; }
{PRED}                      { yylval.stringvalue=(char*)strdup(yytext);
                             return PRED; }
{SAY}                       { return SAY; }
{ASK}                       { return ASK; }
{IF}                        { return IF; }
{THEN}                      { return THEN; }
{ELSE}                      { return ELSE; }
{WHILE}                     { return WHILE; }

{NUMBER}                    { yylval.stringvalue = strdup(yytext);
                              return NUMBER; }
{IDENTIFIER}                { yylval.stringvalue = strdup(yytext);
                              return IDENTIFIER; }
{STR}                       { yylval.stringvalue = strdup(yytext);
                              return STR; }

[\ \n\t\r]+                 {;}

<*>.|\n                     { fprintf(stderr,"ERRO: %d '%s'\n",yylineno, yytext);}
```

Aqui está representado como é que o nosso programa consegue corresponder determinado comando a uma determinada acção: caso encontre qualquer coia que esteja do lado esquerdo, vai retornar o que está do lado direito. Esse retorno irá ser utilizado no ficheiro YACC:

```
%token PROGRAM DECLARATIONS STATEMENTS ARROW
%token INTEGER BOOLEAN STRING FORWARD BACKWARD
%token RRIGHT RLEFT
```

```
%token PEN UP DOWN GOTO WHERE OR AND POW EQUAL
%token DIF MINOREQUAL MAJOREQUAL IN SAY ASK IF
%token THEN ELSE WHILE
%token <stringvalue>TRUE FALSE IDENTIFIER
%token UMBER STR SUCC PRED
```

O que aqui está representado é a passagem de um comando para o seu respectivo "token". Com esta transformação, conseguimos manipular o comando da forma que quisermos.

De seguida, apresentamos as estruturas de dados que utilizamos no nosso trabalho prático:

3.2 Estruturas de Dados

Para armazenarmos a informação relevante à medida que o nosso compilador percorre o programa, criamos várias estruturas de dados que nos ajudam nessa tarefa.

De seguida, está uma pequena explicação de cada uma dessas estruturas de dados (ver código em anexo):

3.2.1 VarHashTable

A máquina virtual, sendo uma máquina de pilhas que vai acumulando valores de diversos tipos, precisa de um mecanismo para associar, em runtime, esses valores a um tipo, um identificador e o respectivo endereço na pilha.

Decidimos que tal mecanismo seria composto por uma tabela de hash que a cada identificador associa um nodo da tabela. Cada nodo contem a informação enunciada acima, o próprio identificador, o tipo do valor ao qual o identificador está associado e a endereço na stack onde esse valor foi alocado.

Esta tabela, permite, em runtime, verificar se determinado identificador existe, e se existir, obtém informação suficiente para poder aceder ao seu valor na stack. Também poderá validar os valores de input verificando o tipo do identificador.

As funções associadas à tabela de hash correspondem às funcionalidades básicas de inserção e procura:

initHash() inicializa a tabela de hash;

hash() função de hash que cria uma chave dado um identificador;

searchVar(char* id) procura a variável na tabela e retorna caso exista;

insertVar(char* id, int type, int address) insere uma nova variável na tabela hash.

3.2.2 VarTipo

Estrutura que no decorrer do parsing, quando encontra uma variável, guarda o identificador, o valor e o tipo de dados.

3.2.3 ConstTipo

Estrutura que no decorrer do parsing, quando encontra uma constante, guarda o valor e o tipo. Tipo este que pode representar valor nulo.

3.2.4 ListaVars

Lista ligada que, no decorrer do parsing, na Declaração de Variáveis, guarda as variáveis por declaração. Esta lista ligada é posteriormente utilizada para armazenar as variáveis na tabela de hash.

3.3 Expressions

Por uma questão de simplificação, uma vez que as "Expressions" são expressões com um valor associado, declaramos que estes estados nada mais são que inteiros, que corresponde ao seu valor.

Basicamente, percorre a sua árvore de derivação até aos comandos base (adição, multiplicação de factores, etc.), calcula o seu valor e retorna-o.

Este estado é utilizado nos mais variáveis comandos LogoLISS, desde comandos da Tartaruga até uma simples atribuição.

3.4 Declarations

O Declarations: Variables percorre as declarações de variáveis e por cada declaração vai adicionando as variáveis a uma lista ligada de variáveis. Cada nodo da lista ligada corresponde a uma estrutura de dados que guarda as informações de cada variável, o identificador, o valor e o tipo de dados.Chegando ao fim de uma declaração, as variáveis contidas na lista ligada

são armazenadas na tabela de hash, os respectivos valores são "empilhados"na stack e a lista ligada é reinicializada para guardar novas variáveis.

3.5 Statements

O Assignment Statement, por cada Assignment encontrado, calcula o valor da expressão (Expression) e coloca no topo da pilha. De seguida, o endereço do variável (Variable) é retornado da tabela de hash para que a atribuição do resultado da expressão à variável, mais especificamente, à posição de memória da stack associada à variável seja possível.

3.6 Os Comandos da Turtle

Este era o principal objectivo do trabalho: fazer um programa em LogoLISS capaz de criar uma tartaruga e capaz de a manipular, deslocando-a em várias direcções e diferentes sentidos.

Estes comandos representam todos os comandos que podemos fazer com a tartaruga (que no nosso projeto significa uma circunferência com centro na posição (300, 200), com raio de 25, desenhado num ecrã 600x800, direcionada para cima (direcção = up) e com o rasto activo (mode = 1)). Os comandos implementados foram:

3.6.1 Setp

Este comando subdivide-se em 2 comandos, o FORWARD e o BACKWARD. Em primeiro lugar, a partir do endereço das coordenadas, vai buscar a posição actual da tartaruga e guarda-a na stack, testa qual é a direcção ("up", "right", "down"ou "left") dada há tartaruga. Dependendo da rotação soma ou subtrai o valor da "expression"na respectiva cordenada (x,y) da tartaruga e acualiza a nova posição da tartaruga. Desenha o seu rasto atráves das antigas e novas coordenadas calculadas, caso solicitado, e só por fim desenha a nova posição da tartaruga.

3.6.2 Rotate

Representa as duas rotações possíveis dadas há tartaruga dependendo da actual direcção que tem. Subdivide-se em dois comandos diferentes: o RRIGhT e o RLEFT. O que eles fazem é o seguinte: dependendo da direcção actual da tartaruga e da rotação dada calcula-se e atualiza-se a nova

direcção na tartaruga.

3.6.3 Mode

representa os dois modos de desenho do rastro, para representar o movimento (Step) da tartaruga:PEN UP e PENDOWN. O que fazem é alterar o "mode" para 0 ou 1 caso seja para desenhar ou nao o rastro da tartaruga.

3.6.4 Dialogue

Mostra os comandos de diálogo ("say"e "ask") que podemos aplicar à tartaruga: Say_Statement, que escreve uma "expression", e Ask_Statement, que escreve uma string e guarda numa variável o que foi lido do teclado.

Por sua vez, o Say_Statement divide-se no comando SAY (Expression), que calcula a "Expression" e coloca-a na stack consoante o tipo ("INTEGER", "BOOLEAN" e "STRING") da "Expression" e imprime a respectiva expression; o Ask_Statement divide-se no comando ASK (STR , Variable), que, em caso de a "Variable" nao estar declarada, dá um erro, senão guarda na stack e mostra a string (STR), depois lê uma string do teclado (concluída por um enter), arquiva esta string na heap e o endereço na satck. Converte o valor lido do teclado consoante o tipo da "Variable" e guarda esse valor na respectiva váriavel.

3.6.5 Location

Mostra ou edita a localização da tartaruga. Subdivide-se em GOTO NUMBER , NUMNBER, que em primeiro lugar, a partir do endereço das coordenadas, vai buscar a posição actual da tartaruga e guarda-a na stack para depois poder desenhar a linha de rasto. Em seguida, arquiva e actualiza os valores das novas coordenadas (NUMBER's) da tartaruga. Desenha o rasto da tartaruga, atráves das antigas e novas coordenadas calculadas, caso solicitado e só por fim desenha a nova posição da tartaruga na posição pretendida; subdivide-se também no WHERE ?, que em primeiro lugar, a partir do endereço das coordenadas, vai buscar a posição actual da tartaruga e imprime em primeiro lugar a coordenada x e depois a coordenada y.

4 Conclusão

Ao iniciar esta conclusão, assumida pela equipa responsável por este relatório sobre a realização de um compilador gerando código máquina de stack vitual para a linguagem "LogoLISS - A Toy Language"como uma reflexão crítica, torna-se importante diferenciar o processo de desenvolvimento do resultado final.

Como processo de desenvolvimento consideramos que o grupo de trabalho teve muitas dificuldades em interpretar o enunciado por falta de conhecimento do grupo sobre o funcionamento duma máquina de pilhas.

Tirando o factores acima referido, consideramos que o processo de desenvolvimento decorreu de uma forma bastante boa e eficaz. Como produto final, este responde, quase na totalidade, aquilo que foi pedido no enunciado pela interpretação que efectuamos. Infelizmente, não conseguimos implementar tudo o que deveriamos, tal como condições if, ciclos while, muito devido à pressão que o 3º e último (esperamos) ano da nossa licenciatura nos proporciona.

A Anexos

```
++++++++++++++++++   logo.l   ++++++++++++++++++++

%{
#include "structures.h"
#include "hashFunctions.h"
#include "y.tab.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

%}

%option yylineno

PROGRAM  [Pp][Rr][Oo][Gg][Rr][Aa][Mm]
DECLARATIONS  [Dd][Ee][Cc][Ll][Aa][Rr][Aa][Tt][Ii][Oo][Nn][Ss]
STATEMENTS  [Ss][Tt][Aa][Tt][Ee][Mm][Ee][Nn][Tt][Ss]
SUCC  [Ss][Uu][Cc][Cc]
PRED  [Pp][Rr][Ee][Dd]
IF  [Ii][Ff]
ELSE [Ee][Ll][Ss][Ee]
THEN [Tt][Hh][Ee][Nn]
WHILE [Ww][Hh][Ii][Ll][Ee]
INTEGER [Ii][Nn][Tt][Ee][Gg][Ee][Rr]
BOOLEAN [Bb][Oo][Oo][Ll][Ee][Aa][Nn]
STRING [Ss][Tt][Rr][Ii][Nn][Gg]
ARRAY [Aa][Rr][Rr][Aa][Yy]
SIZE [Ss][Ii][Zz][Ee]
TRUE [T][R][U][E]
FALSE [F][A][L][S][E]
FORWARD [Ff][Oo][Rr][Ww][Aa][Rr][Dd]
BACKWARD [Bb][Aa][Cc][Kk][Ww][Aa][Rr][Dd]
RRIGHT [Rr][Rr][Ii][Gg][Hh][Tt]
RLEFT [Rr][Ll][Ee][Ff][Tt]
PEN [Pp][Ee][Nn]
UP [Uu][Pp]
```

```
DOWN [Dd][Oo][Ww][Nn]
GOTO [Gg][Oo][Tt][Oo]
WHERE [Ww][Hh][Ee][Rr][Ee]
SAY [Ss][Aa][Yy]
ASK [Aa][Ss][Kk]
IN [Ii][Nn]

NUMBER  [0-9]+
IDENTIFIER  [a-zA-Z][a-zA-Z0-9]*
STR \"([^"\n]|\\\")*\"

%%

{PROGRAM} { return PROGRAM; }
{DECLARATIONS} { return DECLARATIONS; }
{STATEMENTS} { return STATEMENTS; }
"->" { return ARROW; }
{INTEGER} { return INTEGER; }
{BOOLEAN} { return BOOLEAN; }
{STRING} { return STRING; }
{ARRAY} { /*return ARRAY;*/ }
{SIZE} { /*return SIZE;*/ }
{TRUE} { return TRUE; }
{FALSE} { return FALSE; }
{FORWARD} { return FORWARD; }
{BACKWARD} { return BACKWARD; }
{RRIGHT} { return RRIGHT; }
{RLEFT} { return RLEFT; }
{PEN} { return PEN; }
{UP} { return UP; }
{DOWN} { return DOWN; }
{GOTO} { return GOTO; }
{WHERE} { return WHERE; }
"||" { return OR; }
"&&" { return AND; }
"**" { return POW; }
"==" { return EQUAL; }
"!=" { return DIF; }
"<" { return MINOR; }
">" { return MAJOR; }
"<=" { return MINOREQUAL; }
```

```
">=" { return MAJOREQUAL; }
{IN} { return IN; }
[=\-,;!?/()\[\]\{\}*+] {return(yytext[0]);}
{SUCC} { return SUCC; }
{PRED} { return PRED; }
{SAY} { return SAY; }
{ASK} { return ASK; }
{IF} { return IF; }
{THEN} { return THEN; }
{ELSE} { return ELSE; }
{WHILE} { return WHILE; }

{NUMBER} { yylval.stringvalue = strdup(yytext); return NUMBER; }
{IDENTIFIER} { yylval.stringvalue = strdup(yytext); return IDENTIFIER; }
{STR} { yylval.stringvalue = strdup(yytext); return STR; }

[\ \n\t\r]+ {;}

<*>.|\n { fprintf(stderr,"ERRO: %d '%s'\n",yylineno, yytext);}

%%

int yywrap() {
 return 1;
}

/*int yyerror(char *s){
fprintf(stderr,"ERRO: %d %s\n",yylineno, s);
return 0;
}*/
```

++++++++++++++++++++ logo.y ++++++++++++++++++++++++

```
%{
```

```
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include "structures.h"
#include "hashFunctions.h"

int addressG = 0;
int height, width, xpos, ypos, raio;
Direccao direccao;
int mode; // PEN UP

extern char* yytext;
extern int yylineno;

extern ListaVars *nodo;
extern VarHashTable varHashTable;

%}
%error-verbose

%token PROGRAM DECLARATIONS STATEMENTS ARROW
%token INTEGER BOOLEAN STRING FORWARD BACKWARD
%token RRIGHT RLEFT
%token PEN UP DOWN GOTO WHERE OR AND POW EQUAL
%token DIF MINOREQUAL MAJOREQUAL IN SAY ASK IF
%token THEN ELSE WHILE
%token <stringvalue>TRUE FALSE IDENTIFIER
%token UMBER STR SUCC PRED

%left MINOR MAJOR MINOREQUAL MAJOREQUAL EQUAL AND POW DIF OR
%left '+' '-'
%left '*' '/'


%type <stringvalue>Add_Op Mul_Op Rel_Op
%type <intvalue>Type SuccOrPred Factor Term Single_Expression  Expression
%type <intvalue>SuccPred /*Array_Acess*/
%type <varTipo>Var Variable
%type <constTipo>Constant Value_Var Inic_Var
```

```
%union{
int intvalue;
char* stringvalue;
VarTipo varTipo;
ConstTipo constTipo;
}



%start Liss

%%

/**************************Program**************/

Liss  : PROGRAM IDENTIFIER '{' Body '}' {printf("STOP\n");}
;

Body  : DECLARATIONS  {
height = 100;
width = 100;
xpos = 300;
ypos = 200;
raio = 25;
mode = 1; //PEN UP
direccao = up;
init();
}Declarations

    STATEMENTS {/*printHash();*/} Statements
;


/**************************Declarations**************/

Declarations : Declaration
| Declarations Declaration
;

Declaration  : Variable_Declaration
;
```

```
/***********Declarations: Variables************/

Variable_Declaration  : Vars ARROW Type ';'  {saveVars($3);}
;

Vars  : Var  {insertInListaVars($1, 1);
}
| Vars ',' Var  {insertInListaVars($3, 0);
}
;

Var  : IDENTIFIER Value_Var {
$$.id=$1;
$$.type=$2.type;
if ($2.type != -1) {
$$.value=$2.value;
}
       }
;

Value_Var  : {$$.type=-1;}
| '=' Inic_Var {$$=$2;}
;

Type  : INTEGER {$$ = 0;}
| BOOLEAN {$$ = 1;}
| STRING  {$$ = 2;}
/*| ARRAY SIZE NUMBER*/
;

Inic_Var  : Constant {$$ = $1;}
| '(' Constant ')' {$$ = $2;}
/*| Array_Definition*/
;

Constant   : NUMBER {$$.value = $1; $$.type=0;}
| STR   {$$.value = $1; $$.type=2;}
| TRUE   {$$.value = "TRUE"; $$.type=1;}
| FALSE  {$$.value = "FALSE"; $$.type=1;}
```

```
;


/**************************Declarations: Variables: Array_Definition***********


/*
Array_Definition  : '[' Array_Initialization ']'
;

Array_Initialization  : Elem
| Array_Initialization ',' Elem
;

Elem  : 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  {
VarData aux2 = searchVar("xpos"), aux3 = searchVar("ypos");
printf("PUSHG %d\n", aux2->address); //para o drawline
printf("PUSHG %d\n", aux3->address);     //para o drawline
  }
Expression  {
VarData aux = NULL;
if (direccao == up || direccao == down)
aux = searchVar("ypos");
else if (direccao == right || direccao == left)
aux = searchVar("xpos");

printf("PUSHG %d\n", aux->address);
printf("SWAP\n"); //para a subtracao ser bem feita

switch(direccao){
case(up):
printf("SUB\n");
break;
case(down):
printf("ADD\n");
break;
case(right):
printf("ADD\n");
break;
case(left):
printf("SUB\n");
break;
default:
break;
}
printf("STOREG %d\n", aux->address);
drawLine();
drawTurtle();
}
| BACKWARD {
VarData aux2 = searchVar("xpos"), aux3 = searchVar("ypos");
printf("PUSHG %d\n", aux2->address); //para o drawline
printf("PUSHG %d\n", aux3->address);     //para o drawline
 }
```

```
 Expression   {
VarData aux = NULL;

if (direccao == up || direccao == down)
aux = searchVar("ypos");
else if (direccao == right || direccao == left)
aux = searchVar("xpos");

printf("PUSHG %d\n", aux->address);
printf("SWAP\n"); //para a subtracao ser bem feita

  switch(direccao){
  case(up):
  printf("ADD\n");
  break;
  case(down):
  printf("SUB\n");
  break;
  case(right):
  printf("SUB\n");
  break;
  case(left):
  printf("ADD\n");
  break;
  default:
  break;
}
printf("STOREG %d\n", aux->address);
drawLine();
drawTurtle();
}
;

Rotate  : RRIGHT {
switch(direccao){
case(up):
direccao = right;
break;
case(right):
direccao = down;
break;
```

```
case(down):
direccao = left;
break;
default:
direccao = up;
break;
}
}
| RLEFT {
switch(direccao){
case(up):
direccao = left;
break;
case(left):
direccao = down;
break;
case(down):
direccao = right;
break;
default:
direccao = up;
break;
}
}
;

Mode  : PEN UP { mode = 0; }
| PEN DOWN { mode = 1; }
;

Dialogue  : Say_Statement
| Ask_Statement
;

Location  : GOTO NUMBER ',' NUMBER {
VarData aux2 = searchVar("xpos"), aux3 = searchVar("ypos");
printf("PUSHG %d\n", aux2->address); //para o drawline
printf("PUSHG %d\n", aux3->address); //para o drawline
VarData aux = searchVar("xpos"), aux1 = searchVar("ypos");
printf("PUSHI %s\n", $2);
printf("STOREG %d\n", aux->address);
```

```
printf("PUSHI %s\n", $4);
printf("STOREG %d\n", aux1->address);
drawLine();
drawTurtle();

}
| WHERE  '?'{
VarData aux2 = searchVar("xpos"), aux3 = searchVar("ypos");
printf("PUSHG %d\n", aux2->address); //para o drawline
printf("PUSHG %d\n", aux3->address);     //para o drawline
aux2 = searchVar("xpos"), searchVar("ypos");
printf("PUSHG %d\n", aux2->address);
printf("WRITEI\n");
printf("PUSHG %d\n", aux3->address);
printf("WRITEI\n");
}
;


/**************************Assignment Statement**************/


Assignment  : Variable '=' Expression  {
VarData var =  searchVar($1.id);
if (var) printf("STOREG %d\n",var->address);
else yyerror("Variable undeclared!\n");
}
;

Variable  : IDENTIFIER    {  VarData var = searchVar($1);
if(var){
$$.id=var->id;
$$.type=var->type;
}
else $$.type = -1;
        }
;
/*
Array_Acess  :
| '[' Single_Expression ']'{ $$ = $2; }
;
```

```
*/

/**************************Expression*************/

Expression  : Single_Expression { $$ = $1; }
| Expression Rel_Op Single_Expression {
if (strcmp($2,"DIF")!=0)
printf("%s\n",$2);
else{
printf("EQUAL\n");
printf("NOT\n");
}
}
;


/**************************Single Expression*************/

Single_Expression  : Term { $$ = $1; }
| Single_Expression Add_Op Term {
if (strcmp($2,"OR")!=0)
printf("%s\n",$2);
else{
printf("ADD\n");
printf("PUSHI 0\n");
printf("EQUAL\n");
printf("NOT\n");
}
}
;


/**************************Term*************/

Term  : Factor { $$ = $1; }
| Term Mul_Op Factor {
if(strcmp($2,"AND")==0){
printf("MUL\n");
printf("PUSHI 0\n");
printf("EQUAL\n");
        printf("NOT\n");
```

```
}
else if (strcmp($2,"POW")==0){
// TODO calcular a potencia
}
else printf("%s\n",$2);
}
;


/**************************Factor**************/

Factor  : Constant {pushValues($1.type,0,$1.value); $$ = $1.type;}
| Variable {
  VarData var = searchVar ($1.id);
  if(var)printf("PUSHG %d\n", var->address);
  else yyerror("Variable undeclared!\n");
  $$ = var->type;
}
| SuccOrPred { $$ = $1; }
| '(' Expression ')'{ $$ = $2; }
;


/**************************Operators**************/

Add_Op   : '+'{ $$ = "ADD"; }
| '-'{ $$ = "SUB"; }
| OR { $$ = "OR"; }
;

Mul_Op  : '*'{ $$ = "MUL"; }
| '/'{ $$ = "DIV"; }
| AND { $$ = "AND"; }
| POW { $$ = "POW"; }
;

Rel_Op  : EQUAL { $$ = "EQUAL"; }
| DIF  { $$ = "DIF"; }
| MAJOR  { $$ = "SUP"; }
| MINOR { $$ = "INF"; }
| MAJOREQUAL { $$ = "SUPEQ"; }
```

```
| MINOREQUAL { $$ = "INFEQ"; }
/*| IN { $$ = $1; }*/       //PENSO QUE SEJA SO PARA ARRAYS
;


/**************************SuccOrPred**************/

SuccOrPred  : SuccPred IDENTIFIER { VarData var = searchVar($2);
  if (var) {
     printf("PUSHG %d\n", var->address);
  printf("PUSHI %d\n", $1);
  printf("ADD\n");
  }
  else yyerror("Variable undeclared!\n");
}
;

SuccPred  : SUCC { $$ = 1; }
| PRED { $$ = -1; }
;


/**************************IO Statements***********/

Say_Statement  : SAY '(' Expression ')'{ switch ($3){ // Expression Type
case 0: // INTEGER
printf("WRITEI\n");
break;
case 1: // BOOLEAN
printf("WRITEI\n");
break;
case 2: // STRING
printf("WRITES\n");
break;
}
}
;


Ask_Statement  : ASK '(' STR ',' Variable ')'{
  VarData var = searchVar($5.id);
```

```
  if(!var) yyerror("Variable undeclared!\n");
  else{
printf("PUSHS %s\n",$3);  // guardar na stack a STR a perguntar
   printf("WRITES\n");  // escrever a STR a perguntar
printf("READ\n");  /* lê uma string do teclado (concluída por um "\n")
           e arquiva esta string (sem o "\n") na heap e coloca

             */
switch ($5.type){ // Expression Type
case 0: // INTEGER
printf("atoi\n");
break;
case 1: // BOOLEAN
printf("atoi\n");
break;
case 2: // STRING
break;
default :
yyerror("Variable undeclared!\n");
break;
  }
  printf("STOREG %d\n",var->address);
}
  }
;


/**************************Consitional & 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 '}'
;

%%

void insertInListaVars(VarTipo var, int first){
ListaVars *aux = (ListaVars*)malloc(sizeof(ListaVars));
aux->id = var.id;
aux->value = var.value;
aux->type = var.type;
if(first == 1){aux->next = NULL;}
else {aux->next = nodo;}
nodo = aux;

}

void saveVars(int type){
ListaVars *aux = nodo;
while(aux) {
if(!searchVar(aux->id)){
// insere nome, tipo e address na hashtable
insertVar(aux->id, type, addressG);
pushValues(type,aux->type, aux->value);
addressG++;
}
aux=aux->next;
}
nodo = NULL;

}

void pushValues(int varType, int nullType, char* value){
switch(varType) {
```

```
case 0://INTEGER
if (nullType == -1) //VAZIO
printf("PUSHI 0\n");
else
printf("PUSHI %d\n",atoi(value));
break;
case 1://BOOLEAN
if (nullType==-1 || strcmp(value,"TRUE")==0)
printf("PUSHI 1\n");
else if (strcmp(value, "FALSE")==0)
printf("PUSHI 0\n");
break;
case 2://STRING
if (nullType == -1)
printf("pushs \"\"\n");
else
printf("pushs %s\n",value);
break;
// nao estamos a fazer arrays para ja
}
}

void drawTurtle(){
VarData aux, aux2, aux3;
aux3 = searchVar("xpos");
        printf("PUSHG %d\n", aux3->address);
aux2 = searchVar("ypos");
        printf("PUSHG %d\n", aux2->address);
aux = searchVar("raio");
printf("PUSHG %d\n", aux->address);
printf("DRAWCIRCLE\n");
printf("REFRESH\n");
}

void drawLine(){
printf("CLEARDRAWINGAREA\n");
if(mode == 1){ // PEN DOWN
VarData aux1, aux2;
aux1 = searchVar("xpos");
         printf("PUSHG %d\n", aux1->address);
aux2 = searchVar("ypos");
```

```
         printf("PUSHG %d\n", aux2->address);
printf("DRAWLINE\n");
}
}

void init() {
varHashTable = initHash();
VarTipo var;

var.id = "xpos";
var.value = (char*)malloc(sizeof(10));
sprintf(var.value, "%d", xpos);
insertInListaVars(var, 1);

var.id = "ypos";
var.value = (char*)malloc(sizeof(10));
sprintf(var.value, "%d", ypos);
insertInListaVars(var, 0);

var.id = "raio";
var.value = (char*)malloc(sizeof(10));
sprintf(var.value, "%d", raio);
insertInListaVars(var, 0);

saveVars(0);
printf("START\n");
initWindow();
drawTurtle();
}

void initWindow(){
printf("PUSHI %d\n",800);
        printf("PUSHI %d\n",600);
        printf("opendrawingarea\n");
}


int yyerror(char *s){
       fprintf(stderr,"ERRO: %s na linha:%d antes de:%s\n",s,yylineno,yytext);
       return 0;
}
```

```
int main() {
yyparse();
return 0;
}
```

```
++++++++++++++++++++   structures.h  ++++++++++++++++
```

```
#ifndef _STRUCTS
#define _STRUCTS

typedef enum Direccoes{
up,
down,
right,
left
} Direccao;

typedef struct VarTipos {
char* id;
char* value;
int type;
} VarTipo;

typedef struct ConstTipos {
char* value;
int type;
} ConstTipo;

typedef struct NodoVar {
char* id;
char* value;
int type;
struct NodoVar *next;
} ListaVars;


int height, width, xpos, ypos, raio, mode;
Direccao direccao;
```

```
ListaVars *nodo;

void insertInListaVars(VarTipo var, int first);
void saveVars(int type);
void drawTurtle();
void drawLine();
void init();
void initWindow();
void pushValues(int varType, int nullType, char* value);

#endif
```

++++++++++++++++ hashFunctions.c ++++++++++++++++++++++++

```
#include "hashFunctions.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>


VarHashTable varHashTable;

VarHashTable initHash(){
int i;
VarHashTable v;
v = (VarHashTable)malloc(sizeof(VarData)*HASH_SIZE);
for (i = 0; i < HASH_SIZE; i++)
v[i]=NULL;
return v;
}

int hash(char* s) {
int h = 0, i, n=strlen(s);
for (i = 0; i < n; i++) {
h = 31*h + s[i];
```

```
}
return abs(h%HASH_SIZE);
}

VarData searchVar(char* id){
int h = hash(id);
VarData varData = varHashTable[h];
while (varData && strcmp(varData->id, id)!=0){
varData = varData->next;
}
return varData;
}

int insertVar(char* id, int type, int address) {
int h = hash(id);
VarData new, var;
new = (VarData)malloc(sizeof(struct varData));
var = varHashTable[h];
varHashTable[h]=new;
new->id=id;
new->type=type;
new->address=address;
new->next=var;
return 1;
}
```

++++++++++++++++ hashFunctions.h ++++++++++++++++++++++

```
#ifndef _HASH
#define _HASH

#define HASH_SIZE 30

typedef struct varData {
```

```
char* id;
int type;
int address;
struct varData *next;
} *VarData, **VarHashTable;

VarHashTable initHash();
int hash(char* s);
VarData searchVar(char* id);
int insertVar(char* id, int type, int address);
void printHash();

#endif
```

++++++++++++++++++ Makefile ++++++++++++++++++++++++++

```
logoliss: y.tab.o lex.yy.o hashFunctions.o structures.h
cc -o logoliss y.tab.o lex.yy.o hashFunctions.o -lfl

y.tab.o: y.tab.c
cc -c y.tab.c

lex.yy.o: lex.yy.c
cc -c lex.yy.c


lex.yy.c: logo.l y.tab.h structures.h
flex logo.l

y.tab.c y.tab.h: logo.y
yacc -d -v logo.y

hashFunctions.o: hashFunctions.h hashFunctions.c
cc -c hashFunctions.c
```

```
clean :
rm -Rf *.o y.* lex.yy.c
```