# Patrones Léxicos

// \*\*\*\*\*\*\*\*\*\*\*\* Patrones (macros) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Rubbish = [ \t\n\r]

CommentV1 = #.\*

CommentV2 = \"""~\"""

Letter = [a-zA-Z]

Digit = [0-9]

Ident = [\_a-zA-Z][a-zA-Z\_0-9]\*

IntConstant = [0-9]\*

RealType = [0-9]+[.][0-9]\* | [.][0-9]+

RealConstant = {RealType}|{RealType}E[+|-][0-9]+|[0-9]+e[+|-][0-9]+

Character = \'.\'

CharacterASCII = [']\\[0-9]\*[']

%%

// \*\*\*\*\*\*\*\*\*\*\*\* Acciones \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \* THINGS TO IGNORE

{Rubbish}

{CommentV1}

{CommentV2}

// \* RESERVED WORDS

Input

print

def

while

if

else

int

double

char

struct

return

void

main

// \* OPERATORS

"+"

"-"

"\*"

"/"

"%"

">"

"<"

">="

"<="

"="

==

"!="

"!"

"||"

"&&"

// \* CONSTANTS

{IntConstant}

{RealConstant}

{Ident}

'\\n'

'\\t'

{CharacterASCII}

{Character}

// \* DELIMITERS

"{"

"}"

"("

")"

"["

"]"

";"

","

":"

"."

// \* Other

.

# GLC

// \* Declaraciones Yacc

%token INT\_CONSTANT

%token INPUT

%token PRINT

%token DEF

%token WHILE

%token IF

%token ELSE

%token INT

%token DOUBLE

%token CHAR

%token STRUCT

%token RETURN

%token VOID

%token ID

%token REAL\_CONSTANT

%token CHAR\_CONSTANT

%token GREATER

%token SMALLER

%token EQUALS

%token NEGATION

%token MAIN

%token OR

%token AND

%right '='

%left OR AND

%left EQUALS NEGATION SMALLER '<' GREATER '>'

%left '-' '+'

%left '\*' '/' '%'

%nonassoc CAST

%right UNARIO

%nonassoc '!'

%left '.'

%nonassoc '[' ']'

%nonassoc '(' ')'

%nonassoc ':'

%nonassoc ELSE

%%

// \* Gramática y acciones Yacc

programa : definiciones DEF MAIN '(' ')'':'VOID '{' body '}';

definiciones: definiciones definicion

| /\* empty \*/

;

definicion: def ';'

| funcion

;

// \*\*\*\*\*\*\*\*\* FUNCIONES \*\*\*\*\*\*\*\*\*

funcion: DEF ID '(' params ')' ':' retorno '{' body '}';

retorno: tipo

| VOID

;

body: defs

| sentencias

| defs sentencias

|

;

params: /\* empty \*/

| param

;

param: par

| param ',' par

par: ID ':' tipo;

// \*\*\*\*\*\*\*\*\* DEFINICIONES \*\*\*\*\*\*\*\*\*

defs: def ';'

| defs def ';'

;

def: ids ':' tipo

ids: ID

| ids ',' ID

;

tipo: INT

| DOUBLE

| CHAR

|'['INT\_CONSTANT']' tipo

| STRUCT '{' campos '}'

;

campos: campo

|campos campo

;

campo: ids ':' tipo ';';

// \*\*\*\*\*\*\*\*\* SENTENCIAS \*\*\*\*\*\*\*\*\*

sentencias: sentencia

| sentencias sentencia

;

sentencia: PRINT list ';'

| INPUT list ';'

| RETURN expresion ';'

| condicionalSimple

| condicionalComplejo

| while

| asignacion ';'

| invocacion ';'

;

expresion: ID

| INT\_CONSTANT

| CHAR\_CONSTANT

| REAL\_CONSTANT

| '(' expresion ')'

| expresion '[' expresion ']'

| expresion '.' ID

| '(' tipo ')' expresion %prec CAST

| '-' expresion %prec UNARIO

| '!' expresion

| expresion '\*' expresion

| expresion '/' expresion

| expresion '%' expresion

| expresion '+' expresion

| expresion '-' expresion

| expresion '>' expresion

| expresion GREATER expresion

| expresion '<' expresion

| expresion SMALLER expresion

| expresion NEGATION expresion

| expresion EQUALS expresion

| expresion AND expresion

| expresion OR expresion

| ID '(' args ')'

;

list: expresion

| list ',' expresion

;

asignacion: expresion '=' expresion ;

invocacion: ID '(' args ')'

// \*\*\*\*\*\*\*\*\* WHILE \*\*\*\*\*\*\*\*\*

while: WHILE expresion ':' '{' sentencias '}' ;

// \*\*\*\*\*\*\*\*\* IF-ELSE \*\*\*\*\*\*\*\*\*

condicionalSimple: IF expresion ':' cuerpo;

condicionalComplejo: IF expresion ':' cuerpo else;

else: ELSE cuerpo ;

cuerpo: sentencia

| '{' sentencias '}'

;

// \*\*\*\*\*\*\*\*\* INVOCACIÓN DE FUNCIONES \*\*\*\*\*\*\*\*\*

args: /\* empty \*/

| arg

;

arg: expresion

| arg ',' expresion

# Gramática Abstracta

Program: Program -> Definition\*

VarDefinition: Definition -> Type ID

FunDefinition: Definition -> Type Statement\*

Write: Statement -> Exp

Read: Statement -> Exp

Assigment: Statement -> Exp1 Exp2

IfStatement: Statement -> Exp if:Statement\* else:Statement\*

WhileStatement: Statement -> Exp Statement\*

Invocation: Statement -> Variable Exp\*

Return: Statement -> Exp

IntLiteral: Exp -> IntConstant

ChaLiteral: Exp -> CharConstant

RealLiteral: Exp -> RealConstant

Variable: Exp -> ID

Arithmetic: Exp -> left:Exp right:Exp

Comparison: Exp -> left:Exp right:Exp

Cast: Exp -> CastType valor:Exp

Logical: Exp -> left:Exp right:Exp

UnaryNot: Exp -> valor:Exp

FieldAcces: Exp -> valor:Exp ID

Indexing: Exp -> left:Exp right:Exp

Invocation: Exp -> Variable Exp\*