Pp Language Final Documentation

Dr. Héctor Ceballos
Ing. Elda Quiroga
TC3048 Compilers Design

Made by
Eduardo Enrique Trujillo Ramos
A01187313

Hugo Oswaldo Garcia Perez
A00815354

Project found at:
https://github.com/eduardotru/Pp-language
Demo at:
http://bit.ly/PpLanguageDemo

2019-11-27

1. Project Description

a. Purpose of the Project

The purpose of this project is to develop a language focused mainly on engineers and scientists who do statistical analysis and use linear algebra in their day to day life. The main data structure the language will support is the matrix or dataset, which will allow the storage of two-dimensional data. This language will allow the users to speed up their workflow and have a nicer syntax than other languages with this focus such as R but also not be so advanced and difficult to learn for non-programmers such as Python.

b. Language Main Objective

The main objective of Pp is to be readable by both, developers and non-developers, to be a common ground of collaboration between them and be intuitive and easy to program. The language will be a high-level imperative scripting programming language for statistical analysis, having as the basic data structure the matrix, supporting all of its operations as well as basic statistic functions.

c. Scope of the Project

The scope of the project is to create a Programming Language which includes a Compiler and a Virtual Machine to execute the code. The Compiler should have at least the following stages: Lexical Analysis, Syntax Analysis, Semantic Analysis and Generation of Intermediate Code. Intermediate code optimization and generation of machine code is not part of the project scope. The Virtual Machine only needs to execute the intermediate code generated by the compiler and manage the memory. Code optimization or any other execution time optimizations like branch prediction are out of the scope.

Specifically for this project, features like Classes and Objects, Multithreading, File manipulation, Dynamic Typing, and others not described in this document are out of the scope of the project.

d. Language Requirements

1. The Language must have the following statements: Assignment, Conditions, Cycles, Input, and Output
2. The Language must have the following expressions: Arithmetic, Logic, and Relational.
3. The Language must include Modules, such as Functions with and without parameters as well as have local and global variables.
4. The Language must contain at least ONE structured element such as Arrays, Matrices, Lists, etc.

e. General Use Cases

With our project, the developer can:

- Compile Pp code
- Execute compiled Pp code
- Interact with the virtual machine executioning code through I/O
- Code modules/functions
- Use flow control and decision making
- Use basic statistical functions such as mean, mode, etc
- Calculate probabilities for some statistical distributions
- Plot data
- Read runtime, semantic or syntax errors

f. Main Test Cases

To test the functionality of this programming languages, the following test cases were designed:

Test Case Name	Description	Purpose
Fibonacci	This program calculates the nth Fibonacci number using an iterative approach.	The purpose of this test is to make sure loops, assignments, and functions work.
Recursive Fibonacci	This program calculates the nth Fibonacci number using a recursive approach.	The purpose of this test is to confirm recursion and conditionals work. This test makes sure the Virtual Machine makes good management of the stack memory.
Matrix exponentiation Fibonacci	This program calculates the nth Fibonacci number making use of Matrix Exponentiation.	The purpose of this test is to work with matrices and its operations, in this case the exponentiation. It also makes a small test over matrix literals.
Factorial	This program calculates the Factorial function of a number, for example 10!, using an iterative approach.	The purpose of this test is to make sure loops, assignments, and functions work.
Recursive Factorial	This program calculates the Factorial function of a number, for example 10!, using a recursive approach.	The purpose of this test is to confirm recursion and conditionals work. This test makes sure the Virtual Machine makes good management of the stack memory.
Merge Sort	This program implements the merge sort algorithm for sorting a list.	The purpose of this test is to check matrices work well with functions and you can pass matrices as parameters and return matrices in functions. It also tests recursion and function calls.
Find Matrix	This program implements an iterative search over a matrix and returns the index of the value that is being searched.	This test makes sure you can iterate and index matrices correctly.
Linear Regression	This program calculates the linear regression for a set of 10 points using the mean square error estimator and calculating the intercept and slope using matrix operations	This test makes sure matrices can be plotted and that the inverse of a matrix can be calculated as well as matrix multiplication.
Histogram	This program generates 1000	This test makes sure probability

	random variables which follow a gaussian distribution and makes a histogram with them.	distributions work and that we can generate random variables. It also tests the hist function to make histograms.
Stat Functions	This program makes use of some statistical analysis functions over a matrix.	This test makes sure all the statistical functions such as mean, median, mode, variance, and stdev work.

g. Development Process and Our Thoughts

The development process followed was an iterative process. The first thing we did was design the first data types, variable declaration, statements, etc, the basic functions of every programming language and after that we decided what was out language going to specialize in. Before developing, first we focused on the whole grammar and verified that it did not contain any left recursions or ambiguities. When we were satisfied with the grammar we started the developing process. At first, the only thing we were checking was variable declaration and functions declarations. With the SymbolsTable we were able to print each and every variable declared in our program and it's scope (we didn't even have the type). The process continued, making expressions work, types, semantic checking, control flow, starting to print quadruples, and finally executing code with the Virtual Machine.

The way the developers split the work is Hugo was in charge of the data structures in the compiles, such as Quadruples and SymbolsTable, and Eduardo was in charge of the Grammar and PpListener. Hugo was also mainly in charge of the execution of code in the VirtualMachine and Eduardo of the mapping of Memory in the VirtualMachine.

i. Hugo Garcia Thoughts

Despite not being the first compiler I've built it raised up to my mind many new problems and challenges. What I like about compilers is that they are deterministic, so as long as you hold the current flow thread you can find exactly what it's going to do next. Of course, to successfully survive this project we needed strong knowledge from previous classes and specially from problem solving skills.

Signature: _____

ii. Eduardo Trujillo Thoughts

The process of building a compiler and creating your own language was challenging because of the size of the project since you had to think ahead all of the steps of the compilation, from the lexical analysis all the way to the execution of the code. It was a fun project since you were free to do as you wish with the language and the implementation, the only requirements were that you had the basics of a programming language. I think that this project is only doable at this point in our university career since we need a lot of built-up knowledge from previous classes such as Data Structures, Algorithms, Computational Math, Discrete Math, and others. I feel more ready to go out into the world and be a software engineer than I was at the beginning of the semester.

Signature: _____

2. Language Description

a. Language Name

The programming language here described is called Pp (pronounced Peh-peh). The name just comes from naming the language as a popular name in Mexico, Jose, but Jose's are informally called Pepe, hence the name Pp was born.

b. Language Characteristics

Pp is a high-level imperative scripting language with strict typing. Pp overloads operations over matrices such as addition and multiplication for ease of use of scientists and statisticians. This language is intended to be as readable by programmers as it is to non programmers who need a powerful tool for numerical analysis.

Pp offers a variety of builtin functions such as probability distributions, statistical analysis functions, and plotting capabilities and hist to display data in a more visual way. Pp is single threaded and does not allow variable declaration in any scope, only being able to declare variables in a global and a local scope in functions.

c. Compile and Execution errors

i. Compilation Errors

Syntax and Lexical errors are managed by ANTLR. The line where the error occurred will be shown as well as a message such as "mismatched input", "missing token" or "no viable alternative at input" and the compilation will finish.

Semantic errors are managed by us and all of them include a and they can be of the following types:

Error message:	
`Redefinition of function {func_name} at {line}:{column}`	`Use of undefined variable {var_name} at {line}:{column}`
`Redefinition of variable '{var_name}' at {line}:{column}`	`Incorrect number of parameters given to {func_name} at {line}:{column}`
`Incompatible types on operation {left_type}{operator}{right_type} at {line}:{column}`	`Incompatible parameter type. Expected {type}, found {type} at {line}:{column}`
`Use of undeclared function {func_name} at {line}:{column}`	`Cannot read into structured type at {line}:{column}`
`Incompatible return type. Expected {type}, got {type} at {line}:{column}`	`{num_index} index of matrix must be integer at {line}:{column}`
`Invalid indexation to non matrix at {line}:{column}`	`Matrix literal has incompatible types at {line}:{column}`
`Matrix literal has extraneous dimensions at {line}:{column}`	`Cannot get {stat_func} of non matrix type at {line}:{column}`
`Cannot get {stat_func} of non numeric type at {line}:{column}`	

ii. Execution Errors

There are only three types of errors that occur during the execution of Pp code which are segmentation faults, division by zero, and trying to get the inverse of a matrix with no inverse.

3. Compiler Description

a. Development Environment

The programming language here specified will be developed using ANTLR as the syntax and lexical analyzer with Python. The environment in which it will be developed and tested will be Linux Ubuntu 16.04+

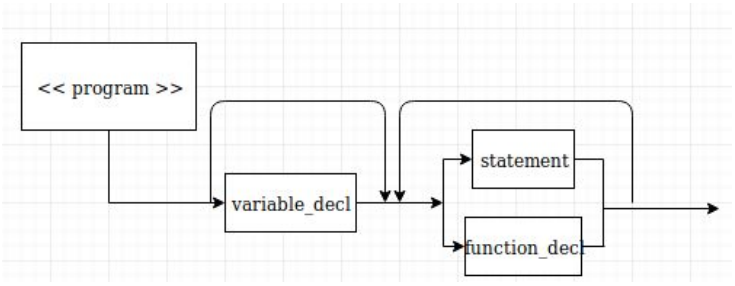
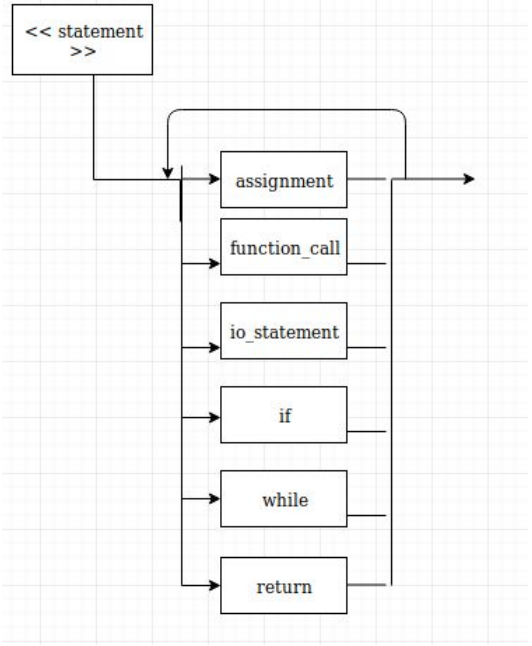
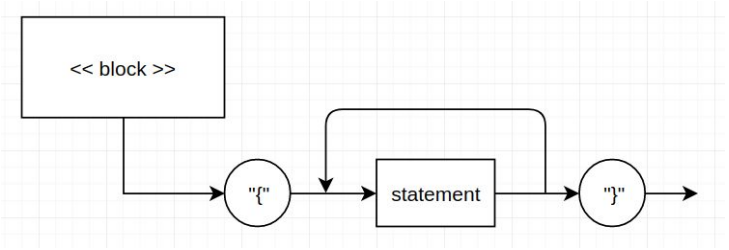
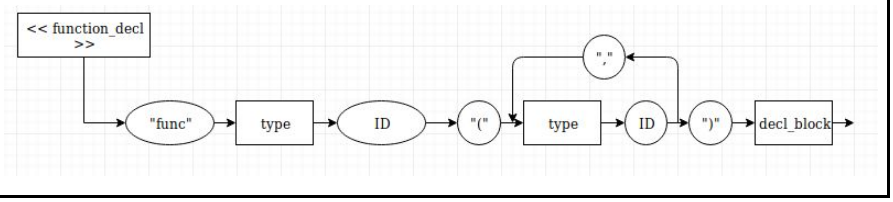
b. Lexical Analysis (Tokens)

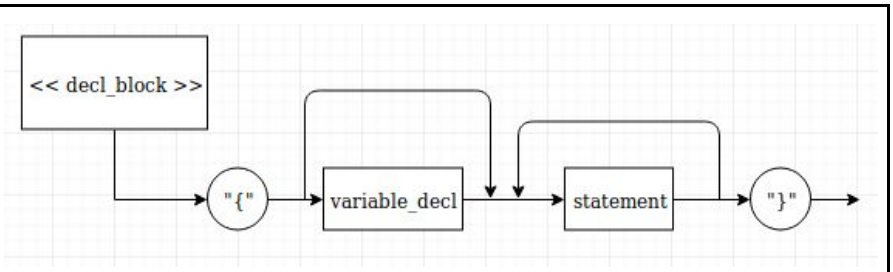
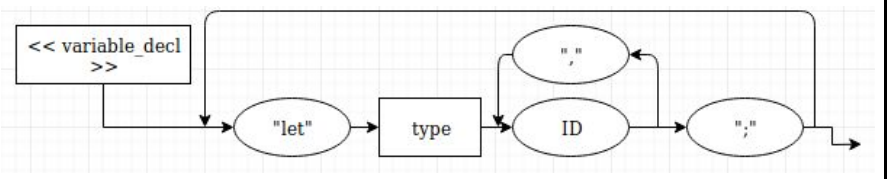
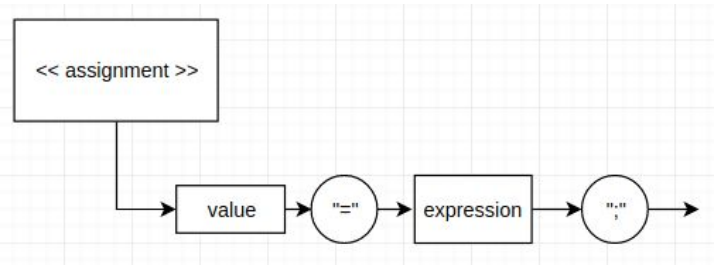
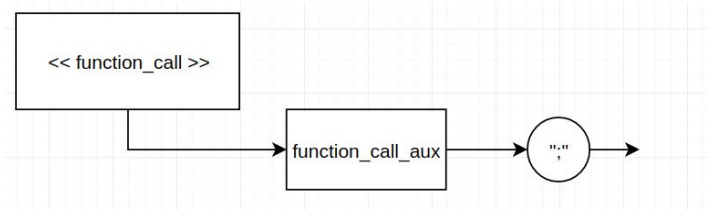
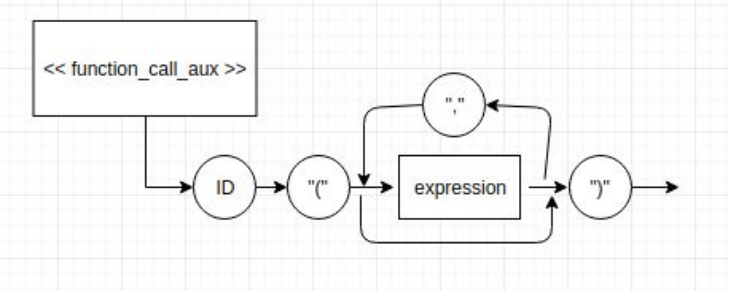
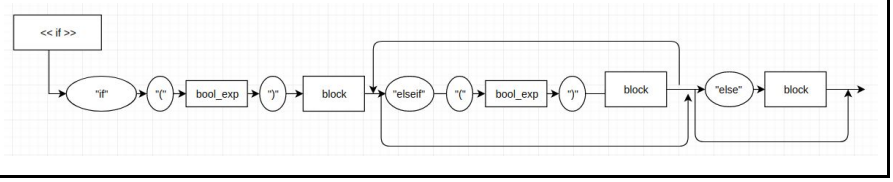
Allowed Tokens	**Regex**	**Allowed Tokens**	**Regex**
ADDITION_OP	“+”	COMMA_DELIMITER	“,”
SUBTRACTION_OP	“-”	INPUT_STATEMENT	“read”
MULTIPLICATION_OP	“*”	OUTPUT_STATEMENT	“write”
DIVISION_OP	“/”	PLOT_STATEMENT	“plot”
EXPONENTIATION_OP	“^”	HIST_STATEMENT	“hist”
MODULUS_OP	“%”	SHOWPLOT_STATEMENT	“showplot”
ASSIGNMENT_OP	“=”	VARIABLE_STATEMENT	“let”
BOOLEAN_AND_OP	“and”	FUNCTION_STATEMENT	“func”
BOOLEAN_OR_OP	“or”	RETURN_STATEMENT	“return”
BOOLEAN_NOT_OP	“not”	IF_STATEMENT	“if”
EQUALITY_OP	“==”	ELSE_IF_STATEMENT	“elseif”
INEQUALITY_OP	“!=”	ELSE_STATEMENT	“else”
LESS_THAN_OP	“<”	WHILE_LOOP_STATEMENT	“while”
LESS_THAN_EQUAL_OP	“<=”	LEFT_SQUARE_BRACKET	“[“
GREATER_THAN_OP	“>”	RIGHT_SQUARE_BRACKET	“]”
GREATER_THAN_EQUAL_OP	“>=”	LEFT_CURLY_BRACKET	“{“
SEMICOLON_DELIMITER	“;”	RIGHT_CURLY_BRACKET	“}”
LEFT_PARENTHESIS	“(“	STAT_FUNCTION_DGAMMA	“dgamma”
RIGHT_PARENTHESIS	“)”	STAT_FUNCTION_DGEOM	“dgeom”

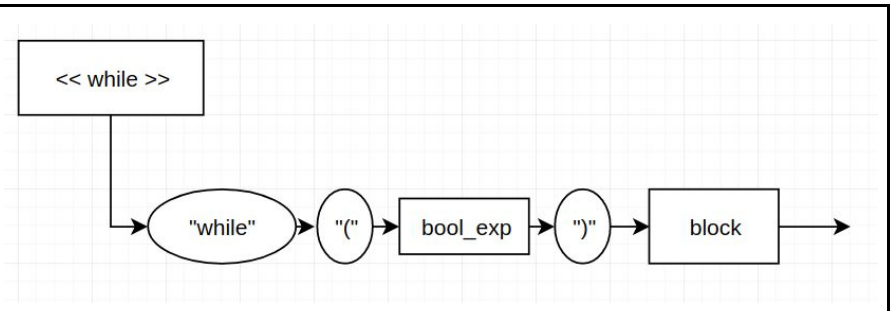
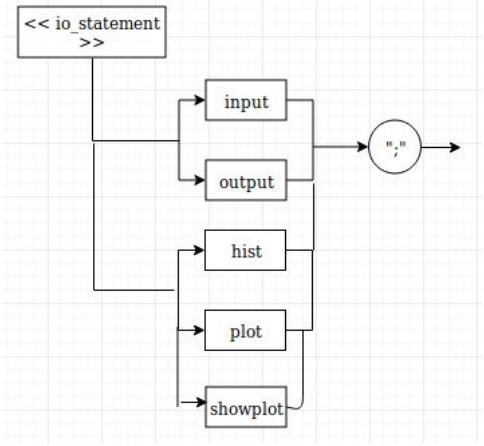
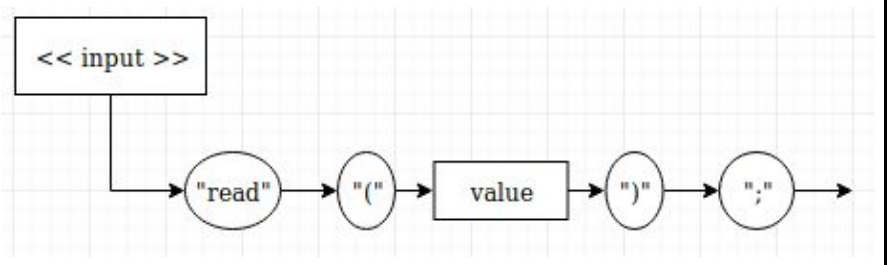
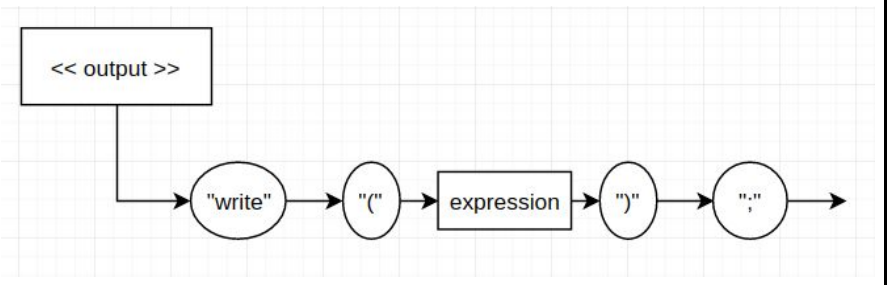
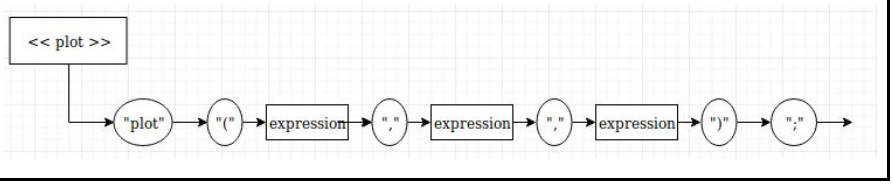
INT_TYPE	“int”	STAT_FUNCTION_DNORM	“dnorm”
BOOLEAN_TYPE	“bool”	STAT_FUNCTION_DPOIS	“dpois”
FLOAT_TYPE	“float”	STAT_FUNCTION_DUNIF	“dunif”
STRING_TYPE	“string”	STAT_FUNCTION_CBETA	“cbeta”
MATRIX_TYPE	“matrix”	STAT_FUNCTION_CBINOM	“cbinom”
ID	“_”?[a-zA-Z][a-zA-Z0-9_]*	STAT_FUNCTION_CEXP	“cexp”
INT_NUMBER	[+|-]?[0-9]+	STAT_FUNCTION_CGAMMA	“cgamma”
FLOAT_NUMBER	[+|-]?[0-9]+”.”([0-9]+)(E[-|+]?[0-9]+)	STAT_FUNCTION_CGEOM	“cgeom”
BOOLEAN_LITERAL	(“true” | ”false”)	STAT_FUNCTION_CNORM	“cnorm”
STRING_LITERAL	“‘.*?’”	STAT_FUNCTION_CPOIS	“cpois”
STAT_FUNCTION_MEAN	“mean”	STAT_FUNCTION_CUNIF	“cunif”
STAT_FUNCTION_MEDIAN	“median”	STAT_FUNCTION_RBETA	“rbeta”
STAT_FUNCTION_MODE	“mode”	STAT_FUNCTION_RBINOM	“rbinom”
STAT_FUNCTION_STDEV	“stdev”	STAT_FUNCTION_REXP	“rexp”
STAT_FUNCTION_VARIANCE	“variance”	STAT_FUNCTION_RGAMMA	“rgamma”
MATRIX_FUNCTION_TRANSPOSE	“transpose”	STAT_FUNCTION_RGEOM	“rgeom”
STAT_FUNCTION_DBETA	“dbeta”	STAT_FUNCTION_RNORM	“rnorm”
STAT_FUNCTION_DBINOM	“dbinom”	STAT_FUNCTION_RPOIS	“rpois”
STAT_FUNCTION_DEXP	“dexp”	STAT_FUNCTION_RUNIF	“runif”

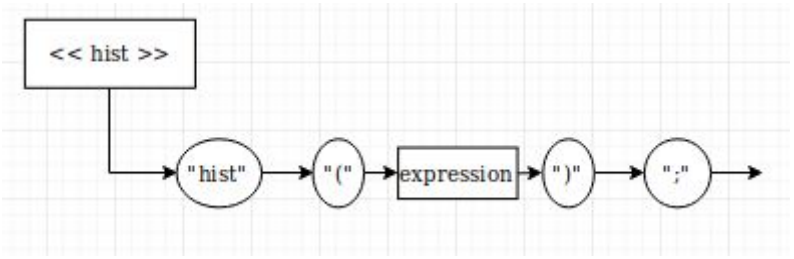
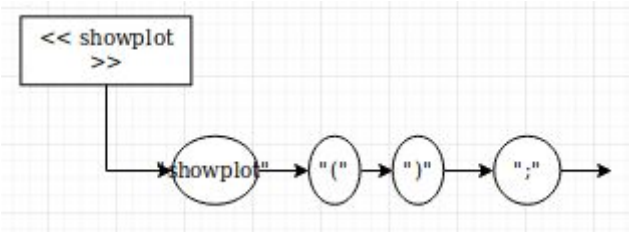
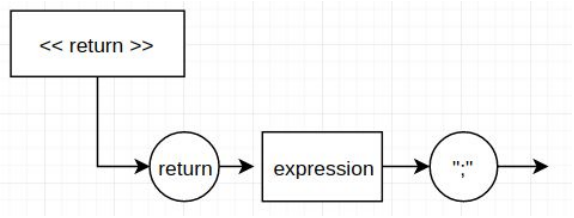
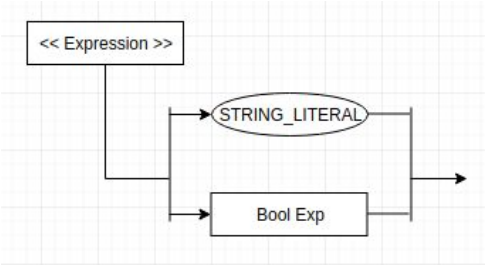
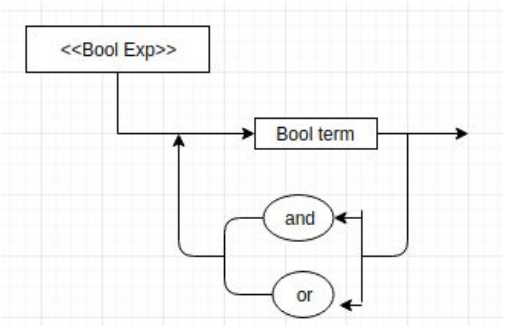
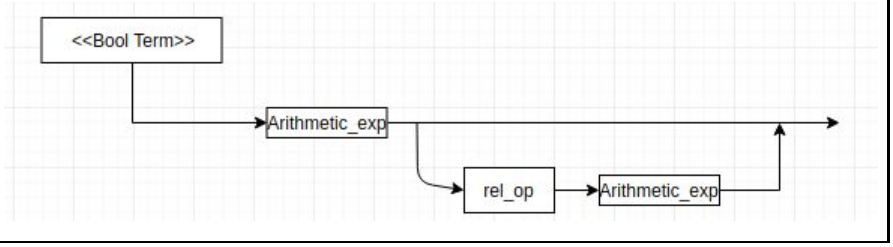
c. Syntax Analysis (Grammar)

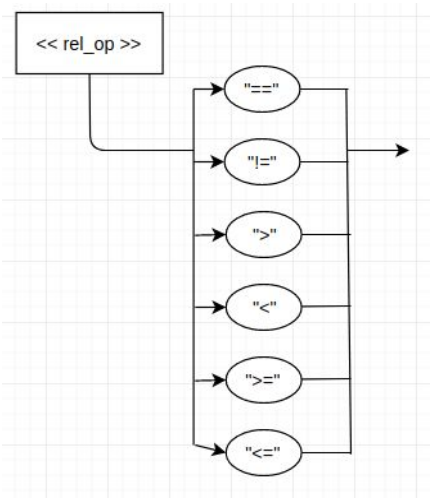
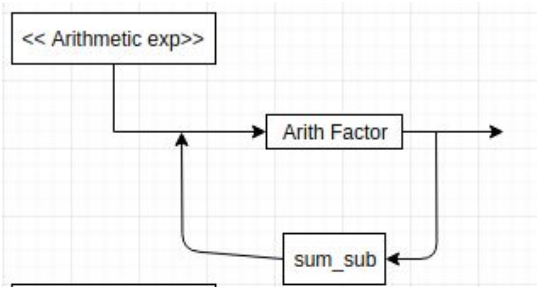
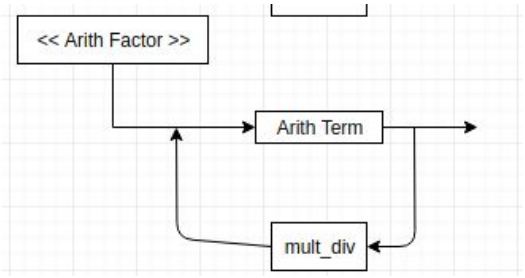
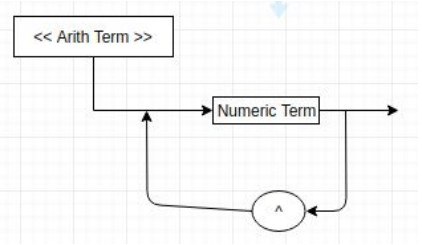
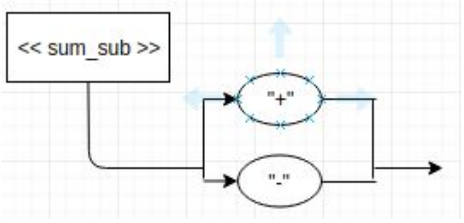
i. Syntax Diagrams

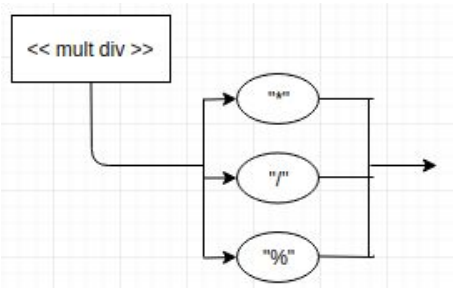
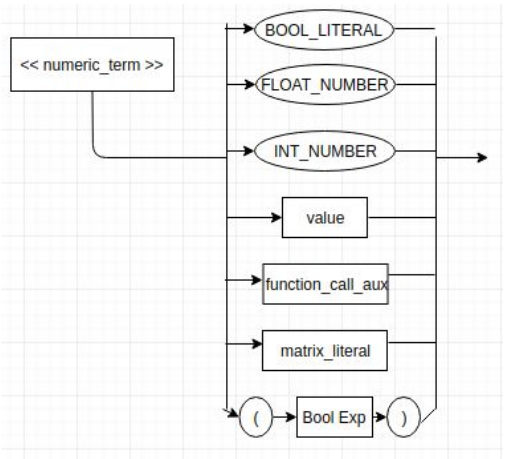
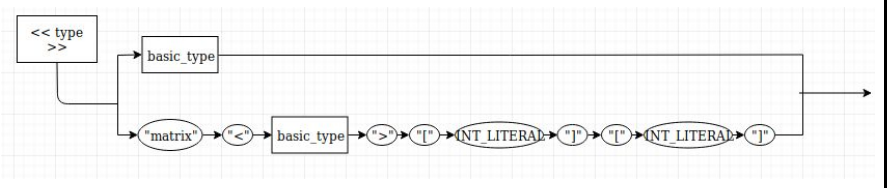
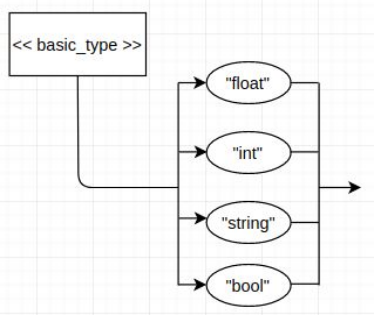
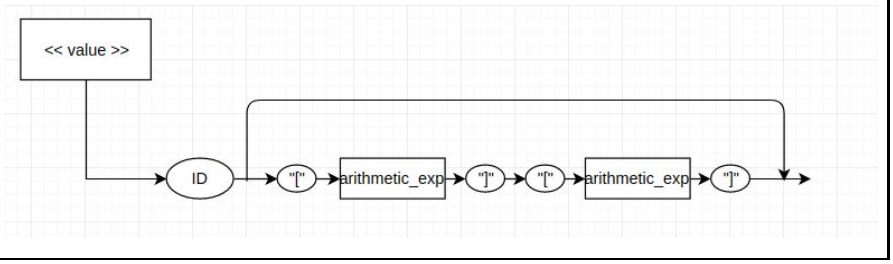
Rule	Diagram
program	
statement	
block	
function_decl	

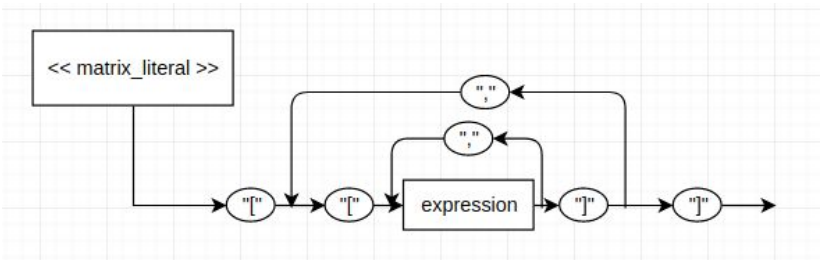
decl_block	
variable_decl	
assignment	
function_call	
function_call_aux	
if	

while	
io_statement	
input	
output	
plot	

hist	
showplot	
return	
expression	
bool_exp	
bool_term	

rel_op	
arithmetic_exp	
Arith Factor	
Arith Term	
Sum_sub	

Mult_div	
numeric_term	
type	
basic_type	
value	

matrix_literal	

ii. Formal Grammar

```
grammar Pp;

r : program0 ;

// Regular Expressions

INT_NUMBER : [+|-]?[0-9]+
;
FLOAT_NUMBER :
[+|-]?[0-9]+'.'([0-9]+)?(
'E'[+|-]?[0-9]+)? ;
STRING_LITERAL :
'"'.*?'"' ;
BOOL_LITERAL : 'true' |
'false' ;
ID :
'_'?[a-zA-Z][a-zA-Z0-9_]*
;
WS : [\t\r\n ]+ -> skip ;

// Grammar

LINE_COMMENT : '//'
~[\r\n]* -> skip
  ;

program0 :
  variable_decl0 program1
  | program1
  ;
program1 :
  statement0 program1
  | function_decl0
program1
  | // empty
  ;

statement0 :
  assignment0
  | function_call0
  | io_statement0
  | if0
  | while0
  | return0
  ;

block0 :
  '{' block1 '}'
  ;

block1 :
  statement0 block1
  | // empty
  ;

function_decl0 :
  'func' function_type0
ID '('
parameters_or_empty0 ')'
decl_block0
  ;

function_type0 :
  'void'
  | type0
  ;

decl_block0 :
  '{' variable_decl0
decl_block1 '}'
  ;

decl_block1 :
  statement0 decl_block1
  | // empty
  ;

parameters_or_empty0:
  parameters0
  | // empty
  ;

parameters0 :
  type0 ID parameters1
  ;

parameters1 :
  ',' parameters0
  | // empty
  ;

variable_decl0 :
  'let' type0 ID
variables_decl1 ';'
variable_decl0
  | // empty
  ;

variables_decl1 :
  ',' ID variables_decl1
  | // empty
  ;

assignment0 :
  value0 '=' expression0
';'
  ;

function_call0 :
  function_call_aux0 ';'
  ;

function_call_aux0 :
  ID '('
function_call_aux1 ')'
  ;

function_call_aux1 :
  expression0
function_call_aux2
  | // empty
  ;

function_call_aux2 :
  ',' expression0
function_call_aux2
  | // empty
  ;

if0 :
  'if' '(' bool_exp0 ')'
block0 else0
  ;

else0 :
  'elseif' '(' bool_exp0
')' block0 else0
  | 'else' block0
  | // empty
  ;

while0 :
  'while' '(' bool_exp0
')' block0
  ;


io_statement0 :
  input0
  | output0
  | plot0
  | hist0
  | showplot0
  ;

input0 :
  'read' '(' value0 ')'
';'
  ;

output0 :
  'write' '(' expression0
')' ';'
  ;
```

```
plot0 :
  'plot' '(' expression0
',' expression0 ','
expression0 ')' ';'
  ;

hist0 :
  'hist' '(' expression0
')' ';'
  ;

showplot0 :
  'showplot' '(' ')' ';'
  ;

return0 :
  'return' expression0
';'
  ;

expression0 :
  STRING_LITERAL
  | bool_exp0
  ;

bool_exp0 :
  | bool_term0 bool_exp1
  | bool_not0 bool_exp0
bool_exp1
  ;

bool_exp1 :
  bool_op0 bool_exp0
bool_exp1
  | // empty
  ;

// X -> Xa | b => X ->
bX', X' -> aX' | eps;

bool_term0 :
  arithmetic_exp0
bool_term1
  ;

bool_term1 :
  rel_op0 arithmetic_exp0
  | // empty
  ;

bool_not0 :
  'not'
  ;

bool_op0 :
  'and'
  | 'or'
  ;

rel_op0 :
  '=='
  | '!='
  | '>'
  | '<'
  | '>='
  | '<='
  ;

arithmetic_exp0 :
  arithmetic_factor0
arithmetic_exp1
  ;

arithmetic_exp1 :
  addition_subtraction0
arithmetic_factor0
arithmetic_exp1
  | // empty
  ;

addition_subtraction0 :
  '+'
  | '-'
  ;

arithmetic_factor0 :
  arithmetic_term0
arithmetic_factor1
  ;

arithmetic_factor1 :

multiplication_division0
arithmetic_term0
arithmetic_factor1
  | // empty
  ;

multiplication_division0
:
  '*'
  | '/'
  | '%'
  ;

arithmetic_term0 :
  numeric_term0
arithmetic_term1
  ;

arithmetic_term1 :
  exponent0 numeric_term0
  | // empty
  ;

exponent0 :
  '^'
  ;

numeric_term0 :
  INT_NUMBER
  | FLOAT_NUMBER
  | BOOL_LITERAL
  | sign0 numeric_term1
  |  '(' bool_exp0 ')'
  ;

numeric_term1 :
  value0
  | stat_functions0
  | function_call_aux0
  | matrix_literal0
  ;

sign0 :
  '-'
  | // empty
  ;

type0 :
  basic_type0
  | 'matrix' '<'
basic_type0 '>' '['
INT_NUMBER ']' '['
INT_NUMBER ']'
  ;

basic_type0 :
  'float'
  | 'int'
  | 'string'
  | 'bool'
  ;

value0 :
 ID value1
 ;

value1 :
  '[' expression0 ']' '['
expression0 ']'
  | // empty
  ;

matrix_literal0 :
  '[' matrix_literal1 ']'
  ;

matrix_literal1 :
  '[' matrix_literal2 ']'
matrix_literal3
  | // empty
  ;

matrix_literal2 :
  expression0
matrix_literal4
  | // empty
  ;

matrix_literal3 :
  ',' '[' matrix_literal2
']' matrix_literal3
  | // empty
  ;

matrix_literal4 :
  ',' expression0
matrix_literal4
  | // empty
  ;

stat_functions0 :
  mean0
  | median0
  | mode0
  | stdev0
  | variance0
  | transpose0
  | beta0
  | binom0
  | exp0
  | gamma0
  | geom0
  | norm0
  | pois0
  | unif0
  ;

mean0 :
```

```
  'mean' '(' expression0
')'
  ;

median0 :
  'median' '('
expression0  ')'
  ;

mode0 :
  'mode' '(' expression0
')'
  ;

stdev0 :
  'stdev' '(' expression0
')'
  ;

variance0 :
  'variance' '('
expression0  ')'
  ;

transpose0 :
  'transpose' '('
expression0  ')'
  ;

beta0 :
  dbeta0
  | cbeta0
  | rbeta0
  ;

dbeta0 :
  'dbeta' '(' expression0
',' expression0 ','
expression0 ')'
  ;

cbeta0 :
  'cbeta' '(' expression0
',' expression0 ','
expression0 ')'
  ;

rbeta0 :
  'rbeta' '(' expression0
',' expression0 ')'
  ;

binom0 :
  dbinom0
  | cbinom0
  | rbinom0
  ;

dbinom0 :
  'dbinom' '('
expression0 ','
expression0 ','
expression0 ')'
  ;

cbinom0 :
  'cbinom' '('
expression0 ','
expression0 ','
expression0 ')'
  ;

rbinom0 :
  'rbinom' '('
expression0 ','
expression0 ')'
  ;

exp0 :
  dexp0
  | cexp0
  | rexp0
  ;

dexp0 :
  'dexp' '(' expression0
',' expression0 ')'
  ;

cexp0 :
  'cexp' '(' expression0
',' expression0 ')'
  ;

rexp0 :
  'rexp' '(' expression0
')'
  ;

gamma0 :
  dgamma0
  | cgamma0
  | rgamma0
  ;

dgamma0 :
  'dgamma' '('
expression0 ','
expression0 ','
expression0 ')'
  ;

cgamma0 :
  'cgamma' '('
expression0 ','
expression0 ','
expression0 ')'
  ;

rgamma0 :
  'rgamma' '('
expression0 ','
expression0 ')'
  ;

geom0 :
  dgeom0
  | cgeom0
  | rgeom0
  ;

dgeom0 :
  'dgeom' '(' expression0
',' expression0 ')'
  ;

cgeom0 :
  'cgeom' '(' expression0
',' expression0 ')'
  ;

rgeom0:
  'rgeom' '(' expression0
')'
  ;

norm0 :
  dnorm0
  | cnorm0
  | rnorm0
  ;

dnorm0 :
  'dnorm' '(' expression0
',' expression0 ','
expression0 ')'
  ;

cnorm0 :
  'cnorm' '(' expression0
',' expression0 ','
expression0 ')'
  ;

rnorm0 :
  'rnorm' '(' expression0
',' expression0 ')'
  ;

pois0 :
  dpois0
  | cpois0
  | rpois0
  ;

dpois0 :
  'dpois' '(' expression0
',' expression0  ')'
  ;

cpois0 :
  'cpois' '(' expression0
',' expression0  ')'
  ;

rpois0 :
  'rpois' '(' expression0
')'
  ;

unif0 :
  dunif0
  | cunif0
  | runif0
  ;

dunif0 :
  'dunif' '(' expression0
',' expression0 ','
expression0 ')'
  ;

cunif0 :
  'cunif' '(' expression0
',' expression0 ','
expression0 ')'
  ;

runif0 :
  'runif' '(' expression0
',' expression0 ')'
  ;
```

d. Semantic Analysis and Intermediate Code Generation

i. Data types

The basic data types supported by this programming language are:

- int: Signed integer with no memory limitation. Will work as BigIntegers in Java or like integers in Python.
- float: Signed double-precision floating-point number.
- bool: True or false values
- string: Chain of 8 bit ASCII characters.

The other more complex data types supported will be:

- matrix: 2-dimensional arrays of contiguous memory. The matrix is a collection of the same basic data type. This data type will also be used for the traditional array, being either a column matrix (n*1 dimensions) or row matrix (1*n dimensions).

ii. Allowed operations

The following arithmetic operation will be supported (order of operands does not matter):

Left operand	operator	Right operand	Result
bool	and, or, ==, !=	bool	bool
	not	bool	bool
int	>, <, >=, <=, ==, !=	int	bool
float	>, <, >=, <=, ==, !=	int	bool
float	>, <, >=, <=, ==, !=	float	bool
int	+, -, *, %, ^	int	int
int	/	int	float
int	+, -, *, /, ^	float	float
float	+, -, *, /, ^	float	float
matrix	+, -, *	matrix	matrix (The dimensions must be compatible and of type int or float)
int	+, -, *	matrix	matrix
float	+, -, *	matrix	matrix
matrix	^	int	matrix (The order does matter here!)

iii. Special Functions specification

The language will include the common matrix operations overloaded so it will be as easy to add two matrices as it is to add two integers. It will also have the capacity to plot points and join them given

two matrices representing the x values and y values as well as make histograms of the values of a matrix.

We will include a small set of statistical functions as part of the standard library such as: (These functions will operate on numerical matrices)

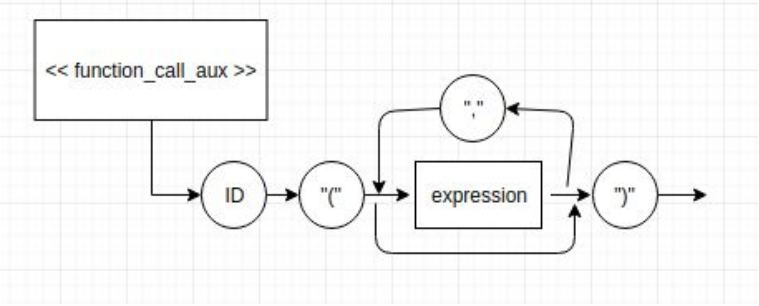
- mean(): Arithmetic mean of the data
- median(): Middle value of the sorted data
- mode(): Most common value of the data
- stdev(): Standard deviation of data
- variance(): Variance of data

The language will also include some probability distributions:

- dbeta(): Beta distribution
- dbinom(): Binomial distribution (Bernoulli included)
- dexp(): Exponential distribution
- dgamma(): Gamma distribution
- dgeom(): Geometric distribution
- dnorm(): Normal distribution
- dpois(): Poisson distribution
- dunif(): Uniform distribution

For every distribution function there are two more versions where the first letter changes, for example cbeta() is the cumulative beta function and rbeta() is the random variable beta function.

The basic syntax for all of the special functions follow the function_call_aux form:



Where the first ID is the name of the special function and has a determined number of parameters depending on the definition of the function. All the functions return a numeric value.

Functions definition:
The following functions on matrices of type int or float. The mode function can operate on strings and booleans as well. For simplicity we will just define the syntax for the matrix type and the variable T is any of the types supported.

- Mean

Function:
```
func float mean(matrix<T> m)
```

- Median

Function:
```
func float median(matrix<T> m)
```

- Mode

Function:
```
func T mode(matrix<T> m)
```

- Standard Deviation

Function:
```
func float stdev(matrix<T> m)
```

- Variance

Function:

```
func float variance(matrix<T> m)
```

- Beta distribution

Γ(a+b)/(Γ(a)Γ(b))x^(a-1)(1-x)^(b-1)

Functions:

```
func float dbeta(float x, float a, float b)
func float cbeta(float x, float a, float b)
func float rbeta(float a, float b)
```

- Binomial distribution (Bernoulli included)

choose(n, x) p^x (1-p)^(n-x)

Functions:

```
func float dbinom(int x, int n, float p)
func float cbinom(int x, int n, float p)
func int rbinom(int n, float p)
```

- Exponential distribution

λ {e}^{- λ x}

Functions:

```
func float dexp(float x, float lambda)
func float cexp(float x, float lambda)
func float rexp(float lambda)
```

- Gamma distribution

1/(s^a Gamma(a)) x^(a-1) e^-(x/s)

Functions:

```
func float dgamma(float x, float a, float s)
func float cgamma(float x, float a, float s)
func float rgamma(float a, float s)
```

- Geometric distribution

p (1-p)^x

Functions:

```
func float dgeom(int x, float p)
func float cgeom(int x, float p)
func int rgeom(float p)
```

- Normal distribution

1/(√(2 π) σ) e^-((x - μ)^2/(2 σ^2))

Functions:

```
func float dnorm(float x, float mean, float sd)
func float cnorm(float x, float mean, float sd)
func float rnorm(float mean, float sd)
```

- Poisson distribution

λ^x exp(-λ)/x!

Functions:

```
func float dpois(int x, float lambda)
```

```
func float dpois(int x, float lambda)
func int rpois(float lambda)
```

- Uniform distribution

1/(max-min)

Functions:

```
func float dunif(float x, float min, float max)
func float cunif(float x, float min, float max)
func float runif(float min, float max)
```

iv. Intermediate Code Generation

Something special about out language is that ANTLR generated code with functions that represent neuralgic points, those are two functions for every rule, one when entering the rule and one when exiting. This produces cleaner code without loss in functionality.

Rule	Enter point	Exit point
R (Root)	-	Adds the exit quadruple, adds the global quadruples to the ObjGenerator and generates both obj and memory files
Program0	-	-
Program1	-	-
Statement0	-	-
Block0	Generates quadruples for if, elseif and while. Also handles their jumps (gotof)	-
Block1	-	-
Function_decl0	Generates objects to handle a function such as its name and creates its Quadruples.	Adds the function Quadruples to the obj generator and changes the scope back to global.
Function_type0	-	-
Decl_block0	-	-
Decl_block1	-	-
Parameters_or_empty	-	Tries to add the function to the symbols table object and creates the quadruples to handle its parameters.
Parameters0	-	-
Parameters1	-	-
Variable_decl0	Tries to add a variable to the symbols table handling its type.	-
Variable_decl1	Same as Variable_decl0 but the type has already been processed.	-

Assigment0	Pushes the operator = to the operator's stack	Gets the left and right operands from the stack, as well as their types and the last operator. It verifies the semantic and generates the assignment quadruple.
Function_call0	-	-
Function_call_aux0	Pushes the operator (, verifies that the function exists, adds the function to the call stack and pushes its parameters to the parameter stack.	Adds the quadruple era and gosub. Also checks if there is a return value.
Function_call_aux1	-	Verifies the correct number of parameters and handles them to become available in that scope.
Function_call_aux2	-	Same as Function_call_1 but after a comma.
If0	Adds "if" to block_reason.	Pops from block_reason.
Else0	Checks if it is an else or elseif and adds its jump quadruples.	Pops from the jumps stack to add the current pointer value to the original jump quadruple.
While0	Sets up the jump quadruple	Completes the jump quadruples
Io_statement0	-	-
Input0	-	Checks where we are trying to input data and checks that it is not a structured type. Also adds its quadruple.
Output0	-	Adds the write quadruple.
Plot0	-	Verifies the parameters we are trying to plot and generates the plot quadruple.
Hist0	-	Verifies the parameter we are trying to plot to be a matrix and generates the hist quadruple.
Showplot0	-	Generates the showplot quadruple.
Return0	-	Pops an operand and type, as well as the return type. Verifies that both types match and generates the return quadruple as well as handling the value assignment.
Expression0	In case of being a string literal, checks its existence in the symbols table and adds it. It pushes its memory address and type.	-
Bool_exp0	-	If the top operator is not, it verifies the

		semantic of what we are trying to negate and generates its quadruple.
Bool_exp1	Checks if there is an ‘and’ or an ‘or’ to apply next and process its quadruple and pushes the next operator	Same as Bool_exp1 enter but without pushing the next operator
Bool_term0	-	-
Bool_term1	The same as Bool_exp1 enter but with relational operators	The same as Bool_exp1 exit but with relational operators
Bool_not0	Pushes the operator not.	-
Bool_op0	-	-
Rel_op0	-	-
Arithmetic_exp0	-	-
Arithmetic_exp1	The same as Bool_exp1 enter but with addition and subtraction	The same as Bool_exp1 exit but with addition and subtraction
Addition_subtraction0	-	-
Arithmetic_factor0	-	-
Arithmetic_factor1	The same as Bool_exp1 enter but with multiplication and division.	The same as Bool_exp1 exit but with multiplication and division.
Multiplication_division0	-	-
Arithmetic_term0	-	-
Arithmetic_term1	The same as Bool_exp1 enter but with exponentiation.	The same as Bool_exp1 exit but with exponentiation.
Exponent0	-	-
Numeric_term0	If it is a constant, it adds it to the symbols table, creates its address and pushes the operand and type. If it is entering from bool_exp0, pushes the operator (.	If it is coming from a bool_exp0, pops and operator.
Numeric_term1	-	If the top operator is the minus unary, it checks the semantics with its operand and generates a quadruple for it.
Sign0	-	If it is a minus sign, generates the minus unary operator and pushes it.
Type0	-	-

Basic_type0	-	-
Value0	Checks the variable in the simbols table and pushes it and its type.	-
Value1	Pushes the operator (.	Solves expression on matrix indices and verifies them.
Matrix_literal0	Sets the matrix_literal array to empty.	Produces the quads for solving a matrix literal.
Matrix_literal1	If coming from matrix_literal2, adds a new empty array to matrix_literal object.	-
Matrix_literal2	-	To the last matrix_literal array it appends the last operand and type.
Matrix_literal3	If coming from matrix_literal2 it appends a new array to matrix_literal.	-
Matrix_literal4	-	Appends to the last array in matrix_literal the last operand and its type.
Stat_functions0	pushes the operator (.	Pops an operator.
Mean0	-	Pops the last operand and type, checks that the type is a matrix and generates its quadruple.
Median0	-	Pops the last operand and type, checks that the type is a matrix and generates its quadruple.
Mode0	-	Pops the last operand and type, checks that the type is a matrix and generates its quadruple.
Stdev0	-	Pops the last operand and type, checks that the type is a matrix and generates its quadruple.
Variance0	-	Pops the last operand and type, checks that the type is a matrix and generates its quadruple.
Transpose0	-	Pops the last operand and type, checks that the type is a matrix and generates its quadruple.
Beta0	-	-
Dbeta0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.

Cbeta0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Rbeta0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Binom0	-	-
Dbinom0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Cbinom0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Rbinom0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Exp0	-	-
Dexp0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Cexp0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Rexp0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Gamma0	-	-
Dgamma0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Cgamma0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Rgamma0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Geom0	-	-
Dgeom0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Cgeom0	-	Generates the quadruples for pushing params, creates a new temporal and

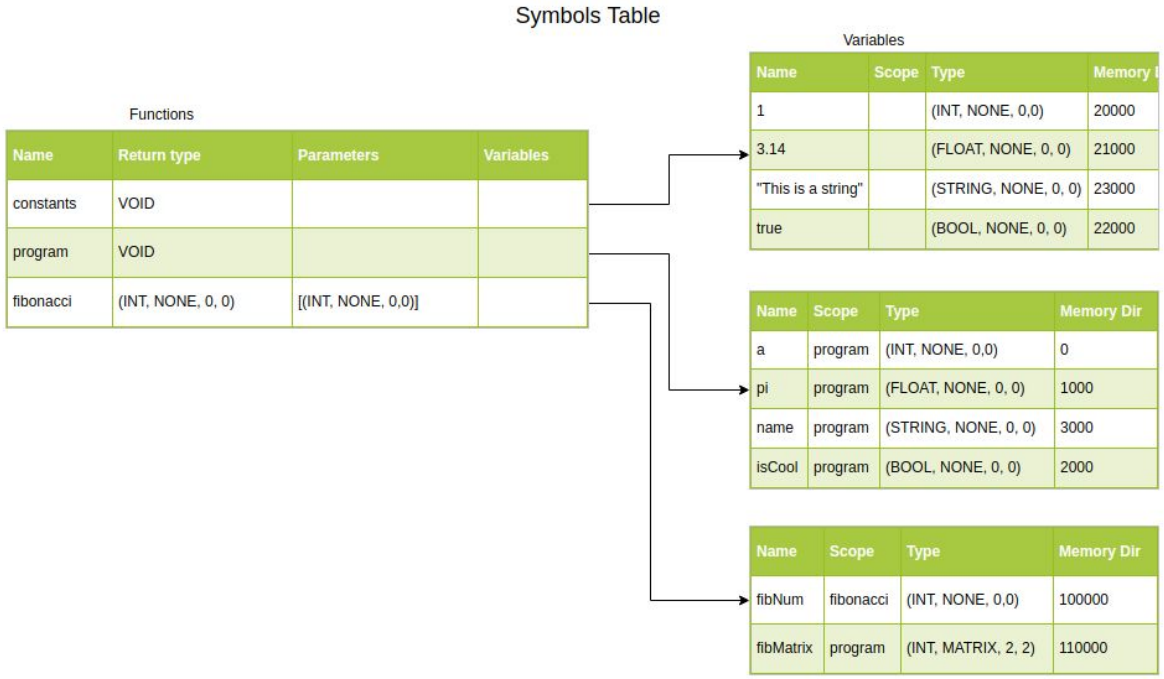
		generates the stat_func quadruple.
Rgeom0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Norm0	-	-
Dnorm0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Cnorm0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Rnorm0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Pois0	-	-
Dpois0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
CPois0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Rpois0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Unif0	-	-
Dunif0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Cunif0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.
Runif0	-	Generates the quadruples for pushing params, creates a new temporal and generates the stat_func quadruple.

e. Memory Administration during Compilation

During the compilation process, the two main memory data structures used where SymbolsTable and Quadruples. The tree listener also has some auxiliary stacks to manage the parameters being received, and the Quadruples objects but they are simply stacks and they will not be described here.

The SymbolsTable class is the main data structure for variable and function declarations. This utilizes a Function class which can store a list of variables, parameters, return type, and its name. For

each variable there are four things that are stored, which are the name, the scope, its type (which includes its basic type and structured type as well as dimensions), and its memory direction. The SymbolsTable is in charge of adding variables to each function, assigning them an address, and making sure that there is no repeated variable in the same scope. SymbolsTable also has some utility functions to get the type of a variable given the name and scope of it, converting names to memory addresses and vice versa as well as working with functions, getting its parameter types and return type. The SymbolsTable contains two special Function objects to represent constants and global variables.



The Quadruples class is the main data structure for intermediate code generation and temporal memory management. It contains four stacks which contain operands, operators, jumps, and types, and also contains a Function object such as the ones used in the SymbolsTable for ease of generating the temporal memory representation for use in the virtual machine. For each function in Pp code there will exist a Quadruples object. This is due to the allowance of declaring functions in any point of the program in the global scope.

When the semantic analysis is done, the compiler then generates the Obj file and a JSON file with the description of the memory. The Obj file is made by the ObjGenerator, which takes all the Quadruples objects generated for the code and combines them, translating the gosub calls function name to the index of the correct quadruple where the code for the function lies, and gotos and gotofs to consider the appending of all quadruples into a single array. Then the memory representation of the program is generated. The MemoryGenerator receives every function, its local variables encoding and its temporal variables encoding, the global variables and the global temporals, and the constants representation and the values of each constant. A Function is represented with a MemoryRepresentation object which contains the range of addresses for each type as well as the dimension of each matrix it has. The MemoryGenerator finally encodes everything into a JSON.

4. Virtual Machine Description

a. Development Environment

Everything was developed and tested on Linux (Ubuntu 16.04+) . The language of choice is Python3 and a couple of libraries used for supporting plot and statistics are numpy and scipy. The lexical and syntax analyzer was built on top of ANTLR.

b. Memory and Execution Processes

The process starts with the object code as an input (basically processed quadruples). Take the follow lines as an example:

```
write   0       0       23000
read    0       0       0
<       0       20001 32000
gotof   32000   0       6
```

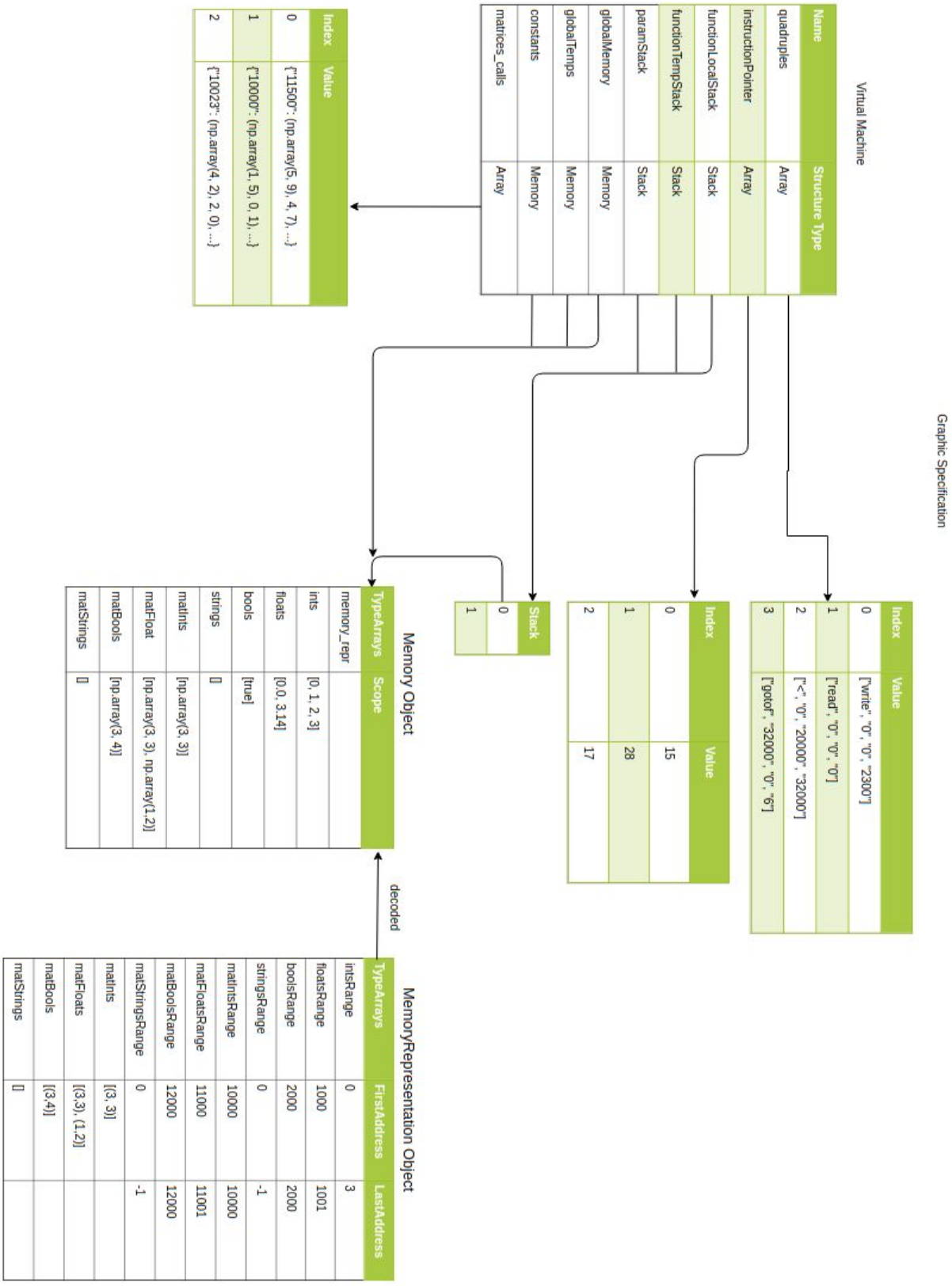
The VirtualMachine object contains a pointer that is used to indicate what to execute next. Once the next quadruple is ready to be executed, four elements of it are available, in order from left to right: the operator, the left operand, right operand and where the result is going to. These four elements may not be present in every quadruple and sometimes it does not represent exactly what explained above. For example, the gotof operator means that the instruction pointer has to be moved to the index value stored in “res”. Also, when any field has the number 0 (in the example above there are many of those), it means that value won’t be used (basically null).

You may notice in the quadruples we have the memory direction of some fields. A memory direction such as 32000 in the last one means that the value we should use from the left operand is stored in the address 32000, and an address in the result field means the result must be stored in that direction. To translate between directions and the actual values. The VirtualMachine object talks to Memory objects generated from the MemoryGenerator object. There are memory objects for many things:

- one stack for the local variables on functions
- one stack for the temporals on functions
- one object for the global variables
- one object for the global temporals
- one object for constants

The Memory object exposes two methods for reading and storing values into memory given a memory direction. A Memory object is created with a MemoryRepresentation object which is created and encoded as a JSON during the compilation process. Before the VirtualMachine starts the execution of quadruples, it first reads the JSON file with the MemoryRepresentation and creates the global variables, temporals, and constants Memory objects. A MemoryRepresentation object contains the range of the addresses of each of the data types (int, float, bool, string, and matrix of each of the previous types) as well as the dimensions of each of the matrices.
The Memory object is in charge of mapping the virtual address into the real address. Whenever a value wants to be stored or retrieved in the Memory, the virtual address is used and the Memory object checks each of the ranges it has to see if any of them contain it and accesses the list which did contain it.

c. Graphic representation



5. Language Functionality Tests

a. Test 1: Fibonacci

```
let int fib;

func int fibonacci (int n) {
  let int f1, f2, i, aux;
  f1 = 0;
  f2 = 1;
  i = 0;
  while (i < n) {
    aux = f2;
    f2 = f1 + f2;
    f1 = aux;
    i = i + 1;
  }
  return f1;
}

write("Which Fibonacci
number do you want?");
read(fib);
if (fib < 0) {
  write("Invalid number");
} else {
  write(fibonacci(fib));
}
```

```
write 0 0 23000
read 0 0 0
< 0 20000 32000
gotof 32000 0 6
write 0 0 23001
goto 0 0 11
push_param 0 0 0
era fibonacci 0 0
gosub 0 0 12
= retVal 0 30000
write 0 0 30000
exit 0 0 0
= pop_param 0 100000
= 20000 0 100001
= 20001 0 100002
= 20000 0 100003
< 100003 100000 132000
gotof 132000 0 25
= 100002 0 100004
+ 100001 100002 130000
= 130000 0 100002
= 100004 0 100001
+ 100003 20001 130001
= 130001 0 100003
goto 0 0 16
return 0 0 100001
end 0 0 0
```

```
Which Fibonacci number do
you want?
7
13
```

b. Test 2: Recursive Fibonacci

```
let int fib;

func int fibonacci (int n) {
  if (n == 0) {
       return 0;
  } elseif (n == 1) {
       return 1;
  }
  return fibonacci(n - 1) +
fibonacci(n - 2);
}

write("Which Fibonacci
number do you want?");
read(fib);
if (fib < 0) {
  write("Invalid number");
} else {
  write(fibonacci(fib));
}
```

```
write 0 0 23000
read 0 0 0
< 0 20000 32000
gotof 32000 0 6
write 0 0 23001
goto 0 0 11
push_param 0 0 0
era fibonacci 0 0
gosub 0 0 12
= retVal 0 30000
write 0 0 30000
exit 0 0 0
= pop_param 0 100000
== 100000 20000 132000
gotof 132000 0 17
return 0 0 20000
goto 0 0 21
== 100000 20001 132001
gotof 132001 0 21
return 0 0 20001
goto 0 0 21
- 100000 20001 130000
push_param 130000 0 0
era fibonacci 0 0
gosub 0 0 12
= retVal 0 130001
- 100000 20002 130002
push_param 130002 0 0
era fibonacci 0 0
gosub 0 0 12
= retVal 0 130003
+ 130001 130003 130004
return 0 0 130004
```

```
Which Fibonacci number do
you want?
7
13
```

	end 0 0 0	

c. Test 3: Matrix Exponentiation Fibonacci

let int fib;
 	
func int fibonacci (int n) {
  let matrix<int>[2][2] fib;
  fib = [[1, 1], [1, 0]];
  if (n == 0) {
  return 0;
  } else {
  fib = fib^n;
  return fib[0][1];
  }
}
 	
write("Which Fibonacci number do you want?");
read(fib);
if (fib < 0) {
  write("Invalid number");
} else {
  write(fibonacci(fib));
}
	write 0 0 23000
	read 0 0 0
	< 0 20001 32000
	gotof 32000 0 6
	write 0 0 23001
	goto 0 0 11
	push_param 0 0 0
	era fibonacci 0 0
	gosub 0 0 12
	= retVal 0 30000
	write 0 0 30000
	exit 0 0 0
	= pop_param 0 100000
	140000 20002 20002 130000
	= 20000 0 130000
	140000 20002 20003 130001
	= 20000 0 130001
	140000 20003 20002 130002
	= 20000 0 130002
	140000 20003 20003 130003
	= 20001 0 130003
	= 140000 0 110000
	== 100000 20001 132000
	gotof 132000 0 26
	return 0 0 20001
	goto 0 0 32
	^ 110000 100000 140001
	= 140001 0 110000
	ver 20001 0 2
	ver 20000 0 2
	110000 20001 20000 130004
	return 0 0 130004
	end 0 0 0
		Which Fibonacci number do you want?
		7
		13

d. Test 4: Factorial

let int fact;
 	
func int factorial (int n) {
  let int i, res;
  i = 1;
  res = 1;
  while (i <= n) {
  res = res*i;
  i = i + 1;
  }
  return res;
}
 	
write("Which number factorial do you want?");
read(fact);
if (fact < 0) {
  write("Invalid number");
} else {
  write(factorial(fact));
}
	write 0 0 23000
	read 0 0 0
	< 0 20001 32000
	gotof 32000 0 6
	write 0 0 23001
	goto 0 0 11
	push_param 0 0 0
	era factorial 0 0
	gosub 0 0 12
	= retVal 0 30000
	write 0 0 30000
	exit 0 0 0
	= pop_param 0 100000
	= 20000 0 100001
	= 20000 0 100002
	<= 100001 100000 132000
	gotof 132000 0 22
	* 100002 100001 130000
	= 130000 0 100002
	+ 100001 20000 130001
	= 130001 0 100001
	goto 0 0 15
	return 0 0 100002
	end 0 0 0
		Which number factorial do you want?
		6
		720

e. Test 5: Recursive Factorial

```
let int fact;

write("Which number
factorial do you want?");
read(fact);

func int rec_factorial (int
n) {
  let int res;
  if (n == 0) {
       write(1);
       return 1;
  }
  res = rec_factorial(n -
1)*n;
  write(res);
  return res;
}

rec_factorial(fact);
```

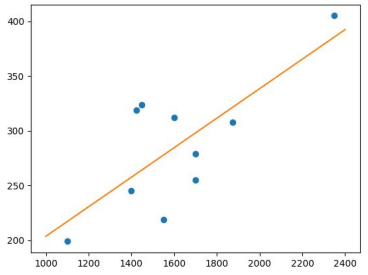
```
write 0 0 23000
read 0 0 0
push_param 0 0 0
era rec_factorial 0 0
gosub 0 0 7
= retVal 0 30000
exit 0 0 0
= pop_param 0 100000
== 100000 20000 132000
gotof 132000 0 13
write 0 0 20001
return 0 0 20001
goto 0 0 13
- 100000 20001 130000
push_param 130000 0 0
era rec_factorial 0 0
gosub 0 0 7
= retVal 0 130001
* 130001 100000 130002
= 130002 0 100001
write 0 0 100001
return 0 0 100001
end 0 0 0
```

```
Which number factorial do
you want?
7
1
1
2
6
24
120
720
5040
```

f. Test 6: Linear Regression

```
let matrix<float>[1][10] X,
Y;
let matrix<float>[2][2] A;
let matrix<float>[2][1] B,
y;
let matrix<float>[1][1] aux;
let int sum, i;

X = [[1400, 1600, 1700,
1875, 1100, 1550, 2350,
1450, 1425, 1700]];
Y = [[245, 312, 279, 308,
199, 219, 405, 324, 319,
255]];

A[0][0] = 10;

sum = 0;
i = 0;
while (i < 10) {
  sum = X[0][i] + sum;
  i = i + 1;
}
A[0][1] = sum;
A[1][0] = sum;

aux = X*transpose(X);
A[1][1] = aux[0][0];

sum = 0;
i = 0;
while (i < 10) {
  sum = Y[0][i] + sum;
  i = i + 1;
}
y[0][0] = sum;
aux = X*transpose(Y);
y[1][0] = aux[0][0];

B = A^-1 * y;
```

```
40000 20009 20009 30000
= 20000 0 30000
40000 20009 20010 30001
= 20001 0 30001
40000 20009 20011 30002
= 20002 0 30002
40000 20009 20012 30003
= 20003 0 30003
40000 20009 20013 30004
= 20004 0 30004
40000 20009 20014 30005
= 20005 0 30005
40000 20009 20015 30006
= 20006 0 30006
40000 20009 20016 30007
= 20007 0 30007
40000 20009 20017 30008
= 20008 0 30008
40000 20009 20018 30009
= 20002 0 30009
= 40000 0 11000
40001 20009 20009 30010
= 20019 0 30010
40001 20009 20010 30011
= 20020 0 30011
40001 20009 20011 30012
= 20021 0 30012
40001 20009 20012 30013
= 20022 0 30013
40001 20009 20013 30014
= 20023 0 30014
40001 20009 20014 30015
= 20024 0 30015
40001 20009 20015 30016
= 20025 0 30016
40001 20009 20016 30017
= 20026 0 30017
40001 20009 20017 30018
= 20027 0 30018
40001 20009 20018 30019
= 20028 0 30019
```



plot(X, Y, "o");
plot([[1000, 2400]],
[[B[0][0] + B[1][0]*1000,
B[0][0] + B[1][0]*2400]],
"-");
showplot();
	= 40001 0 11001
	ver 20029 0 2
	ver 20029 0 2
	11002 20029 20029 31000
	= 20030 0 31000
	= 20029 0 0
	= 20029 0 1
	< 1 20030 32000
	gotof 32000 0 58
	ver 20029 0 1
	ver 1 0 10
	11000 20029 1 31001
	+ 31001 0 31002
	= 31002 0 0
	+ 1 20031 30020
	= 30020 0 1
	goto 0 0 48
	ver 20029 0 2
	ver 20031 0 2
	11002 20029 20031 31003
	= 0 0 31003
	ver 20031 0 2
	ver 20029 0 2
	11002 20031 20029 31004
	= 0 0 31004
	transpose 11000 0 41000
	* 11000 41000 41001
	= 41001 0 11005
	ver 20031 0 2
	ver 20031 0 2
	11002 20031 20031 31005
	ver 20029 0 1
	ver 20029 0 1
	11005 20029 20029 31006
	= 31006 0 31005
	= 20029 0 0
	= 20029 0 1
	< 1 20030 32001
	gotof 32001 0 88
	ver 20029 0 1
	ver 1 0 10
	11001 20029 1 31007
	+ 31007 0 31008
	= 31008 0 0
	+ 1 20031 30021
	= 30021 0 1
	goto 0 0 78
	ver 20029 0 2
	ver 20029 0 1
	11004 20029 20029 31009
	= 0 0 31009
	transpose 11001 0 41002
	* 11000 41002 41003
	= 41003 0 11005
	ver 20031 0 2
	ver 20029 0 1
	11004 20031 20029 31010
	ver 20029 0 1
	ver 20029 0 1
	11005 20029 20029 31011
	= 31011 0 31010
	^ 11002 20032 41004
	* 41004 11004 41005
	= 41005 0 11003
	plot 11000 11001 23000
	40002 20009 20009 30022
	= 20033 0 30022
	40002 20009 20010 30023
	= 20034 0 30023
	ver 20029 0 2	

	ver 20029 0 1 11003 20029 20029 31012 ver 20031 0 2 ver 20029 0 1 11003 20031 20029 31013 * 31013 20033 31014 + 31012 31014 31015 ver 20029 0 2 ver 20029 0 1 11003 20029 20029 31016 ver 20031 0 2 ver 20029 0 1 11003 20031 20029 31017 * 31017 20034 31018 + 31016 31018 31019 41006 20009 20009 31020 = 31015 0 31020 41006 20009 20010 31021 = 31019 0 31021 plot 40002 41006 23001 showplot 0 0 0 exit 0 0 0	

g. Test 7: Merge Sort

let matrix<int>[10][1] arr;
 	
func matrix<int>[10][1] merge(matrix<int>[10][1] left, matrix<int>[10][1] right, int length_left, int lenght_right) {
let matrix<int>[10][1] sorted_list;
let int i, l, r;
i = 0;
l = 0;
r = 0;
while (i < length_left + lenght_right) {
if (l < length_left and r < lenght_right) {
if (left[l][0] < right[r][0]) {
sorted_list[i][0] = left[l][0];
l = l + 1;
} else {
sorted_list[i][0] = right[r][0];
r = r + 1;
}
} elseif (l < length_left) {
sorted_list[i][0] = left[l][0];
l = l + 1;
} elseif (r < lenght_right) {
sorted_list[i][0] = right[r][0];
r = r + 1;
}
	40000 20011 20011 30000
	= 20003 0 30000
	40000 20011 20012 30001
	= 20004 0 30001
	40000 20011 20013 30002
	= 20005 0 30002
	40000 20011 20014 30003
	= 20006 0 30003
	40000 20011 20015 30004
	= 20007 0 30004
	40000 20011 20016 30005
	= 20001 0 30005
	40000 20011 20017 30006
	= 20008 0 30006
	40000 20011 20018 30007
	= 20002 0 30007
	40000 20011 20019 30008
	= 20009 0 30008
	40000 20011 20020 30009
	= 20010 0 30009
	transpose 40000 0 40001
	= 40001 0 10000
	push_param 20003 0 0
	push_param 10000 0 0
	era mergesort 0 0
	gosub 0 0 101
	= retVal 0 40002
	= 40002 0 10000
	write 0 0 10000
	exit 0 0 0
	= pop_param 0 110000
	= pop_param 0 110001
	= pop_param 0 100000
	= pop_param 0 100001
	= 20000 0 100002
	= 20000 0 100003
	= 20000 0 100004
	+ 100000 100001 130000
	< 100002 130000 132000
	gotof 132000 0 99
	< 100003 100000 132001
	< 100004 100001 132002
	and 132001 132002 132003
	gotof 132003 0 72
		[[1]
		[2]
		[3]
		[4]
		[5]
		[6]
		[7]
		[8]
		[9]
		[10]]

```
        i = i + 1;
        }
        return sorted_list;
}

func matrix<int>[10][1]
mergesort(matrix<int>[10][1]
arr, int length) {
        let
matrix<int>[10][1] left,
right;
        let int mid, i, j;

        if (length <= 1) {
        return arr;
        }

        mid = length/2;
        i = 0;
        j = 0;
        while (i < mid) {
        left[j][0] =
arr[i][0];
        j = j + 1;
        i = i + 1;
        }
        j = 0;
        while (i < length) {
        right[j][0] =
arr[i][0];
        j = j + 1;
        i = i + 1;
        }
        return
merge(mergesort(left, mid),
mergesort(right, length -
mid), mid, length - mid);
}

arr = transpose([[10, 4, 5,
7, 8, 1, 3, 2, 6, 9]]);
arr = mergesort(arr, 10);
write(arr);
```

```
ver 100003 0 10
ver 20000 0 1
110000 100003 20000 130001
ver 100004 0 10
ver 20000 0 1
110001 100004 20000 130002
< 130001 130002 132004
gotof 132004 0 62
ver 100002 0 10
ver 20000 0 1
110002 100002 20000 130003
ver 100003 0 10
ver 20000 0 1
110000 100003 20000 130004
= 130004 0 130003
+ 100003 20001 130005
= 130005 0 100003
goto 0 0 71
ver 100002 0 10
ver 20000 0 1
110002 100002 20000 130006
ver 100004 0 10
ver 20000 0 1
110001 100004 20000 130007
= 130007 0 130006
+ 100004 20001 130008
= 130008 0 100004
goto 0 0 96
< 100003 100000 132005
gotof 132005 0 84
ver 100002 0 10
ver 20000 0 1
110002 100002 20000 130009
ver 100003 0 10
ver 20000 0 1
110000 100003 20000 130010
= 130010 0 130009
+ 100003 20001 130011
= 130011 0 100003
goto 0 0 96
< 100004 100001 132006
gotof 132006 0 96
ver 100002 0 10
ver 20000 0 1
110002 100002 20000 130012
ver 100004 0 10
ver 20000 0 1
110001 100004 20000 130013
= 130013 0 130012
+ 100004 20001 130014
= 130014 0 100004
goto 0 0 96
+ 100002 20001 130015
= 130015 0 100002
goto 0 0 37
return 0 0 110002
end 0 0 0
= pop_param 0 110003
= pop_param 0 100005
<= 100005 20001 132000
gotof 132000 0 107
return 0 0 110003
goto 0 0 107
/ 100005 20002 131000
= 131000 0 100006
= 20000 0 100007
= 20000 0 100008
< 100007 100006 132001
gotof 132001 0 125
ver 100008 0 10
```

	ver 20000 0 1 110004 100008 20000 130000 ver 100007 0 10 ver 20000 0 1 110003 100007 20000 130001 = 130001 0 130000 + 100008 20001 130002 = 130002 0 100008 + 100007 20001 130003 = 130003 0 100007 goto 0 0 111 = 20000 0 100008 < 100007 100005 132002 gotof 132002 0 140 ver 100008 0 10 ver 20000 0 1 110005 100008 20000 130004 ver 100007 0 10 ver 20000 0 1 110003 100007 20000 130005 = 130005 0 130004 + 100008 20001 130006 = 130006 0 100008 + 100007 20001 130007 = 130007 0 100007 goto 0 0 126 push_param 100006 0 0 push_param 110004 0 0 era mergesort 0 0 gosub 0 0 101 = retVal 0 140000 - 100005 100006 130008 push_param 130008 0 0 push_param 110005 0 0 era mergesort 0 0 gosub 0 0 101 = retVal 0 140001 - 100005 100006 130009 push_param 130009 0 0 push_param 100006 0 0 push_param 140001 0 0 push_param 140000 0 0 era merge 0 0 gosub 0 0 30 = retVal 0 140002 return 0 0 140002 end 0 0 0	

h. Test 8: Find Matrix

let matrix<int>[4][4] mat;
let int i, j, aux;
 	
func matrix<int>[1][2]
find(matrix<int>[4][4] mat,
int num) {
  let int i, j;
  i = 0;
  while (i < 4) {
  j = 0;
  while (j < 4) {
  if (mat[i][j]
== num) {
  return [[i,
j]];
  }
  j = j + 1;
  }
  i = i + 1;
	write 0 0 23000
	= 20000 0 0
	< 0 20001 32000
	gotof 32000 0 17
	= 20000 0 1
	< 1 20001 32001
	gotof 32001 0 14
	ver 0 0 4
	ver 1 0 4
	10000 0 1 30000
	read 0 0 30000
	+ 1 20004 30001
	= 30001 0 1
	goto 0 0 5
	+ 0 20004 30002
	= 30002 0 0
	goto 0 0 2
	write 0 0 10000
	= 20006 0 0
		Escribe 16 numeros separados
		por enter
		1
		2
		3
		4
		5
		6
		7
		8
		9
		10
		12
		11
		13
		14
		15
		16
		[[1 2 3 4]

```
        }
        return [[-1, -1]];
}

write("Escribe 16 numeros
separados por enter");
i = 0;
while (i < 4) {
        j = 0;
        while (j < 4) {
        read(mat[i][j]);
        j = j + 1;
        }
        i = i + 1;
}

write(mat);
i = 10;
while (i > 0) {
        write("Que numero
deseas buscar?");
        read(aux);
        write(find(mat,
aux));
        i = i - 1;
}
```

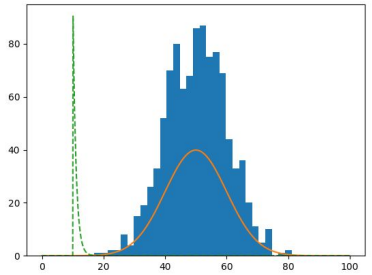
```
> 0 20000 32002
gotof 32002 0 32
write 0 0 23001
read 0 0 2
push_param 2 0 0
push_param 10000 0 0
era find 0 0
gosub 0 0 33
= retVal 0 40000
write 0 0 40000
- 0 20004 30003
= 30003 0 0
goto 0 0 19
exit 0 0 0
= pop_param 0 110000
= pop_param 0 100000
= 20000 0 100001
< 100001 20001 132000
gotof 132000 0 58
= 20000 0 100002
< 100002 20001 132001
gotof 132001 0 55
ver 100001 0 4
ver 100002 0 4
110000 100001 100002 130000
== 130000 100000 132002
gotof 132002 0 52
140000 20002 20002 130001
= 100001 0 130001
140000 20002 20003 130002
= 100002 0 130002
return 0 0 140000
goto 0 0 52
+ 100002 20004 130003
= 130003 0 100002
goto 0 0 39
+ 100001 20004 130004
= 130004 0 100001
goto 0 0 36
140001 20002 20002 130005
= 20005 0 130005
140001 20002 20003 130006
= 20005 0 130006
return 0 0 140001
end 0 0 0
```

```
 [ 5  6  7  8]
 [ 9 10 12 11]
 [13 14 15 16]]
Que numero deseas buscar?
12
[[2 2]]
Que numero deseas buscar?
13
[[3 0]]
Que numero deseas buscar?
...
```

i. Test 9: Histogram

```
let matrix<float>[1000][1]
his, nor, x, ex;
let int n;
let float delta;
n = 999;
delta = 0.0;
while (n >= 0) {
        his[n][0] =
rnorm(50, 10);
        nor[n][0] =
dnorm(delta, 50, 10) * 1000;
        ex[n][0] =
dexp(delta, 10) * 100;
        x[n][0] = delta;
        delta = delta + 0.1;
        n = n - 1;
}
// Comentario
hist(his);
plot(x, nor, "-");
plot(x, ex, "--");
```

```
= 20000 0 0
= 21000 0 1000
>= 0 20001 32000
gotof 32000 0 37
ver 0 0 1000
ver 20001 0 1
11000 0 20001 31000
push_param 20003 0 0
push_param 20002 0 0
stat_func rnorm 2 31001
= 31001 0 31000
ver 0 0 1000
ver 20001 0 1
11001 0 20001 31002
push_param 20003 0 0
push_param 20002 0 0
push_param 1000 0 0
stat_func dnorm 3 31003
* 31003 20004 31004
= 31004 0 31002
ver 0 0 1000
```



showplot();	ver 20001 0 1 11003 0 20001 31005 push_param 20003 0 0 push_param 1000 0 0 stat_func dexp 2 31006 * 31006 20005 31007 = 31007 0 31005 ver 0 0 1000 ver 20001 0 1 11002 0 20001 31008 = 1000 0 31008 + 1000 21001 31009 = 31009 0 1000 - 0 20006 30000 = 30000 0 0 goto 0 0 2 hist 11000 0 0 plot 11002 11001 23000 plot 11002 11003 23001 showplot 0 0 0 exit 0 0 0	

j. Test 10: Stat Functions

let matrix<int>[10][1] arr;
 	
arr = transpose([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]]);
 	
write("Mean:");
write(mean(arr));
 	
write("Median:");
write(median(arr));
 	
write("Mode:");
write(mode(arr));
 	
write("Variance:");
write(variance(arr));
 	
write("Std Dev:");
write(stdev(arr));
	40000 20010 20010 30000
	= 20000 0 30000
	40000 20010 20011 30001
	= 20001 0 30001
	40000 20010 20012 30002
	= 20002 0 30002
	40000 20010 20013 30003
	= 20003 0 30003
	40000 20010 20014 30004
	= 20004 0 30004
	40000 20010 20015 30005
	= 20005 0 30005
	40000 20010 20016 30006
	= 20006 0 30006
	40000 20010 20017 30007
	= 20007 0 30007
	40000 20010 20018 30008
	= 20008 0 30008
	40000 20010 20019 30009
	= 20009 0 30009
	transpose 40000 0 40001
	= 40001 0 10000
	write 0 0 23000
	mean 10000 0 31000
	write 0 0 31000
	write 0 0 23001
	median 10000 0 31001
	write 0 0 31001
	write 0 0 23002
	mode 10000 0 30010
	write 0 0 30010
	write 0 0 23003
	variance 10000 0 31002
	write 0 0 31002
	write 0 0 23004
	stdev 10000 0 31003
	write 0 0 31003
	exit 0 0 0
		Mean:
		5.5
		Median:
		5.5
		Mode:
		1
		Variance:
		8.25
		Std Dev:
		2.8722813232690143

6. Code

a. VirtualMachine.py

This object is the final part of the process. It takes the object code (basically quadruples) and executes them one by one. It has five memory objects: for global variables, global temporals, constants, local variables and local temporals. It has three functions, daro, dar and daw that helps get the memory objects, writing and reading from a memory direction. Also it stores the matrices used in a given program.

```
import numpy as np
from scipy import stats

import matplotlib.pyplot as plt

from Memory import Memory
from MemoryGenerator import MemoryGenerator
from StatFunctions import StatFunctions


class VirtualMachine:
   def __init__(self, filename):
       self.filename = filename
       self.quadruples = []
       self.instructionPointer = [0]
       self.functionLocalStack = []
       self.functionTempStack = []
       self.retVal = None
       self.paramStack = []
       self.parse_quadruples()
       self.memory = MemoryGenerator.decode(self.filename + ".json")
       self.globalMemory = Memory(self.memory["program"]["locals"])
       self.globalTemps = Memory(self.memory["program"]["temps"])
       self.constants = Memory(self.memory["constants"]["repr"])
       for [addr, val] in self.memory["constants"]["vals"]:
           self.constants.set_value(addr, val)
       self.matrices_calls = [{}]

   # Parses the quadruples from a .ppo file and puts them in the quadruples property
   def parse_quadruples(self):
       with open(self.filename + ".ppo", 'r') as f:
           for line in f.readlines():
               self.quadruples.append(line.split())

   # Creates memory when you are about to need it, for example in a function call
   def expand_activation_record(self, func):
       self.functionLocalStack.append(
           Memory(self.memory["functions"][func]["locals"]))
       self.functionTempStack.append(
           Memory(self.memory["functions"][func]["temps"]))
       self.matrices_calls.append({})

   # Given a quadruple, try to execute it and advance the instruction pointer
   def execute(self):
       while True:
           [op, left, right, res] = self.quadruples[self.instructionPointer[-1]]
           if (self.execute_quadruple(op, left, right, res)):
               self.instructionPointer[-1] = self.instructionPointer[-1] + 1
           else:
               raise Exception("Execution error on quad #" +
                               str(self.instructionPointer[-1]))

   # Decode and return the memory object for a memory location
   def daro(self, mem):
       if mem >= 100000:
           if mem >= 120000:
               return self.functionTempStack[-1]
           else:
```

```
            return self.functionLocalStack[-1]
    else:
        if mem >= 30000:
            return self.globalTemps
        elif mem >= 20000:
            return self.constants
        else:
            return self.globalMemory

# Decode and read. Returns the value in a memory direction
def dar(self, mem):
    if mem == "pop_param":
        return self.paramStack.pop()
    elif mem == "retVal":
        return self.retVal
    else:
        mem = int(mem)
        if mem in self.matrices_calls[-1]:
            mat, i, j = self.matrices_calls[-1][mem]
            self.matrices_calls[-1].pop(mem)
            return mat[i][j]
        return self.daro(mem).get_value(mem)

# Decode and write. Writes a value in a memory direction
def daw(self, val, mem):
    mem = int(mem)
    if mem in self.matrices_calls[-1]:
        mat, i, j = self.matrices_calls[-1][mem]
        mat[i][j] = val
        self.matrices_calls[-1].pop(mem)
    else:
        self.daro(mem).set_value(mem, val)

# Receive a quadruple and execute it. Returns true if success.
def execute_quadruple(self, op, left, right, res):
    # print(self.instructionPointer[-1], op, left, right, res)
    if op == "+":
        self.daw(self.dar(left) + self.dar(right), res)
    elif op == "-":
        self.daw(self.dar(left) - self.dar(right), res)
    elif op == "*":
        self.daw(np.dot(self.dar(left), self.dar(right)), res)
    elif op == "/":
        try:
            self.daw(self.dar(left) / self.dar(right), res)
        except Exception:
            print("Division by zero")
            exit()
    elif op == "%":
        self.daw(self.dar(left) % self.dar(right), res)
    elif op == "^":
        if isinstance(self.dar(left), np.ndarray):
            try:
                self.daw(np.linalg.matrix_power(
                    self.dar(left), self.dar(right)), res)
            except Exception:
                print("Matrix does not have inverse")
                exit()
        else:
            self.daw(self.dar(left) ** self.dar(right), res)
    elif op == "=":
        if left == "pop_param":
            self.daw(self.paramStack.pop(), res)
        else:
            self.daw(self.dar(left), res)
    elif op == ">":
        self.daw(self.dar(left) > self.dar(right), res)
    elif op == "<":
        self.daw(self.dar(left) < self.dar(right), res)
    elif op == ">=":
        self.daw(self.dar(left) >= self.dar(right), res)
```

```
elif op == "<=":
    self.daw(self.dar(left) <= self.dar(right), res)
elif op == "==":
    self.daw(self.dar(left) == self.dar(right), res)
elif op == "!=":
    self.daw(self.dar(left) != self.dar(right), res)
elif op == "and":
    self.daw(self.dar(left) and self.dar(right), res)
elif op == "or":
    self.daw(self.dar(left) or self.dar(right), res)
elif op == "not":
    self.daw(not self.dar(left), res)
elif op == "read":
    self.daw(input(), res)
elif op == "write":
    print(self.dar(res))
elif op == "goto":
    self.instructionPointer[-1] = int(res) - 1
elif op == "gotof":
    if not self.dar(left):
        self.instructionPointer[-1] = int(res) - 1
elif op == "push_param":
    self.paramStack.append(self.dar(left))
elif op == "era":
    self.expand_activation_record(left)
elif op == "gosub":
    self.instructionPointer.append(int(res) - 1)
elif op == "return":
    self.retVal = self.dar(res)
    self.instructionPointer.pop()
    self.functionLocalStack.pop()
    self.functionTempStack.pop()
    self.matrices_calls.pop()
elif op == "end":
    self.instructionPointer.pop()
    self.functionLocalStack.pop()
    self.functionTempStack.pop()
    self.matrices_calls.pop()
elif op == "exit":
    exit()
elif op == "ver":
    if self.dar(left) < int(right) or self.dar(left) >= int(res):
        print("Segmentation Fault")
        exit()
elif op == "plot":
    x = self.dar(left)
    y = self.dar(right)
    fmt = self.dar(res)
    plt.plot(x.flatten(), y.flatten(), fmt)
elif op == "hist":
    x = self.dar(left)
    plt.hist(x.flatten(), bins=30)
elif op == "showplot":
    plt.show()
elif op == "transpose":
    mat = self.dar(left)
    self.daw(np.transpose(mat), res)
elif op == "mean":
    mat = self.dar(left)
    self.daw(np.mean(mat), res)
elif op == "median":
    mat = self.dar(left)
    self.daw(np.median(mat), res)
elif op == "mode":
    mat = self.dar(left)
    self.daw(stats.mode(mat, axis=None)[0][0], res)
elif op == "stdev":
    mat = self.dar(left)
    self.daw(np.std(mat), res)
elif op == "variance":
    mat = self.dar(left)
```

```
            self.daw(np.var(mat), res)
        elif op.isdigit():
            mat = self.dar(op)
            index1 = self.dar(left)
            index2 = self.dar(right)
            self.daw(mat[index1][index2], res)
            res = int(res)
            self.matrices_calls[-1][res] = (mat, index1, index2)
        elif op == "-u":
            self.daw(-self.dar(left), res)
        elif op == "stat_func":
            params = []
            for _ in range(int(right)):
                params.append(self.paramStack.pop())
            self.daw(StatFunctions.execute(left, params), res)
        else:
            return False
        return True
```

b. SymbolsTable.py

In order of apparition: BasicTypes is an enumerator for “primitive” types in the language. Structured types is to state if it is a single memory location or if it is a matrix (remember arrays are matrices too). Then the class Type is a wrapper for both mentioned classes. Function and Variable classes store information about function and variables such as their names, return type/type, parameters and local variables. The SymbolsTable class stores the range in memory for every type of variable or constant. This class contains methods to add new elements such as a new declared variable to a certain function, handles validations to check that something exists, formats itself to be stringified and encodes and decodes itself for better portability and communication with other modules.

```
from enum import Enum
from typing import List, Tuple

from MemoryGenerator import MemoryRepresentation


class BasicTypes(Enum):
        BOOL = "bool"
        STRING = "string"
        INT = "int"
        FLOAT = "float"
        VOID = "void"


class StructuredTypes(Enum):
        NONE = "none"
        MATRIX = "matrix"
        # Not yet supported
        # DATASET = "dataset"


class Type:
        def __init__(self, basic_type, struct_type, rows=None, cols=None):
        self.basic_type = basic_type
        self.struct_type = struct_type
        self.rows = rows
        self.cols = cols

        def __str__(self):
        struct_type_value = self.struct_type.value
        if struct_type_value == "none":
                struct_type_value = ""
        dims = "" if struct_type_value == "" else f"[{self.rows}][{self.cols}]"
        return f"{struct_type_value}<{self.basic_type.value}{dims}>"

        def __repr__(self):
        return str(self)


class Function:
```

```
        def __init__(self, name, return_type, parameters):
        self.name = name
        self.return_type = return_type
        self.parameters = []
        self.variables = parameters
        self.memory_size = 0

        def get_dirs(self, basic_type, struct_type):
        return [
                var.memory_dir
                for var in self.variables.values()
                if var.type.basic_type == basic_type and
                var.type.struct_type == struct_type
        ]

        def encode_matrix_data(self, basic_type):
        matrix_dirs = self.get_dirs(basic_type, StructuredTypes.MATRIX)
        matrices = [var for var in self.variables.values()
                        if var.memory_dir in matrix_dirs]
        matrices.sort(key=(lambda x: x.memory_dir))
        return [(matrix.type.rows, matrix.type.cols) for matrix in matrices]

        def encode(self):
        return MemoryRepresentation(
                [min(self.get_dirs(BasicTypes.INT, StructuredTypes.NONE), default=0),
                max(self.get_dirs(BasicTypes.INT, StructuredTypes.NONE), default=-1)
                ],
                [min(self.get_dirs(BasicTypes.FLOAT, StructuredTypes.NONE), default=0),
                max(self.get_dirs(BasicTypes.FLOAT,
                                        StructuredTypes.NONE), default=-1)
                ],
                [min(self.get_dirs(BasicTypes.BOOL, StructuredTypes.NONE), default=0),
                max(self.get_dirs(BasicTypes.BOOL, StructuredTypes.NONE), default=-1)
                ],
                [min(self.get_dirs(BasicTypes.STRING, StructuredTypes.NONE), default=0),
                max(self.get_dirs(BasicTypes.STRING,
                                        StructuredTypes.NONE), default=-1)
                ],
                [min(self.get_dirs(BasicTypes.INT, StructuredTypes.MATRIX), default=0),
                max(self.get_dirs(BasicTypes.INT, StructuredTypes.MATRIX), default=-1)
                ],
                [min(self.get_dirs(BasicTypes.FLOAT, StructuredTypes.MATRIX), default=0),
                max(self.get_dirs(BasicTypes.FLOAT,
                                        StructuredTypes.MATRIX), default=-1)
                ],
                [min(self.get_dirs(BasicTypes.BOOL, StructuredTypes.MATRIX), default=0),
                max(self.get_dirs(BasicTypes.BOOL,
                                        StructuredTypes.MATRIX), default=-1)
                ],
                [min(self.get_dirs(BasicTypes.STRING, StructuredTypes.MATRIX), default=0),
                max(self.get_dirs(BasicTypes.STRING,
                                        StructuredTypes.MATRIX), default=-1)
                ],
                self.encode_matrix_data(BasicTypes.INT),
                self.encode_matrix_data(BasicTypes.FLOAT),
                self.encode_matrix_data(BasicTypes.BOOL),
                self.encode_matrix_data(BasicTypes.STRING),
        )


class Variable:
        def __init__(self, name, data_type, scope, memory_dir=0):
        self.name = name
        self.type = data_type
        self.scope = scope
        self.memory_dir = memory_dir


class SymbolsTable:
        def __init__(self):
        self.globVarInt = 0
```

```
self.globVarFloat = 1000
self.globVarBool = 2000
self.globVarString = 3000
self.globMatInt = 10000
self.globMatFloat = 11000
self.globMatBool = 12000
self.globMatString = 13000
self.constInt = 20000
self.constFloat = 21000
self.constBool = 22000
self.constString = 23000

# Just for reference, used in quadruples
self.globTempInt = 30000
self.globTempFloat = 31000
self.globTempBool = 32000
self.globTempString = 33000
self.globTempMatInt = 40000
self.globTempMatFloat = 41000
self.globTempMatBool = 42000
self.globTempMatString = 43000

self.locVarInt = 100000
self.locVarFloat = 101000
self.locVarBool = 102000
self.locVarString = 103000
self.locMatInt = 110000
self.locMatFloat = 111000
self.locMatBool = 112000
self.locMatString = 113000

# Just for reference, used in quadruples
self.locTempInt = 130000
self.locTempFloat = 131000
self.locTempBool = 132000
self.locTempString = 133000
self.locTempMatInt = 140000
self.locTempMatFloat = 141000
self.locTempMatBool = 142000
self.locTempMatString = 143000

self.constants = Function("constants", BasicTypes.VOID, {})
self.dir_to_memory_dict = {}
self.functions = {}
self.add_function("program", BasicTypes.VOID, {})

def add_function(self, name, return_type, parameters):
if name in self.functions:
       raise Exception("The function already exists.")
else:
       self.functions[name] = Function(
       name, return_type, {})
       self.functions[name].parameters = parameters
       for parameter in parameters:
       self.add_variable(
              parameter.name, parameter.type, name)

# Given a function name checks its existence
def exists_function(self, name, parameters):
if name in self.functions:
       return True
return False

# Adds a variable to a scope and assigns memory to it
def add_variable(self, name, data_type, scope):
if scope not in self.functions:
       raise Exception("The scope does not exists.")
if name in self.functions[scope].variables:
       raise Exception("The variable already exists in this scope.")
else:
       if scope == "program":
```

```
                if data_type.struct_type == StructuredTypes.NONE:
                        if data_type.basic_type == BasicTypes.INT:
                        ptr = self.globVarInt
                        self.globVarInt += 1
                        elif data_type.basic_type == BasicTypes.FLOAT:
                        ptr = self.globVarFloat
                        self.globVarFloat += 1
                        elif data_type.basic_type == BasicTypes.BOOL:
                        ptr = self.globVarBool
                        self.globVarBool += 1
                        elif data_type.basic_type == BasicTypes.STRING:
                        ptr = self.globVarString
                        self.globVarString += 1
                elif data_type.struct_type == StructuredTypes.MATRIX:
                        if data_type.basic_type == BasicTypes.INT:
                        ptr = self.globMatInt
                        self.globMatInt += 1
                        elif data_type.basic_type == BasicTypes.FLOAT:
                        ptr = self.globMatFloat
                        self.globMatFloat += 1
                        elif data_type.basic_type == BasicTypes.BOOL:
                        ptr = self.globMatBool
                        self.globMatBool += 1
                        elif data_type.basic_type == BasicTypes.STRING:
                        ptr = self.globMatString
                        self.globMatString += 1
                else:
                if data_type.struct_type == StructuredTypes.NONE:
                        if data_type.basic_type == BasicTypes.INT:
                        ptr = self.locVarInt
                        self.locVarInt += 1
                        elif data_type.basic_type == BasicTypes.FLOAT:
                        ptr = self.locVarFloat
                        self.locVarFloat += 1
                        elif data_type.basic_type == BasicTypes.BOOL:
                        ptr = self.locVarBool
                        self.locVarBool += 1
                        elif data_type.basic_type == BasicTypes.STRING:
                        ptr = self.locVarString
                        self.locVarString += 1
                elif data_type.struct_type == StructuredTypes.MATRIX:
                        if data_type.basic_type == BasicTypes.INT:
                        ptr = self.locMatInt
                        self.locMatInt += 1
                        elif data_type.basic_type == BasicTypes.FLOAT:
                        ptr = self.locMatFloat
                        self.locMatFloat += 1
                        elif data_type.basic_type == BasicTypes.BOOL:
                        ptr = self.locMatBool
                        self.locMatBool += 1
                        elif data_type.basic_type == BasicTypes.STRING:
                        ptr = self.locMatString
                        self.locMatString += 1
                var = Variable(name, data_type, scope, ptr)
                self.functions[scope].variables[name] = var
                self.dir_to_memory_dict[ptr] = name

        # Checks existence of a variable in a scope
        def exists_variable(self, name, scope):
        if name in self.functions[scope].variables or name in
self.functions["program"].variables:
                return True
        return False

        # Adds a constant, remember they are global
        def add_constant(self, value, data_type):
        if value in self.constants.variables:
                raise Exception("The contant already exists in this scope.")
        else:
                if data_type.basic_type == BasicTypes.INT:
                ptr = self.constInt
```

```
                self.constInt += 1
                elif data_type.basic_type == BasicTypes.FLOAT:
                ptr = self.constFloat
                self.constFloat += 1
                elif data_type.basic_type == BasicTypes.BOOL:
                ptr = self.constBool
                self.constBool += 1
                elif data_type.basic_type == BasicTypes.STRING:
                ptr = self.constString
                self.constString += 1

                var = Variable(value, data_type, "program", ptr)
                self.constants.variables[value] = var
                self.dir_to_memory_dict[ptr] = value

        # Checks if a constant already has been processed
        def exists_constant(self, value):
        if value in self.constants.variables:
                return True
        else:
                return False

        # Given a constant name, returns its memory direction
        def constant_to_dir(self, value):
        if self.exists_constant(value):
                return self.constants.variables[value].memory_dir

        # Given a memory direction, returns its value
        def dir_to_name(self, memory_dir):
        return self.dir_to_memory_dict[memory_dir]

        # Given a variable name and a scope, returns its memory direction
        def name_to_dir(self, name, scope):
        if self.exists_variable(name, scope):
                if name in self.functions[scope].variables:
                return self.functions[scope].variables[name].memory_dir
                else:
                return self.functions["program"].variables[name].memory_dir
        else:
                raise Exception("Variable does not exist in this scope.")

        # Returns the type of a memory direction. You should provide the scope to
facilitate things
        def get_type(self, memory_dir, scope):
        name = self.dir_to_name(memory_dir)
        if self.exists_variable(name, scope):
                if name in self.functions[scope].variables:
                return self.functions[scope].variables[name].type
                else:
                return self.functions["program"].variables[name].type
        else:
                raise Exception("Variable does not exist in this scope.")

        def get_return_type(self, name):
        return self.functions[name].return_type

        def get_function_param_type(self, name, index):
        return self.functions[name].parameters[index].type

        def get_function_num_params(self, name):
        return len(self.functions[name].parameters)

        def set_function_memory(self, name, temp_memory):
        self.functions[name].memory_size = temp_memory + \
                len(self.functions[name].variables)

        def get_function_memory(self, name):
        return self.functions[name].memory_size

        def __str__(self):
        ret = "===============SYMBOLS=TABLE==============\n"
```

```
        for i in self.functions:
                func = self.functions[i]
                ret += "-----> " + func.name + " : " + \
                str(func.return_type) + "\n"
                for j in self.functions[func.name].variables:
                variable = self.functions[func.name].variables[j]
                ret += "NAME: " + variable.name + "\n"
                ret += "TYPE: " + str(variable.type) + "\n\n"

                ret += "\n"

        ret += "=========================================\n"
        return ret
```

c. Quadruples.py

Suppose you have a single scope, a Quadruples object is what you use to build your quadruples for that context. You store the stacks for operators, operands, types and jumps. Also it stores the initial memory directions for each type. It includes a function to create new temporal registers, manage the mentioned stacks and print everything inside a Quadruples object.

```
from SymbolsTable import BasicTypes, StructuredTypes, Variable, Function

# To store the four elements of a quadruple
class Quadruple:
        def __init__(self, op, left, right, res):
        self.op = op
        self.left = left
        self.right = right
        self.res = res


class Quadruples:
        def __init__(self, isFunc = True):
        self.poper = []
        self.pilao = []
        self.psaltos = []
        self.ptipos = []
        self.quadruples = []
        self.temp_register_ptr = 0

        self.vars = Function("temps", None, {})

        padding = 0 if not isFunc else 100000

        self.tempInt = 30000 + padding
        self.tempFloat = 31000 + padding
        self.tempBool = 32000 + padding
        self.tempString = 33000 + padding
        self.matTempInt = 40000 + padding
        self.matTempFloat = 41000 + padding
        self.matTempBool = 42000 + padding
        self.matTempString = 43000 + padding

        # Creates a new temporal register and returns its memory direction
        def new_temp_register(self, register_type):
        ret = None
        if register_type.struct_type == StructuredTypes.MATRIX:
                if register_type.basic_type == BasicTypes.INT:
                ret = self.matTempInt
                self.matTempInt += 1
                elif register_type.basic_type == BasicTypes.FLOAT:
                ret = self.matTempFloat
                self.matTempFloat += 1
                elif register_type.basic_type == BasicTypes.BOOL:
                ret = self.matTempBool
                self.matTempBool += 1
                elif register_type.basic_type == BasicTypes.STRING:
                ret = self.matTempString
                self.matTempString += 1
```

```
elif register_type.struct_type == StructuredTypes.NONE:
        if register_type.basic_type == BasicTypes.INT:
        ret = self.tempInt
        self.tempInt += 1
        elif register_type.basic_type == BasicTypes.FLOAT:
        ret = self.tempFloat
        self.tempFloat += 1
        elif register_type.basic_type == BasicTypes.BOOL:
        ret = self.tempBool
        self.tempBool += 1
        elif register_type.basic_type == BasicTypes.STRING:
        ret = self.tempString
        self.tempString += 1
self.vars.variables[ret] = Variable("temp", register_type, "temp", ret)
return ret

def get_quad_count(self):
return len(self.quadruples)

def add_quadruple(self, op, left, right, res):
self.quadruples.append(Quadruple(op, left, right, res))

def has_operator(self):
return len(self.poper) > 0

def push_operator(self, o):
self.poper.append(o)

def pop_operator(self):
if not self.has_operator():
        raise Exception("There are no elements left.")
else:
        return self.poper.pop(-1)

def top_operator(self):
if not self.has_operator():
        raise Exception("There are no elements left.")
else:
        return self.poper[-1]

def has_operand(self):
return len(self.pilao) > 0

def push_operand(self, o):
self.pilao.append(o)

def pop_operand(self):
if not self.has_operand():
        raise Exception("There are no elements left.")
else:
        return self.pilao.pop(-1)

def top_operand(self):
if not self.has_operand():
        raise Exception("There are no elements left.")
else:
        return self.pilao[-1]

def has_jump(self):
return len(self.psaltos) > 0

def push_jump(self, j):
self.psaltos.append(j)

def pop_jump(self):
if not self.has_jump():
        raise Exception("There are no elements left.")
else:
        return self.psaltos.pop(-1)

def top_jump(self):
```

```
        if not self.has_jump():
                raise Exception("There are no elements left.")
        else:
                return self.psaltos[-1]

        def has_type(self):
        return len(self.ptipos) > 0

        def push_type(self, t):
        self.ptipos.append(t)

        def pop_type(self):
        if not self.has_type():
                raise Exception("There are no elements left.")
        else:
                return self.ptipos.pop(-1)

        def top_type(self):
        if not self.has_type():
                raise Exception("There are no elements left.")
        else:
                return self.ptipos[-1]

        def encode_temp_memory(self):
        return self.vars.encode()

        def __str__(self):
        ret = "==============================POPER======================================\n"
        ret += '[%s]' % ', '.join(map(str, self.poper)) + "\n\n"

        ret +=
"==============================PILAO======================================\n"
        ret += '[%s]' % ', '.join(map(str, self.pilao)) + "\n\n"

        ret +=
"==============================PSALTOS====================================\n"
        ret += '[%s]' % ', '.join(map(str, self.psaltos)) + "\n\n"

        ret += "==============================4QUADRUPLES4==============================\n"
        ret += "#\t\tOp\t\tLeft\t\tRight\t\tRes\n"
        for i in range(len(self.quadruples)):
                ret += str(i) + "\t\t" + str(self.quadruples[i].op) + "\t\t" +
str(self.quadruples[i].left) + \
                "\t\t" + str(self.quadruples[i].right) + \
                "\t\t" + str(self.quadruples[i].res) + "\n"

        ret += "\n"

        ret +=
"=========================================================================\n"
        return ret
```

d. ObjGenerator.py

Given the Quadruples object from everything, it builds the final object code (basically quadruples) in the proper execution order.

```
from Quadruples import Quadruple
from MemoryGenerator import MemoryGenerator, MemoryRepresentation
from Memory import Memory

class ObjGenerator:
        def __init__(self):
        self.functions = {}
        self.program = None
        self.instruction_ptr = 0

        # Adds the quadruples for a new function (new scope)
        def add_function_quadruples(self, name, quadruples):
        self.functions[name] = quadruples
```

```
        # The main quadruples
        def add_global_quadruples(self, quadruples):
        self.program = quadruples

        # Unifies all the Quadruples
        def joinQuads(self, programQuads):
        retQuads = []
        origin = []
        for quad in programQuads:
                retQuads.append(quad)
                origin.append("program")
        funStarts = {"program": 0}

        for key in self.functions:
                funStarts[key] = len(retQuads)
                for quad in self.functions[key].quadruples:
                retQuads.append(quad)
                origin.append(key)

        for i in range(len(retQuads)):
                retQuads[i]
                if retQuads[i].op == "gosub":
                retQuads[i].res = funStarts[retQuads[i].res]

                elif retQuads[i].op == "goto" or retQuads[i].op == "gotof":
                retQuads[i].res += funStarts[origin[i]]

        return retQuads

        # Writes a .ppo file using the unified quadruples
        def gen_obj_file(self, filename):
        quads = self.joinQuads(self.program.quadruples)
        with open(filename + ".ppo", 'w') as obj:
                for quad in quads:
                if quad.left is None:
                        quad.left = 0
                if quad.right is None:
                        quad.right = 0
                if quad.res is None:
                        quad.res = 0

                obj.write(f'{quad.op} {quad.left} {quad.right} {quad.res}\n')

        # Writes the memory file for the prepared quadruples
        def gen_mem_file(self, symbols_table, filename):
        mem = MemoryGenerator()
        for func_name, func in symbols_table.functions.items():
                if func_name == "program":
                continue
                quads = self.functions[func_name]
                mem.add_function(func_name, func.encode(), quads.encode_temp_memory())

        mem.add_globals(symbols_table.functions["program"].encode(),
self.program.encode_temp_memory())
        mem.add_constants(
                symbols_table.constants.encode(),
                [
                [var.memory_dir, var.name]
                for var in sorted(symbols_table.constants.variables.values(), key=(lambda v:
v.memory_dir))
                ]
        )
        mem.encode(filename + ".json")
```