MANUAL TÉCNICO

Herramientas utilizadas para el desarrollo del código:

- JISON como herramienta léxica y sintáctica.
- NODE JS versión 16.9.1.
- ANGULAR para el front end versión 11.2.8.
- WORKER (usado para back end), como herramienta semántica y de cuartetos..
- ACE EDITOR para angular

Que tipo de estructuras no acepta el código:

- Enviar dirección de una variable ubicada en el heap.
- Enviar dirección de una posición del arreglo, se puede enviar únicamente la dirección de las variables.

Se utilizó nodemon para realizar pruebas del lado del back end.

El programa genera únicamente código 3 direcciones de la interpretación del código multilenguaje establecido previamente.
Para reconocer arreglos se debe de declarar de la siguiente manera.
Arr [50], indicando siempre el espacio en blanco que hay en ellos.
Detalles de gramáticas:

Expresión Regular	**Token**
“[“	OPEN_BRACKET
“]”	CLOSE_BRACKET
“{“	OPEN_CURLY
“}”	CLOSE_CURLY
“(“	OPEN_PARENTHESIS
“)”	CLOSE_PARENTHESIS
java “.”[aA-zZ|_|$][0-9|aA-zZ|_|$]*	METODO_CLASE
PY”.”[aA-zZ|_|$][0-9|aA-zZ|_|$]*	METODO
java “.”[aA-zZ|_|$][0-9|aA-zZ|_|$]*	Clase
if	IF
else	ELSE
switch	SWITCH

while	while
#include	INCLUDE
for	FOR
do	DO
int	INT
char	CHAR
float	FLOAT
const	CONST
case	CASE
break	BREAK
default	DEFAULT
continue	CONTINUE
scanf	SCANF
printf	PRINTF
clrscr	CLEAN_SCREEN
getch	GETCH
void	VOID
main	MAIN
[0-9]+	LIT_ENTERO
[0-9]+[“?”[0-9]+]?	LIT_DECIMAL
\”[^\”]*\”	LIT_STRING
\’[^\’]*\’	LIT_CARACTER
[aA-zZ|_|$][0-9|aA-zZ|_|$]*	IDENTIFICADOR
>	MAYOR
<	MENOR
==	COMPARACION
!=	DIFERENTE
&&	AND

||	OR
!	NOT
++	INCREMENTO
--	DECREMENTO
+	SUMA
-	RESTA
*	POR
/	DIV
%	MOD
=	IGUAL
;	COLON
:	SEMI_COLON
&	PUNTERO
,	COMA

gramatica C

```
parametros
   : expresion parametros_re
   | /*empty*/    ;

parametros_re
   :  COMA expresion parametros_re
   |  /*empty*/    ;

metodo_stmt
   : METODO OPEN_PARENTHESIS parametros CLOSE_PARENTHESIS
variable_clase
   :  COMA IDENTIFICADOR variable_clase
asignacion_clase
   : variable_clase
   | OPEN_PARENTHESIS parametros
clase_stmt
   : CLASE IDENTIFICADOR asignacion_clase

clean_stmt
```

```
   : CLEAN_SCREEN OPEN_PARENTHESIS CLOSE_PARENTHESIS
getch_stmt
:GETCH OPEN_PARENTHESIS CLOSE_PARENTHESIS
   }
   ;

print_parametros_re
   : COMA expresion print_parametros_re
   | /*empty*/
   ;

print_stmt
   : PRINTF OPEN_PARENTHESIS print_parametros CLOSE_PARENTHESIS


nuevo_arreglo
   : OPEN_BRACKET expresion CLOSE_BRACKET nuevo_arreglo_re


nuevo_arreglo_re
   : OPEN_BRACKET expresion CLOSE_BRACKET nuevo_arreglo_re
arreglo_stmt
   : IDENTIFICADOR nuevo_arreglo    ;


magnitud
   :   OPEN_BRACKET expresion CLOSE_BRACKET magnitud_re
   | /*empty*/    ;

magnitud_re
   :   OPEN_BRACKET expresion CLOSE_BRACKET magnitud_re
   |  /*empty*/    ;

expresion
   : expresion AND expresion
 | expresion OR expresion
| NOT expresion
| expresion MAYOR expresion
|expresionMENORexpresion
   | expresion COMPARACION expresion
   | LIT_ENTERO
   | LIT_DECIMAL
   | LIT_CARACTER
```

```
    | LIT_STRING
    | IDENTIFICADOR
    | arreglo_stmt
    | PUNTERO IDENTIFICADOR magnitud %prec UPUNTERO
    | metodo_stmt
    | metodo_clase_stmt
    | OPEN_PARENTHESIS expresion CLOSE_PARENTHESIS {$$=$2;}
    ;

ini
    : code_c EOF {return reversaArreglo($1);}
    ;

paqueteria
    : INCLUDE LIT_STRING

code_c
    : paqueteria code_c
    | var_stmt COLON code_c
    | clase_stmt COLON code_c
    | main code_c

/*Start of Main*/
main
    : VOID MAIN OPEN_PARENTHESIS CLOSE_PARENTHESIS OPEN_CURLY statements
    ;

/*End of Main*/
/*Start of metodo y clases*/

/**/
/*Init of statements*/
empty_statements
    :  statement empty_statements
    | /*empty*/    ;

block_statements
    : OPEN_CURLY statements
    | statement    ;

statements
    :  statement statements {
    | CLOSE_CURLY
```

```
   ;


statement
   : var_stmt COLON
   | if_stmt
   | for_stmt
   | while_stmt      | do_stmt COLON
   | switch_stmt
 | CONTINUE COLON
/*End of statements*/

/*Init of For statement*/

accion_for
   : var_stmt
 | metodo_clase_stmt
   | clase_stmt
   | print_stmt
   | scan_stmt
   | clean_stmt
   | getch_stmt
   ;

for_inicio
   :var_stmt COLON {$$=$1;}
   ;

for_condicion
   : expresion COLON {$$=$1;}


for_stmt
       :  FOR  OPEN_PARENTHESIS  for_inicio  for_condicion  accion_for
CLOSE_PARENTHESIS   ;

/*End of for statement*/

/*Init of Var statement*/
data_type
   : INT
   | FLOAT
   | CHAR     ;
```

```
const_data
   : CONST data_type

arreglo
   : OPEN_BRACKET expresion CLOSE_BRACKET arreglo_re

arreglo_re
   : OPEN_BRACKET expresion CLOSE_BRACKET arreglo_re

valor_asignacion
   : IGUAL expresion
   | IGUAL getch_stmt
   | INCREMENTO
   | DECREMENTO
   | /*empty*/
   ;

var_stmt
   : const_data IDENTIFICADOR arreglo valor_asignacionlumna);
   }
         |   const_data   error      {addSyntaxError("Se   esperaba   un
identificador",$2,linea(this._$.first_line),
columna(this._$.first_column));}
   | IDENTIFICADOR arreglo valor_asignacion
   ;
/*End of Statement*/

/*Start of IF*/
if_stmt
   : IF OPEN_PARENTHESIS expresion CLOSE_PARENTHESIS block_statements
   | ELSE block_statements
/*Close of if*/
/*Start of while*/
while_stmt
          :   WHILE   OPEN_PARENTHESIS   expresion   CLOSE_PARENTHESIS
block_statements
/*End of while*/

/*Start of Do*/
do_stmt
   : DO OPEN_CURLY empty_statements CLOSE_CURLY while_do
```

```
while_do
   : WHILE OPEN_PARENTHESIS expresion CLOSE_PARENTHESIS
/*End of Do*/
/*Start of switch*/
switch_stmt
   : SWITCH OPEN_PARENTHESIS IDENTIFICADOR CLOSE_PARENTHESIS OPEN_CURLY
cases CLOSE_CURLY
;

cases
   :case_stmt cases
   |default_stmt
   | /*empty*/



switch_statement
   : var_stmt COLON
   | scan_stmt COLON
   | print_stmt COLON
   | clean_stmt COLON
   | getch_stmt COLON
   | metodo_stmt COLON
   | metodo_clase_stmt COLON
   | metodo_clase_stmt
   | clase_stmt COLON
   | if_stmt
   | for_stmt
   | while_stmt
   | do_stmt COLON
   | switch_stmt
   | CONTINUE COLON

switch_instructions
   : switch_statement switch_instructions
   | BREAK COLON
   ;

default_instructions
   : switch_statement default_instructions
   | /*empty*/
   ;
```

```
case_stmt
   : CASE expresion SEMI_COLON switch_instructions
   ;

default_stmt
   :DEFAULT SEMI_COLON default_instructions }
   ;
```

Gramatica Java

Expresion regular	token
package	PACKAGE
>=	MAYOR_IGUAL
<=	MENOR_IGUAL
!=	DIFERENTE
==	COMPARACION
&&	AND
&	PUNTERO
||	OR
!	NOT
+=	O_MAS
-=	O_MENOS
*=	O_POR
/=	O_DIV
%=	O_MOD
^=	O_POW
=	IGUAL
++	INCREMENTO
--	DECREMENTO
+	SUMA
-	RESTA

*	POR
/	DIV
%	MOD
^	POW
>	MAYOR
<	MENOR
int	INT
float	FLOAT
double	DOUBLE
boolean	BOOLEAN
char	CHAR
String	STRING
void	VOID
true	TRUE
false	FALSE
(	OPEN_PARENTHESIS
)	CLOSE_PARENTHESIS
[	OPEN_BRACKET
]	CLOSE_BRACKET
{	OPEN_CURLY
}	CLOSE_CURLY
;	COLON
:	SEMI_COLON
,	COMA
public	PUBLIC
private	PRIVATE
intinput	INTINPUT
floatinput	FLOATINPUT

charinput	CHARINPUT
return	RETURN
for	FOR
while	WHILE
do	DO
extends	EXTENDS
switch	SWITCH
default	DEFAULT
if	IF
else	ELSE
print	PRINT
println	PRINTLN
continue	CONTINUE
break	BREAK
case	CASE
class	CLASS
this	THIS
\"[^\"]*\"	CADENA
\'[^\']\'	CARACTER
[0-9]+(“.“[0-9]+)	DECIMAL
[0-9]+	ENTERO
[aA-zZ|"_"|"$"]([aA-zZ]|[0-9]|"_"|"$")*	IDENTIFICADOR
.	DOT

```
init_java
   : ini EOF
   ;

ini
```

```
   : class_stmt ini
   | /*empty*/
       ;

concatenate_values
       :expresion concatenate_values_re

concatenate_values_re
       :COMA expresion concatenate_values_re
       |CLOSE_PARENTHESIS
       ;

print
       :PRINT
       |PRINTLN
       ;

print_stmt
       :print OPEN_PARENTHESIS concatenate_values
       ;


identifier
       : PUBLIC
       | PRIVATE
       ;
data_type
       : INT
       | STRING
       | CHAR
       | BOOLEAN
       | FLOAT
       | DOUBLE
       | VOID
       ;

entry_stmt
   : INTINPUT
   | FLOATINPUT
   | CHARINPUT
   ;
```

```
this_stmt
   : THIS DOT IDENTIFICADOR
   | IDENTIFICADOR

extends_re
   :EXTENDS IDENTIFICADOR
   | /*empty*/    ;

class_stmt
       : PUBLIC CLASS IDENTIFICADOR extends_re OPEN_CURLY
class_instructions


class_instructions
       :  identifier class_instruction class_instructions
      | CLOSE_CURLY
       ;
ides
   :IDENTIFICADOR
   |PUNTERO IDENTIFICADOR
function_parameters
       : data_type ides function_parameters_re
       | CLOSE_PARENTHESIS
       ;

function_parameters_re
       :  COMA data_type ides function_parameters_re
       | CLOSE_PARENTHESIS

function_stmt
       :  IDENTIFICADOR OPEN_PARENTHESIS function_parameters OPEN_CURLY
instructions
variable_stmt
       : IDENTIFICADOR asignacion_variable variable_stmt_re

variable_stmt_re
       :  COMA IDENTIFICADOR asignacion_variable variable_stmt_re
       | COLON
       ;

class_statements
       : variable_stmt
       | function_stmt
```

```
constructor_class
       : IDENTIFICADOR OPEN_PARENTHESIS function_parameters OPEN_CURLY
instructions

class_instruction
       : data_type class_statements
       | constructor_class
       ;

instructions
       : instruction instructions
       | CLOSE_CURLY
       ;

instruction
       : variable
       | if_stmt
       | else_stmt
       | switch_stmt
       | for_stmt
       | while_stmt
       | do_stmt COLON
       | metodo COLON
       | print_stmt COLON
       | CONTINUE COLON
       | RETURN expresion COLON        ;

stmt_enclusure
   : OPEN_CURLY instructions
   | instruction
   ;
/*Variable*/
increm
   : INCREMENTO
   | DECREMENTO       ;

valor_variable
   :expresion
   |entry_stmt
   ;

metodo_asignacion
   :IGUAL
```

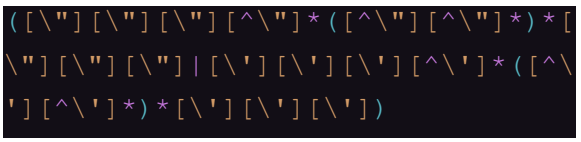
```
   | O_MAS
   | O_MENOS
   | O_POR
   | O_DIV
   | O_MOD
   | O_POW
   ;

igualacion_re
   : metodo_asignacion valor_variable igualacion_re

asignacion_variable
   : metodo_asignacion valor_variable igualacion_re
   |  /*empty*/
   ;

asignacion
   : metodo_asignacion valor_variable igualacion_re
   | increm
   |  /*empty*/
   ;

asignacion_post
   : metodo_asignacion valor_variable igualacion_re
   | increm
   ;

nombre_variables
   : this_stmt asignacion_variable nombre_variables_re

nombre_variables_re
   :  COMA this_stmt asignacion_variable nombre_variables_re
   | COLON    ;

variable
   : data_type nombre_variables
/*INIT of metodo*/
parameters
   : expresion parameters_re

expresion
   :expresion AND expresion
   |expresion OR expresion
```

```
   |NOT expresion
   |expresion MENOR expresion
   |expresion MENOR_IGUAL expresion
   |expresion MAYOR expresion
   |expresion MAYOR_IGUAL expresion
   |expresion DIFERENTE expresion
   |expresion COMPARACION expresion
   |expresion SUMA expresion
   |expresion RESTA expresion
   |expresion POR expresion
   |expresion DIV expresion
   |expresion MOD expresion
   |expresion POW expresion
   |OPEN_PARENTHESIS expresion CLOSE_PARENTHESIS
   |ENTERO
   |DECIMAL
   |RESTA expresion
   |this_stmt accion_increm
   |this_stmt
   |CADENA
   |CARACTER
   |booleanos
   |metodo
   ;

booleanos
   :TRUE
   |FALSE
   ;

accion_increm
   : INCREMENTO
   | DECREMENTO
   ;

metodo
   : IDENTIFICADOR OPEN_PARENTHESIS parameters
if_stmt
   : IF OPEN_PARENTHESIS block_condition stmt_enclusure

else_stmt
   : ELSE stmt_enclusure    ;
```

```
/*End of IF*/
/*Init of Switch*/

switch_instructions
       : switch_instruction switch_instruction
       | BREAK COLON


default_instructions
       : switch_instruction default_instructions
       | CLOSE_CURLY

switch_instruction
       : variable
       | if_stmt
       | else_stmt
       | switch_stmt
       | for_stmt
       | while_stmt
       | do_stmt COLON
       | CONTINUE COLON
       | RETURN expresion COLON  {
switch_stmt
   : SWITCH OPEN_PARENTHESIS IDENTIFICADOR CLOSE_PARENTHESIS OPEN_CURLY
cases_stmt

cases_stmt
   : CASE expresion SEMI_COLON switch_instructions cases_stmt
   | DEFAULT SEMI_COLON default_instructions
   | CLOSE_CURLY
   ;
/*End of Switch*/
/*Init For*/


this_asignacion
   :   this_stmt asignacion

declaracion_for
   : data_type this_stmt asignacion declaracion_for_re

   | this_asignacion declaracion_for_re
```

```
declaracion_for_re
   : COMA  this_asignacion declaracion_for_re{
   | COLON
   ;



for_stmt
   : FOR OPEN_PARENTHESIS for_inicio for_condition for_asignacion
stmt_enclusure
for_inicio
   : declaracion_for
   //| declaracion_for error {addSyntaxError("Una condicion era
esperada", $2, linea(this._$.first_line),
columna(this._$.first_column));}
   ;

for_condition
: expresion COLON

for_asignacion
   : declaraciones_post

/*End of For*/
/*Init of While*/

while_stmt
   : WHILE OPEN_PARENTHESIS block_condition stmt_enclusure
/*End of While*/

/*Init of DoWhile*/

do_stmt
   : DO stmt_enclusure while_do

while_do
   : WHILE OPEN_PARENTHESIS block_condition

   ;
```

GRAMATICA PYTHON

Expresion Regular	Token
\t	INDENTATION
	INDENTATION
	LIT_CADENA
def	DEF
print	PRINT
&	PUNTERO
println	PRINTLN
if	IF
else	ELSE
elif	ELIF
input	INPUT
return	RETURN
while	WHILE
break	BREAK
continue	CONTINUE
for	FOR
in	IN
range	RANGE
and	AND
or	OR
not	NOT
(	OPEN_PARENTHESIS
)	CLOSE_PARENTHESIS
[	OPEN_BRACKET
]	CLOSE_BRACKET

{	OPEN_CURLY
}	CLOSE_CURLY
,	COMA
>=	MAYOR_IGUAL
>	MAYOR
<	MENOR
<=	MENOR_IGUAL
!=	DIFERENTE
==	COMPARACION
+=	O_MAS
-=	O_RESTA
/=	O_DIV
*=	O_POR
%=	O_MOD
^=	O_POW
=	IGUAL
:	SEMI_COLON
+	SUMA
-	RESTA
/	DIV
*	POR
%	MOD
^	POW
`[0-9]+("."[0-9]+)`	LIT_DECIMAL
[0-9]+	LIT_ENTERO
`([\"][^\"]*[\"])`	LIT_CADENA

`([\'''][^\']*[\'])`	LIT_CADENA
True	LIT_TRUE
False	LIT_FALSE
`[aA-zZ|"_"]([aA-zZ]|[0-9]|"_")*`	IDENTIFICADOR

SINTACTICO

```
input
   :INPUT OPEN_PARENTHESIS CLOSE_PARENTHESIS


expresion
   :expresion AND expresion

   |expresion OR expresion

   |NOT expresion

   |expresion MAYOR expresion

   |expresion MAYOR_IGUAL expresion

   |expresion MENOR expresion

   |expresion MENOR_IGUAL expresion

   |expresion DIFERENTE expresion

   |expresion COMPARACION expresion

   |expresion SUMA expresion

   |expresion RESTA expresion

   |expresion POR expresion

   |expresion DIV expresion

   |expresion MOD expresion

   |expresion POW expresion
```

```
   |RESTA expresio

   |LIT_ENTERO

   |LIT_DECIMAL

   |LIT_CADENA

   |LIT_TRUE

   |LIT_FALSE

   |IDENTIFICADOR

   |OPEN_PARENTHESIS expresion CLOSE_PARENTHESIS

   ;

ini
   : statements EOF

   ;

ides
   :IDENTIFICADOR
   |PUNTERO IDENTIFICADOR


/*Funcion*/
parameters
   : ides parameters_re

   }
   |  /*empty*/

   ;

parameters_re
   :  COMA  ides parameters_re
   | /*empty*/

   ;
```

```
function_stmt
   :DEF IDENTIFICADOR OPEN_PARENTHESIS parameters CLOSE_PARENTHESIS
SEMI_COLON SPACE


/*End of Funcion*/

/*Declaraciones*/


statements
   : statements statement

   | statements function_stmt

   | /*empty*/


met_params
   :expresion met_params_re

   |/*empty*/

   ;

met_params_re

   |/*empty*/

   ;

metodo_stmt
   :IDENTIFICADOR OPEN_PARENTHESIS met_params CLOSE_PARENTHESIS SPACE

statement
   : var_stmt
   | if_stmt
   | for_stmt
   | while_stmt
   | print_stmt
   | metodo_stmt
   | CONTINUE SPACE
```

```
   | BREAK SPACE
   | RETURN expresion SPACE
   ;
/*End of Declaraciones*/

/*Print statement*/
print_parameter
   : expresion print_parameter_re
   ;

print_parameter_re
   :  COMA expresion print_parameter_re
   |  /*empty*/

print_method
   :PRINT
   |PRINTLN
   ;

print_stmt
   : print_method  OPEN_PARENTHESIS print_parameter CLOSE_PARENTHESIS
SPACE
/
if_stmt
   : IF expresion SEMI_COLON SPACE
   | ELIF expresion SEMI_COLON SPACE
   | ELSE SEMI_COLON SPACE

/*End of statement*/

/*For statement*/
for_parameters
   :expresion for_parameters_re


for_parameters_re
   :  COMA expresion for_parameters_re


   |  /*empty*/
   ;

rango
```

```
   :RANGE OPEN_PARENTHESIS for_parameters CLOSE_PARENTHESIS


for_stmt
   : FOR IDENTIFICADOR IN rango SEMI_COLON SPACE


/*While statement*/
while_stmt

/*Variable statement*/
igualaciones
   : IGUAL
   | O_MAS
   | O_RESTA
   | O_POR
   | O_DIV
   | O_POW
   | O_MOD
   ;

nombre_variables
   : IDENTIFICADOR nombre_variables_re

nombre_variables_re
   : COMA IDENTIFICADOR nombre_variables_re


expresiones
   :expresion expresiones_re

expresiones_re
   : COMA expresion expresiones_re

   |/*empty*/
   ;

asignacion
   : igualaciones expresiones asignacion_re

asignacion_re
   :  igualaciones expresiones asignacion_re
   |  igualaciones input asignacion_re
```

```
var_stmt
   : nombre_variables asignacion SPACE
```