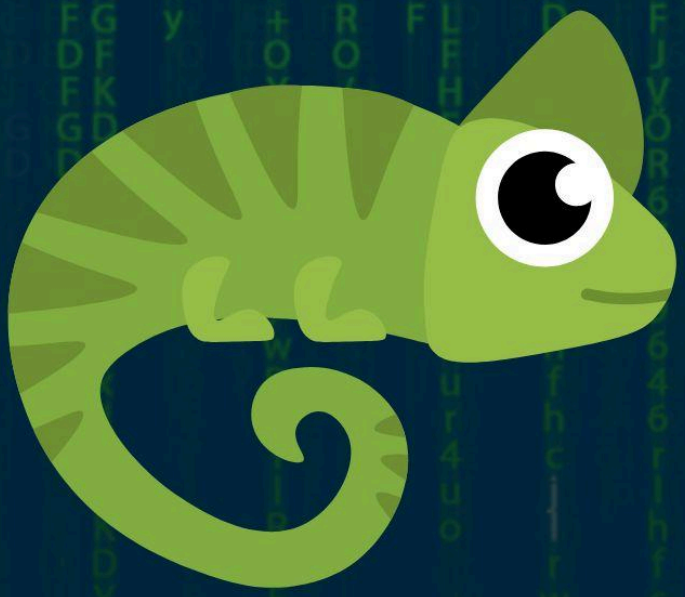
PASCAL IDE

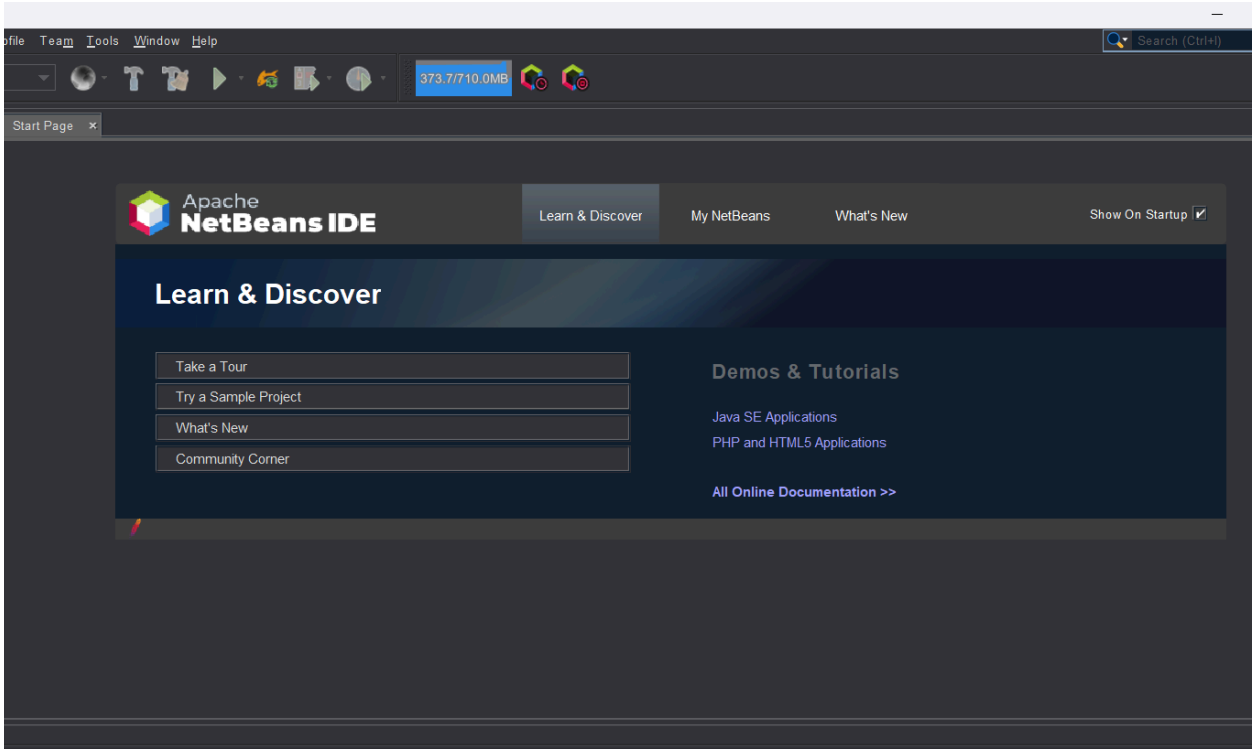


TECHNICAL MANUAL

ÍNDICE

TECNOLOGÍA USADA EN EL PROYECTO

- Ide: ApacheNetBeans IDE 16



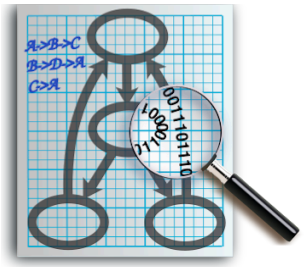
- JFlex 1.9.1
    - Enlace para la descarga: https://jflex.de/download.html
- Java-cup-11b
    - Enlace para la descarga: https://www2.cs.tum.edu/projects/cup/

REQUERIMIENTOS PARA EL FUNCIONAMIENTO DEL PROYECTO

- **JAVA**

openjdk 23-ea 2024-09-17
OpenJDK Runtime Environment (build 23-ea+36-Debian-1)
OpenJDK 64-Bit Server VM (build 23-ea+36-Debian-1, mixed mode, sharing)

- **Graphviz**



dot - graphviz version 2.43.0 (0)
libdir = "/usr/lib/x86_64-linux-gnu/graphviz"
Activated plugin library: libgvplugin_dot_layout.so.6
Using layout: dot:dot_layout
Activated plugin library: libgvplugin_core.so.6
Using render: dot:core
Using device: dot:dot:core

DIAGRAMA DE CLASES

TABLA DE SÍMBOLOS

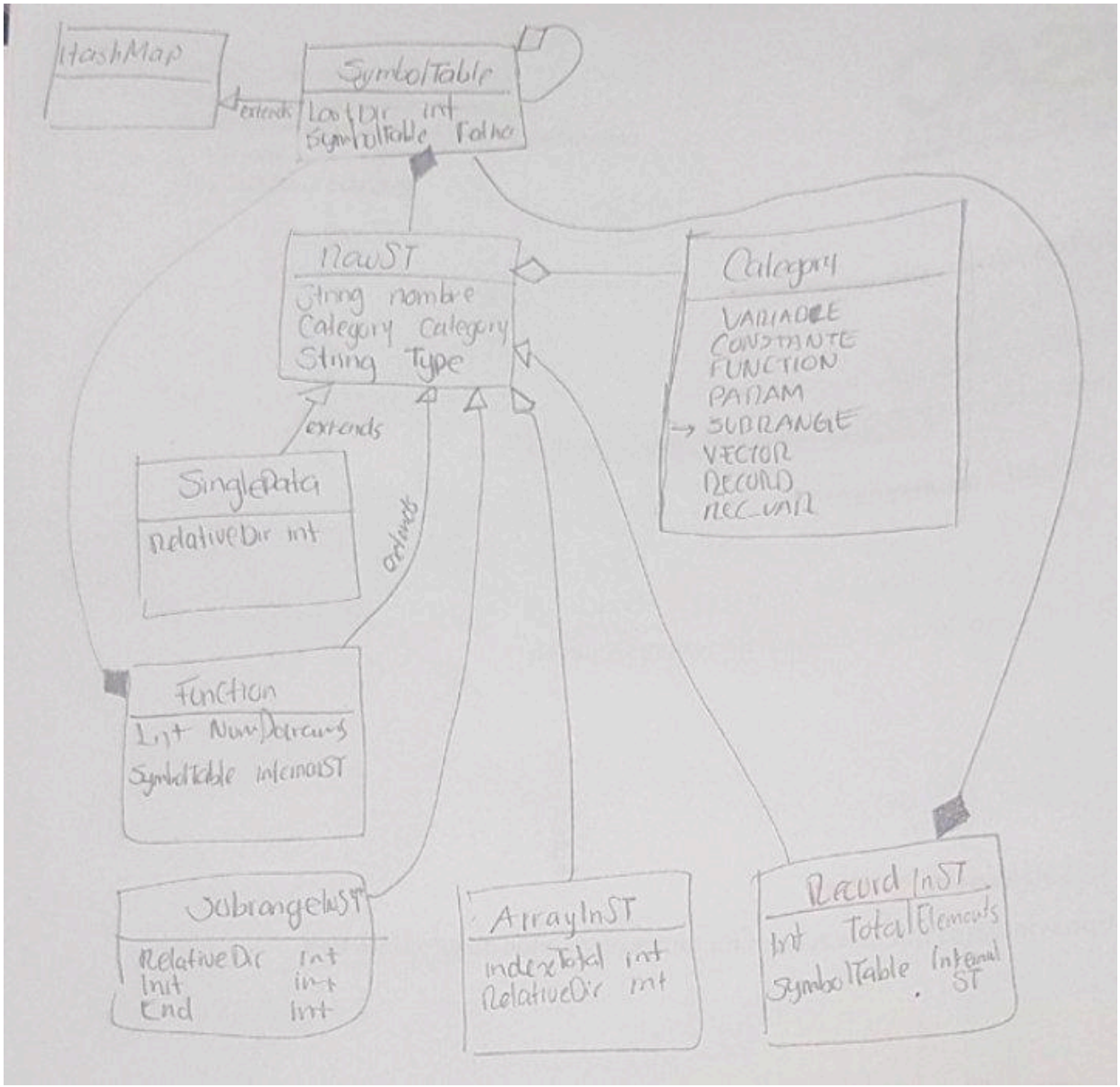
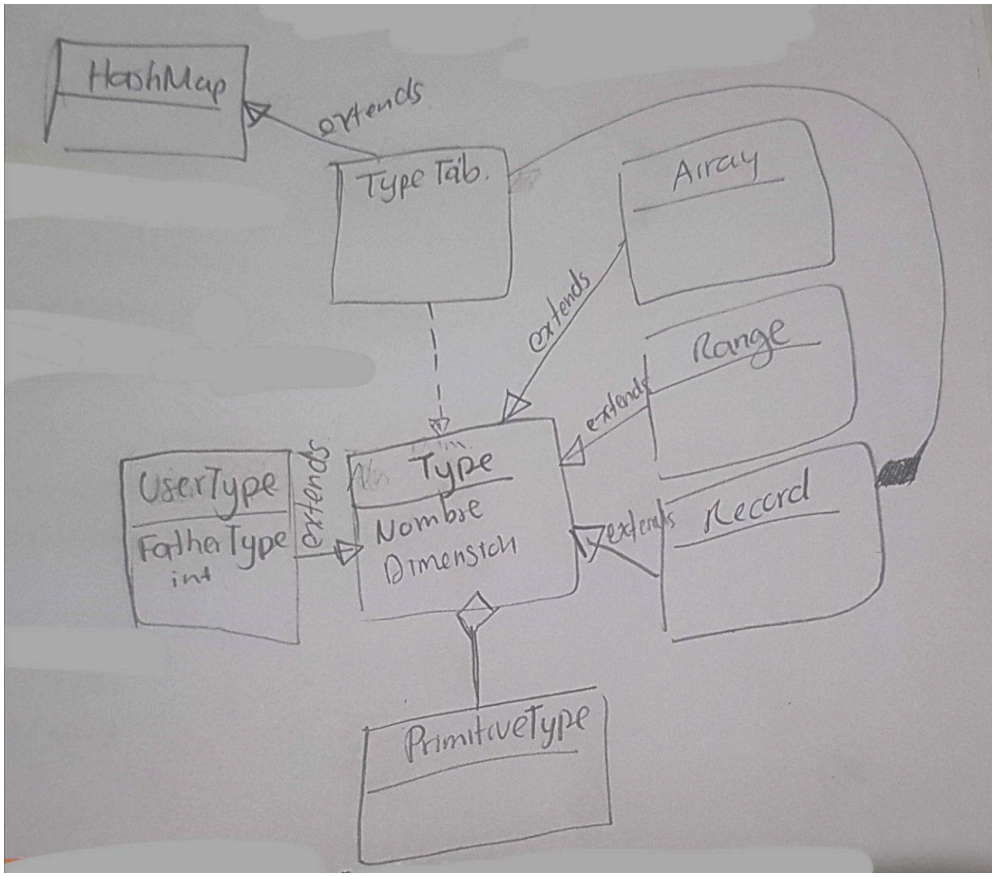
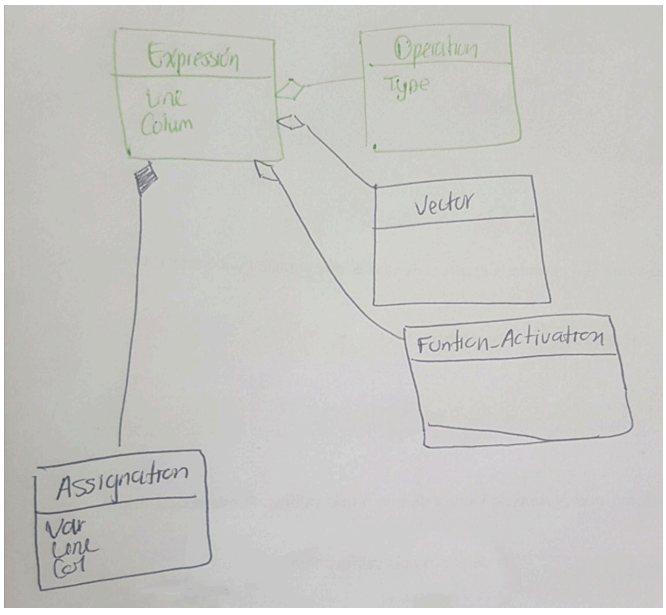


TABLA DE TIPOS



EXPRESIONES



STATEMENTS

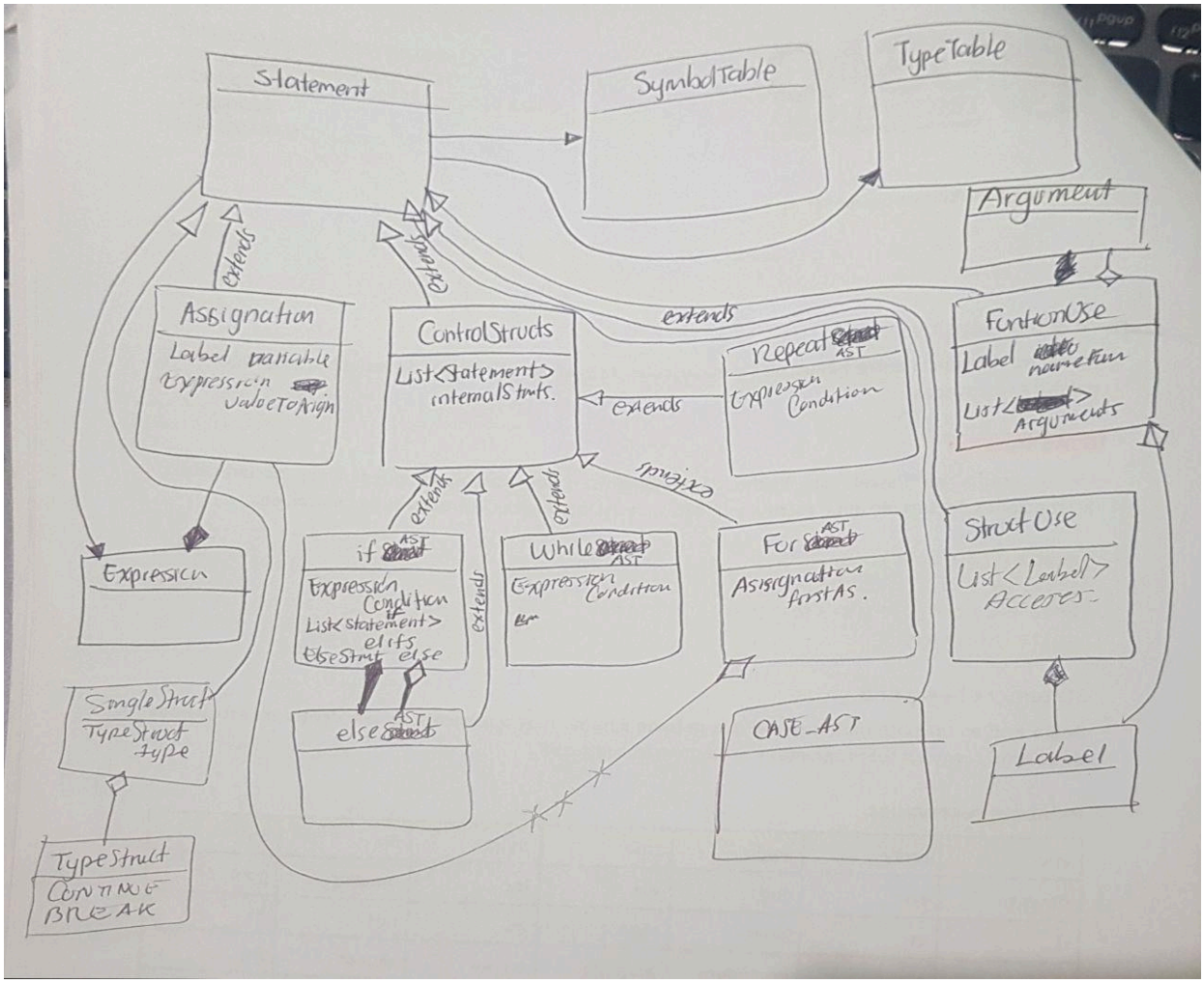
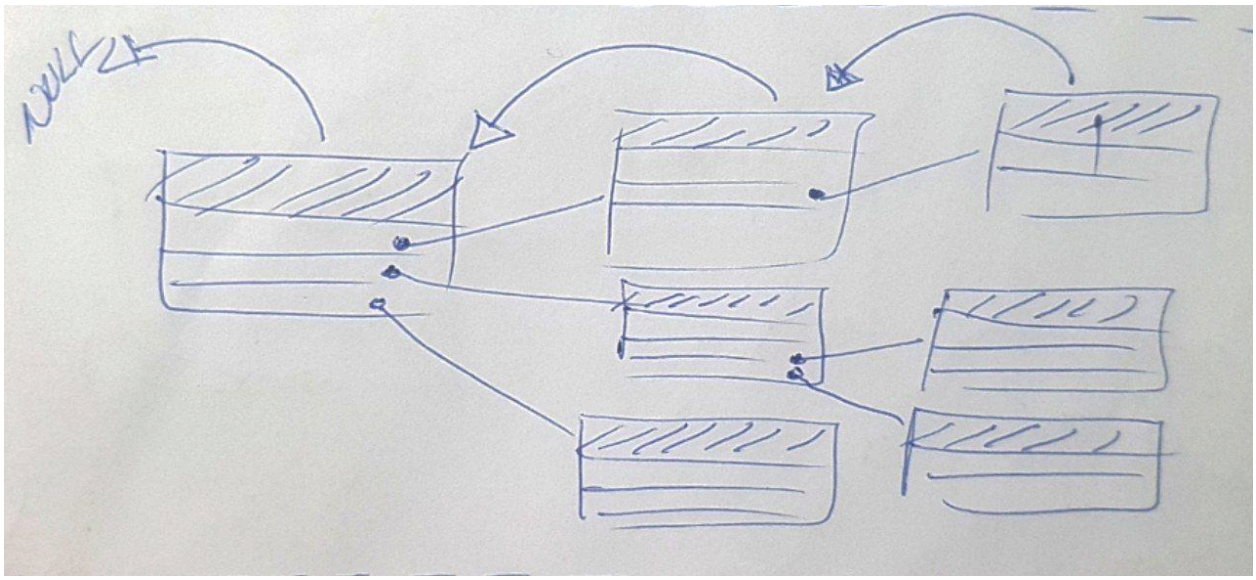

Statement
SymbolTable
TypeTable
Argument
extends
Assignation
Label
Expression
ControlStructs
List<Statement>
internalStmts.
extends
RepeatAST
Expression
Condition
FonctionUse
Label
List<
Arguments
Expression
SingleStruct
TypeStruct
type
TypeStruct
CONTINUE
BREAK
if AST
Expression
Condition
List<Statement>
elifs
ElseStmt
else
elseAST
WhileAST
Expression
Condition
ForAST
Asisignation
firstAS.
CASE_AST
StructUse
List<Label>
Acceres
Label

DIAGRAMA DE RELACIÓN TABLAS DE SÍMBOLOS/TIPOS

Las tablas de símbolos y de tipos van a estar de la siguiente manera, pudiendo tener una interna dependiendo del contexto:



DESCRIPCIÓN DE PAQUETES

- **compi2.pascal.valitations**

Clases de inicializacion, fronted y otros elementos superficiales

- **compi2.pascal.valitations.analysis**

Comprende la parte de las primeras fases de análisis del compilador, se incluyen clases de análisis léxico y sintáctico.

- **compi2.pascal.valitations.analysis.symbolt**

Incluye validaciones de la tabla de símbolos y otros elementos inherentes a esta.

- **compi2.pascal.valitations.analysis.typet**

Incluye validaciones de la tabla de tipos y otros elementos inherentes a esta.

- **compi2.pascal.valitations.analysis.typet.conver**

Todo lo relacionado a la conversión implícita de tipos

- **compi2.pascal.valitations.analizator**

Contiene clases master para realizar el análisis sobre el compilador, son una especie de controlados y clases que contienen todos los elementos principales.

- **compi2.pascal.valitations.exceptions**

Contiene las excepciones personalizadas del proyecto.

- **compi2.pascal.valitations.files**

Incluye las clases que controlan y validan todo lo relacionado con los archivos.

- **compi2.pascal.valitations.graphs**

Incluye las clases necesarias para graficar la tabla de tipos y la de símbolos.

- **compi2.pascal.valitations.semantic**

Todo lo relacionado a paso de datos en el analizador sintáctico.

- **compi2.pascal.valitations.semantic.ast**

Contiene los ast de los statements del lenguaje.

- **compi2.pascal.valitations.semantic.expr**

Contiene todo lo relacionado con las expresiones.

- **compi2.pascal.valitations.semantic.module**

Incluye el ast de las funciones y procedimientos, así como clases importantes para su validación.

- **compi2.pascal.valitations.obj**

Contiene los árboles de sintaxis abstracta (AST por sus siglas en inglés) de los objetos que se asignan en la tabla de tipos como la de símbolos.

- **compi2.pascal.valitations.util**

Contiene clases útiles para todo el proyecto.

ESPECIFICACIONES DEL LENGUAJE

PARA EL LEXER

SÍMBOLOS RESERVADOS

ARITMÉTICOS

+	PLUS
-	MINUS
*	TIMES
/	BARRA

DELIMITADORES

[	CORCHETE L
]	CORCHETE R
(	PARENTESIS L
)	PARENTESIS R
{	LLAVE L
}	LLAVE R

COMPARADORES

=	IGUAL
<>	DIFERENTE
>=	MAYOR O IGUAL QUE
<=	MENOR O IGUAL QUE
>	MAYOR QUE
<	MENOR QUE

OTROS

:=	ASIGNACIÓN
,	COMA
.	PUNTO
;	PUNTO Y COMA
:	DOS PUNTOS

PALABRAS RESERVADAS

Nota: Estas son case insensitive

- ★ AND
- ★ ARRAY
- ★ BEGIN
- ★ BOOLEAN
- ★ BREAK
- ★ CASE
- ★ CHAR
- ★ CONST
- ★ CONTINUE
- ★ DIV
- ★ DO
- ★ DOWNTO
- ★ ELSE
- ★ FILE
- ★ FOR
- ★ FUNCTION
- ★ GOTO
- ★ IF
- ★ IN
- ★ INTEGER
- ★ LABEL
- ★ LONGINT
- ★ MOD
- ★ NIL
- ★ NOT
- ★ OF
- ★ OR
- ★ PACKED
- ★ PROCEDURE
- ★ PROGRAM
- ★ READLN
- ★ REAL
- ★ RECORD
- ★ REPEAT
- ★ RETURN
- ★ SET
- ★ STRING
- ★ THEN
- ★ TO
- ★ TYPE
- ★ UNTIL
- ★ VAR
- ★ WHILE
- ★ WITH
- ★ WRITELN

FORMATO DE DATOS

```
Identifier = [letra] ([letra]][digito]|_)*
Boolean = 0|1
Entero = [0-9]+
Float = {Entero}\.{Entero}
```

PARA EL PARSER (GRAMÁTICA)

```
start with s;
s ::=   header
      type_b
      const_b
      var_b
      functions_b
      procedure_b
      main
      ;

header ::= PROGRAM ID SEMICOLON ;
```

ÚTILES

```
var_type ::= INTEGER
           | REAL
           | BOOLEAN
           | CHAR
           | STRING
           | ID
           ;

id_list ::= id_list COMA ID
           | ID
           ;

range ::= expression DOT DOT DOT expression
           ;
```

```
arr_range ::= CORCHETE_L expression DOT DOT expression CORCHETE_R
            ;

record_b ::= def_record more_records
            ;

def_record ::= ID COLON var_type SEMICOLON
            | ID COLON PACKED ARRAY arr_range OF var_type SEMICOLON
            ;

more_records ::= def_record more_records
            | /* empty */
            ;

expression_list ::= expression_list COMA expression
            | expression
            ;
```

BLOQUE DE TIPOS

```
type_b ::= TYPE list_typedec
            | /* empty */
            ;

list_typedec ::= type_dec list_typedec
            | type_dec
            ;

type_dec ::=  id_list EQUALS var_type SEMICOLON
            | id_list EQUALS range SEMICOLON
            | id_list EQUALS ARRAY arr_range OF var_type SEMICOLON
            | id_list EQUALS RECORD record_b END SEMICOLON
            ;
```

BLOQUE DE CONSTANTES

```
const_b ::= CONST list_constdec
            | /* empty */
```

```
            ;

list_constdec ::= list_constdec const_dec
            | const_dec
            ;

const_dec ::= ID EQUALS expression SEMICOLON
            ;
```

BLOQUE DE VARIABLES

```
var_b ::= VAR list_vardec
            | /* emtpy */
            ;

list_vardec ::= list_vardec var_dec
            | var_dec
            ;

var_dec ::= id_list COLON var_type SEMICOLON
            | id_list COLON range SEMICOLON
            | id_list COLON ARRAY arr_range OF var_type SEMICOLON
            | id_list EQUALS RECORD record_b END SEMICOLON
            ;
```

EXPRESIONES

```
/*precedencia de booleanos*/
precedence left THEN;
precedence left ELSE;
precedence left OR;
precedence left AND;
precedence left NOT;

/*precedencia de operaciones*/
precedence left PLUS, MINUS;
precedence left TIMES, DIV, MOD;
```

```
precedence right UMINUS;

expression ::= expression AND expression
            | expression AND THEN expression
            | expression OR expression
            | expression OR ELSE expression
            | expression NOT expression
            | bool_exp
            ;

bool_exp ::= arit_exp EQUALS arit_exp
            | arit_exp DIFFERENT arit_exp
            | arit_exp GRATER arit_exp
            | arit_exp LESS arit_exp
            | arit_exp GRATER_EQUALS arit_exp
            | arit_exp LESS_EQUALS arit_exp
            | arit_exp
            ;

arit_exp ::=
            arit_exp PLUS arit_exp
            | arit_exp MINUS arit_exp
            | arit_exp TIMES arit_exp
            | arit_exp DIV arit_exp
            | arit_exp MOD arit_exp
            | literals
            | ID CORCHETE_L arit_exp CORCHETE_R
            | ID DOT list_access
            | ID PARENTESIS_L id_list PARENTESIS_R
            | ID
            %prec UMINUS
            | PARENTESIS_L expression PARENTESIS_R
            %prec UMINUS
            | PLUS arit_exp
            %prec UMINUS
            | MINUS arit_exp
            ;

list_access ::=  ID DOT list_access
            | ID
            ;

literals ::= BOOLEAN_LIT
```

```
            | INTEGER_LIT
            | REAL_LIT
            | CHAR_LIT
            | STRING_LIT
            ;
```

BLOQUE DE INSTRUCCIONES

CONDICIONAL

```
precedence left IF;
precedence left ELSE;

conditional ::= if_stmt pos_if
      ;
if_stmt ::= IF PARENTESIS_L expression PARENTESIS_R THEN
      ;

pos_if ::= simple_stmt elif_stmt
      | simple_stmt SEMICOLON
      | complex_stmt
      | BEGIN block_stmt END SEMICOLON
      | BEGIN block_stmt END elif_stmt
      ;

elif_stmt ::=  ELSE IF PARENTESIS_L expression PARENTESIS_R THEN pos_elif
elif_stmt
      | ELSE IF PARENTESIS_L expression PARENTESIS_R THEN pos_elif
      | else_stmt
      ;

pos_elif ::= simple_stmt else_stmt
      | simple_stmt SEMICOLON
      | BEGIN block_stmt END SEMICOLON
      | BEGIN block_stmt END else_stmt
      ;

else_stmt ::= ELSE statements_wc
      | /* empty */
      ;
```

```
statements_wc ::= simple_stmt SEMICOLON
      | BEGIN block_stmt END SEMICOLON
      ;
```

CASE

```
case_stmt ::= CASE PARENTESIS_L expression PARENTESIS_R OF case_block ELSE
single_stm END SEMICOLON
      ;

case_block ::= labels COLON single_stm SEMICOLON
      ;

labels ::= labels COMA expression
      | expression
      ;

single_stm ::= ID ASSIGNATION expression
      | ID PARENTESIS_L expression_list PARENTESIS_R
      | ID DOT ID
      | definite_fun
      ;
```

CICLOS

```
while_stmt ::= WHILE PARENTESIS_L expression PARENTESIS_R DO statements
      ;

for_stmt ::= FOR ID ASSIGNATION expression TO expression DO statements
      ;

repeat_stmt ::= REPEAT statements UNTIL expression SEMICOLON
      ;
```

OTRAS INSTRUCCIONES

```
statements ::= simple_stmt
      | simple_stmt SEMICOLON
      | complex_stmt
      | BEGIN block_stmt END SEMICOLON
```

```
      ;

block_stmt ::= list_stmts
      | /* empty */
      ;

list_stmts ::= simple_stmt
      | simple_stmt SEMICOLON
      | complex_stmt
      | simple_stmt SEMICOLON more_stmts
      | complex_stmt more_stmts
      ;

more_stmts ::= simple_stmt
      | simple_stmt SEMICOLON
      | simple_stmt SEMICOLON more_stmts
      | complex_stmt more_stmts
      | complex_stmt
      ;

simple_stmt ::= BREAK
      | CONTINUE
      | ID ASSIGNATION expression
      | ID PARENTESIS_L expression_list PARENTESIS_R
      | ID DOT ID ASSIGNATION expression
      | definite_fun
      ;

complex_stmt ::= conditional
      | case_stmt
      | while_stmt
      | for_stmt
      | repeat_stmt
      ;
```

FUNCIONES

```
functions_b ::= functions
      | /* empty */
      ;
```

```
functions ::= functions function_dec
      ;

function_dec ::= function_dec FUNCTION ID PARENTESIS_L arguments
PARENTESIS_R COLON var_type SEMICOLON
            var_b
            BEGIN block_stmt END SEMICOLON
      ;
```

PROCEDIMIENTOS

```
procedure_b ::= procedures
      | /* empty */
      ;

procedures ::= procedures procedure_dec
      | procedure_dec
      ;

procedure_dec ::= PROCEDURE ID PARENTESIS_L arguments PARENTESIS_R
SEMICOLON
            var_b
            BEGIN block_stmt END SEMICOLON
      ;
```

PARAMETROS Y ARGUMENTOS

```
arguments ::= arguments COMA type_arg
      | type_arg
      | /* empty */
      ;

type_arg ::= VAR id_list COLON var_type
      | id_list COLON var_type
      ;
```

BLOQUE PRINCIPAL

```
main ::= BEGIN block_stmt END DOT
```

```
      ;
```

FUNCIONES POR DEFECTO

```
definite_fun ::= writefn
      | readfn
      ;

writefn ::= WRITELN PARENTESIS_L expression_list PARENTESIS_R
      ;
readfn ::= READLN PARENTESIS_L expression_list PARENTESIS_R
      ;
```

CONVERSIÓN IMPLÍCITA DE TIPOS

Operadores Aritméticos

Suma

boolean (0)	0
char (1)	1
integer (2)	2
longint (3)	3
real (4)	4
string (5)	ERROR

No es posible realizar operaciones con un tipo personalizado

+	boolean (0)	char (1)	integer (2)	longint (3)	real (4)	string (5)
boolean	0	1	2	3	4	5
char	1	1	2	3	4	5
integer	2	2	2	3	4	5
longint	3	3	3	3	4	5
real	4	4	4	4	4	5
string	5	5	5	5	5	5

No es posible operar con tipos definidos por el usuario.

Resta

boolean (0)	0
char (1)	1
integer (2)	2
longint (3)	3
real (4)	4
string (5)	ERROR

No es posible realizar operaciones con un tipo personalizado

-	boolean (0)	char (1)	integer (2)	longint (3)	real (4)	string (5)
boolean	0	1	2	3	4	ERROR
char	1	1	2	3	4	ERROR
integer	2	2	2	3	4	ERROR
longint	3	3	3	3	4	ERROR
real	4	4	4	4	4	ERROR
string	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR

No es posible operar con tipos definidos por el usuario.

Multiplicación

*	boolean (0)	char (1)	integer (2)	longint (3)	real (4)	string (5)
boolean	0	1	2	3	4	ERROR
char	1	1	2	3	4	ERROR
integer	2	2	2	3	4	ERROR
longint	3	3	3	3	4	ERROR
real	4	4	4	4	4	ERROR
string	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR

No es posible operar con tipos definidos por el usuario.

División (“/”)

/	boolean (0)	char (1)	integer (2)	longint (3)	real (4)	string (5)
boolean	4	4	4	4	4	ERROR
char	4	4	4	4	4	ERROR
integer	4	4	4	4	4	ERROR
longint	4	4	4	4	4	ERROR
real	4	4	4	4	4	ERROR
string	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR

No es posible operar con tipos definidos por el usuario.

División entera (div)

div	boolean (0)	char (1)	integer (2)	longint (3)	real (4)	string (5)
boolean	2	2	2	2	ERROR	ERROR
char	2	2	2	2	ERROR	ERROR
integer	2	2	2	2	ERROR	ERROR
longint	2	2	2	2	ERROR	ERROR
real	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR
string	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR

No es posible operar con tipos definidos por el usuario.

Módulo (mod)

div	boolean (0)	char (1)	integer (2)	longint (3)	real (4)	string (5)
boolean	2	2	2	2	ERROR	ERROR
char	2	2	2	2	ERROR	ERROR
integer	2	2	2	2	ERROR	ERROR
longint	2	2	2	2	ERROR	ERROR
real	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR
string	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR

No es posible operar con tipos definidos por el usuario.

Operadores Relacionales

Igualdad

=	boolean (0)	char (1)	integer (2)	longint (3)	real (4)	string (5)
boolean	0	0	0	0	0	ERROR
char	0	0	0	0	0	ERROR
integer	0	0	0	0	0	ERROR
longint	0	0	0	0	0	ERROR
real	0	0	0	0	0	ERROR
string	ERROR	ERROR	ERROR	ERROR	ERROR	0

Un tipo personalizado por el usuario solo puede ser comparado por otro igual y regresa un booleano.

Desigualdad

<>	boolean (0)	char (1)	integer (2)	longint (3)	real (4)	string (5)
boolean	0	0	0	0	0	ERROR
char	0	0	0	0	0	ERROR
integer	0	0	0	0	0	ERROR
longint	0	0	0	0	0	ERROR
real	0	0	0	0	0	ERROR
string	ERROR	ERROR	ERROR	ERROR	ERROR	0

Un tipo personalizado por el usuario solo puede ser comparado por otro igual y regresa un booleano.

Comparadores

> >= <= <	boolean (0)	char (1)	integer (2)	longint (3)	real (4)	string (5)
boolean	0	0	0	0	0	ERROR
char	0	0	0	0	0	ERROR
integer	0	0	0	0	0	ERROR
longint	0	0	0	0	0	ERROR
real	0	0	0	0	0	ERROR
string	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR

No es posible realizar operaciones con un tipo personalizado

Operadores Booleanos

and, and then, or, or else	boolean (0)	char (1)	integer (2)	longint (3)	real (4)	string (5)
boolean	0	0	0	0	0	ERROR
char	0	0	0	0	0	ERROR
integer	0	0	0	0	0	ERROR
longint	0	0	0	0	0	ERROR
real	0	0	0	0	0	ERROR
string	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR

No es posible realizar operaciones con un tipo personalizado

Negación (Not)

boolean (0)	0
char (1)	0
integer (2)	0

longint (3)	0
real (4)	0
string (5)	ERROR

No es posible realizar operaciones con un tipo personalizado