Tecnológico de Monterrey

Proyecto Final para la clase Diseño de Compiladores FJ22: Parejas

Lenguaje Mogavity

Profesores:
Elda Quiroga González
Héctor Ceballos

Equipo:

Clarissa Andrea Velásquez Magaña A01281743

Jaime Hisao Yesaki Hinojosa A01720044

[illegible] [illegible]

Monterrey, Nuevo León, México
26 de marzo del 2022

Index

Project Description

I. Purpose and Scope

The purpose of this project is to apply all the knowledge we have acquired from our courses of Computer Science Engineering, including programming fundamentals, object-oriented programming, data structures, algorithms, computational mathematics, and more. We envision an easy to use and friendly programming language. Our scope is to create a basic object-oriented programming language. It will support mechanisms such as defining classes, attributes, methods and single hierarchy. It will be similar to C++ but with some differences and rules.

II. Language requirements and test cases

A. Requirements

- The following are the requirements of Mogavity:
- Have int, float and objects from class as data type
- Have at least one uncontrolled cycle (while, do while)
- Have at least one controlled cycle (for)
- Implement the creation of functions
- Implement the creation of Classes
    - Classes need to have attributes, methods and a constructor
    - Classes can inherit from another class
- Support arithmetic expressions
- Support relational operation expressions
- Support logical operation expressions
- Support function calls
- Functions can be of type void, int or float
- Have at least one conditional (if)
- Have at least one-dimensional and two-dimensional array
- The user can input data
- It supports printing into the console

B. Principal Test Cases

Test - Arrays and arithmetic operations
Assigns values to some cells from arrays, and multiplies arrays.
Test Code

```
program test;

var int A[5];

main {
    A[2] = 7;
    A[3] = 5;
    output -> A[2];
    output -> A[3];
    A[1] = A[2] * A[3];
    output -> A[1];
    A[1] = A[2] * A[3] * A[2] * A[3] * A[2] * A[3] * A[2] * A[3] * A[2]
* A[3] * A[2] * A[3] * A[2] * A[3];
    output -> A[1];
    return 0;
}
```

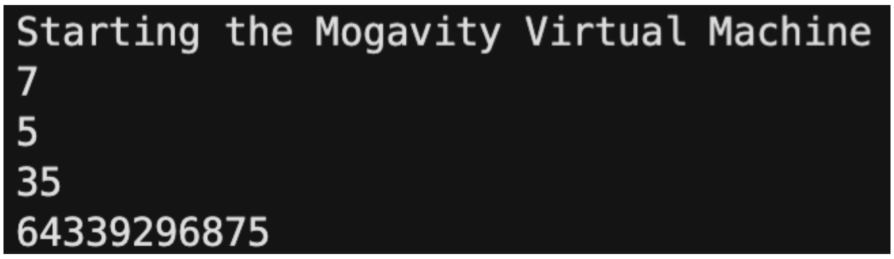
Intermediate Code

```
[1,GOTO,None,None,2]
[2,VERIFY,150000,0,4]
[3,+,150000,150001,140000]
[4,+,140000,150002,160000]
[5,=,150003,None,160000]
[6,VERIFY,150004,0,4]
[7,+,150004,150001,140001]
[8,+,140001,150002,160001]
[9,=,150005,None,160001]
[10,VERIFY,150000,0,4]
[11,+,150000,150001,140002]
[12,+,140002,150002,160002]
[13,OUTPUT,None,None,160002]
[14,VERIFY,150004,0,4]
[15,+,150004,150001,140003]
[16,+,140003,150002,160003]
[17,OUTPUT,None,None,160003]
[18,VERIFY,150006,0,4]
[19,+,150006,150001,140004]
[20,+,140004,150002,160004]
[21,VERIFY,150000,0,4]
[22,+,150000,150001,140005]
[23,+,140005,150002,160005]
[24,VERIFY,150004,0,4]
[25,+,150004,150001,140006]
[26,+,140006,150002,160006]
[27,*,160005,160006,140007]
[28,=,140007,None,160004]
[29,VERIFY,150006,0,4]
[30,+,150006,150001,140008]
[31,+,140008,150002,160007]
[32,OUTPUT,None,None,160007]
```

```
[33,VERIFY,150006,0,4]
[34,+,150006,150001,140009]
[35,+,140009,150002,160008]
[36,VERIFY,150000,0,4]
[37,+,150000,150001,140010]
[38,+,140010,150002,160009]
[39,VERIFY,150004,0,4]
[40,+,150004,150001,140011]
[41,+,140011,150002,160010]
[42,*,160009,160010,140012]
[43,VERIFY,150000,0,4]
[44,+,150000,150001,140013]
[45,+,140013,150002,160011]
[46,*,140012,160011,140014]
[47,VERIFY,150004,0,4]
[48,+,150004,150001,140015]
[49,+,140015,150002,160012]
[50,*,140014,160012,140016]
[51,VERIFY,150000,0,4]
[52,+,150000,150001,140017]
[53,+,140017,150002,160013]
[54,*,140016,160013,140018]
[55,VERIFY,150004,0,4]
[56,+,150004,150001,140019]
[57,+,140019,150002,160014]
[58,*,140018,160014,140020]
[59,VERIFY,150000,0,4]
[60,+,150000,150001,140021]
[61,+,140021,150002,160015]
[62,*,140020,160015,140022]
[63,VERIFY,150004,0,4]
[64,+,150004,150001,140023]
[65,+,140023,150002,160016]
[66,*,140022,160016,140024]
[67,VERIFY,150000,0,4]
[68,+,150000,150001,140025]
[69,+,140025,150002,160017]
[70,*,140024,160017,140026]
[71,VERIFY,150004,0,4]
[72,+,150004,150001,140027]
[73,+,140027,150002,160018]
[74,*,140026,160018,140028]
[75,VERIFY,150000,0,4]
[76,+,150000,150001,140029]
[77,+,140029,150002,160019]
[78,*,140028,160019,140030]
[79,VERIFY,150004,0,4]
[80,+,150004,150001,140031]
[81,+,140031,150002,160020]
[82,*,140030,160020,140032]
[83,VERIFY,150000,0,4]
[84,+,150000,150001,140033]
[85,+,140033,150002,160021]
[86,*,140032,160021,140034]
```

```
[87,VERIFY,150004,0,4]
[88,+,150004,150001,140035]
[89,+,140035,150002,160022]
[90,*,140034,160022,140036]
[91,=,140036,None,160008]
[92,VERIFY,150006,0,4]
[93,+,150006,150001,140037]
[94,+,140037,150002,160023]
[95,OUTPUT,None,None,160023]
[96,RETURN,None,None,None]
[97,ENDFUNC,None,None,None]
[98,EOF,None,None,None]
```

Output



```
Starting the Mogavity Virtual Machine
7
5
35
64339296875
```

Test - While

Does a while loop until the control variable reaches 10

Test Code

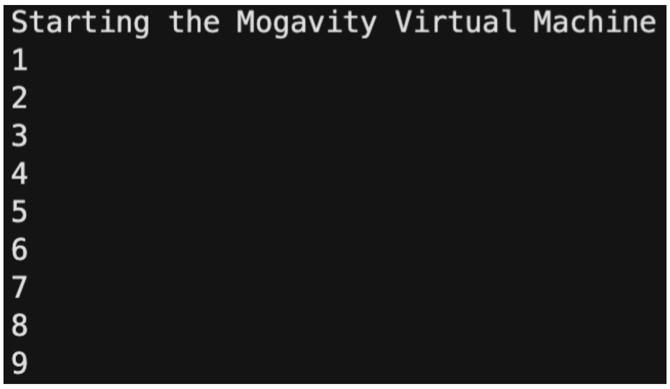
```
program test;

var int a;

main {
   a = 1;
   while (a < 10){
    output -> a;
    a = a + 1;
   }
    return 0;
}
```

Intermediate Code

```
[1,GOTO,None,None,2]
[2,=,150000,None,110001]
[3,<,110001,150001,140000]
[4,GOTOF,140000,None,9]
```

[5,OUTPUT,None,None,110001]
[6,+,110001,150000,140001]
[7,=,140001,None,110001]
[8,GOTO,None,None,3]
[9,RETURN,None,None,None]
[10,ENDFUNC,None,None,None]
[11,EOF,None,None,None]

Output



```
Starting the Mogavity Virtual Machine
1
2
3
4
5
6
7
8
9
```

Test - Function Scope

This test aims to see how Mogavity handles different scopes and function calls. It tests the scope management functionality and function calls.

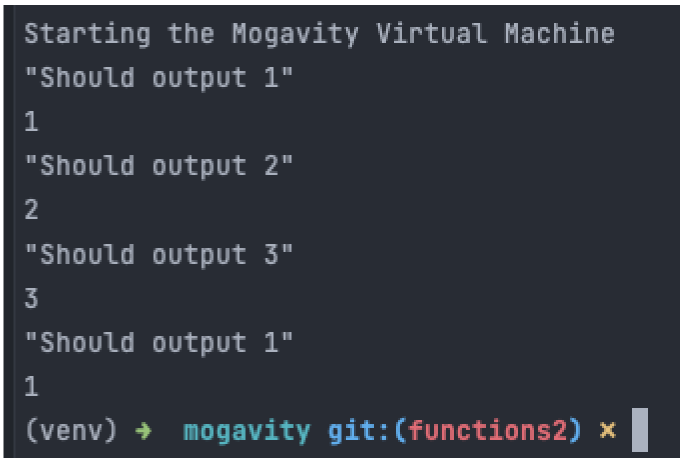
Test Code

```
program test;

var int a, b, c, d;

instr void local2()
{
   var int a, b, c;
   a = 3;
   output -> "Should output 3";
   output -> a;
   return 0;
}

instr void local()
{
   var int a, b, c;
   a = 2;
   output -> "Should output 2";
   output -> a;
   _local2();
   return 0;
}

main{
   output -> "Should output 1";
   a = 1;
   output -> a;
   _local();
   output -> "Should output 1";
   output -> a;
   return 0;
}
```

Intermediate Code

```
[1,GOTO,None,None,14]
[2,=,150000,None,10000]
[3,OUTPUT,None,None,150001]
[4,OUTPUT,None,None,10000]
[5,RETURN,None,None,None]
[6,ENDFUNC,None,None,None]
[7,=,150003,None,10000]
[8,OUTPUT,None,None,150004]
[9,OUTPUT,None,None,10000]
[10,ERA,None,None,local2]
[11,GOSUB,local2,None,2]
[12,RETURN,None,None,None]
[13,ENDFUNC,None,None,None]
[14,OUTPUT,None,None,150005]
[15,=,150006,None,110001]
[16,OUTPUT,None,None,110001]
[17,ERA,None,None,local]
[18,GOSUB,local,None,7]
[19,OUTPUT,None,None,150005]
[20,OUTPUT,None,None,110001]
[21,RETURN,None,None,None]
[22,ENDFUNC,None,None,None]
[23,EOF,None,None,None]
```

Output



```
Starting the Mogavity Virtual Machine
"Should output 1"
1
"Should output 2"
2
"Should output 3"
3
"Should output 1"
1
(venv) ➜  mogavity git:(functions2) ✗
```

Test - And and Or

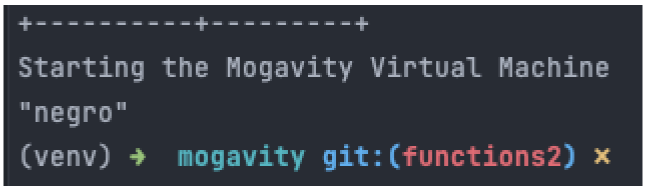
This test will try the functionality of our AND and OR operators when used in conditionals.

Test Code

```
program test;

var int a, b, c, d;

main {

   a = 1;
   b = 2;
   c = 3;
   d = 4;

   if ((a > b or c < d) and (c < d and a < d)) {
       output -> "negro";
   }
   otherwise {
       output -> "blanco";
   }
}
```

Intermediate Code

```
[1,GOTO,None,None,2]
[2,=,150000,None,110001]
[3,=,150001,None,110002]
[4,=,150002,None,110003]
[5,=,150003,None,110004]
[6,>,110001,110002,140000]
[7,<,110003,110004,140001]
[8,or,140000,140001,140002]
```

[9,<,110003,110004,140003] [10,<,110001,110004,140004] [11,and,140003,140004,140005] [12,and,140002,140005,140006] [13,GOTOF,140006,None,16] [14,OUTPUT,None,None,150004] [15,GOTO,None,None,17] [16,OUTPUT,None,None,150005] [17,ENDFUNC,None,None,None] [18,EOF,None,None,None]
Output


```
+---------+---------+
Starting the Mogavity Virtual Machine
"negro"
(venv) ➜  mogavity git:(functions2) ✗
```

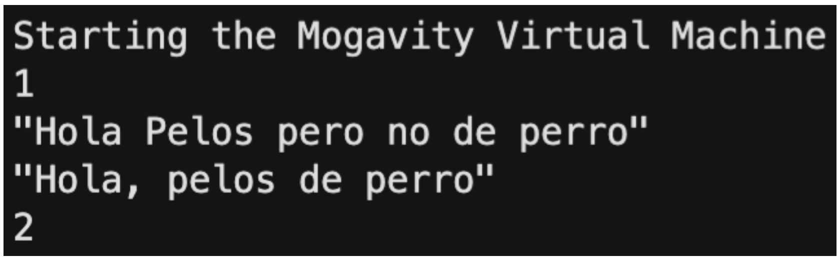
Test - Class
Declares a class “Perro” and assigns values to its attributes and demostrates the functionality of the class’ method.
Test Code

```
program test;

class Perro {
    attr:
        int a, b;
    constr:
        Perro(){

        }
    methods:
        method void pelos(){
            output -> "Hola, pelos de perro";
        }
}

var Perro perro;
var int a;

instr void pelos(){
     output -> "Hola Pelos pero no de perro";
     return 0;
}

main {
    perro.a = 1;
    perro.b = 2;
    output -> perro.a;
    _pelos();
    _perro.pelos();
    output -> perro.b;
    return 0;
}
```

Intermediate Code

```
[1,GOTO,None,None,7]
[2,OUTPUT,None,None,150000]
[3,ENDFUNC,1,None,None]
[4,OUTPUT,None,None,150001]
[5,RETURN,None,None,None]
[6,ENDFUNC,None,None,None]
[7,=,150003,None,110001]
[8,=,150004,None,110002]
[9,OUTPUT,None,None,110001]
[10,ERA,None,None,pelos]
[11,GOSUB,pelos,None,4]
[12,ERA,None,None,perro.pelos]
[13,GOSUB,2,None,2]
[14,OUTPUT,None,None,110002]
[15,RETURN,None,None,None]
[16,ENDFUNC,None,None,None]
[17,EOF,None,None,None]
```

Output



```
Starting the Mogavity Virtual Machine
1
"Hola Pelos pero no de perro"
"Hola, pelos de perro"
2
```

Test - Class inheritance

Declares a class “Animal” and a class “Perro”. Assigns values to its attributes and demonstrates the functionality of the parent class’ method, using the child object.

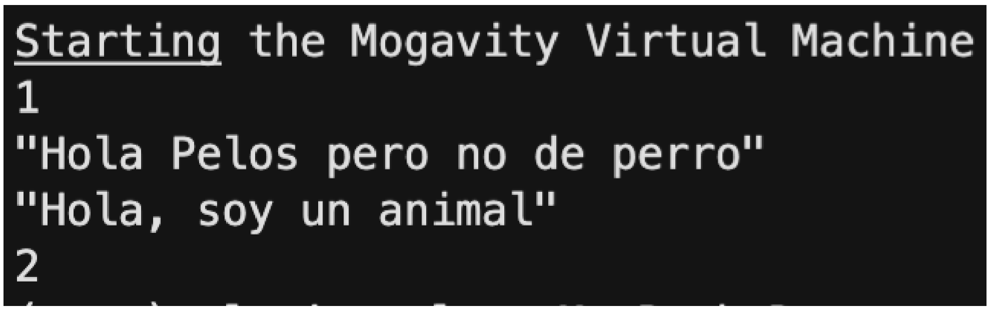
Test Code

```
program test;

class Animal {
    attr:
        int animal;
    constr:
        Animal(){
            output -> "Un animal nace!";
        }
    methods:
        method void que_soy(){
            output -> "Hola, soy un animal";
        }
}


class Perro inherits Animal {
    attr:
        int a, b;
    constr:
        Perro(){

        }
    methods:
        method void pelos2(){
            output -> "Hola, pelos de perro";
        }
}


var Perro perro;
var int a;

instr void pelos(){
     output -> "Hola Pelos pero no de perro";
     return 0;
}

main {
    perro.a = 1;
    perro.b = 2;
    output -> perro.a;
    _pelos();
    _perro.que_soy();
    output -> perro.b;
    return 0;
}
```

Intermediate Code

```
[1,GOTO,None,None,10]
[2,OUTPUT,None,None,150000]
[3,OUTPUT,None,None,150001]
[4,ENDFUNC,1,None,None]
[5,OUTPUT,None,None,150002]
[6,ENDFUNC,1,None,None]
[7,OUTPUT,None,None,150003]
[8,RETURN,None,None,None]
[9,ENDFUNC,None,None,None]
[10,=,150005,None,110002]
[11,=,150006,None,110003]
[12,OUTPUT,None,None,110002]
[13,ERA,None,None,pelos]
[14,GOSUB,pelos,None,7]
[15,ERA,None,None,perro.que_soy]
[16,GOSUB,2,None,3]
[17,OUTPUT,None,None,110003]
[18,RETURN,None,None,None]
[19,ENDFUNC,None,None,None]
[20,EOF,None,None,None]
```

Output



```
Starting the Mogavity Virtual Machine
1
"Hola Pelos pero no de perro"
"Hola, soy un animal"
2
```

Test - For

Inside the for it assigns a value to an array cell and prints the value.

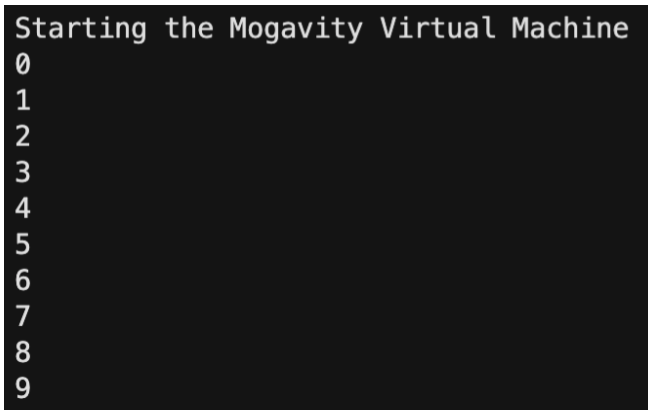
Test Code

```
program test;

var int a,b,c, D[10];

main{
    b = 0;
    for (a = 0; 10; a+=1){
      D[a] = a;
      output -> D[a];

    }
    return 0;
}
```

Intermediate Code

```
[1,GOTO,None,None,2]
[2,=,150000,None,110002]
[3,=,150000,None,110001]
[4,=,110001,None,140000]
[5,=,150001,None,140001]
[6,<,140000,140001,140002]
[7,GOTOF,140002,None,20]
[8,VERIFY,110001,0,9]
[9,+,110001,150000,140003]
[10,+,140003,150002,160000]
[11,=,110001,None,160000]
[12,VERIFY,110001,0,9]
[13,+,110001,150000,140004]
[14,+,140004,150002,160001]
[15,OUTPUT,None,None,160001]
[16,+,140000,150003,140005]
[17,=,140005,None,140000]
[18,=,140005,None,110001]
[19,GOTO,None,None,6]
[20,RETURN,None,None,None]
[21,ENDFUNC,None,None,None]
[22,EOF,None,None,None]
```

Output



```
Starting the Mogavity Virtual Machine
0
1
2
3
4
5
6
7
8
9
```

Test - Input and Output

The program asks the user two numbers and multiplies them and outputs the answer.

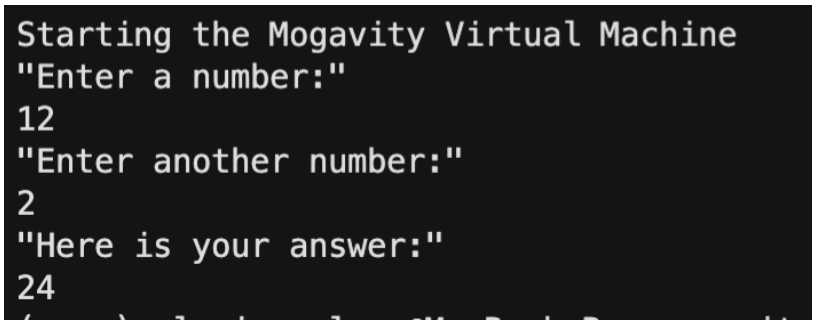
Test Code

```
program test;

var int a, b, c;

main {
    output -> "Enter a number:";
    input <- a;
    output -> "Enter another number:";
    input <- b;

    c = a*b;

    output -> "Here is your answer:", c;
}
```

Intermediate Code

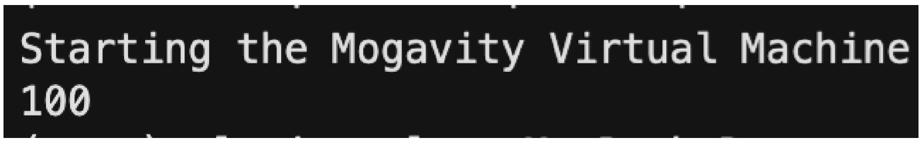
```
[1,GOTO,None,None,2]
[2,OUTPUT,None,None,150000]
[3,INPUT,None,None,110001]
[4,OUTPUT,None,None,150001]
[5,INPUT,None,None,110002]
[6,*,110001,110002,140000]
[7,=,140000,None,110003]
[8,OUTPUT,None,None,150002]
[9,OUTPUT,None,None,110003]
[10,ENDFUNC,None,None,None]
[11,EOF,None,None,None]
```

Output



```
Starting the Mogavity Virtual Machine
"Enter a number:"
12
"Enter another number:"
2
"Here is your answer:"
24
```

Test - Left association

Obtains the number 100 from different arithmetic operations.
Test Code
```
program test;

var int a;

main {
    a = (7 + (6 / 2)) * (20-10);
    output -> a;
    return 0;
}
```
Intermediate Code
[1,GOTO,None,None,2] [2,/,150001,150002,140000] [3,+,150000,140000,140001] [4,-,150003,150004,140002] [5,*,140001,140002,140003] [6,=,140003,None,110001] [7,OUTPUT,None,None,110001] [8,RETURN,None,None,None] [9,ENDFUNC,None,None,None] [10,EOF,None,None,None]
Output


```
Starting the Mogavity Virtual Machine
100
```

III. General Project Development Description

A. Weekly Record

First week of development
We developed the proposal for Mogavity. We met up to see how we will structure the language and see what we will try to accomplish. For the proposal we created the syntax diagrams and the structure for every item will be needed.

Second week of development

For the second week we generated the lexer and the grammatical rules for Mogavity. We are aware that we can make some changes to the lexer and the rules.

Third week of development
We created our function directory, for that we created a class that includes the methods that will help us control our function directory. Inside the function directory, we implemented the variable table. We also established the neurological points for the basic structures. We also implemented the semantic cube.

Fourth week of development
This week we started implementing the creation of quadruples and the stacks needed, such as the Poper and the StackO. We also started implementing the intermediate code for arithmetic, relational and logical expressions. During this week we also decided to remove the chars from the data type we would manage, one, because it was going into the chars rule instead of the integer or float rules, two, because at the end they weren't so relevant.

Fifth week of development
This week we corrected some errors that the function directory and the semantic cube were generating. We completed the intermediate code for expressions, conditionals and for the while cycle. We also decided to stop implementing the "elif" statement because it was causing us a lot of problems. If we had time, we would try to implement it. We also started to generate the code for the *for* cycle. We also started to divide the memory.

Sixth week of development
We finished implementing the intermediate for the *for cycle*. We decided to make our *for cycle* only incremental as we were going to have trouble making sure the control variable and the final variable were obeying the rule. We started making the intermediate code for the functions. We adjusted the generation of quadruples to assign space in memory. The function directory already accepts constants and has its own table. We created the memory manager class.

Seventh week of development
We finished all the intermediate code for the functions. We implemented the execution of the expressions and sequential statements. We also started to implement the execution of the functions but we haven't finished them. The memory is almost finished, we only need to see some stuff about the parameters of the functions.

Eighth week of development
For this week, we implemented the intermediate code for the arrays. The virtual machine already manages different scopes. We also store the values for the

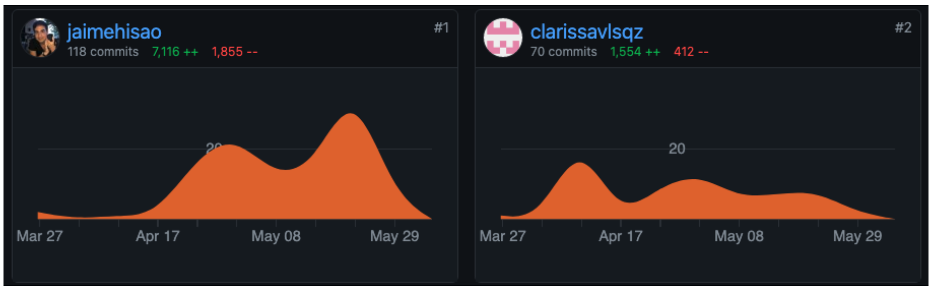
functions that return a value. We finished implementing the intermediate code for functions. We also tested recursive functions.

Ninth week of development

We started implementing the object-oriented part of Mogavity. We only lack the inheritance part. We are trying to fix some aesthetic problems. We started the documentation.

B. Git History

A detailed list of commits can be found in our GitHub repository, branches master and functions2 contain our code and history. Our commit list extracted from GitHub was too large to make it fit in an image.
This image represents our commits history. It is worth mentioning that Jaime commits more frequently than Clarissa, so there are more commits in the project's history.



C. Reflections

Clarissa Andrea Velásquez Magaña

I definitely underestimated what it takes to make a compiler. It is not hard after having the professor explaining to us how some parts work. But it does take time and a lot of testing. Hisao and I tried to be as aligned as possible to the calendar, but even being a little bit behind schedule, it took a toll at the end. We didn't test enough during the previous weeks, that at the end we noticed all the errors that we had. Time was the most valuable thing we had in this project.

Apart from the experience of creating the compiler. I think it was interesting and fun to use all the knowledge I have learned throughout these 8 semesters of university. I have seen how I have better myself as a programmer, each semester and this project is an evidence of it. I believe that my partner and I made a great team as we complemented each other. Some stuff that I wouldn't have come up with, Hisao would do, and vice versa.

[illegible]

Jaime Hisao Yesaki Hinojosa

This project taught me a lot about what a real project looks like, it allowed me to “compile” all of my knowledge acquired in my 8 semesters of university. It also allowed me to experience a more advanced team effort, because in this project, both programmers were involved throughout the development and important decisions. I believe that Clarissa and I worked together to handle problems as they appeared and formed a great team that complemented each other.

[illegible]

IV. Language Description

A. Language Name

Mogavtiy, the inspiration for the name comes from combining Mocha and Gravity, the names each one of us wanted to propose.

B. General characteristics

Mogavity is a simple and basic programming language. Its syntax was made with the intention to be readable and user-friendly . Our intention is that new programmers use Mogavity so they can familiarize with how coding works. From then on, they can go to more extravagant languages. The language has inspiration from C++.

Mogavity is an object oriented programming language. But because it's basic, it can only handle the creation of classes with its attributes and methods. It only has a single inheritance. Mogavity can only handle data types of *integers*, *floats* and objects created from classes. But the user can print *strings*, and use *booleans* for the conditionals. The conditionals are also very basic and simple, as it only includes *if* and *otherwise* statements. We have one uncontrolled cycle that is the *while* and one controlled cycle that is the *for*. The *for* cycle can be incremented by adding any integer or multiplying any number. Mogavity handles one-dimensional and two-dimensional *arrays* of type integer and floating point numbers. It can also do *expressions* that involve arithmetic, logical and relational operators. *Functions* can also be created within Mogavity. The types of functions that can be created are void, integer, and floating point number.

C. Errors Mogavity can handle

Compilation Errors	
Error Message	**Causes**

`"Class you are trying to inherit does not exist!"`	The users tries to inherit from a non-existent class
`"Constructor must have the same name of the class!"`	The user does not create the constructor with the same name as the class
`"No class exists with name class_name"`	The user is trying to access a non-existent class
`"Variable has not been declared"`	The user tries to use a non-existent variable
`"Type mismatch at line_num"`	The user tries to make an operation with invalid types
`"Expected type bool"`	The user is not including a boolean expression inside an if or a while
`"Type mismatch on for statement in line "`	The user is including an non-integer inside the for
`"Type mismatched in parameters in scope"`	One or more of the parameters is not the same type as the function’s signature
`"Amount of parameters is incorrect for scope "`	The amount of parameters is not the same as the function’s signature
`"Too many Integer variables, please optimize!"`	The number of Integer variables exceeds 10000.
`"Too many Float variables, please optimize!"`	The number of Float variables exceeds 10000.
`"Too many Temporary variables, please optimize your operations!"`	The number of Temporal variables exceeds 10000.
`"Too many Constants, please optimize your operations!"`	The number of Constant variables exceeds 10000.
`"Too many Pointers!"`	The number of Pointer variables exceeds 10000.
`"There's no file with that name"`	The file to be read does not exist
Execution errors	
`"Variable you are trying to access has not been declared previously!"`	The variable being accessed in code does not have a value in memory as it is being accessed before being assigned a value.
`"Array access is out of bounds!"`	This error is given when you try to access an element in the array beyond the original space declaration.

V. Compiler Description

A. Language, Libraries and Equipment used

The virtual machine runs on Python, and uses no external libraries to run. However we do use a Stack library of our own creation and our FunctionDirectory and Quadruple objects. Speaking of hardware and equipment, we developed on macOS computers with Python 3.8.2 using PyCharm and VSCode as IDEs.

B. Description of Lexical Analysis

Basic elements (Tokens)

Token name	Regular Expression	VALUE
ID	[a-zA-Z]+(_?[a-zA-Z0-9]+)*	any string
PLUS	\+	+
MINUS	-	-
TIMES	*	*
DIVIDE	/	/
LESSTHAN	<	<
GREATERTHAN	>	>
EQUALS	==	==
NOTEQUAL	!=	!=
ASSIGNMENT	\=	=
EQUALGREATERTHAN	>=	>=
EQUALLESSTHAN	<=	<=
PLUSEQUAL	+=	+=
TIMESEQUAL	*=	*=
LEFTARROW	<-	<-
RIGHTARROW	->	->
LEFTPARENTHESIS	\(	(
RIGHTPARENTHESIS	\)	)
LEFTCURLYBRACKET	\{	{
RIGHTCURLYBRACKET	\}	}

LEFTBRACKET	\[	[
RIGHTBRACKET	\]	]
CTE_STRING	"([^\\"\n]+|\\.)*"	any string
CTE_FLOAT	[0-9]+\.[0-9]+([Ee][+-]?[0-9]*)?	positive float
CTE_INT	[0-9]+	positive integer
CTE_CHAR		
DOT	\.	.
SEMICOLON	;	;
COLON	:	:
COMMA	,	,
UNDERSCORE	_	_

RESERVED WORDS
program
class
instr
int
float
char
void
output
input
or
and
if
elif
for
while

main
inherits
attr
methods
var
return
otherwise

C. Syntax Analysis Description

Name	Grammar Rule
Program	programa : PROGRAM new_program ID save_program SEMICOLON class vars instr MAIN np_main bloque np_end_func end_of_file | PROGRAM new_program ID save_program SEMICOLON class instr MAIN np_main bloque np_end_func end_of_file | PROGRAM new_program ID save_program SEMICOLON vars instr MAIN np_main bloque np_end_func end_of_file | PROGRAM new_program ID save_program SEMICOLON vars MAIN np_main bloque np_end_func end_of_file | PROGRAM new_program ID save_program SEMICOLON instr MAIN np_main bloque np_end_func end_of_file | PROGRAM new_program ID save_program SEMICOLON MAIN np_main bloque np_end_func end_of_file
Class	class : CLASS ID declare_class INHERITS ID inherits_init LEFTCURLYBRACKET ATTR COLON attrs CONSTR COLON constructor METHODS COLON class_method RIGHTCURLYBRACKET class | CLASS ID declare_class LEFTCURLYBRACKET ATTR COLON attrs CONSTR COLON constructor METHODS COLON class_method RIGHTCURLYBRACKET class | CLASS ID declare_class LEFTCURLYBRACKET ATTR COLON attrs METHODS COLON class_method RIGHTCURLYBRACKET class | CLASS ID declare_class LEFTCURLYBRACKET ATTR COLON attrs CONSTR COLON constructor RIGHTCURLYBRACKET class | empty
Attributes	attrs : tipoCompuesto new_attr_set_type ID new_attribute attrs2 SEMICOLON | tipoSimple new_attr_set_type ID new_attribute attrs3 SEMICOLON | empty
attrs2	attrs2 : COMMA ID new_attribute attrs2 | empty
attrs3	attrs3 : LEFTBRACKET set_array set_dim_and_r CTE_INT set_limits RIGHTBRACKET set_each_node set_virtual_address attrs5 | LEFTBRACKET set_array set_dim_and_r CTE_INT set_limits RIGHTBRACKET set_each_node LEFTBRACKET CTE_INT set_limits RIGHTBRACKET set_each_node set_virtual_address attrs5

	| ID new_attribute attrs5 | attrs5
attrs4	attrs4 : VAR tipoCompuesto ID new_attribute attrs2 SEMICOLON attrs4 | VAR tipoSimple ID new_attribute attrs3 SEMICOLON attrs4 | empty
attr5	attrs5 : COMMA attrs3 | empty
Constructor	constructor : ID constructor_verify LEFTPARENTHESIS params RIGHTPARENTHESIS bloque | ID constructor_verify LEFTPARENTHESIS RIGHTPARENTHESIS bloque
Class Method	class_method : METHOD VOID set_void_type ID declare_class_method LEFTPARENTHESIS params set_number_params RIGHTPARENTHESIS LEFTCURLYBRACKET vars save_curr_quad_method bloque2 RIGHTCURLYBRACKET np_end_method class_method | METHOD tipoSimple ID declare_class_method LEFTPARENTHESIS params set_number_params RIGHTPARENTHESIS LEFTCURLYBRACKET vars save_curr_quad_method bloque2 RIGHTCURLYBRACKET np_end_method class_method | empty
Variable declaration	vars : VAR tipoCompuesto new_variable_set_type ID new_variable_from_class vars4 SEMICOLON vars | VAR tipoSimple new_variable_set_type ID new_variable vars3 SEMICOLON vars | empty
Variable declaration 2	vars2 : COMMA ID new_variable vars2 | empty
Variable declaration 3	vars3 : LEFTBRACKET set_array set_dim_and_r CTE_INT set_limits RIGHTBRACKET set_each_node set_virtual_address vars5 | LEFTBRACKET set_array set_dim_and_r CTE_INT set_limits RIGHTBRACKET set_each_node add_dim LEFTBRACKET CTE_INT set_limits RIGHTBRACKET set_each_node set_virtual_address vars5 | ID new_variable vars3 | vars5
Variable declaration 4	'''vars4 : COMMA ID new_variable_from_class vars4 | empty'''
Variable declaration 5	vars5 : COMMA vars3 | empty
Compound Type	tipoCompuesto : ID class_exists new_variable_set_type
Simple Type	tipoSimple : INT new_variable_set_type | FLOAT new_variable_set_type | CHAR new_variable_set_type
Instruction	instr : INSTR VOID set_void_type ID new_function LEFTPARENTHESIS params set_number_params RIGHTPARENTHESIS LEFTCURLYBRACKET vars set_local_vars save_curr_quad bloque2 RIGHTCURLYBRACKET np_end_func instr | INSTR tipoSimple ID new_function LEFTPARENTHESIS params set_number_params RIGHTPARENTHESIS LEFTCURLYBRACKET vars set_local_vars save_curr_quad bloque2

	RIGHTCURLYBRACKET np_end_func instr | empty
Parameters	params : tipoSimple new_variable_set_type set_params ID new_variable_param params | COMMA tipoSimple new_variable_set_type set_params ID new_variable_param params | empty
Block	bloque : LEFTCURLYBRACKET bloque2 RIGHTCURLYBRACKET
Block2	bloque2 : estatuto bloque2 | empty
Statement	estatuto : asignacion | llamada_void | condicion | escritura | lectura | cicloW | cicloFor | return
Assignment	asignacion : variable ASSIGNMENT exp SEMICOLON
Variable	variable : ID | ID verify_class_parent DOT ID verify_class_attr_id | ID add_array_id LEFTBRACKET verify_dims exp array_quads RIGHTBRACKET end_array_call | ID add_array_id LEFTBRACKET verify_dims exp array_quads RIGHTBRACKET update_dim LEFTBRACKET exp array_quads RIGHTBRACKET end_array_call
Variable2	variable2 : LEFTBRACKET verify_dims exp array_quads RIGHTBRACKET update_dim LEFTBRACKET exp array_quads RIGHTBRACKET end_array_call | empty
If condition	condicion : IF LEFTPARENTHESIS exp RIGHTPARENTHESIS np_if_1 bloque np_if_2 | IF LEFTPARENTHESIS exp RIGHTPARENTHESIS np_if_1 bloque condicion2
If condition 2	condicion2 : OTHERWISE np_else bloque np_if_2 | np_if_2 ELIF LEFTPARENTHESIS exp RIGHTPARENTHESIS np_if_1 bloque condicion2
Print	escritura : OUTPUT RIGHTARROW exp generate_write_quad escritura2 SEMICOLON | OUTPUT RIGHTARROW CTE_STRING generate_write_quad escritura2 SEMICOLON
Input	lectura : INPUT LEFTARROW variable SEMICOLON
Function call	llamada : UNDERSCORE ID function_detection LEFTPARENTHESIS llamada2 verify_coherence_of_params RIGHTPARENTHESIS function_gosub | UNDERSCORE ID method_detection_class_save DOT ID method_detection LEFTPARENTHESIS llamada2 verify_coherence_of_params RIGHTPARENTHESIS function_gosub_method
Function call 2	llamada2 : add_to_param_counter exp verify_param llamada2 | empty
Function call 3	llamada3 : COMMA add_to_param_counter exp verify_param llamada3 | empty

Function call for void	llamada_void : UNDERSCORE ID function_detection LEFTPARENTHESIS llamada2 verify_coherence_of_params RIGHTPARENTHESIS function_gosub SEMICOLON | UNDERSCORE ID method_detection_class_save DOT ID method_detection LEFTPARENTHESIS llamada2 verify_coherence_of_params RIGHTPARENTHESIS function_gosub_method SEMICOLON
While	cicloW : WHILE np_while_1 LEFTPARENTHESIS exp RIGHTPARENTHESIS np_while_2 bloque np_while_3
For loop	cicloFor : FOR LEFTPARENTHESIS assign_for SEMICOLON exp for_exp_comp SEMICOLON update RIGHTPARENTHESIS bloque for_update
For Assign	assign_for : ID for_declaration ASSIGNMENT exp for_exp_assign
For update	update : ID PLUSEQUAL CTE_INT | ID TIMESEQUAL CTE_INT
Return	return : RETURN exp end_func_return SEMICOLON
Exp	exp : expA expOR
OR expression	expOR : OR save_op expA add_operator_or expOR | empty
ExpA	expA : expB expAND
AND expression	expAND : AND save_op expB add_operator_and expAND | empty
ExpB	expB : expC add_operator_rop expROP
Relational Operator Expression	expROP : LESSTHAN save_op expB | GREATERTHAN save_op expB | EQUALLESSTHAN save_op expB | EQUALGREATERTHAN save_op expB | EQUALS save_op expB | NOTEQUAL save_op expB | empty
ExpC	expC : termino add_operator_plusminus expPM
ExpPM	expPM : PLUS save_op expC | MINUS save_op expC | empty
Term	termino : factor add_operator_multiplydivide expMD
ExpMD	expMD : TIMES save_op termino | DIVIDE save_op termino | empty
Factor	factor : llamada | LEFTPARENTHESIS add_fake_bottom exp delete_fake_bottom RIGHTPARENTHESIS | CTE_INT save_pvar_int save_constant_int | CTE_FLOAT save_pvar_float save_constant_float | variable save_pvar_var save_id

D. Generation of intermediate Code Description and Semantic Analysis Description

Operation Code and Virtual Addresses

Our compiler uses the following ranges for assigning and storing variables

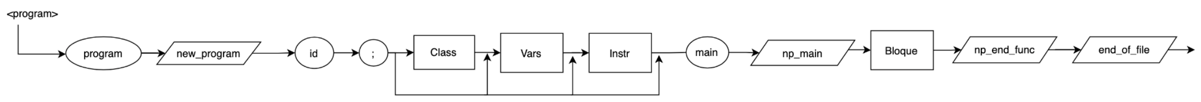
Scope and Type	Range
Local - Integer	10000-19999
Local - Float	20000-29999
Local - Temporal	30000-39999
Local - Constants	40000-49999
Local - Pointers	50000-59999
Global - Integer	110000-119999
Global - Float	120000-129999
Global - Temporal	130000-139999
Global - Constants	140000-149999
Global - Pointers	150000-159999

We also have the following commands in our quadruples, and here is what they do.

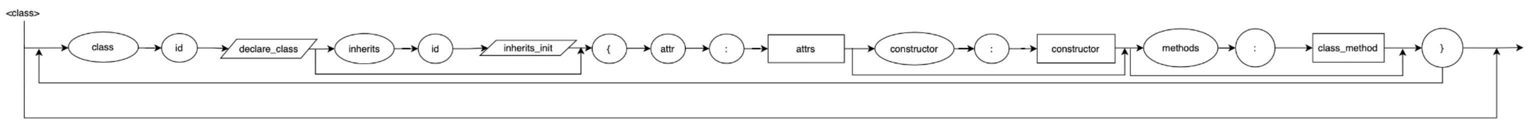
Command	Action
GOTO	Tells the virtual machine to go to the mentioned quadruple.
GOTOF	Tells the virtual machine to go to the mentioned quadruple in case the condition is False.
GOSUB	Tells the virtual machine to change the instruction pointer to the requested function, also, it stores the current instruction pointer in the jump stack, to return later.
ERA	Tells the virtual machine to instantiate new memory for the new function scope and inserts it into the memory stack.

PARAMETER	Tells the virtual machine to bring the parameter into the new function scope for execution.
RETURN	Tells the virtual machine to save the return value and reload the previous scope as the function has ended.
ENDFUNC	Tells the virtual machine to reload the previous scope as the function has ended.
VERIFY	Tells the virtual machine to verify that the array index is inside the bounds.
OUTPUT	Tells the virtual machine to output to the console.
INPUT	Tells the virtual machine to receive input from the user

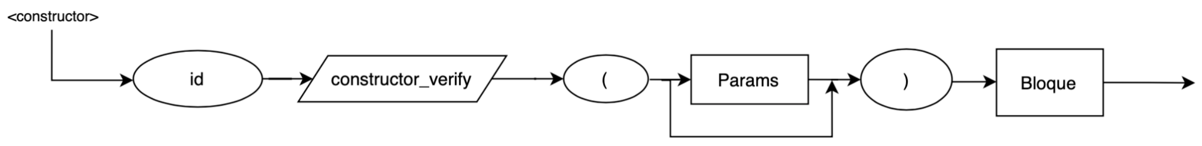
Syntax Diagrams



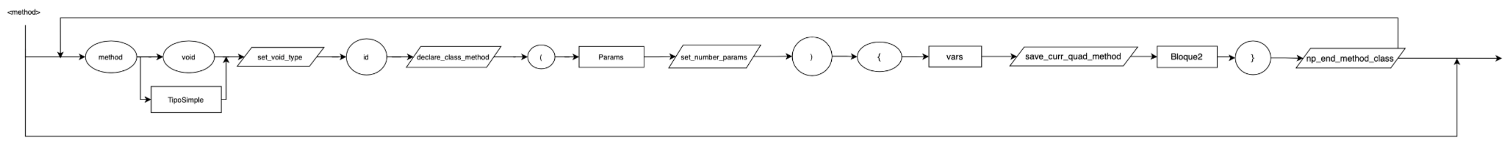
Neuralgic Point	Description
new_program	Generates the initial quad to go to main
np_main	Generates the
np_end_func	Generates the quadruple that declares the end of main.
end_of_file	Generates the EOF quadruple to stop execution.



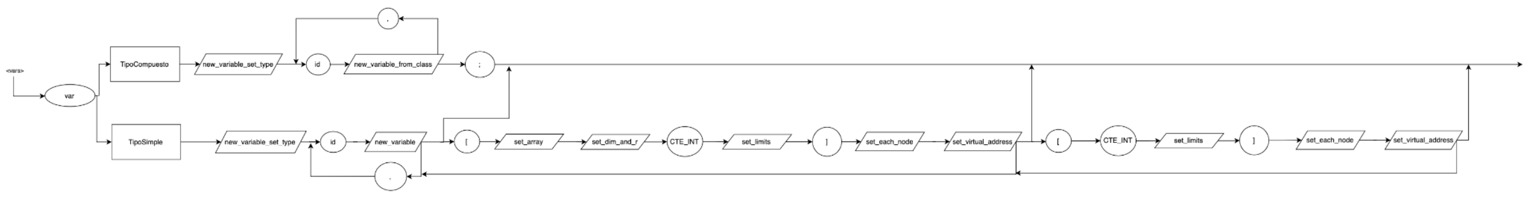
Neuralgic Point	Description
declare_class	Initializes new class into the class directory
inherits_init	Checks if the inherited class exists and adds the attributes and methods of that class into the new one



Neuralgic Point	Description
constructor_verify	Verifies that the constructor has the same name as the class

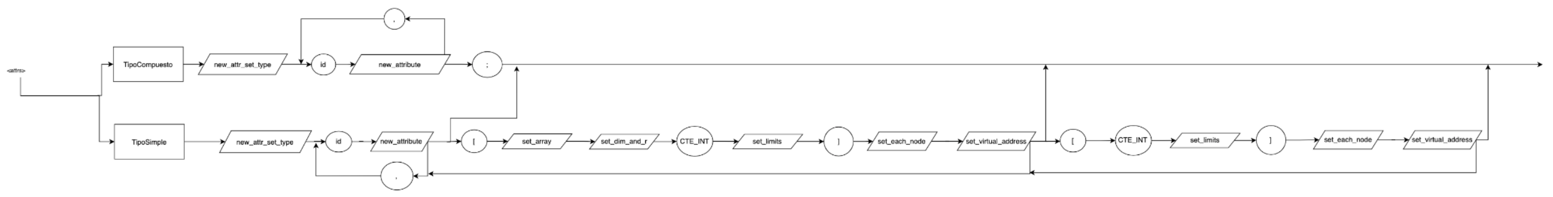


Neuralgic Point	Description
set_void_type	Sets the function type variable to "void".
declare_class_method	Adds the class method to the class table directory
set_number_params	Checks if the number of parameters is correct, if they are it saves the amount of parameters
save_curr_quad	Saves the quadruple where the function starts
np_end_method_class	Generates the ENDFUNC quadruple to know where the method ends.

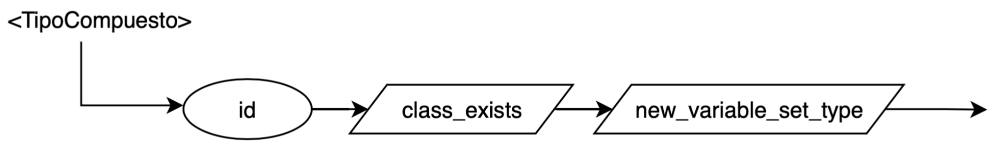


Neuralgic Point	Description
new_variable_set_type	Save the type of the variable
new_variable_from_class	Stores the variable into the variable table with the type of the class
new_variable	Stores the variable into the variable table
set_array	Saves the variable as an array

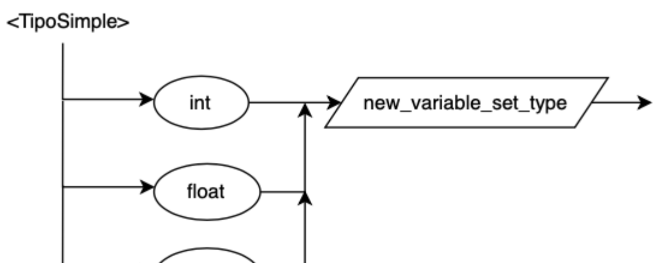
set_dim_and_r	Sets dimension = 1 and r = 1
set_limits	Stores the limits of the dimension of the array
set_each_node	Creates the node with the information required for the dimension
set_virtual_address	Sets each cell of the array with its corresponding address



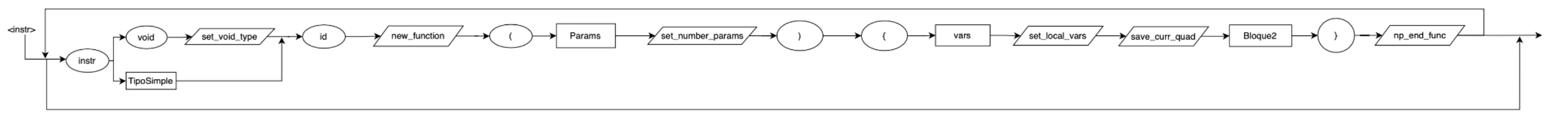
Neuralgic Point	Description
new_attr_set_type	Save the type of the attribute
new_attribute	Stores the attribute into the class directory
set_array	Saves the variable as an array
set_dim_and_r	Sets dimension = 1 and r = 1
set_limits	Stores the limits of the dimension of the array
set_each_node	Creates the node with the information required for the dimension
set_virtual_address	Sets each cell of the array with its corresponding address



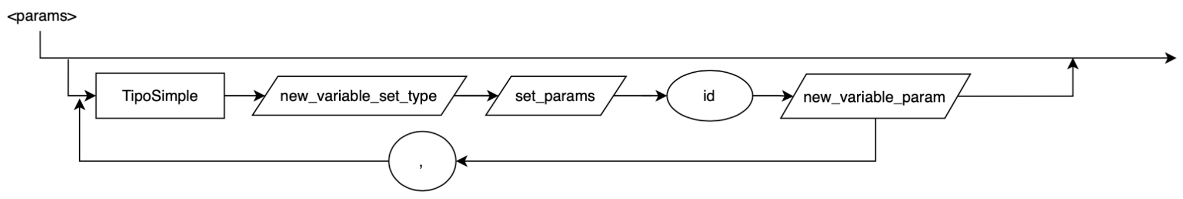
Neuralgic Point	Description
class_exists	Checks if the type of variable is part of a class
new_variable_set_type	Stores the type of the variable in global memory for later use.



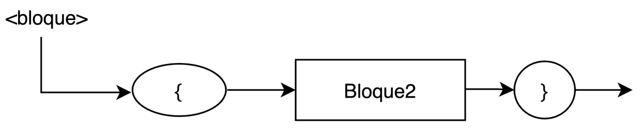
Neuralgic Point	Description
new_variable_set_type	Stores the type of the variable in global memory for later use.

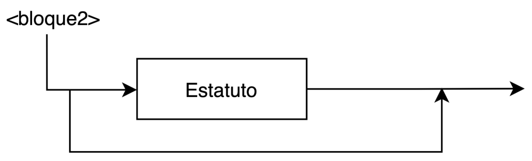


Neuralgic Point	Description
set_void_type	Sets the global function type variable to "void".
new_function	Lays the groundwork when a new function is detected. Adds function to Function_Directory and uses the var set in set_void_type.
set_number_params	Checks if the number of parameters is correct, if they are it saves the amount of parameters
set_local_vars	Saves the amount of local variables the function has
save_curr_quad	Saves the quadruple where the function starts
np_end_func	Generates the ENDFUNC quadruple to know where the function ends.

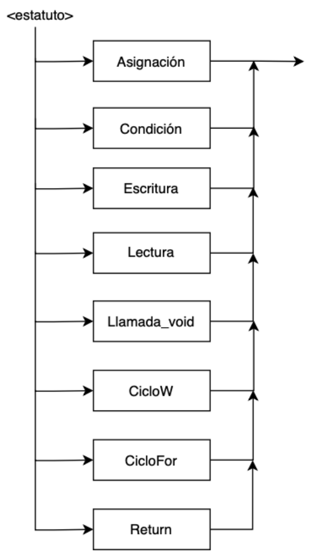


Neuralgic Point	Description
new_variable_set_type	Stores the type of the variable in global memory for later use.
set_params	Adds the type of the parameter to the parameter table
new_variable_param	Adds the parameter to the variable table of the scope.

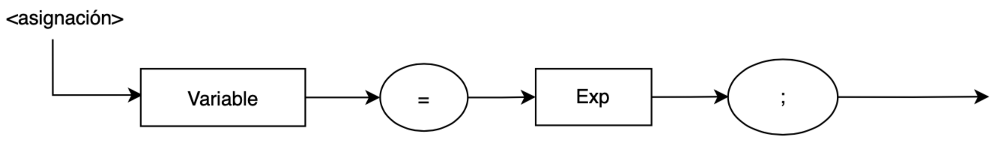




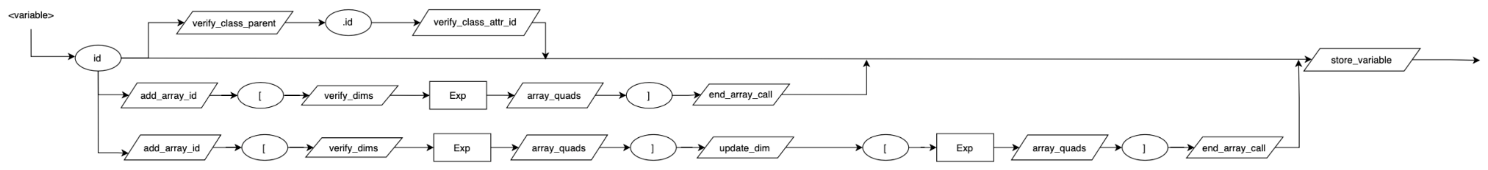
Neuralgic Point	Description
NONE	



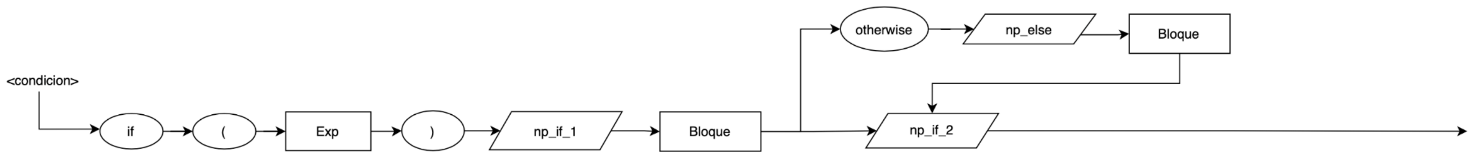
Neuralgic Point	Description
NONE	



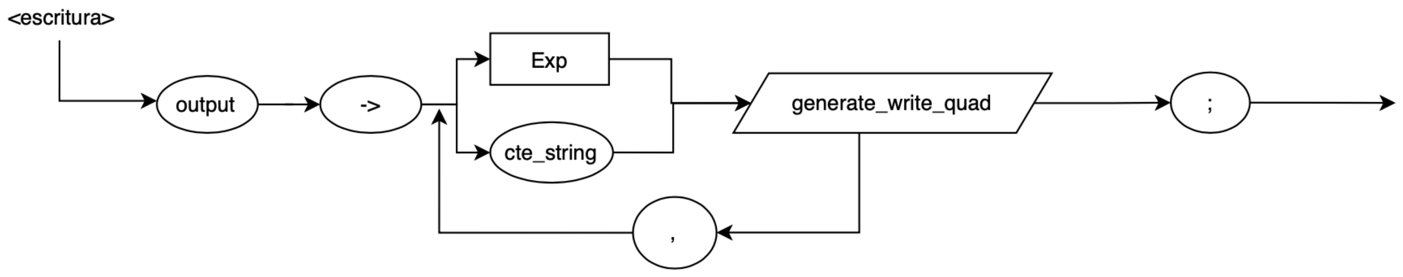
Neuralgic Point	Description
NONE	



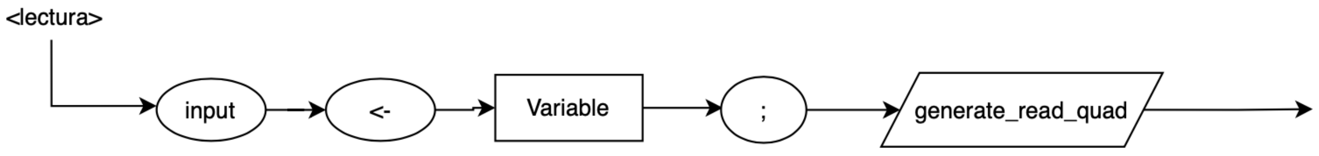
Neuralgic Point	Description
store_variable	Stores the type of the variable in global memory for further use in case of object attributes and method calls.
verify_class_parent	Stores the class variable id
verify_class_attr_id	Stores the class attribute
add_array_id	Checks if the variable is an array and pushes it into the stack
verify_dims	Pushes the dimension into the dimension stack and creates the first node for the dimension of the array
array_quads	Generates the verify quad, and if possible it makes the quads for the operations to access the indicated cell
update_dim	Changes dimension to the next one
end_array_call	Generates the quads to sum the “k” and to add the base address



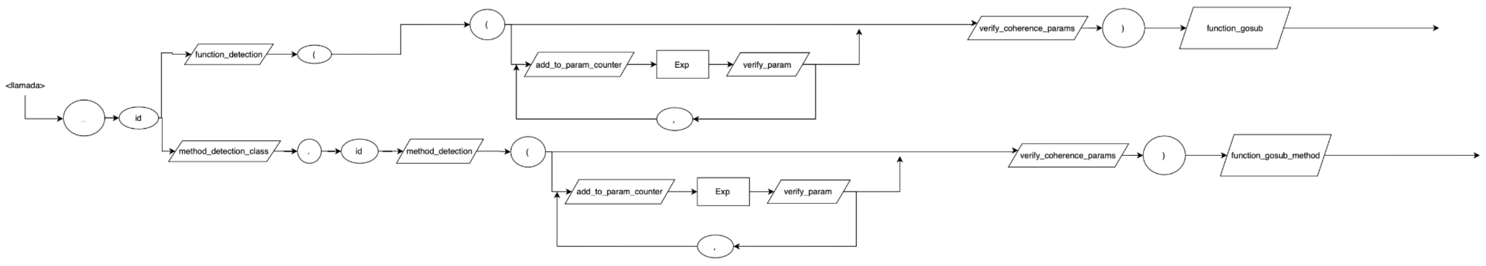
Neuralgic Point	Description
np_if_1	Generates quad if the statement is false
np_if_2	Fills quad



Neuralgic Point	Description
generate_write_quad	Generates quad for the printing statement

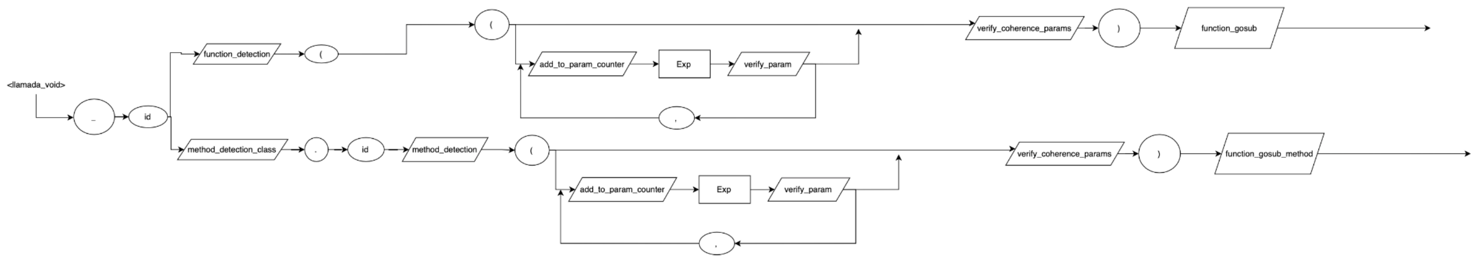


Neuralgic Point	Description
generate_read_quad	Generates the quad for reading variable

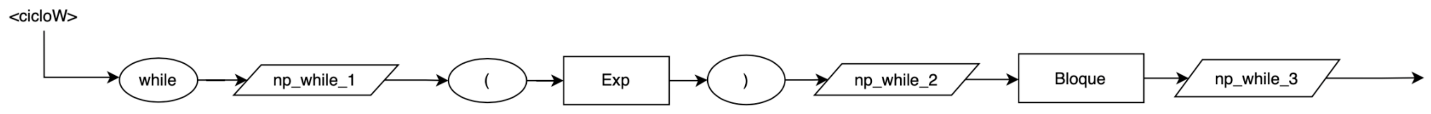


Neuralgic Point	Description
function_detection	Checks if the function exists in the function directory
method_detection_class_save	Saves method name
method_detection	Check if method exists
add_to_param_counter	Adds the to the counter of parameter

verify_param	Checks if the parameter type is the correct one and generates PARAMETER quad
verify_coherence_params	Checks that the amount of parameter is the correct one
function_gosub	Generates the GOSUB quad
function_gosub_method	Generates the GOSUB quad for the class method

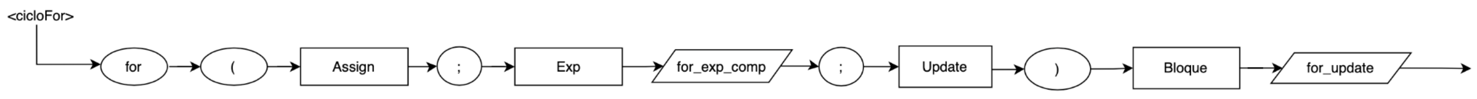


Neuralgic Point	Description
function_detection	Checks if the function exists in the function directory
method_detection_class_save	Saves method name
method_detection	Check if method exists
add_to_param_counter	Adds the to the counter of parameter
verify_param	Checks if the parameter type is the correct one and generates PARAMETER quad
verify_coherence_params	Checks that the amount of parameter is the correct one
function_gosub	Generates the GOSUB quad
function_gosub_method	Generates the GOSUB quad for the class method

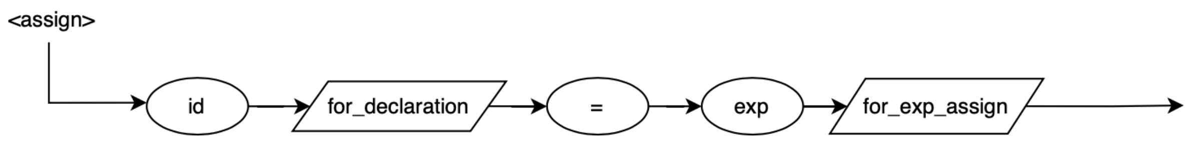


Neuralgic Point	Description
np_while_1	Leaves the number of the quad id in the in the stack_jump to remember to fill it

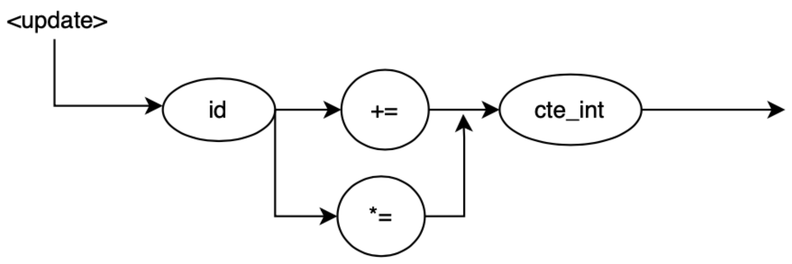
np_while_2	Generates quad if the statement is false
np_while_3	Fills the quad and generates the quad to continue in the cycle



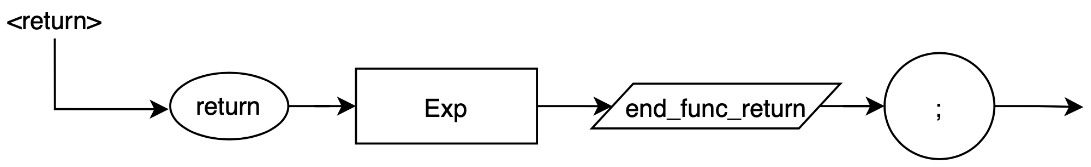
Neuralgic Point	Description
for_exp_comp	Checks if Exp is an int and generates the quads to check Exp after every iteration and the quad if the statement is false
for_update	Generates quad to update the value of the control variable and fills the quads.



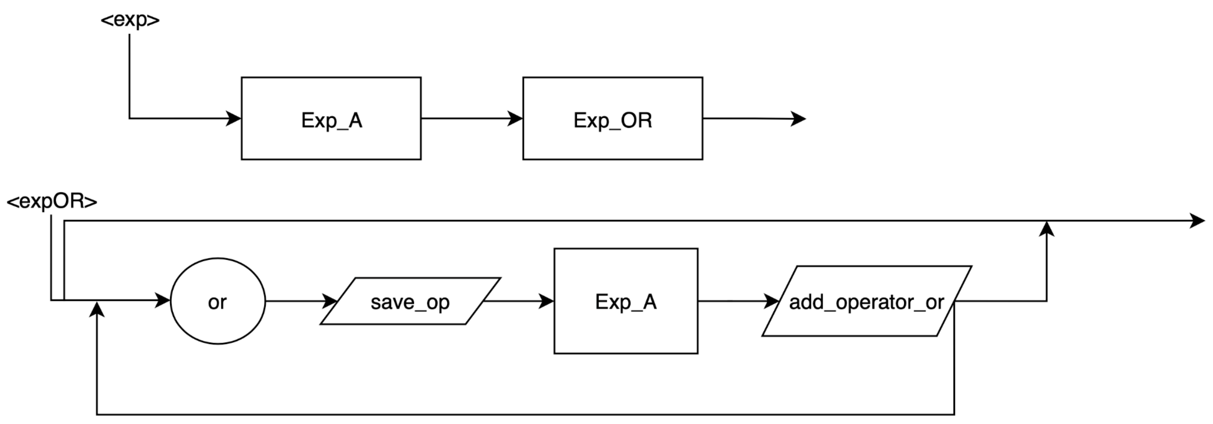
Neuralgic Point	Description
for_declaration	Checks that exp is an int
for_exp_assign	Generates the quads to assign the value to the original id and to create the control variable



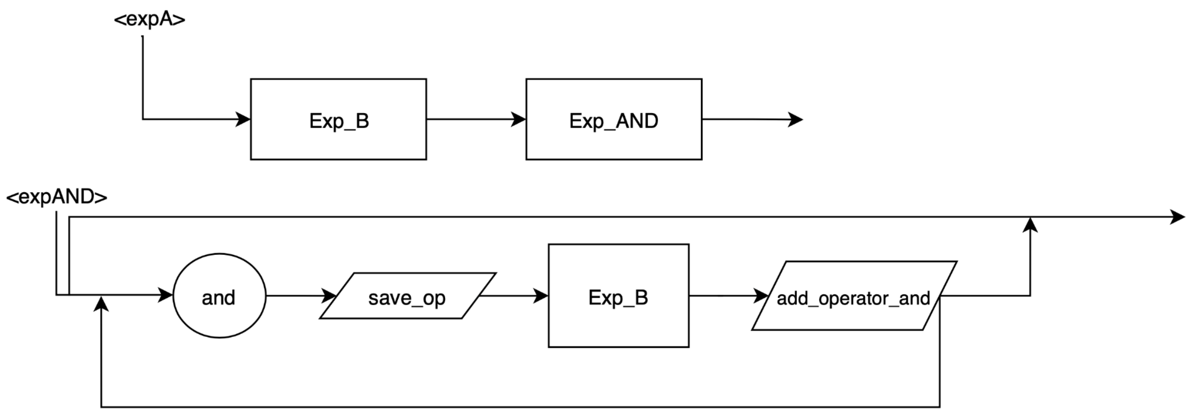
Neuralgic Point	Description
NONE	



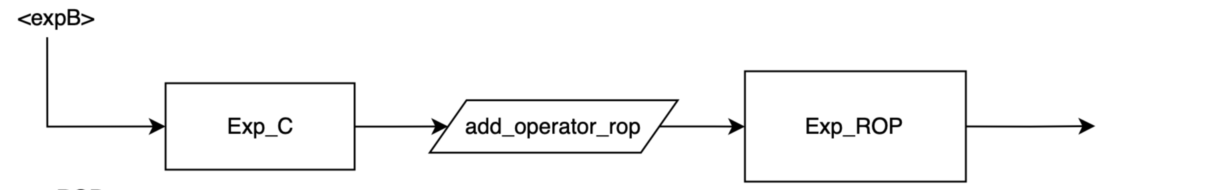
Neuralgic Point	Description
end_func_return	Generates the quad to return the expression established

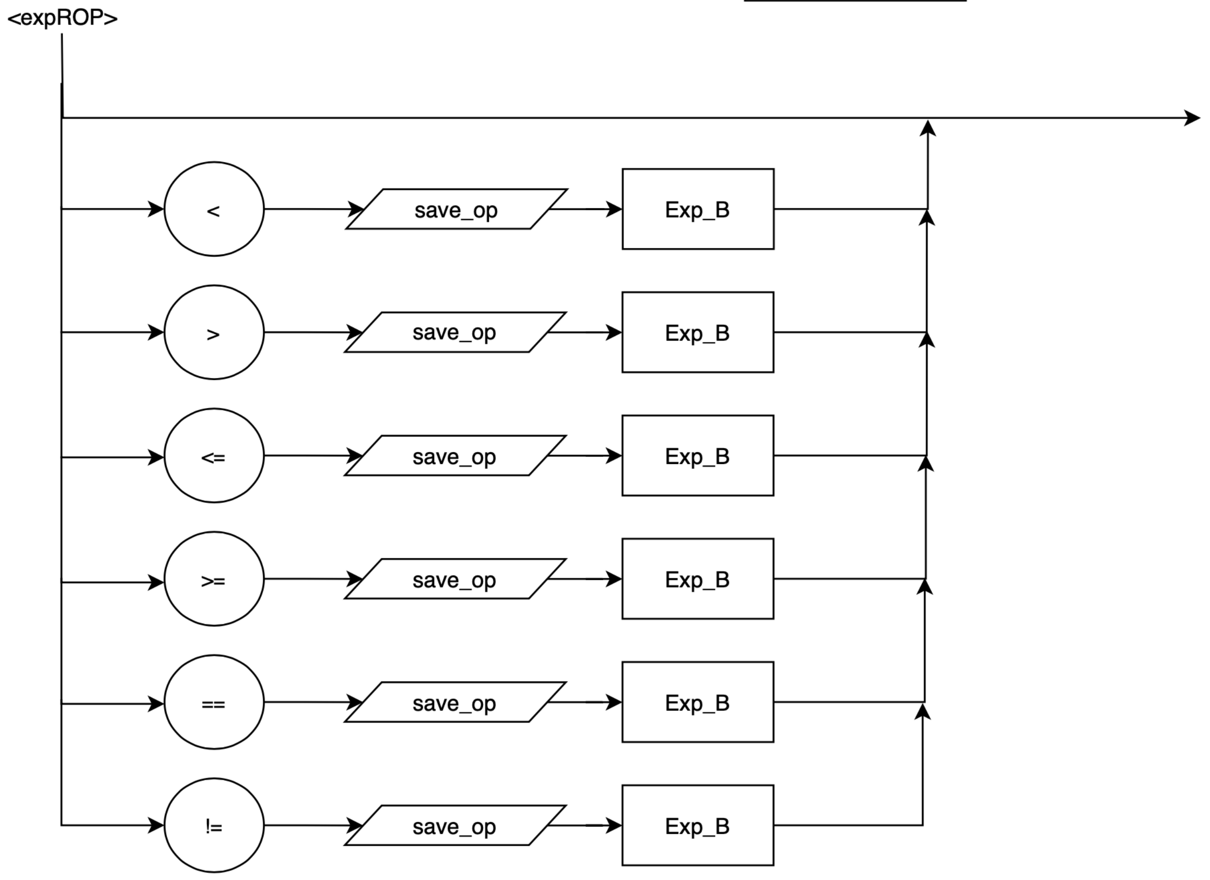


Neuralgic Point	Description
save_op	Stores the operator into poper stack
add_operator_or	Generates the quad for OR operation

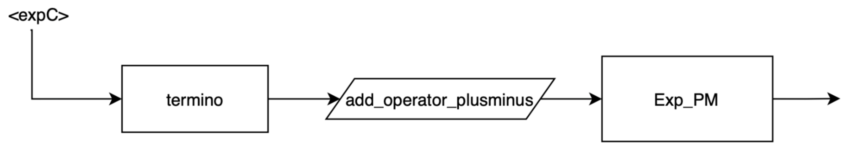


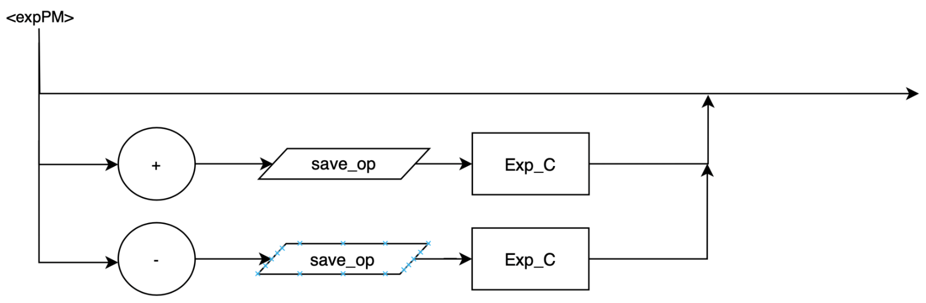
Neuralgic Point	Description
save_op	Stores the operator into poper stack
add_operator_and	Generates the quad for AND operation



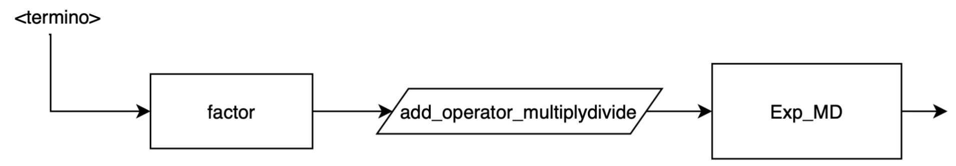


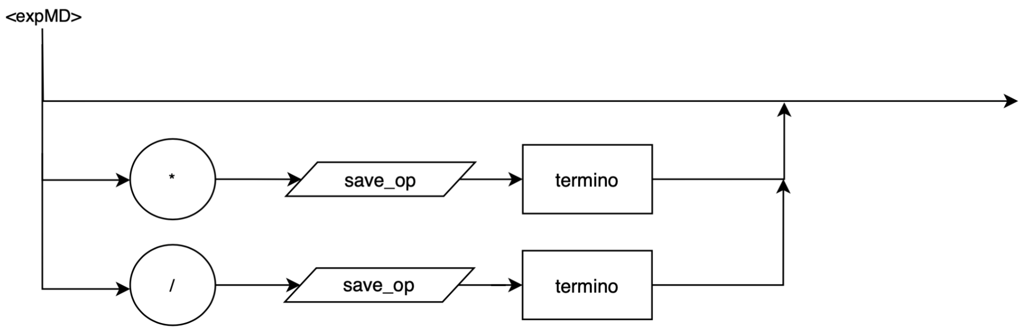
Neuralgic Point	Description
save_op	Stores the operator into poper stack
add_operator_ROP	Generates the quad for any relational operation



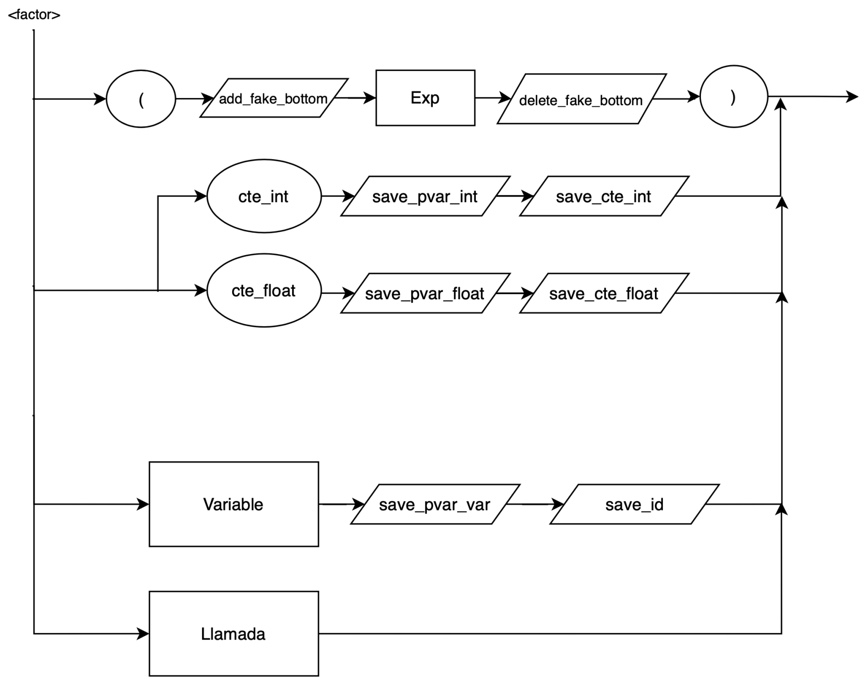


Neuralgic Point	Description
save_op	Stores the operator into poper stack
add_operator_plusminus	Generates the quad for any PLUS or MINUS operation





Neuralgic Point	Description
save_op	Stores the operator into poper stack
add_operator_multiplydivide	Generates the quad for any MULTIPLY or DIVIDE operation



Neuralgic Point	Description
add_fake_bottom	Inserts a “(“ into the poper stack

delete_fake_bottom	Pops the "(" from the poper stack
save_pvar_int	Saves the CTE INT into a variable and converts it into an int
save_constant_int	Obtains the address from the int variable and stores it into the stack
save_pvar_float	Saves the CTE FLOAT into a variable and converts it into a float
save_constant_float	Obtains the address from the float variable and stores it into the stack
save_pvar_var	Saves the variable into a variable
save_id	Obtains the address from the variable and stores it into the stack

Structure of the language

The general structure of a program in the **Mogavity** language will be like the following:

```
program nombre_prog;
<Class declaration>               // follows Class syntax
<Global variable declaration>     // follows Vars syntax
<Functions' definition>           // follows Instr syntax
main()
{
    <Statement>
}
```

Italics means optional
Bold means reserved words

Class declaration

```
class name_class inherits inherited_class {
attr:
<Attributes declaration>        // follows Attributes syntax
constructor:
<Constructor declaration>       // follows Constructor syntax
method:
<Methods definition>            // follows Methods syntax
}
```

Variable declaration

var type ids;

where

type: can be **int**, **float**, or id (it can be a class name). The type id can't define arrays, meaning that there won't be arrays from classes.

ids: can be separated by a comma, or have the structure of 1D or 2D arrays

```
            id[N,M]             // N and M are indexable ints from 0 to N-1, M-1
    Ej.     int id1[3][2];
            float id1;
            Car id1, id2;           // there are two objects of type Car
```

** This declaration is for either local or global variables

__Function declaration__

instr type id **(**Params**)**

{

<Local variable declaration>

<Estatuto>

}

where

type: can be **int**, **float**, o **void**

Params: follows Params syntax

__Statement__

We will define the definition for every statement:

Assignment

Variable **=** Exp;

**Variable can have the value of an expression

Variable **=** instr_name(*param*);

**Variable can have the value of return from a function

Variable = isntr_name(*Param*) + cte;

** Variable can have the value of return from a function combined with an arithmetic operation.

** Variable: follows the Variable syntax

Variable: can be id, id[N,M], id[N] ó id.id

Condition

if (Exp**) {** *Statement* **}**

otherwise {*Statement****}***

There won't be an "if else" expression, so the user will need to make that amount of if's they require.

Printing

output -> Exp | cte_string*, Exp* | *cte_string***;**

We can output the result of an expression or strings separated by commas.

Lectura

input <- Variable**;**

If the user inputs an incorrect value based on the type of the variable, it will get an error.

Llamada

instr_name**(***Exp,Exp***);**

```
class_name.method_name(Exp,Exp);
```

** An instruction (function) can have 0 or more parameters
** A void instruction can be called

While Cycle

```
while (Exp)
{
   Statement
}
```

The statement will be completed while the expression is true

For Cycle

```
for (Assign;Exp;Update)
{
  Statement
}
```

Every iteration the value of the id will be updated and it will enter the statement while the expression is true. Our For loop can only make the ID value increase, that’s why we only accept a sum or multiplication, our for loop doesn't support going back on a variable value.

where

```
Assign: id = N
        * N is an integer
Exp: can be arithmetic operations, variable or a constant integer
Update: id += N
         id *= N
```

Return

```
return Exp;
```

The return statement will be used for the functions that return a value. Return can only support expressions such as variables and arithmetic expressions.

Expressions

Expressions can have the following logical, relational and arithmetic operators:
+, **-**, *, **/**, **AND**, **OR**, **<**, **>**, **<=**, **>=**, **!=**, **==**
They can be used with parenthesis and will manage left association.

Semantic principles

left_op	right_op	+	-	*	/	AND	OR	<	>	<=	>=	!=	==
int	int	int	int	int	int	error	error	bool	bool	bool	bool	bool	bool
int	float	float	float	float	float	error	error	bool	bool	bool	bool	bool	bool
float	int	float	float	float	float	error	error	bool	bool	bool	bool	bool	bool
float	float	float	float	float	float	error	error	bool	bool	bool	bool	bool	bool
bool	bool	error	error	error	error	bool	bool	error	error	error	error	error	error
bool	int	error	error	error	error	error	error	error	error	error	error	error	error
bool	float	error	error	error	error	error	error	error	error	error	error	error	error

Special Functions

Because our language is object-oriented, we don't have any special functions. We will focus on the creation of classes, their instances and the management of attributes and methods.

Data types

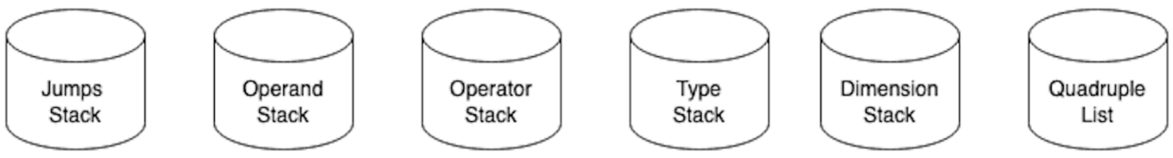
The data types that Mogavity manages are the following:

- int (positive integers)
- float (positive decimals)
- id (object of a class)

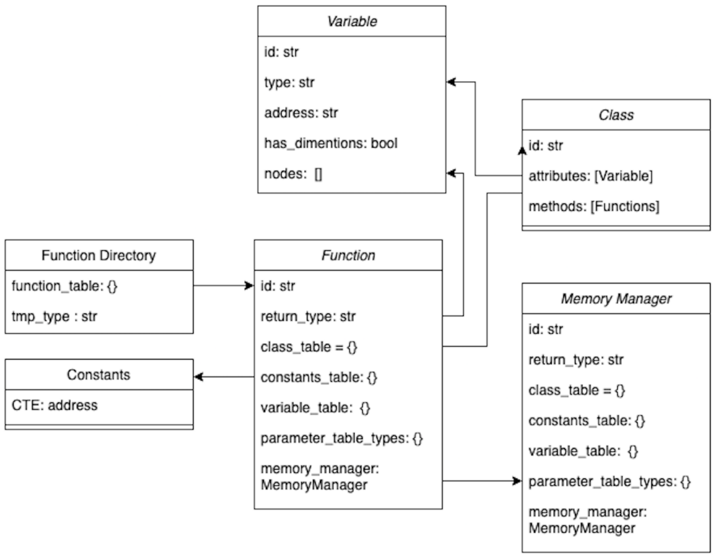
From the data types **int** and **float** we can create 1D and 2D arrays. Because we only control positive numbers, the user will have to create the negative numbers by subtracting zero from the number to convert to negative.
We can output *strings* and for conditions *booleans* will be used.

Memory Management

Mogavity uses a simple system of objects to manage memory for items during compilation. A graphic description is found below. For all the stacks used in compilation, we maintain a simple structure, each stack has its own function during the compilation phase and assist in the generation of quadruples, which are then stored in the Quadruple list.



To manage the rest of the items, we use a complex structure of objects that we found were best suited for this application. Below is a diagram explaining the objects and the relation between objects.



The Function Directory object holds all of our Function objects, which can also be treated as scopes. Inside the Function Directory, a dictionary with the name function_table holds all of the objects. We chose to create an object, rather than interacting plainly with the dictionary because this way we could also associate actions to the object (methods).

A Function object also holds dictionaries, these store all of our classes, variables and constants, depending on the current scope. We also store parameters here and we also have a Memory Manager object. We decided to keep using dictionaries as the base of our memory because they allow us quick access using keys and simplify the way of the compiler knowing if an item exists inside it or not.

The Variable object is created once per variable, and groups together the ID, type and address of the variable always. In the case the variable is an array, it will also store an attribute to know if it is a dimensioned variable and also store its nodes to assist with array calculations.

The Memory Manager object is assigned once per Function (scope) and is in charge of assigning memory addresses to variables. We chose to do an object too because we could instantiate one per scope and handle memory the same way without worrying about addresses getting mixed with ones from other scopes.

The Class object exists to hold the "skeleton" of a class and allow for easy instantiation once called. An object made it simple to keep all information together.

Constants are simply stored on a dictionary, using the value of the constant as the key and the value of the dictionary is the address of such constant. Constants are only assigned in the global scope, so they can be recycled through any scope without wasting additional memory.

VI. Virtual Machine Description

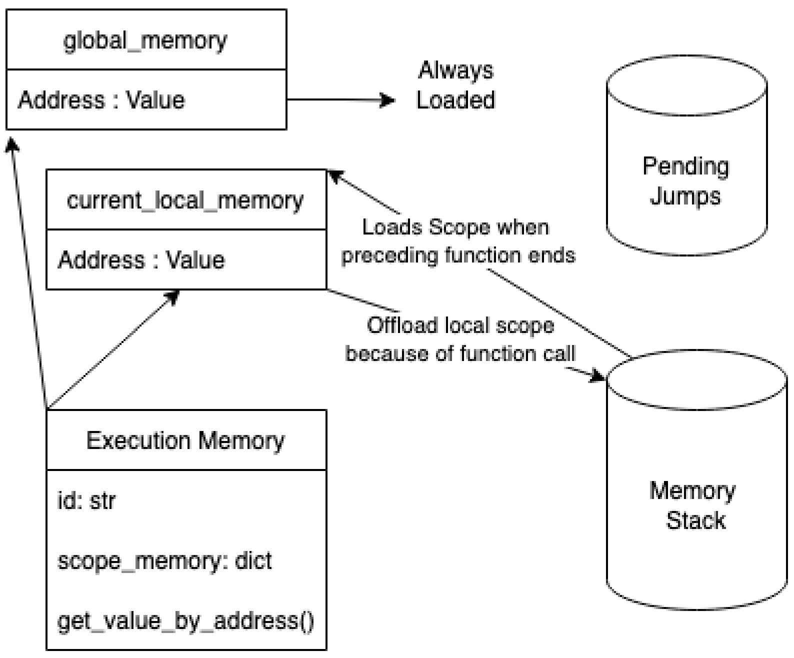
A. Language, Libraries and Equipment used

The virtual machine runs on Python, and uses no external libraries to run. However we do use a Stack library of our own creation and our FunctionDirectory and Quadruple objects. Speaking of hardware and equipment, we developed on macOS computers with Python 3.8.2 using PyCharm and VSCode as IDEs.

B. Memory Administration

For managing the memory during execution, we store our variables in two objects, one stores the global scope and the other one stores the current local scope. When we need to change scope, the global object remains unchanged, while the local one is pushed to the stack to make way for the new scope. Then, the current quadruple is added to the jump stack. When we need to revert to the previous scope, the object is popped and instantiated, then the jump stack tells us where to go.

1. **ExecutionMemory -** An object we created that pairs a dictionary with an ID, we also use the object to simplify our operations like inserting to memory and getting a value from an address.
2. **global_memory -** Is a dictionary that stores the global scope, using the address as key for the value.
3. **current_local_memory** - Is a dictionary that stores the local scope and uses the address as key for the value.
4. **memory_stack -** A stack that stores the “sleeping” memory when scopes are changed.
5. **pending_jumps -** A simple stack that stores the jumps that the instruction pointer needs to do.



Using an object wrapped dictionary (Execution Memory) made the most sense to us, as this way we only load in memory the objects that we use, so we do not waste space, plus is provides us an easy way of throwing an error when the VM tries to access a variable that does not exist or has not been declared previously

We use a simple relationship between virtual addresses and real ones during execution. Variables are stored in a dictionary as seen above, so unallocated addresses are not given a space in real memory, thus not wasting space.

VII. Language Testing

Test - QuickSort
Executes a sorting algorithm on an array. Tests array access, loops, output and conditionals.
Test Code

```
program test;

var int A[20];
var int a,b,c,LEN;

instr void sort() {
   var int pivot, low, high, temp, pv;
   pv = LEN/2;
   pivot = A[pv];
   low = 0;
   high = LEN - 1;
   a = 1;

   while (a == 1) {
       while (low < (LEN - 1) and A[low] < pivot) {
           low = low + 1;
       }
       while (high > 0 and A[high] > pivot) {
           high = high - 1;
       }

       if (low >= high) {
           a = 0;
       } otherwise {
           temp = A[low];
           A[low] = A[high];
           A[high] = temp;
       }
   }
}

main{
LEN = 20;
A[0] = 20;
A[1] = 19;
A[2] = 18;
A[3] = 17;
A[4] = 16;
A[5] = 15;
A[6] = 14;
A[7] = 13;
A[8] = 12;
A[9] = 11;
A[10] = 10;
A[11] = 9;
A[12] = 8;
A[13] = 7;
A[14] = 6;
A[15] = 5;
A[16] = 4;
A[17] = 3;
A[18] = 2;
A[19] = 1;
```

```
    a = 0;
    while(a<LEN){
        output -> A[a];
        a = a+1;
    }

    _sort();
    output -> "===========SORTING=============";

    a = 0;
    while(a<LEN){
        output -> A[a];
        a = a+1;
    }



    return 0;
}
```

Intermediate Code

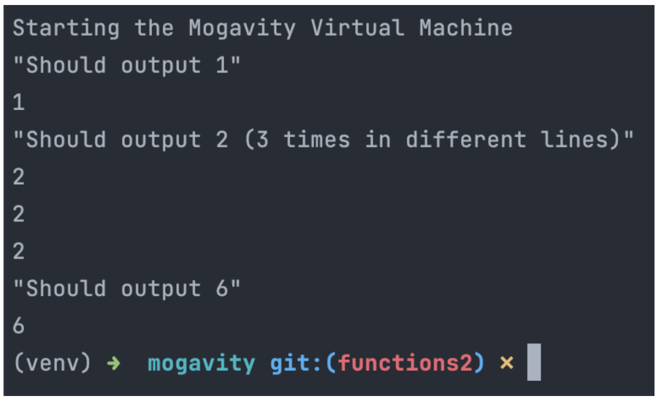
```
[1,GOTO,None,None,56]
[2,/,110025,150000,40000]
[3,=,40000,None,10004]
[4,VERIFY,10004,0,19]
[5,+,10004,150001,40001]
[6,+,40001,150002,60000]
[7,=,60000,None,10000]
[8,=,150001,None,10001]
[9,-,110025,150003,40002]
[10,=,40002,None,10002]
[11,=,150003,None,110022]
[12,==,110022,150003,40003]
[13,GOTOF,40003,None,55]
[14,-,110025,150003,40004]
[15,<,10001,40004,40005]
[16,VERIFY,10001,0,19]
[17,+,10001,150001,40006]
[18,+,40006,150002,60001]
[19,<,60001,10000,40007]
[20,and,40005,40007,40008]
[21,GOTOF,40008,None,25]
[22,+,10001,150003,40009]
[23,=,40009,None,10001]
[24,GOTO,None,None,14]
[25,>,10002,150001,40010]
[26,VERIFY,10002,0,19]
[27,+,10002,150001,40011]
[28,+,40011,150002,60002]
[29,>,60002,10000,40012]
```

```
[30,and,40010,40012,40013]
[31,GOTOF,40013,None,35]
[32,-,10002,150003,40014]
[33,=,40014,None,10002]
[34,GOTO,None,None,25]
[35,>=,10001,10002,40015]
[36,GOTOF,40015,None,39]
[37,=,150001,None,110022]
[38,GOTO,None,None,54]
[39,VERIFY,10001,0,19]
[40,+,10001,150001,40016]
[41,+,40016,150002,60003]
[42,=,60003,None,10003]
[43,VERIFY,10001,0,19]
[44,+,10001,150001,40017]
[45,+,40017,150002,60004]
[46,VERIFY,10002,0,19]
[47,+,10002,150001,40018]
[48,+,40018,150002,60005]
[49,=,60005,None,60004]
[50,VERIFY,10002,0,19]
[51,+,10002,150001,40019]
[52,+,40019,150002,60006]
[53,=,10003,None,60006]
[54,GOTO,None,None,12]
[55,ENDFUNC,None,None,None]
[56,=,150004,None,110025]
[57,VERIFY,150001,0,19]
[58,+,150001,150001,140000]
[59,+,140000,150002,160000]
[60,=,150004,None,160000]
[61,VERIFY,150003,0,19]
[62,+,150003,150001,140001]
[63,+,140001,150002,160001]
[64,=,150005,None,160001]
[65,VERIFY,150000,0,19]
[66,+,150000,150001,140002]
[67,+,140002,150002,160002]
[68,=,150006,None,160002]
[69,VERIFY,150007,0,19]
[70,+,150007,150001,140003]
[71,+,140003,150002,160003]
[72,=,150008,None,160003]
[73,VERIFY,150009,0,19]
[74,+,150009,150001,140004]
[75,+,140004,150002,160004]
[76,=,150010,None,160004]
[77,VERIFY,150011,0,19]
[78,+,150011,150001,140005]
[79,+,140005,150002,160005]
[80,=,150012,None,160005]
[81,VERIFY,150013,0,19]
[82,+,150013,150001,140006]
[83,+,140006,150002,160006]
```

```
[84,=,150014,None,160006]
[85,VERIFY,150015,0,19]
[86,+,150015,150001,140007]
[87,+,140007,150002,160007]
[88,=,150016,None,160007]
[89,VERIFY,150017,0,19]
[90,+,150017,150001,140008]
[91,+,140008,150002,160008]
[92,=,150018,None,160008]
[93,VERIFY,150019,0,19]
[94,+,150019,150001,140009]
[95,+,140009,150002,160009]
[96,=,150020,None,160009]
[97,VERIFY,150021,0,19]
[98,+,150021,150001,140010]
[99,+,140010,150002,160010]
[100,=,150021,None,160010]
[101,VERIFY,150020,0,19]
[102,+,150020,150001,140011]
[103,+,140011,150002,160011]
[104,=,150019,None,160011]
[105,VERIFY,150018,0,19]
[106,+,150018,150001,140012]
[107,+,140012,150002,160012]
[108,=,150017,None,160012]
[109,VERIFY,150016,0,19]
[110,+,150016,150001,140013]
[111,+,140013,150002,160013]
[112,=,150015,None,160013]
[113,VERIFY,150014,0,19]
[114,+,150014,150001,140014]
[115,+,140014,150002,160014]
[116,=,150013,None,160014]
[117,VERIFY,150012,0,19]
[118,+,150012,150001,140015]
[119,+,140015,150002,160015]
[120,=,150011,None,160015]
[121,VERIFY,150010,0,19]
[122,+,150010,150001,140016]
[123,+,140016,150002,160016]
[124,=,150009,None,160016]
[125,VERIFY,150008,0,19]
[126,+,150008,150001,140017]
[127,+,140017,150002,160017]
[128,=,150007,None,160017]
[129,VERIFY,150006,0,19]
[130,+,150006,150001,140018]
[131,+,140018,150002,160018]
[132,=,150000,None,160018]
[133,VERIFY,150005,0,19]
[134,+,150005,150001,140019]
[135,+,140019,150002,160019]
[136,=,150003,None,160019]
[137,=,150001,None,110022]
```

```
[138,<,110022,110025,140020]
[139,GOTOF,140020,None,147]
[140,VERIFY,110022,0,19]
[141,+,110022,150001,140021]
[142,+,140021,150002,160020]
[143,OUTPUT,None,None,160020]
[144,+,110022,150003,140022]
[145,=,140022,None,110022]
[146,GOTO,None,None,138]
[147,ERA,None,None,sort]
[148,GOSUB,sort,None,2]
[149,OUTPUT,None,None,150022]
[150,=,150001,None,110022]
[151,<,110022,110025,140023]
[152,GOTOF,140023,None,160]
[153,VERIFY,110022,0,19]
[154,+,110022,150001,140024]
[155,+,140024,150002,160021]
[156,OUTPUT,None,None,160021]
[157,+,110022,150003,140025]
[158,=,140025,None,110022]
[159,GOTO,None,None,151]
[160,RETURN,None,None,None]
[161,ENDFUNC,None,None,None]
[162,EOF,None,None,None]
```

Output



```
Starting the Mogavity Virtual Machine
"Should output 1"
1
"Should output 2 (3 times in different lines)"
2
2
2
"Should output 6"
6
(venv) ➜  mogavity git:(functions2) ✗
```

Test - Matrix Multiplication from Input
Does operations on two matrices with input from the user. The user fills two 2x2 matrices and then receives the printed result. Tests loops, matrices, input, output and arrays.

Test Code

```
program test;

var int matrixA[2][2];
var int matrixB[2][2];
var int product[2][2];
var int a,b,c,LEN, res;


main{

   output -> "matrixA[0][0]";
   input <- res;
   matrixA[0][0] = res;
   output -> "matrixA[0][1]";
   input <- res;
   matrixA[0][1] = res;
   output -> "matrixA[1][0]";
   input <- res;
   matrixA[1][0] = res;
   output -> "matrixA[1][1]";
   input <- res;
   matrixA[1][1] = res;

   output -> "matrixB[0][0]";
   input <- res;
   matrixB[0][0] = res;
   output -> "matrixB[0][1]";
   input <- res;
   matrixB[0][1] = res;
   output -> "matrixB[1][0]";
   input <- res;
   matrixB[1][0] = res;
   output -> "matrixB[1][1]";
   input <- res;
   matrixB[1][1] = res;

   LEN = 2;

   a = 0;
   b = 0;
   c = 0;
   while(a<LEN){
       b = 0;
       while(b<LEN){
            product[a][b] = 0;
            b = b+1;
        }
        a = a+1;
    }

    a = 0;
    b = 0;
    c = 0;
    while(a<LEN){
        b = 0;
        while(b<LEN){
            c = 0;
            while(c<LEN){
                product[a][b] = product[a][b]+ matrixA[a][c] *
matrixB[c][b];
                c = c + 1;
            }
            b = b+1;
        }
        a = a+1;
    }
    output -> "====================================";
    a = 0;
    b = 0;
    while(a<LEN){
        b = 0;
        while(b<LEN){
            output -> product[a][b];
            b = b+1;
        }
        a = a+1;
    }

    return 0;
}
```

Intermediate Code

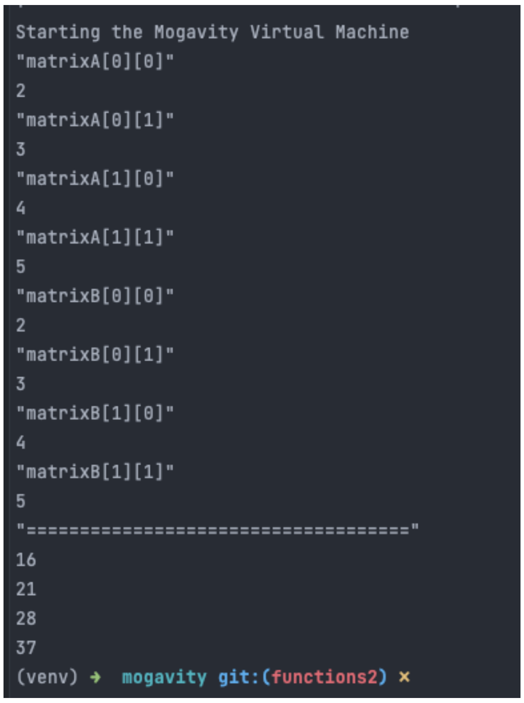
```
[1,GOTO,None,None,2]
[2,OUTPUT,None,None,150000]
[3,INPUT,None,None,110020]
[4,VERIFY,150001,0,1]
[5,*,150001,150002,140000]
[6,VERIFY,150001,0,1]
[7,+,140000,150001,140001]
[8,+,140001,150001,140002]
[9,+,140002,150003,160000]
```

```
[10,=,110020,None,160000]
[11,OUTPUT,None,None,150004]
[12,INPUT,None,None,110020]
[13,VERIFY,150001,0,1]
[14,*,150001,150002,140003]
[15,VERIFY,150005,0,1]
[16,+,140003,150005,140004]
[17,+,140004,150001,140005]
[18,+,140005,150003,160001]
[19,=,110020,None,160001]
[20,OUTPUT,None,None,150006]
[21,INPUT,None,None,110020]
[22,VERIFY,150005,0,1]
[23,*,150005,150002,140006]
[24,VERIFY,150001,0,1]
[25,+,140006,150001,140007]
[26,+,140007,150001,140008]
[27,+,140008,150003,160002]
[28,=,110020,None,160002]
[29,OUTPUT,None,None,150007]
[30,INPUT,None,None,110020]
[31,VERIFY,150005,0,1]
[32,*,150005,150002,140009]
[33,VERIFY,150005,0,1]
[34,+,140009,150005,140010]
[35,+,140010,150001,140011]
[36,+,140011,150003,160003]
[37,=,110020,None,160003]
[38,OUTPUT,None,None,150008]
[39,INPUT,None,None,110020]
[40,VERIFY,150001,0,1]
[41,*,150001,150002,140012]
[42,VERIFY,150001,0,1]
[43,+,140012,150001,140013]
[44,+,140013,150001,140014]
[45,+,140014,150009,160004]
[46,=,110020,None,160004]
[47,OUTPUT,None,None,150010]
[48,INPUT,None,None,110020]
[49,VERIFY,150001,0,1]
[50,*,150001,150002,140015]
[51,VERIFY,150005,0,1]
[52,+,140015,150005,140016]
[53,+,140016,150001,140017]
[54,+,140017,150009,160005]
[55,=,110020,None,160005]
[56,OUTPUT,None,None,150011]
[57,INPUT,None,None,110020]
[58,VERIFY,150005,0,1]
[59,*,150005,150002,140018]
[60,VERIFY,150001,0,1]
[61,+,140018,150001,140019]
[62,+,140019,150001,140020]
[63,+,140020,150009,160006]
```

```
[64,=,110020,None,160006]
[65,OUTPUT,None,None,150012]
[66,INPUT,None,None,110020]
[67,VERIFY,150005,0,1]
[68,*,150005,150002,140021]
[69,VERIFY,150005,0,1]
[70,+,140021,150005,140022]
[71,+,140022,150001,140023]
[72,+,140023,150009,160007]
[73,=,110020,None,160007]
[74,=,150002,None,110019]
[75,=,150001,None,110016]
[76,=,150001,None,110017]
[77,=,150001,None,110018]
[78,<,110016,110019,140024]
[79,GOTOF,140024,None,96]
[80,=,150001,None,110017]
[81,<,110017,110019,140025]
[82,GOTOF,140025,None,93]
[83,VERIFY,110016,0,1]
[84,*,110016,150002,140026]
[85,VERIFY,110017,0,1]
[86,+,140026,110017,140027]
[87,+,140027,150001,140028]
[88,+,140028,150013,160008]
[89,=,150001,None,160008]
[90,+,110017,150005,140029]
[91,=,140029,None,110017]
[92,GOTO,None,None,81]
[93,+,110016,150005,140030]
[94,=,140030,None,110016]
[95,GOTO,None,None,78]
[96,=,150001,None,110016]
[97,=,150001,None,110017]
[98,=,150001,None,110018]
[99,<,110016,110019,140031]
[100,GOTOF,140031,None,143]
[101,=,150001,None,110017]
[102,<,110017,110019,140032]
[103,GOTOF,140032,None,140]
[104,=,150001,None,110018]
[105,<,110018,110019,140033]
[106,GOTOF,140033,None,137]
[107,VERIFY,110016,0,1]
[108,*,110016,150002,140034]
[109,VERIFY,110017,0,1]
[110,+,140034,110017,140035]
[111,+,140035,150001,140036]
[112,+,140036,150013,160009]
[113,VERIFY,110016,0,1]
[114,*,110016,150002,140037]
[115,VERIFY,110017,0,1]
[116,+,140037,110017,140038]
[117,+,140038,150001,140039]
```

```
[118,+,140039,150013,160010]
[119,VERIFY,110016,0,1]
[120,*,110016,150002,140040]
[121,VERIFY,110018,0,1]
[122,+,140040,110018,140041]
[123,+,140041,150001,140042]
[124,+,140042,150003,160011]
[125,VERIFY,110018,0,1]
[126,*,110018,150002,140043]
[127,VERIFY,110017,0,1]
[128,+,140043,110017,140044]
[129,+,140044,150001,140045]
[130,+,140045,150009,160012]
[131,*,160011,160012,140046]
[132,+,160010,140046,140047]
[133,=,140047,None,160009]
[134,+,110018,150005,140048]
[135,=,140048,None,110018]
[136,GOTO,None,None,105]
[137,+,110017,150005,140049]
[138,=,140049,None,110017]
[139,GOTO,None,None,102]
[140,+,110016,150005,140050]
[141,=,140050,None,110016]
[142,GOTO,None,None,99]
[143,OUTPUT,None,None,150014]
[144,=,150001,None,110016]
[145,=,150001,None,110017]
[146,<,110016,110019,140051]
[147,GOTOF,140051,None,164]
[148,=,150001,None,110017]
[149,<,110017,110019,140052]
[150,GOTOF,140052,None,161]
[151,VERIFY,110016,0,1]
[152,*,110016,150002,140053]
[153,VERIFY,110017,0,1]
[154,+,140053,110017,140054]
[155,+,140054,150001,140055]
[156,+,140055,150013,160013]
[157,OUTPUT,None,None,160013]
[158,+,110017,150005,140056]
[159,=,140056,None,110017]
[160,GOTO,None,None,149]
[161,+,110016,150005,140057]
[162,=,140057,None,110016]
[163,GOTO,None,None,146]
[164,RETURN,None,None,None]
[165,ENDFUNC,None,None,None]
[166,EOF,None,None,None]
```

Output



```
Starting the Mogavity Virtual Machine
"matrixA[0][0]"
2
"matrixA[0][1]"
3
"matrixA[1][0]"
4
"matrixA[1][1]"
5
"matrixB[0][0]"
2
"matrixB[0][1]"
3
"matrixB[1][0]"
4
"matrixB[1][1]"
5
"=================================="
16
21
28
37
(venv) ➜  mogavity git:(functions2) ✗
```

Test - Fibonacci
This test aims to get the Nth number in the fibonacci sequence, we test it both iteratively and recursively. With this test, we test arrays, loops, return, function calls, recursion and scope management.
Test Code

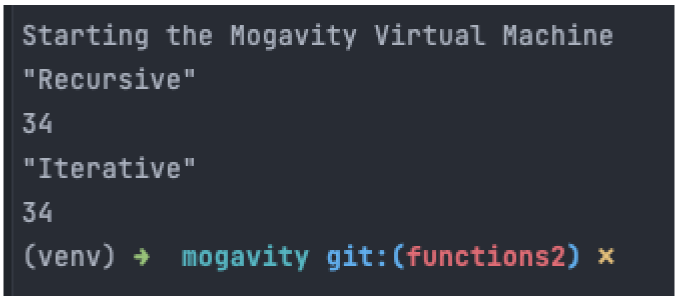
```
program test;

var int a;

instr int fibo_recursive(int num, int a, int b){
   if(num == 0){
       return a;
   }
   if(num == 1){
       return b;
   }
   a = _fibo_recursive(num-1, b, a+b);
   return a;
}

instr int fibo_iterative(int num){
   var int sequence[100];
   var int i;
   i = 2;
   sequence[0] = 0;
   sequence[1] = 1;

   while(i<=num){
       sequence[i] = sequence[i-2] + sequence[i-1];
       i = i + 1;
   }
   return sequence[num];
}

main{
   output -> "Recursive";
   a = _fibo_recursive(9,0,1);
   output -> a;
   output -> "Iterative";
   a = _fibo_iterative(9);
   output -> a;
   return 0;
}
```

Intermediate Code

```
[1,GOTO,None,None,55]
[2,==,10000,150000,40000]
[3,GOTOF,40000,None,6]
[4,=,10001,None,110002]
[5,RETURN,None,None,110002]
[6,==,10000,150001,40001]
[7,GOTOF,40001,None,10]
[8,=,10002,None,110002]
[9,RETURN,None,None,110002]
[10,ERA,None,None,fibo_recursive]
[11,-,10000,150001,40002]
[12,PARAMETER,40002,None,10000]
[13,PARAMETER,10002,None,10001]
[14,+,10001,10002,40003]
[15,PARAMETER,40003,None,10002]
[16,GOSUB,fibo_recursive,None,2]
[17,=,110002,None,40004]
[18,=,40004,None,10001]
[19,=,10001,None,110002]
[20,RETURN,None,None,110002]
[21,ENDFUNC,None,None,None]
[22,=,150002,None,10102]
[23,VERIFY,150000,0,99]
[24,+,150000,150000,40000]
[25,+,40000,150003,60000]
```

```
[26,=,150000,None,60000]
[27,VERIFY,150001,0,99]
[28,+,150001,150000,40001]
[29,+,40001,150003,60001]
[30,=,150001,None,60001]
[31,<=,10102,10000,40002]
[32,GOTOF,40002,None,49]
[33,VERIFY,10102,0,99]
[34,+,10102,150000,40003]
[35,+,40003,150003,60002]
[36,-,10102,150002,40004]
[37,VERIFY,40004,0,99]
[38,+,40004,150000,40005]
[39,+,40005,150003,60003]
[40,-,10102,150001,40006]
[41,VERIFY,40006,0,99]
[42,+,40006,150000,40007]
[43,+,40007,150003,60004]
[44,+,60003,60004,40008]
[45,=,40008,None,60002]
[46,+,10102,150001,40009]
[47,=,40009,None,10102]
[48,GOTO,None,None,31]
[49,VERIFY,10000,0,99]
[50,+,10000,150000,40010]
[51,+,40010,150003,60005]
[52,=,60005,None,110003]
[53,RETURN,None,None,110003]
[54,ENDFUNC,None,None,None]
[55,OUTPUT,None,None,150004]
[56,ERA,None,None,fibo_recursive]
[57,PARAMETER,150005,None,10000]
[58,PARAMETER,150000,None,10001]
[59,PARAMETER,150001,None,10002]
[60,GOSUB,fibo_recursive,None,2]
[61,=,110002,None,140000]
[62,=,140000,None,110001]
[63,OUTPUT,None,None,110001]
[64,OUTPUT,None,None,150006]
[65,ERA,None,None,fibo_iterative]
[66,PARAMETER,150005,None,10000]
[67,GOSUB,fibo_iterative,None,22]
[68,=,110003,None,140001]
[69,=,140001,None,110001]
[70,OUTPUT,None,None,110001]
[71,RETURN,None,None,None]
[72,ENDFUNC,None,None,None]
[73,EOF,None,None,None]
```

Output



```
Starting the Mogavity Virtual Machine
"Recursive"
34
"Iterative"
34
(venv) ➜  mogavity git:(functions2) ✗
```

Test - And and Or

This test will try the functionality of our AND and OR operators when used in conditionals.

Test Code

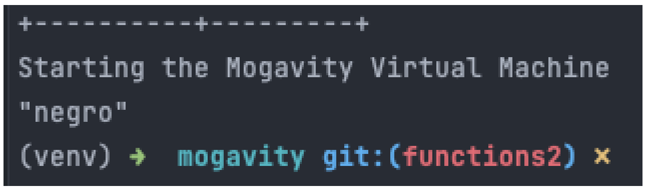
```
program test;

var int a, b, c, d;

main {

   a = 1;
   b = 2;
   c = 3;
   d = 4;

   if ((a > b or c < d) and (c < d and a < d)) {
       output -> "negro";
   }
   otherwise {
       output -> "blanco";
   }
}
```

Intermediate Code

```
[1,GOTO,None,None,2]
[2,=,150000,None,110001]
[3,=,150001,None,110002]
[4,=,150002,None,110003]
[5,=,150003,None,110004]
[6,>,110001,110002,140000]
[7,<,110003,110004,140001]
[8,or,140000,140001,140002]
[9,<,110003,110004,140003]
[10,<,110001,110004,140004]
[11,and,140003,140004,140005]
[12,and,140002,140005,140006]
[13,GOTOF,140006,None,16]
```

[14,OUTPUT,None,None,150004]
[15,GOTO,None,None,17]
[16,OUTPUT,None,None,150005]
[17,ENDFUNC,None,None,None]
[18,EOF,None,None,None]

Output



```
+----------+---------+
Starting the Mogavity Virtual Machine
"negro"
(venv) ➜  mogavity git:(functions2) ✗
```

Test - Arrays

This test does multiplications on arrays, using multiple operands in the same statement as to simulate a very complex operation.

Test Code

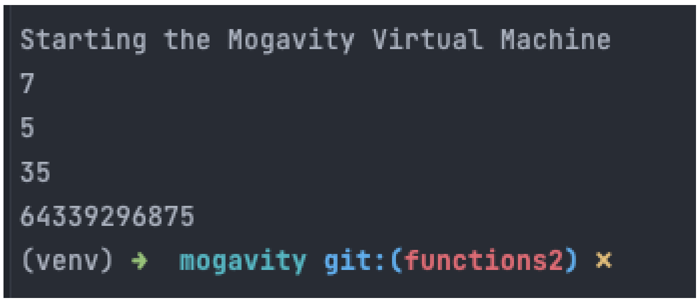
```
program test;

var int A[5];

main {
   A[2] = 7;
   A[3] = 5;
   output -> A[2];
   output -> A[3];
   A[1] = A[2] * A[3];
   output -> A[1];
   A[1] = A[2] * A[3] * A[2] * A[3] * A[2] * A[3] * A[2] * A[3] * A[2] *
A[3] * A[2] * A[3] * A[2] * A[3];
   output -> A[1];
   return 0;
}
```

Intermediate Code

[1,GOTO,None,None,2]
[2,VERIFY,150000,0,4]
[3,+,150000,150001,140000]
[4,+,140000,150002,160000]
[5,=,150003,None,160000]
[6,VERIFY,150004,0,4]
[7,+,150004,150001,140001]
[8,+,140001,150002,160001]
[9,=,150005,None,160001]
[10,VERIFY,150000,0,4]

```
[11,+,150000,150001,140002]
[12,+,140002,150002,160002]
[13,OUTPUT,None,None,160002]
[14,VERIFY,150004,0,4]
[15,+,150004,150001,140003]
[16,+,140003,150002,160003]
[17,OUTPUT,None,None,160003]
[18,VERIFY,150006,0,4]
[19,+,150006,150001,140004]
[20,+,140004,150002,160004]
[21,VERIFY,150000,0,4]
[22,+,150000,150001,140005]
[23,+,140005,150002,160005]
[24,VERIFY,150004,0,4]
[25,+,150004,150001,140006]
[26,+,140006,150002,160006]
[27,*,160005,160006,140007]
[28,=,140007,None,160004]
[29,VERIFY,150006,0,4]
[30,+,150006,150001,140008]
[31,+,140008,150002,160007]
[32,OUTPUT,None,None,160007]
[33,VERIFY,150006,0,4]
[34,+,150006,150001,140009]
[35,+,140009,150002,160008]
[36,VERIFY,150000,0,4]
[37,+,150000,150001,140010]
[38,+,140010,150002,160009]
[39,VERIFY,150004,0,4]
[40,+,150004,150001,140011]
[41,+,140011,150002,160010]
[42,*,160009,160010,140012]
[43,VERIFY,150000,0,4]
[44,+,150000,150001,140013]
[45,+,140013,150002,160011]
[46,*,140012,160011,140014]
[47,VERIFY,150004,0,4]
[48,+,150004,150001,140015]
[49,+,140015,150002,160012]
[50,*,140014,160012,140016]
[51,VERIFY,150000,0,4]
[52,+,150000,150001,140017]
[53,+,140017,150002,160013]
[54,*,140016,160013,140018]
[55,VERIFY,150004,0,4]
[56,+,150004,150001,140019]
[57,+,140019,150002,160014]
[58,*,140018,160014,140020]
[59,VERIFY,150000,0,4]
[60,+,150000,150001,140021]
[61,+,140021,150002,160015]
[62,*,140020,160015,140022]
[63,VERIFY,150004,0,4]
[64,+,150004,150001,140023]
```

[65,+,140023,150002,160016] [66,*,140022,160016,140024] [67,VERIFY,150000,0,4] [68,+,150000,150001,140025] [69,+,140025,150002,160017] [70,*,140024,160017,140026] [71,VERIFY,150004,0,4] [72,+,150004,150001,140027] [73,+,140027,150002,160018] [74,*,140026,160018,140028] [75,VERIFY,150000,0,4] [76,+,150000,150001,140029] [77,+,140029,150002,160019] [78,*,140028,160019,140030] [79,VERIFY,150004,0,4] [80,+,150004,150001,140031] [81,+,140031,150002,160020] [82,*,140030,160020,140032] [83,VERIFY,150000,0,4] [84,+,150000,150001,140033] [85,+,140033,150002,160021] [86,*,140032,160021,140034] [87,VERIFY,150004,0,4] [88,+,150004,150001,140035] [89,+,140035,150002,160022] [90,*,140034,160022,140036] [91,=,140036,None,160008] [92,VERIFY,150006,0,4] [93,+,150006,150001,140037] [94,+,140037,150002,160023] [95,OUTPUT,None,None,160023] [96,RETURN,None,None,None] [97,ENDFUNC,None,None,None] [98,EOF,None,None,None]
Output



```
Starting the Mogavity Virtual Machine
7
5
35
64339296875
(venv) ➜  mogavity git:(functions2) ✗
```

Test - Search
Using a linear search, we generate an array with numbers and then print it out, then the user selects a number and the program returns the index for such a number in the array or an error message if the number was not found. Using this test, we verify the functionality of arrays, function calls, IFs, and loops.

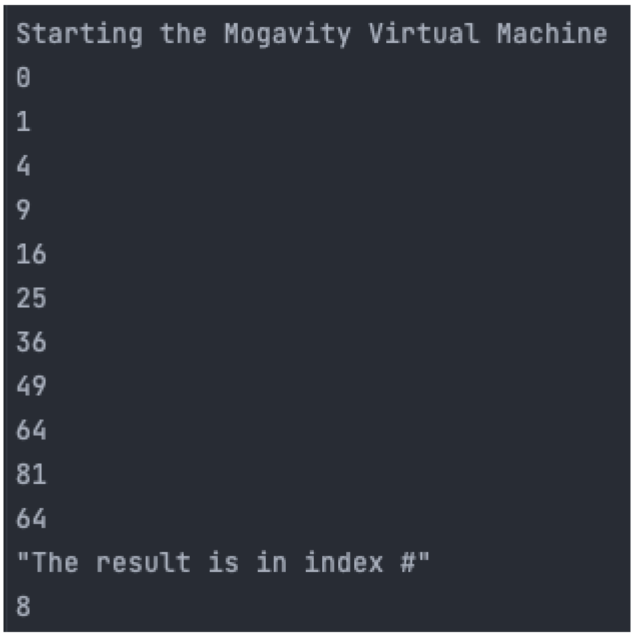
Test Code

```
program search;

var int A[10];
var int result, LEN, a, b;

instr int search(int to_find){
   a = 0;
   while(a<LEN){
       if(A[a] == to_find){
           return a;
       }
       a = a + 1;
   }
   return 999999;
}

main{
   LEN = 10;
   a = 0;
   while(a<LEN){
       A[a] = a*a;
       a = a + 1;
   }

   a = 0;
   while(a<LEN){
       output -> A[a];
       a = a + 1;
   }

   input <- b;
   result = _search(b);

   if(result == 999999){
       output -> "Not found";
   }otherwise{
       output -> "The result is in index #";
       output -> result;
   }

   return 0;
}
```

Intermediate Code

```
[1,GOTO,None,None,18]
[2,=,140000,None,110014]
[3,<,110014,110013,30000]
[4,GOTOF,30000,None,15]
[5,VERIFY,110014,0,9]
[6,+,110014,140000,30001]
[7,+,30001,140001,50000]
[8,==,50000,10000,30002]
[9,GOTOF,30002,None,12]
[10,=,110014,None,110016]
[11,RETURN,None,None,110016]
[12,+,110014,140002,30003]
[13,=,30003,None,110014]
[14,GOTO,None,None,3]
[15,=,140003,None,110016]
[16,RETURN,None,None,110016]
[17,ENDFUNC,None,None,None]
[18,=,140004,None,110013]
[19,=,140000,None,110014]
[20,<,110014,110013,130000]
[21,GOTOF,130000,None,30]
[22,VERIFY,110014,0,9]
[23,+,110014,140000,130001]
[24,+,130001,140001,150000]
[25,*,110014,110014,130002]
[26,=,130002,None,150000]
[27,+,110014,140002,130003]
[28,=,130003,None,110014]
[29,GOTO,None,None,20]
[30,=,140000,None,110014]
[31,<,110014,110013,130004]
[32,GOTOF,130004,None,40]
[33,VERIFY,110014,0,9]
[34,+,110014,140000,130005]
[35,+,130005,140001,150001]
[36,OUTPUT,None,None,150001]
[37,+,110014,140002,130006]
[38,=,130006,None,110014]
[39,GOTO,None,None,31]
[40,INPUT,None,None,110015]
[41,ERA,None,None,search]
[42,PARAMETER,110015,None,10000]
[43,GOSUB,search,None,2]
[44,=,110016,None,130007]
[45,=,130007,None,110012]
[46,==,110012,140003,130008]
[47,GOTOF,130008,None,50]
[48,OUTPUT,None,None,140005]
[49,GOTO,None,None,52]
[50,OUTPUT,None,None,140006]
[51,OUTPUT,None,None,110012]
[52,RETURN,None,None,None]
[53,ENDFUNC,None,None,None]
[54,EOF,None,None,None]
```

Output



```
Starting the Mogavity Virtual Machine
0
1
4
9
16
25
36
49
64
81
64
"The result is in index #"
8
```

Test - Class inheritance

Declares a class “Animal” and a class “Perro”. Assigns values to its attributes and demonstrates the functionality of the parent class’ method, using the child object.

Test Code

```
program test;

class Animal {
    attr:
        int animal;
    constr:
        Animal(){
            output -> "Un animal nace!";
        }
    methods:
        method void que_soy(){
            output -> "Hola, soy un animal";
        }
}
```

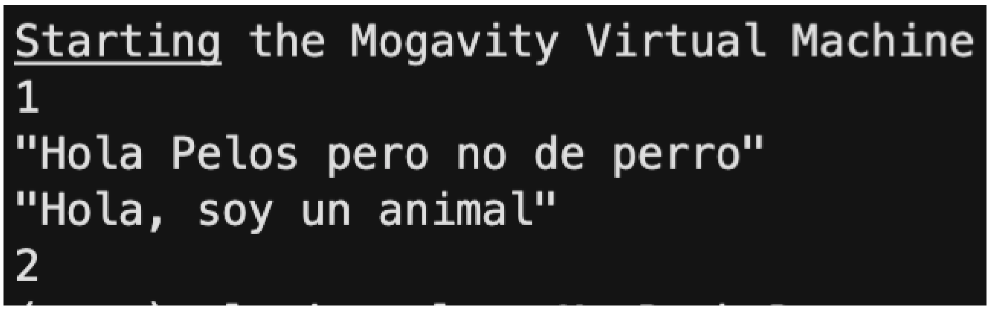
```
class Perro inherits Animal {
    attr:
        int a, b;
    constr:
        Perro(){

        }
    methods:
        method void pelos2(){
            output -> "Hola, pelos de perro";
        }
}


var Perro perro;
var int a;

instr void pelos(){
     output -> "Hola Pelos pero no de perro";
     return 0;
}

main {
    perro.a = 1;
    perro.b = 2;
    output -> perro.a;
    _pelos();
    _perro.que_soy();
    output -> perro.b;
    return 0;
}
```

Intermediate Code

```
[1,GOTO,None,None,10]
[2,OUTPUT,None,None,150000]
[3,OUTPUT,None,None,150001]
[4,ENDFUNC,1,None,None]
[5,OUTPUT,None,None,150002]
[6,ENDFUNC,1,None,None]
[7,OUTPUT,None,None,150003]
[8,RETURN,None,None,None]
[9,ENDFUNC,None,None,None]
[10,=,150005,None,110002]
[11,=,150006,None,110003]
[12,OUTPUT,None,None,110002]
[13,ERA,None,None,pelos]
[14,GOSUB,pelos,None,7]
[15,ERA,None,None,perro.que_soy]
```

```
[16,GOSUB,2,None,3]
[17,OUTPUT,None,None,110003]
[18,RETURN,None,None,None]
[19,ENDFUNC,None,None,None]
[20,EOF,None,None,None]
```

Output



```
Starting the Mogavity Virtual Machine
1
"Hola Pelos pero no de perro"
"Hola, soy un animal"
2
```

VIII. Referencias

PLY: http://www.dabeaz.com/ply/