// Un lenguaje de programación

B-minor () {

```
string Por ="Felipe Sanchez"
```

}

Contenidos

void Origen {

/* RWLZ Idealmente un lenguaje inspirado en la sintaxis de c#, que buscaba una compatibilidad con .net */

```
int main(){
    int a = 5;
    int b = 3;
    int c = a * b + 2;
    print(c);

    int result = c / 2;
    print(result);

    result -= 3;
    print(result);

    return 0;
}
```

RWLZ Idealmente un lenguaje inspirado en la sintaxis de c#, que buscaba una compatibilidad con .net

Pero por problemas en el camino y la falta de timpo, el lenguaje termino conviertiendose en la entrega de B-minor

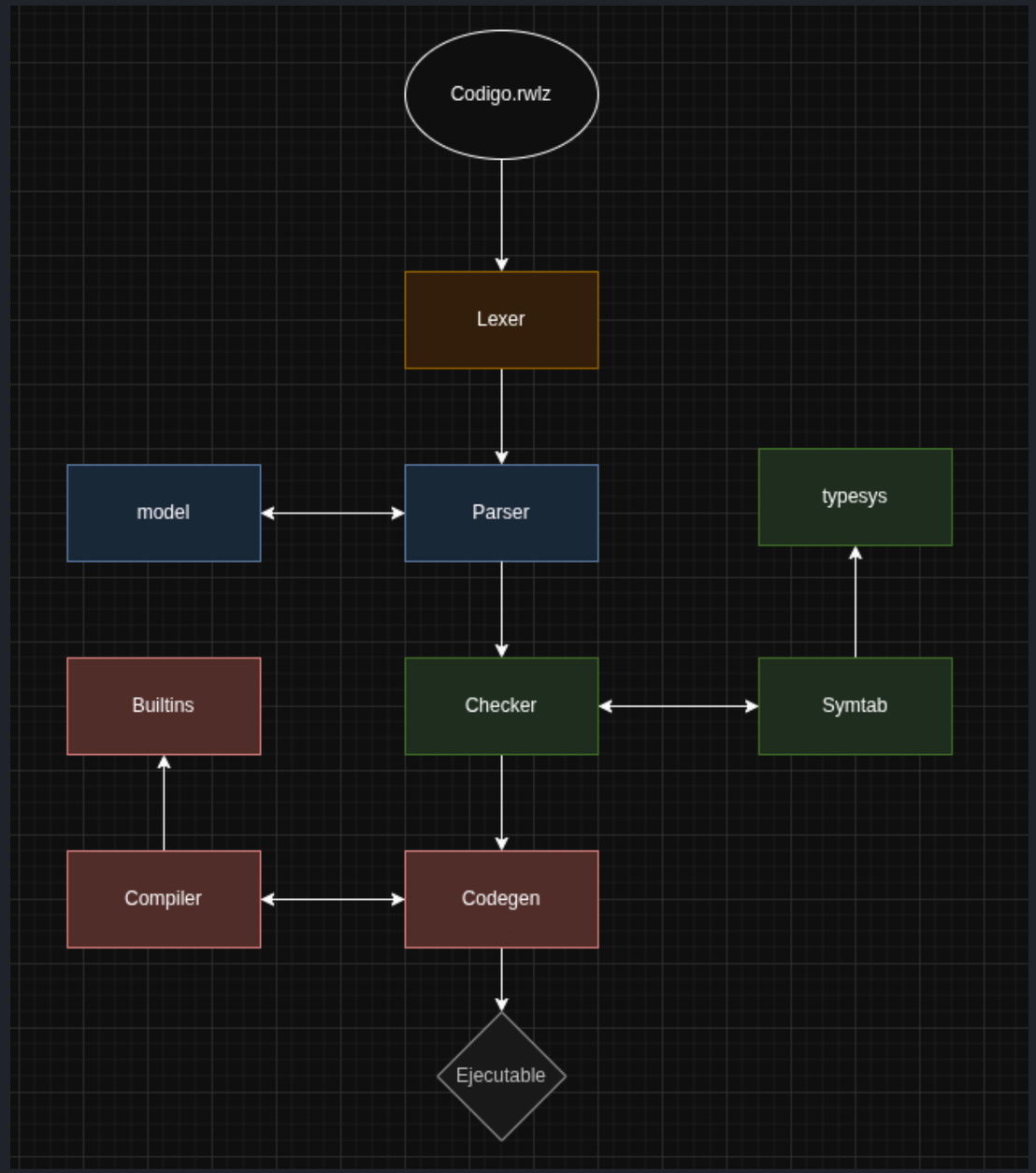
}

Ejemplo {

```
int show_result(int r1x, int r1y, int r2x,  int r2y) {
    print("=== Result ===");
    print("Closest pair found!");
    print("These two points are distance 1 apart in both x and y");


    print("Point 1: (" + r1x + ", " + r1y + ")");
    print("Point 2: (" + r2x + ", " + r2y + ")");

    print("Algorithm completed successfully!");
    return 1;
}


int main() {
    print("=== Closest Pair of Points - Brute Force O(n^2) ===");
    show_result(0, 0, 1, 1);
}
```

}

Ejemplo {


Codigo.rwlz
Lexer
model
Parser
typesys
Builtins
Checker
Symtab
Compiler
Codegen
Ejecutable

}

Lexer {

```
tokens = {
    BASE, BREED, PROP, HOOK,
    PRINT,
    ID, ARRAY, AUTO,
    INTEGER_LITERAL, FLOAT_LITERAL, STRING_LITERAL, CHAR_LITERAL,
    INT, FLOAT, BOOL, CHAR, STRING, VOID,
    IF, ELSE, RETURN, TRUE, FALSE,
    FOR, WHILE, BREAK, CONTINUE,
    EQ, NEQ, LE, GE, LT, GT, ASSIGN,
    PLUS, MINUS, TIMES, DIVIDE, MODULO,
    INCREMENT, DECREMENT,
    PLUS_ASSIGN, MINUS_ASSIGN, TIMES_ASSIGN, DIVIDE_ASSIGN,
    AND, OR, NOT,
    CONST,
    BEPINPLUGIN
}
```

BOOL	bool	5	126	130
ID	main	5	131	135
(	(	5	135	136
INT	int	5	136	139
ID	speed	5	140	145
,	,	5	145	146
FLOAT	float	5	147	152
ID	power	5	153	158
)	)	5	158	159
{	{	5	160	161
INT	int	6	166	169
ID	steps	6	170	175
ASSIGN	=	6	176	177
INTEGER_LIT…	10	6	178	180
;	;	6	180	181
PRINT	print	7	186	191
(	(	7	191	192
STRING_LITE…	"moviendo al jugador"	7	192	213
)	)	7	213	214
;	;	7	214	215
IF	if	8	220	222
(	(	8	223	224
ID	steps	8	224	229
GT	>	8	230	231
INTEGER_LIT…	5	8	232	233
)	)	8	233	234
{	{	8	235	236
RETURN	return	1	275	281
FALSE	false	1	282	287
;	;	1	287	288
}	}	2	293	294
ELSE	else	2	295	299
{	{	2	300	301
RETURN	return	3	310	316
TRUE	true	3	317	321
;	;	3	321	322
}	}	4	327	328
}	}	5	329	330

}

Reglas para Identificadores {

REGEX: [A-Za-z_][A-Za-z0-9_]*

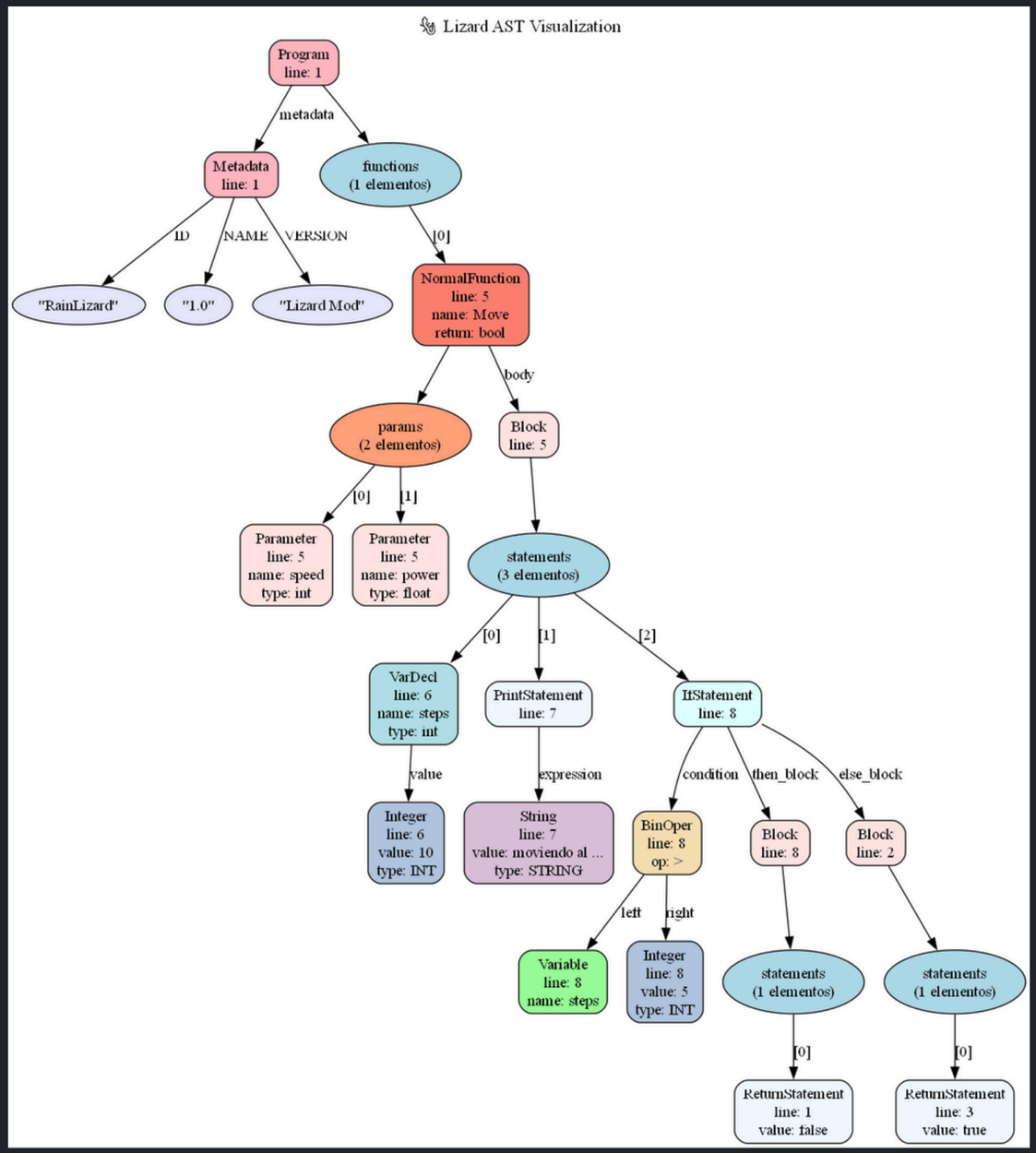
Deben comenzar con una letra o “_”

Pueden contener letras, números y “_”

Token asignado: ID

}

Parser {


Lizard AST Visualization
Program
line: 1
metadata
Metadata
line: 1
functions
(1 elementos)
ID
NAME
VERSION
[0]
"RainLizard"
"1.0"
"Lizard Mod"
NormalFunction
line: 5
name: Move
return: bool
body
params
(2 elementos)
Block
line: 5
[0]
[1]
Parameter
line: 5
name: speed
type: int
Parameter
line: 5
name: power
type: float
statements
(3 elementos)
[0]
[1]
[2]
VarDecl
line: 6
name: steps
type: int
PrintStatement
line: 7
IfStatement
line: 8
value
expression
condition
then_block
else_block
Integer
line: 6
value: 10
type: INT
String
line: 7
value: moviendo al ...
type: STRING
BinOper
line: 8
op: >
Block
line: 8
Block
line: 2
left
right
Variable
line: 8
name: steps
Integer
line: 8
value: 5
type: INT
statements
(1 elementos)
statements
(1 elementos)
[0]
[0]
ReturnStatement
line: 1
value: false
ReturnStatement
line: 3
value: true

}

Semantico {

Sintaxis inspirada en c

symbol table: global
(parent: *None*)
(symbols listed in declaration order)

key	value
print	builtin function: (string value) -> void
Move	user function: (int speed, float power) -> bool

symbol table: Move (defined at line 5)
parent: global
return type: bool
parameters:
(symbols listed in declaration order)

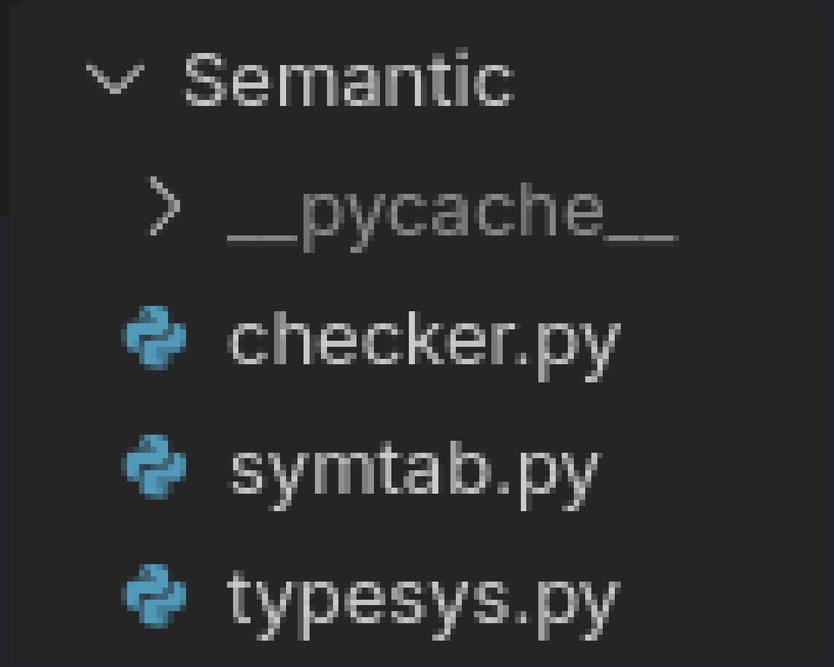
key	value
speed	int [init: ✓, used: ✗]
power	float [init: ✓, used: ✗]

block:
(symbols listed in declaration order)

key	value
steps	variable: int [init: ✓, used: ✓]

}

```python
def check(self, ast: Program) -> bool:
    """
    Main entry point for semantic checking.
    Returns True if no errors were found.
    """
    self.errors = 0
    self.warnings = 0

    try:
        self.visit(ast)
    except Exception as e:
        error(f"Internal error during semantic analysis: {e}")
        self.errors += 1

    return self.errors == 0
```



```python
class RWLZType:
    base_type: BaseType
    is_array: bool = False
    is_const: bool = False
```

```python
class BaseType(Enum):
    INT = "int"
    FLOAT = "float"
    BOOL = "bool"
    CHAR = "char"
    STRING = "string"
    VOID = "void"
```

```python
PROMOTION_RULES = {
    BaseType.INT: {BaseType.FLOAT},
    BaseType.FLOAT: {BaseType.INT},
    BaseType.CHAR: {BaseType.INT, BaseType.STRING},
    BaseType.BOOL: {BaseType.INT},
}
```

LLVM {

Sintaxis inspirada en c

```
self.type_map = {
    BaseType.INT: ir.IntType(32),
    BaseType.FLOAT: ir.DoubleType(),
    BaseType.BOOL: ir.IntType(1),
    BaseType.CHAR: ir.IntType(8),
    BaseType.VOID: ir.VoidType(),
    BaseType.STRING: ir.IntType(8).as_pointer()
}
```

```
# Platform-specific linking
if self.platform == "Linux":
    # Use gcc with -no-pie flag for Linux
    link_cmd = ['gcc', obj_filename, '-o', output_filename, '-no-pie']

elif self.platform == "Darwin":  # macOS
    # Use clang for macOS
    link_cmd = ['clang', obj_filename, '-o', output_filename]

elif self.platform == "Windows":
    # Use gcc with .exe extension for Windows
    if not output_filename.endswith('.exe'):
        output_filename += '.exe'
    link_cmd = ['gcc', obj_filename, '-o', output_filename]
```

}