Instituto Tecnológico y de Estudios

Superiores de Monterrey

Proyecto Final:

Diseño de Compiladores

PyCoffee

Esteban Torres

A01193925

Bruno Méndez

A01194018

Project Description And User Manual

Project Description

Purpose

The purpose of the project is to integrate all the knowledge acquired during our Bachelor's Degree in Computer Science. It includes data structures, algorithms, computer theory, programming languages by building a compiler capable of receiving a set of instructions and delivering results in a web framework. We are building the compiler using the concepts we learned throughout the semester: Lexical Analysis, Syntax Analysis, Semantic Analysis, the translation process and the generation of the intermediate code. The execution of it is made in our design of a virtual machine and the result is returned via Flask and the front-end is done in React.

Objective

The objective of this project is to design and create a compiler that is capable of executing in any device that has a browser receiving a code similar to C or C++. The results of the compiler will be filled in a div called output and it supports user inputs. This way it keeps the code and the results in a well structured manner.

Scope

The language contains all the basic elements of a programming language:

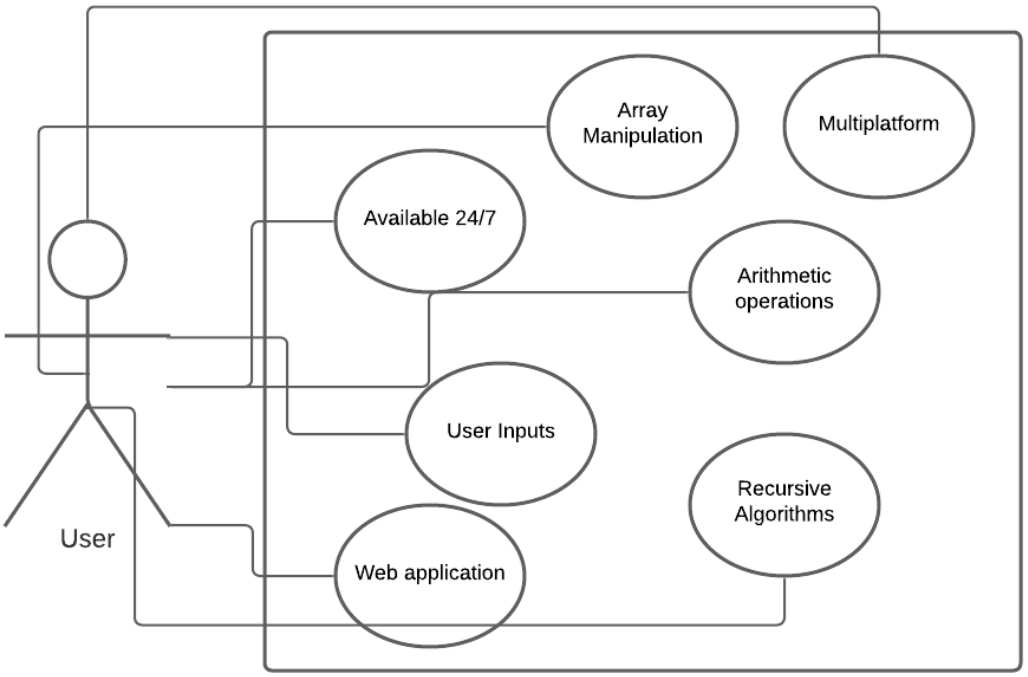
- Variable Declaration
- Function Declaration
- Assignment Expressions
- Void Function Calls
- Value-Returning Function
- Decision Statements (If/else)
- Cycle Statements (While & For)

- Mathematical and Boolean Expressions
- Arrays
- Reading of Inputs
- Printing of Outputs

Requirements

Non-Functional	Functional
Programs are read from the front-end and sent as a JSON file	The code received is initialized with the ‘program’ token
Syntax is similar to C or C++	Function declaration
Project is accesible in any web browser	Function Calling
The backend (compiler) is served in Heroku and the frontend is served in Netlify	Arrays
	Can read values from the frontend
	Can print values to the frontend
	Error handling and displayed in the frontend

General Use Cases (diagram)



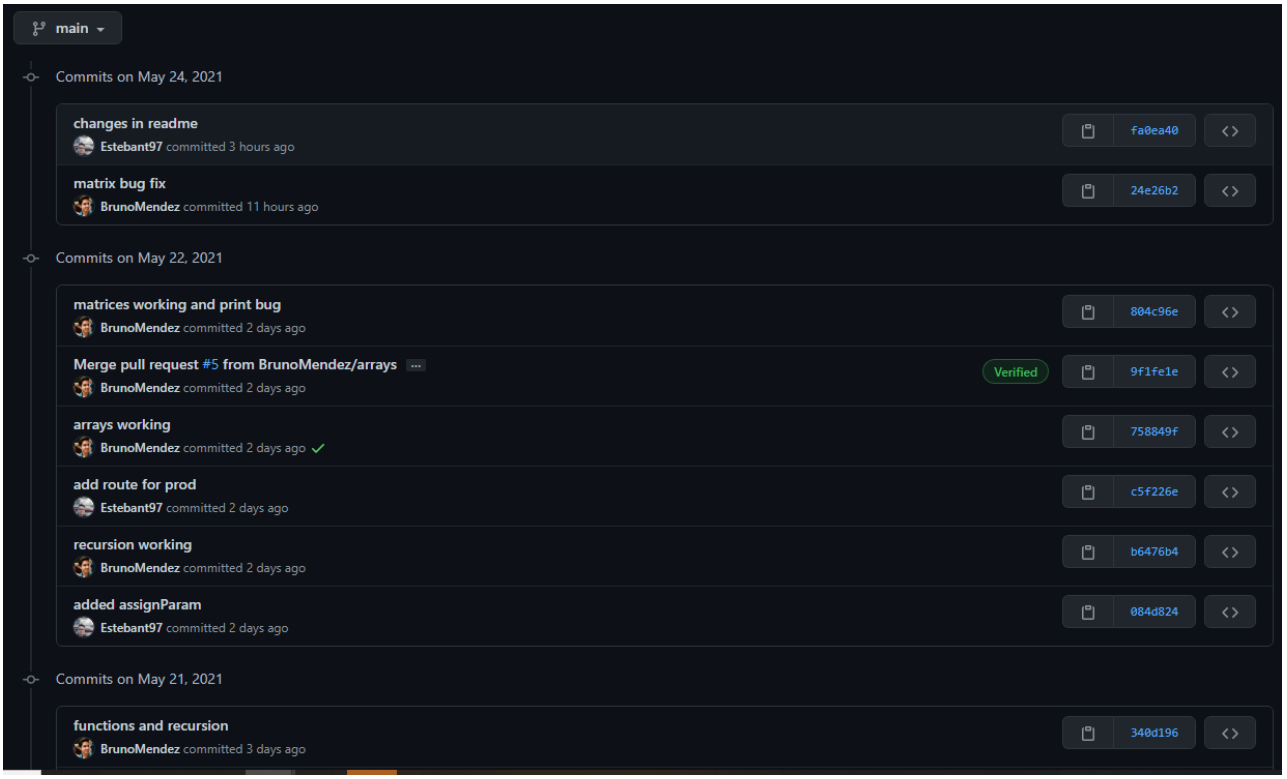
Test Cases Description (Table)

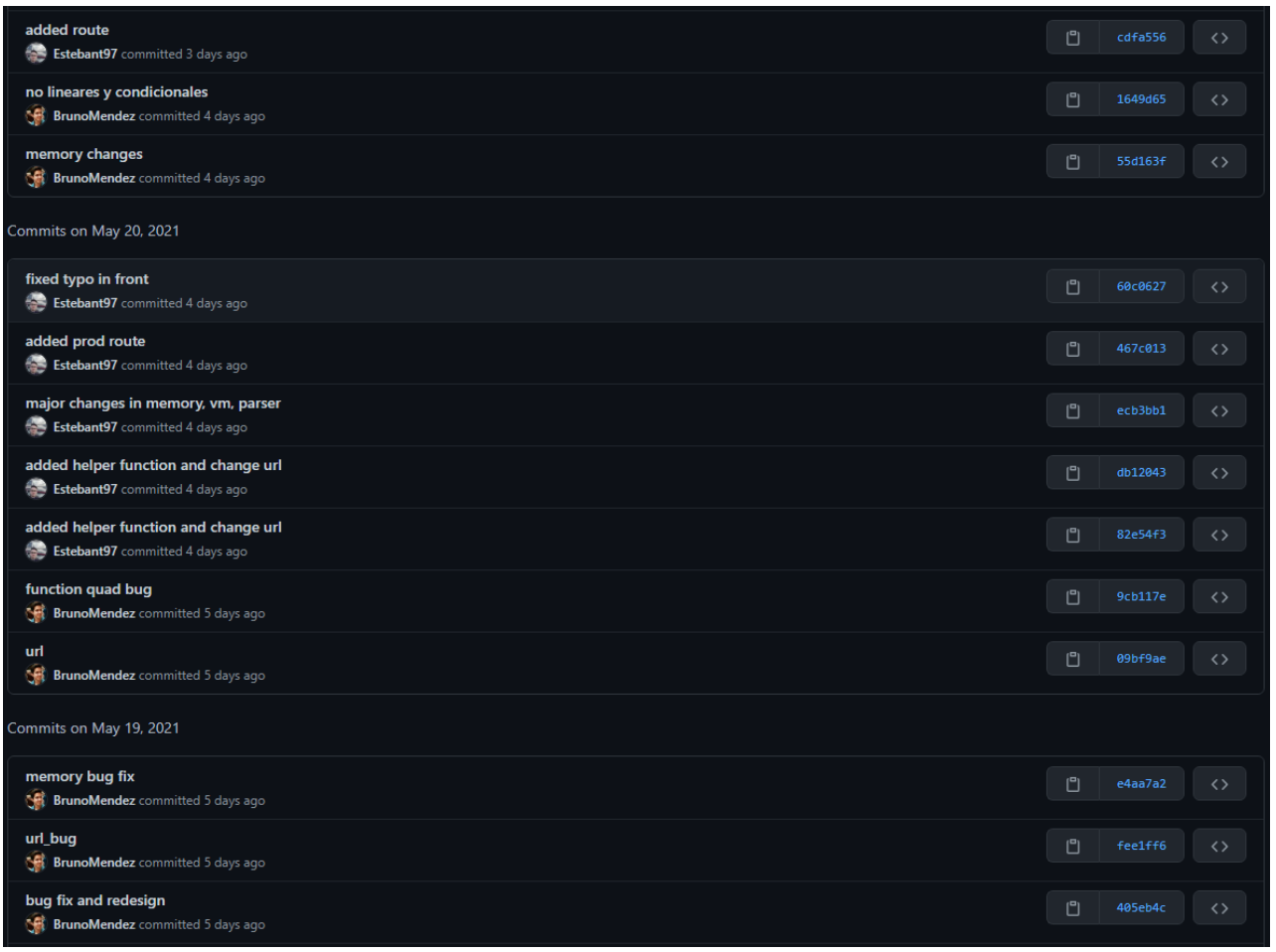
Recursive factorial	Recursive approach to calculating the factorial of a number
Cyclical factorial	Cyclical factorial approach to calculating the factorial of a number
Recursive fibonacci	Recursive approach to calculating the fibonacci series
Cyclical fibonacci	Cyclical approach to calculating the fibonacci series
Operations in 2d arrays	Demonstrate operations using 2d arrays
Bubble Sort	Sort an array using nested while loops

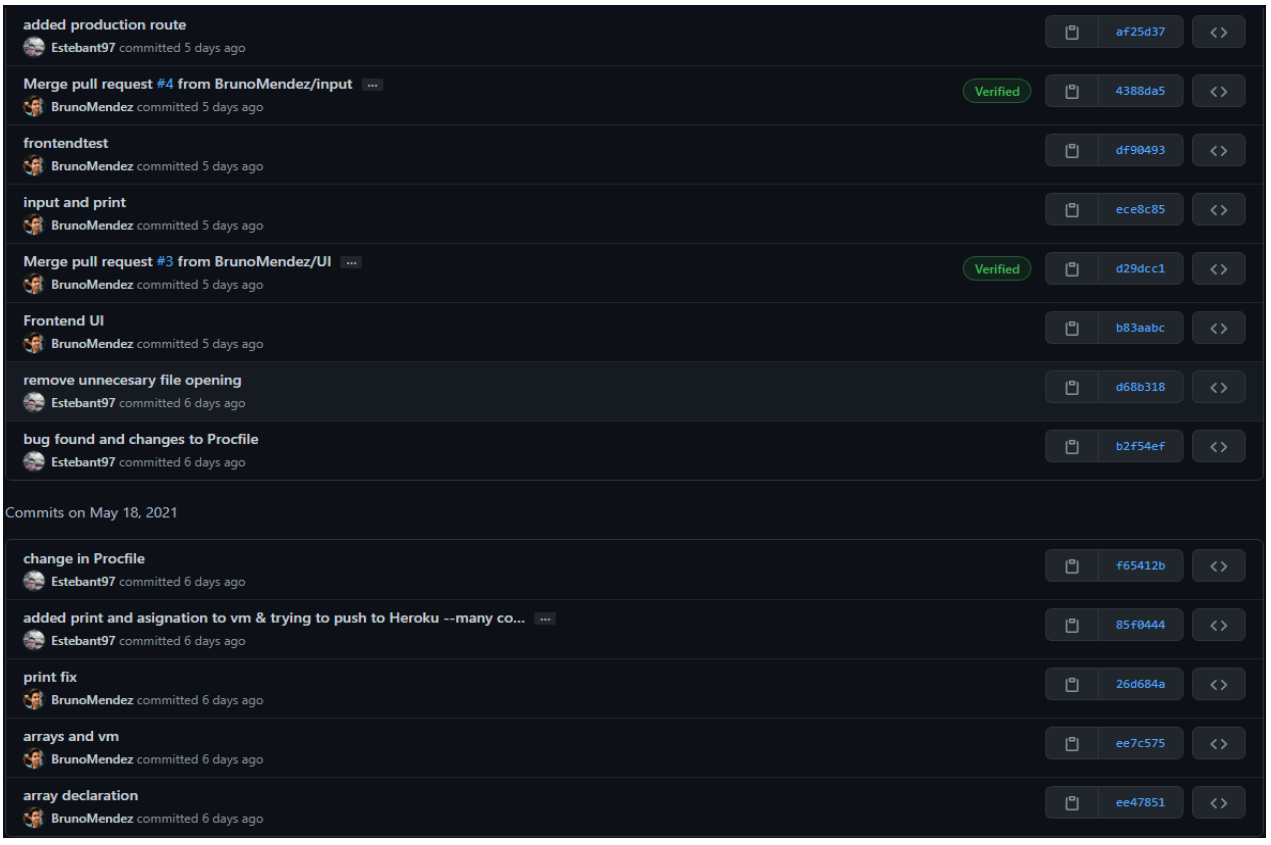
Project Development Process (Github & Screenshots)

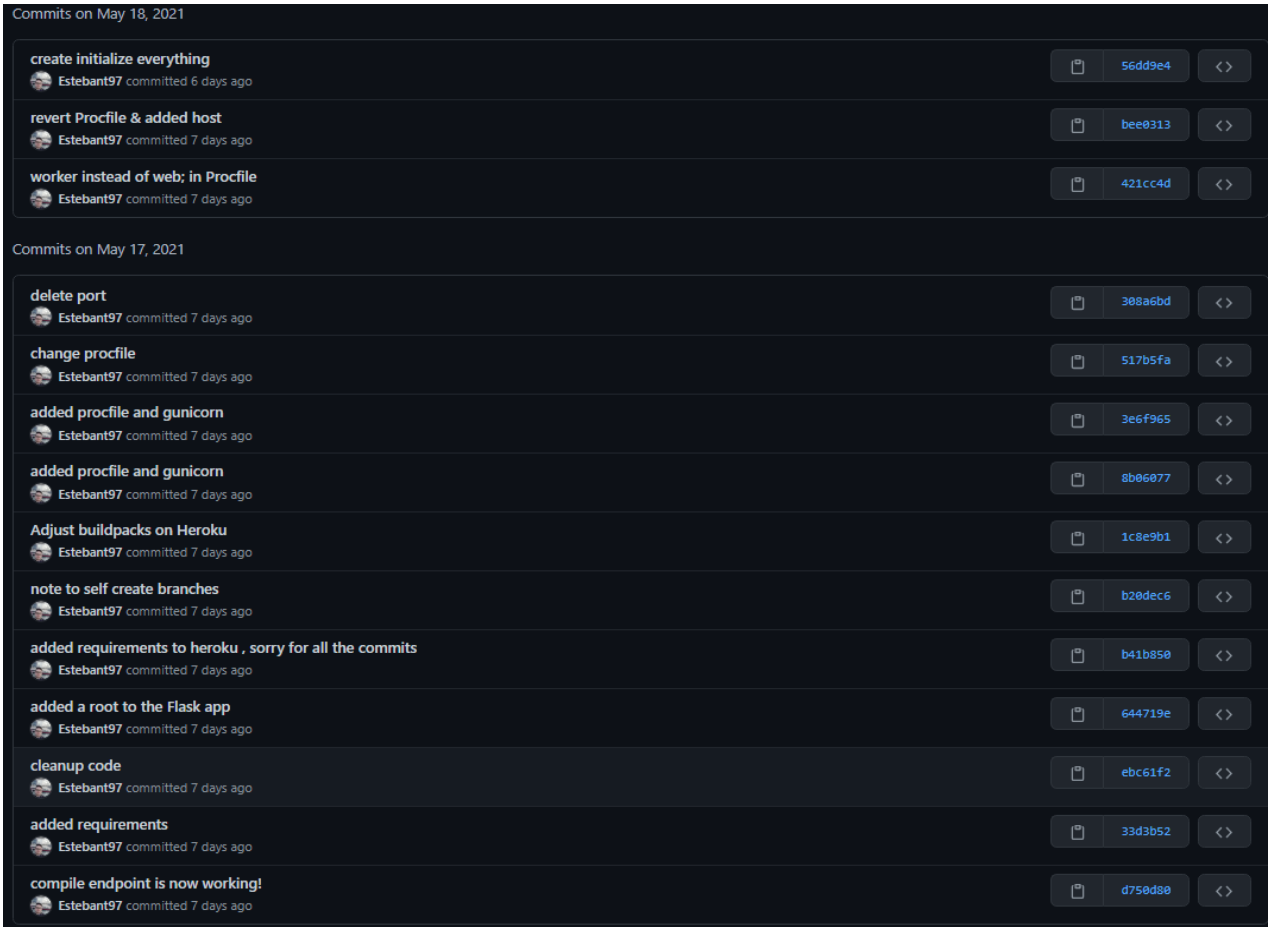
For version control we use Github to make the contributions in the branches and maintain records of the work done. Pair programming done twice a week with a worklog updated each iteration following an explanation of each commit done in the repository

Commits




added route
Estebant97 committed 3 days ago
cdfa556
no lineares y condicionales
BrunoMendez committed 4 days ago
1649d65
memory changes
BrunoMendez committed 4 days ago
55d163f
Commits on May 20, 2021
fixed typo in front
Estebant97 committed 4 days ago
60c0627
added prod route
Estebant97 committed 4 days ago
467c013
major changes in memory, vm, parser
Estebant97 committed 4 days ago
ecb3bb1
added helper function and change url
Estebant97 committed 4 days ago
db12043
added helper function and change url
Estebant97 committed 4 days ago
82e54f3
function quad bug
BrunoMendez committed 5 days ago
9cb117e
url
BrunoMendez committed 5 days ago
09bf9ae
Commits on May 19, 2021
memory bug fix
BrunoMendez committed 5 days ago
e4aa7a2
url_bug
BrunoMendez committed 5 days ago
fee1ff6
bug fix and redesign
BrunoMendez committed 5 days ago
405eb4c


added production route
Estebant97 committed 5 days ago
af25d37
Merge pull request #4 from BrunoMendez/input
BrunoMendez committed 5 days ago
Verified
4388da5
frontendtest
BrunoMendez committed 5 days ago
df90493
input and print
BrunoMendez committed 5 days ago
ece8c85
Merge pull request #3 from BrunoMendez/UI
BrunoMendez committed 5 days ago
Verified
d29dcc1
Frontend UI
BrunoMendez committed 5 days ago
b83aabc
remove unnecesary file opening
Estebant97 committed 6 days ago
d68b318
bug found and changes to Procfile
Estebant97 committed 6 days ago
b2f54ef
Commits on May 18, 2021
change in Procfile
Estebant97 committed 6 days ago
f65412b
added print and asignation to vm & trying to push to Heroku --many co...
Estebant97 committed 6 days ago
85f0444
print fix
BrunoMendez committed 6 days ago
26d684a
arrays and vm
BrunoMendez committed 6 days ago
ee7c575
array declaration
BrunoMendez committed 6 days ago
ee47851



Bitácora de Entregas fechas pendientes

Apr 9 by 11:59pm

En nuestro primer avance logramos definir el lex y el parser de nuestro compilador para dispositivos móviles tenemos pendiente ver unos warnings que nos aparecieron al correr el codigo y se generaron las tablas LALR que queremos checar para el próximo avance

Apr 16 by 11:59pm

En nuestro segundo avance lograremos corregir todos los warnings que nos aparecen al ejecutar el parser, al igual que definimos archivos para testear y verificamos su funcionamiento. Definimos el directorio de funciones y la tabla de variables (programadas) y el diseño de la tabla de consideraciones semánticas.

Apr 23 by 11:59pm

En nuestro tercer avance logramos corregir algunos errores de la gramática al igual que programamos la tabla de consideraciones semánticas, generación de código de expresiones aritméticas y los estatutos secuenciales. Al igual que avanzamos definiendo la pauta para los siguientes entregables a realizar.

May 1 by 11:59pm
En el cuarto avance logramos agregar todo el código intermedio para los estatutos no-lineales for, while, if, if-else. También creamos un avail para guardar temporales y arreglamos bugs de entregas pasadas.

May 8 by 11:59pm
En el quinto avance logramos agregar todo el código intermedio para llamar a funciones void y declarar funciones void. También realizamos unos cambios al parser para poder agregar lo desarrollado correctamente.

May 16 by 11:59pm
En el sexto avance logramos terminar todo el código para declarar y llamar funciones con tipo. También realizamos el código intermedio para memoria y un primer avance del front/backend para comunicarnos con el Compilador usando Flask y NodeJS. El front-end no es final solo es para visualizar.

May 25 by 11:59pm
En el séptimo avance logramos corregir varios errores que teníamos de entregas pasadas, logramos declarar e inicializar arreglos de 1 y 2 dimensiones. También realizamos el codigo de ejecucion de funciones y llamadas recursivas. Para esta entrega tenemos ya casi toda la funcionalidad del compilador, aún falta checar un detalle para tener un input dentro de una función. Al igual ya esta lista el front/backend utilizando React y Flask.

Reflexión Esteban

En este proyecto pudimos darle una refrescada a todos los conocimientos aprendidos de nuestra carrera desde programación I hasta lo más avanzado. Me pareció muy retador y interesante, poniendo a prueba nuestros conocimientos. Al igual que me pareció increíble pasar de la teoría a la aplicación real y terminar con un producto final funcional fue lo que mas me gusto de la materia.

Reflexión Bruno

Este proyecto ha sido de los favoritos de mi carrera porque fue bastante retador, lo cual me obligó a practicar la mayoría de los conocimientos que aprendí en mi carrera. Me gusto que en nuestra carrera pasamos de aprender a programar a crear nuestro propio lenguaje y compilador. Finalmente me gustó el proyecto porque me dio un entendimiento mucho mayor sobre cómo funcionan los compiladores que estoy seguro que me hará ser mejor programador.

Bruno Mendez

Bruno Mendez

Eta

Esteban Torres

What were the things we had the most issues with?

With the “input” we had a lot of issues because the compiler is running in a server separate from the front, therefore whenever an input was detected we needed to tell the front end to expect an input value and save the quadruples generated until that moment. For this part we created another Flask route that receives the quadruples generated until that moment and we send the current quadruple and the input value passed to this route to the VM.

The next function helps us with the input from the frontend to the backend.

```
# User input request from front end.
@app.route('/user-input', methods=["POST"])
def userInput():
    content = request.get_json()
    # JSON request example:
    # {
    #   input_value: 2,
    #   current_quad: 18,
    # }
    try:
        # Uses previously generated quadruples (in compile request).
        return vm.start(quadruples,
                        currentQuad=content[CURRENT_QUAD],
                        inputValue=content[INPUT_VALUE])
    except Exception as e:
        print("Error", e)
        errors = {1: "Error: " + str(e)}
        return errors
```

In the front end this route is called when the state of the input div is changed (in other words, when it has a value), this information is sent to the backend and the virtual machine returns all the generated results, so that can be displayed in the frontend.

The other thing we have lots of issues was with deploying the compiler to a server, all the documentation online explained it for other type of projects therefore a lot of research and trial and error was done in this part:

For this to work we had to add a requirements.txt (to add all the libraries needed to work)

```
antlr4-python3-runtime==4.9.1
Flask==1.1.2
Flask-Compress==1.9.0
flask-core==2.9.0
Flask-Cors==3.0.10
gunicorn==20.1.0
json5==0.9.5
jsonschema==3.2.0
ply==3.11
Werkzeug==1.0.1
```

, and a Procfile (this is like running the parser e.g. python parser.py), but in this case we needed to use a library named gunicorn to handle our app, the command used is : web: gunicorn parser:app.

Language Description

Language Name

PYCOFFEE

Language Characteristics Description

PyCoffee is a programming language very similar to C/C++ that contains simple arithmetic with basic uses of array declaration and manipulation, function declaration and calling, memory storage and input/output capabilities. It can be used in any device with an internet connection and an internet browser so it is really accessible for all the people without taking in consideration their machine architecture, power, etc.

Compile-time and Execution-time Errors (Table)

Compile Time	
Type Mismatch	Variable type does not match
Var Already in Table	Variable already declared
Var Not Defined	Variable not defined
Function Not Declared	Function was not declared
Invalid parameter number	Invalid parameters number
Non-void function return missing	Non-void functions must have a return statement
Invalid Type	The type is invalid
Syntax Error	Syntax error found
StackOverflow	Memory exceeded

Execution Time	
Out of bound	Index is out of bounds (for arrays)
StackOverflow	Memory exceeded

Compiler Description

Computing Equipment, Languages and Utilities

Operating System: Windows 10
Languages Used: Python 3.8
Libraries, frameworks,etc :

- antlr4-python3-runtime==4.9.1
- Flask==1.1.2
- Flask-Compress==1.9.0
- flask-core==2.9.0
- Flask-Cors==3.0.10
- gunicorn==20.1.0
- json5==0.9.5

jsonschema==3.2.0
ply==3.11
Werkzeug==1.0.1
React

Esteban Torres
Brand: ???
Model: ???
Computer was built by Esteban Torres
Bruno Mendez
Brand: Dell
Model: XPS15 9560
Server:
Buildpacks : Heroku/python

Lexical Analysis Description

```
# Lista de nombres de tokens.
tokens = [
    'ID', 'SEMICOLON', 'COLON', 'COMA', 'LBRACE', 'RBRACE', 'LBRACKET',
        'RBRACKET',  'EQUAL',  'PLUS',  'MINUS',  'MULTIPLY',  'DIVIDE',
'LPAREN',
     'RPAREN', 'CST_STRING', 'CST_CHAR', 'CST_INT', 'CST_FLOAT', 'LT',
'GT',
    'NE', 'EQEQ', 'LTEQ', 'GTEQ'
]

# Lista de palabras reservadas.
reserved = {
    'program': 'PROGRAM',
    'var': 'VAR',
    'function': 'FUNCTION',
    'int': 'INT',
    'float': 'FLOAT',
    'char': 'CHAR',
    'void': 'VOID',
    'print': 'PRINT',
    'input': 'INPUT',
    'if': 'IF',
    'else': 'ELSE',
    'main': 'MAIN',
    'return': 'RETURN',
    'while': 'WHILE',
```

```
    'for': 'FOR',
    'and': 'AND',
    'or': 'OR',
    'not': 'NOT'
}
tokens += reserved.values()

# Regexpr para tokens simples
t_SEMICOLON = r';'
t_COLON = r':'
t_COMA = r','
t_LBRACE = r'\{'
t_RBRACE = r'\}'
t_LBRACKET = r'\['
t_RBRACKET = r'\]'
t_EQUAL = r'='
t_PLUS = r'\+'
t_MINUS = r'-'
t_MULTIPLY = r'\*'
t_DIVIDE = r'/'
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_CST_STRING = r'("(\\"|[^"])*")'
t_CST_CHAR = r'\'[a-zA-Z]\''
t_CST_INT = r'[0-9]+'
t_CST_FLOAT = r'[0-9]+\.[0-9]+'
t_LT = r'<'
t_GT = r'>'
t_NE = r'<>'
t_EQEQ = r'=='
t_LTEQ = r'<='
t_GTEQ = r'>='


# Regexpr para tokens que requieren más código
def t_ID(token):
    r'[a-zA-Z][a-zA-Z0-9_]*'
    if token.value in reserved:
        token.type = reserved[token.value]
    return token


# Regla para tomar en cuenta cambios de línea
```

```
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)


# Caracteres a ignorar
t_ignore = ' \t'
t_ignore_COMMENT = r'%%.*'


# Manejo de errores
def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)
```

Syntax Analysis Description

'''program : PROGRAM ID createGlobalTables SEMICOLON vars functions MAIN LPAREN RPAREN mainStart block'''

'''vars : VAR varsPrime | '''

'''varsPrime : listIds COLON type addVars SEMICOLON varsPrime | '''

'''functions : function functions | '''

'''listIds : ids listIdsPrime'''

'''listIdsPrime : COMA ids listIdsPrime | '''

'''ids : ID addId | ID addId LBRACKET CST_INT addArr1 RBRACKET | ID addId LBRACKET CST_INT addArr1 RBRACKET LBRACKET CST_INT addArr2 RBRACKET'''

'''ids2 : ID addIdToStack checkIfNotFunction | ID addIdToStack arrPos'''

'''arrPos : LBRACKET getArr1 exp getArr2 RBRACKET addArr4 | LBRACKET getArr1 exp getArr2 RBRACKET LBRACKET addArr5 exp addArr3 RBRACKET addArr4'''

'''type : INT | FLOAT | CHAR'''

```
'"returnType : type | VOID"'

'"function : returnType FUNCTION ID addFunction1 LPAREN params RPAREN addFunction3 vars addFunction4 block"'

'"params : ids COLON type addFunction2 paramsPrime | "'

'"paramsPrime : COMA params  | "'

'"block : LBRACE statements RBRACE"'

'"statements : statement statements  | "'

'"statement : assignment | write | callVoidF | return | read | decision | repetition | expression"'

'"assignment : ids2 EQUAL addOperator expression addAssignment SEMICOLON"'

'"write : PRINT LPAREN writePrime RPAREN SEMICOLON"'

'"writePrime : addFakeBottom expression popOperator printExpression writePrimePrime | CST_STRING printString writePrimePrime"'

'"writePrimePrime : COMA writePrime | "'

'"callVoidF : ID addIdToStack callFunction SEMICOLON"'

'"callFunction : LPAREN callFunction1 expressions RPAREN callFunction3"'

'"expressions : expression callFunction2 expressionsPrime  | "'

'"expressionsPrime : COMA expressions  | "'

'"return : RETURN LPAREN expression RPAREN SEMICOLON"'

'"read : INPUT addOperator LPAREN readPrime RPAREN SEMICOLON"'

'"readPrime : ids2 readVar readPrimePrime"'

'"readPrimePrime : COMA readPrime | "'

'"repetition : conditional | nonConditional"'

'"decision : IF LPAREN expression addIf1 RPAREN block decisionPrime
```

```
addIf2"'

"'decisionPrime : addIf3 ELSE block | "'

"'conditional : WHILE addWhile1 LPAREN expression addWhile2 RPAREN
block addWhile3"'

"'nonConditional : FOR LPAREN ids2 EQUAL addOperator exp addFor1
COLON exp addFor2 RPAREN block addFor3"'

"'expression : miniExpression addAndOr AND addOperator expression |
miniExpression addAndOr OR addOperator expression | miniExpression
addAndOr"'

"'miniExpression : NOT addOperator LPAREN expression RPAREN addNot
| microExpression addNot"'


"'microExpression : exp addExp GT addOperator microExpression | exp
addExp LT addOperator microExpression | exp addExp NE addOperator
microExpression  | exp addExp EQEQ addOperator microExpression  | exp
addExp LTEQ addOperator microExpression | exp addExp GTEQ
addOperator microExpression | exp addExp"'

"'exp : term addTerm | term addTerm PLUS addOperator exp | term addTerm
MINUS addOperator exp"'

"'term : factor addFactor | factor addFactor MULTIPLY addOperator term |
factor addFactor DIVIDE addOperator term"'

"'factor : LPAREN addOperator expression RPAREN popOperator | varCst"'

"'varCst : CST_FLOAT addFloat | CST_INT addInt | CST_CHAR addChar |
callableCst"'

"'callableCst : ID addIdToStack checkIfNotFunction |  ID addIdToStack
callFunction | ID addIdToStack arrPos"'

"'mainStart : "'

"'createGlobalTables :  "'

"'addVars : "'

"'addFunction2 : "'
```

```
'"addId1 : '"
'"addId2 : '"
'"addId : '"
'"checkIfNotFunction : '"
'"addIdToStack : '"
'"getArr1 : '"
'"addArr5 : '"
'"getArr2 : '"
'"addArr3 : '"
'"addArr4 : '"
'"addFunction4 : '"
'"addFunction3 : '"
'"addFunction1 : '"
'"printExpression : '"
'"printString : '"
'"popOperator : '"
'"addFakeBottom : '"
'"callFunction1 : '"
'"callFunction2 : '"
'"callFunction3 : '"
'"readVar : '"
'"popOperator : '"
```

```
'"addOperator : '"

'"addAssignment : '"

'"addAndOr : '"

'"addNot : '"

'"addExp : '"

'"addTerm : '"

'"addFactor : '"

'"addIf1 : '"

'"addIf2 : '"

'"addIf3 : '"

'"addWhile1 : '"

'"addWhile2 : '"

'"addWhile3 : '"

'"addFor1 : '"

'"addFor2 : '"

'"addFor3 : '"

'"addFloat : '"

'"addInt : '"

'"addChar : '"
```

Code Generation and Semantic Analysis Description

The code generation is done using quadruples that are generated in the following format:

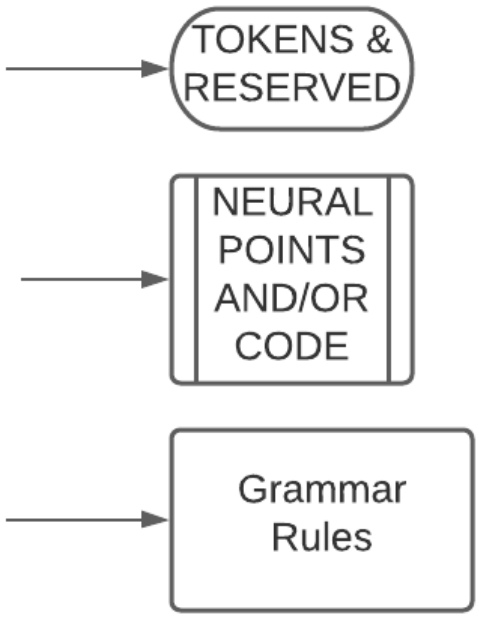
(operator, leftOperand, rightOperand, result)
The operator can be one from the range of operators supported by our language, including logical and mathematical operators and special operators like GOSUB, ERA, GOTO and GOTOF. The GOSUB and ERA are used for function calling and go to the function and switch contexts. The GOTO and GOTOF are used in loops and conditions.

The operands are memory addresses defined below and representing the values each one represents and the result can be memory addresses or jump to quadruple instructions, which moves the context of the program.

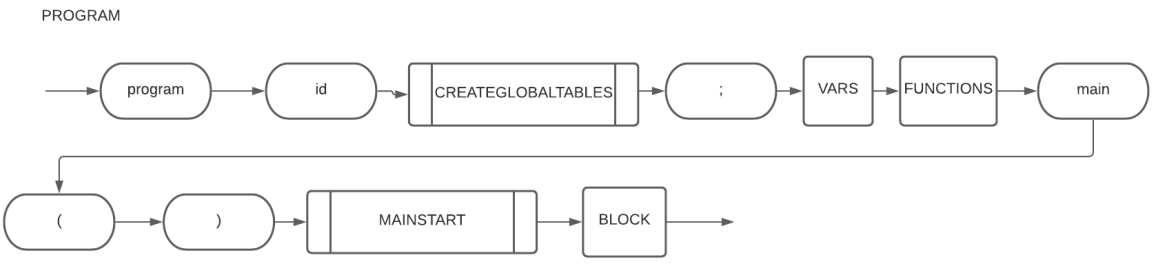
- Global Int : 1000 - 1999
- Global Float : 2000 - 2999
- Global Char : 3000 - 3999
- Local Int : 4000 - 4999
- Local Float : 5000 - 5999
- Local Char : 6000 - 6999
- Temporary Int : 7000 - 7999
- Temporal Float : 8000 - 8999
- Temporal Char : 9000 - 9999
- Constant Int : 10000 - 10999
- Constant Float : 11000 - 11999
- Constant Char : 12000 - 12999
- Void : 13000 - 13999
- Temporal Local Int : 14000 - 14999
- Temporal Local Float : 15000 - 15999
- Temporal Local Char : 16000 - 16999

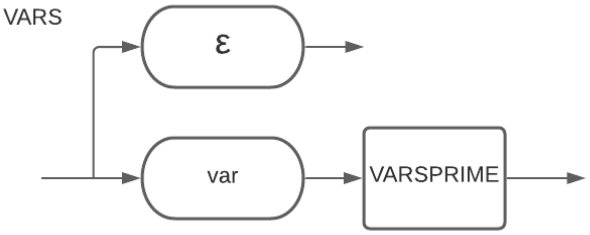
Syntax Diagrams and Action Description

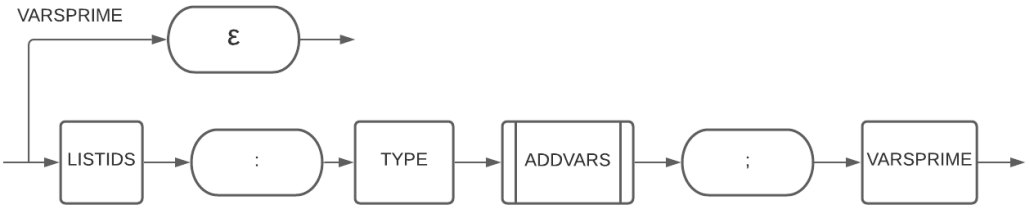
FIGURES TO BE USED

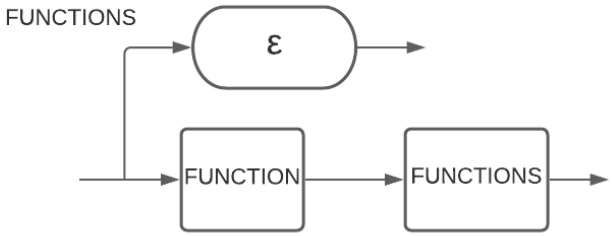


DIAGRAMS

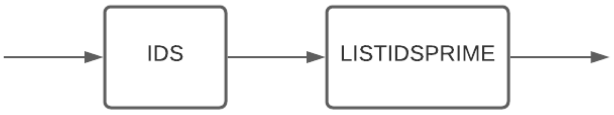


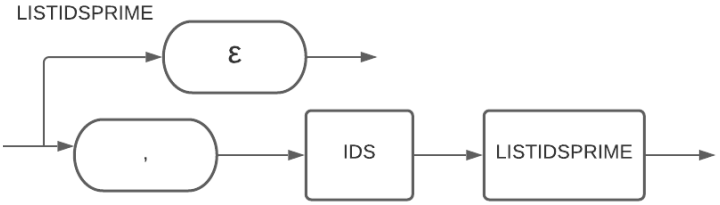



VARSPRIME
ε
LISTIDS
:
TYPE
ADDVARS
;
VARSPRIME

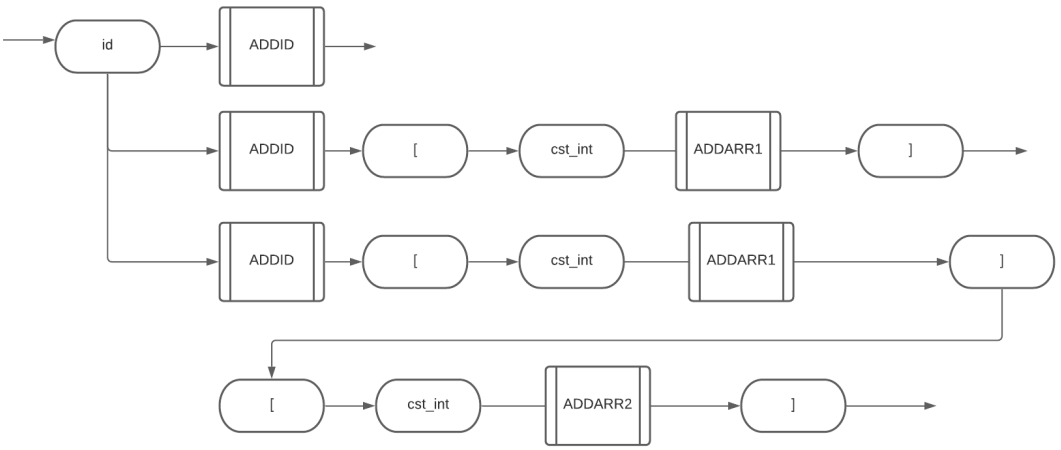

FUNCTIONS
ε
FUNCTION
FUNCTIONS

LISTIDS

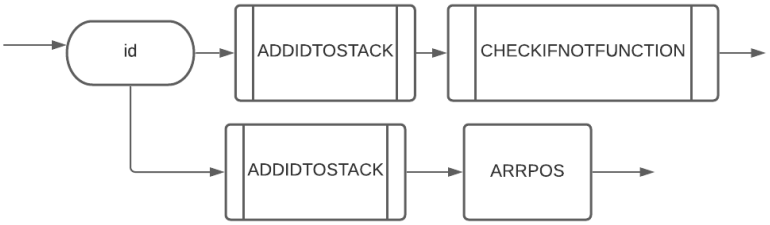

IDS
LISTIDSPRIME


LISTIDSPRIME
ε
,
IDS
LISTIDSPRIME

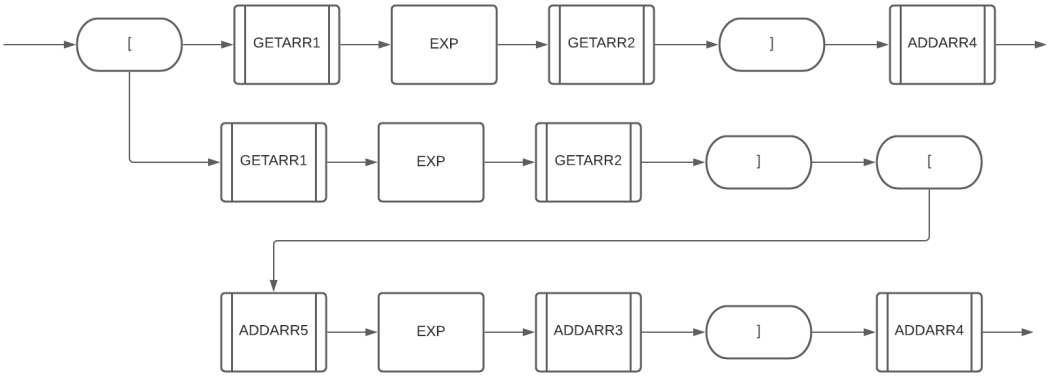
IDS



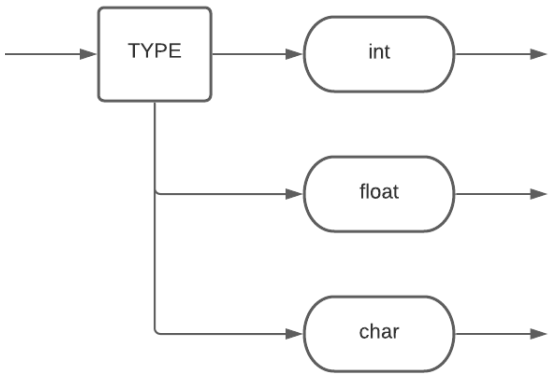
IDS2



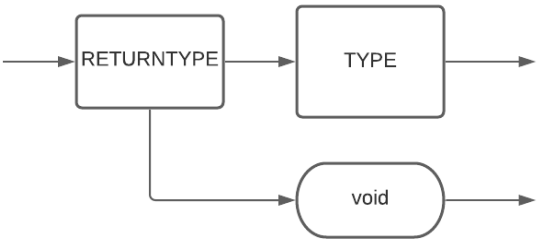
ARRPOS



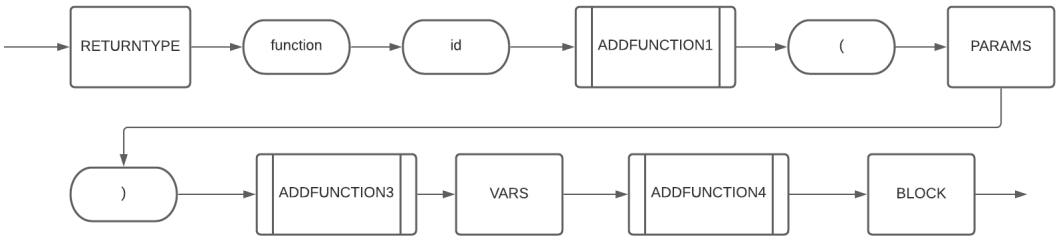
TYPE



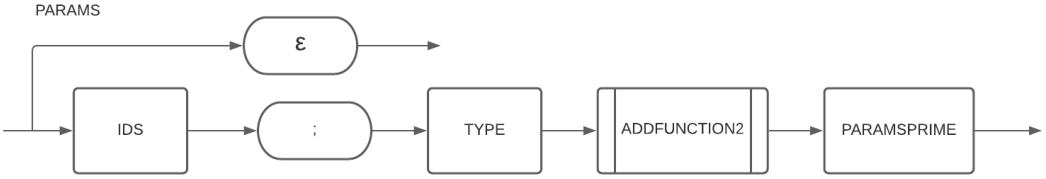
RETURNTYPE



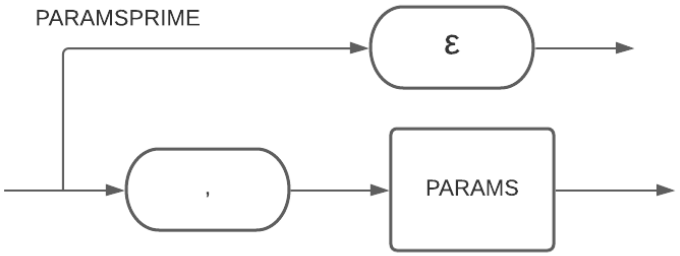
FUNCTION



PARAMS

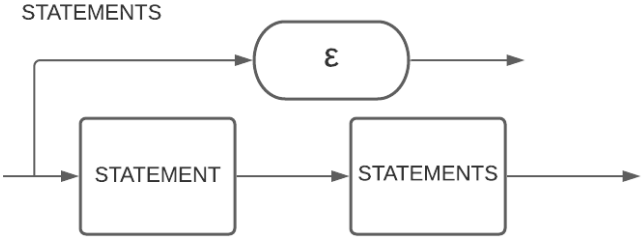


PARAMSPRIME

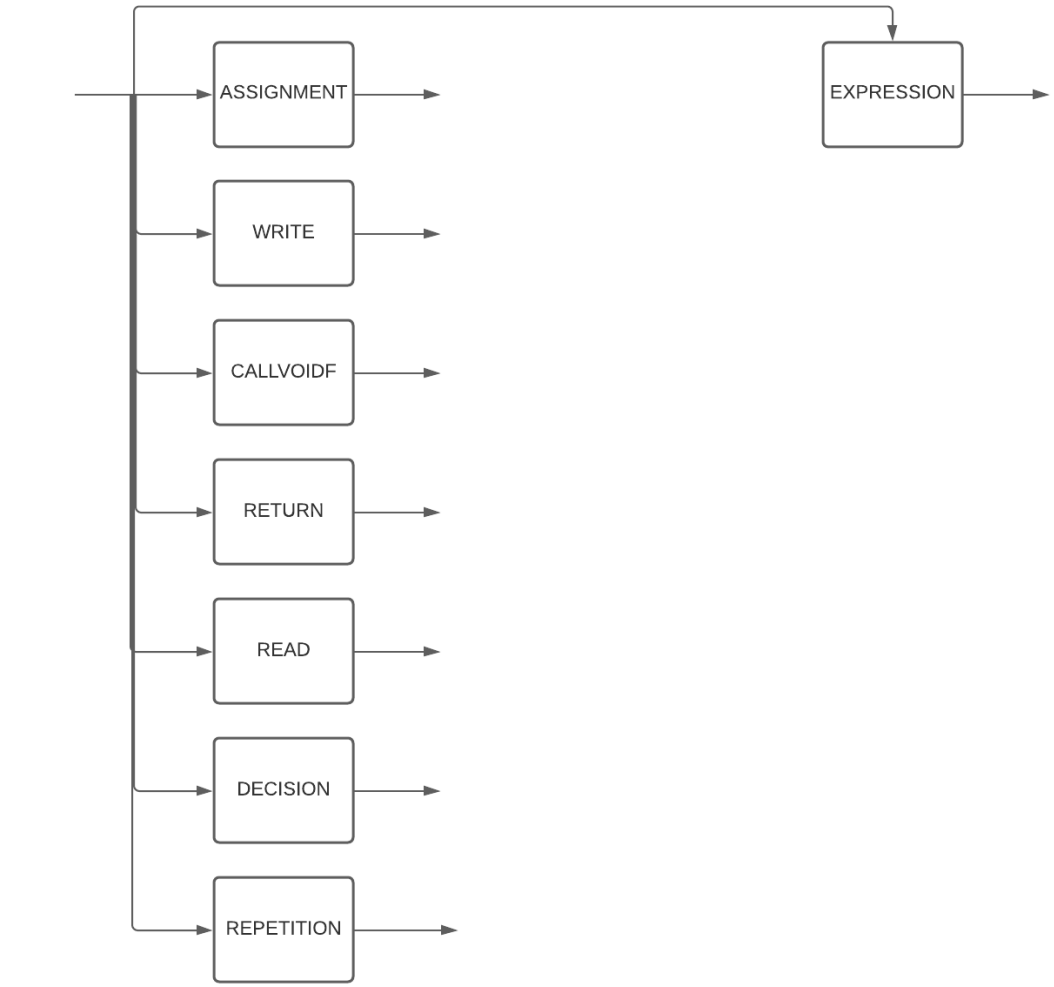


BLOCK

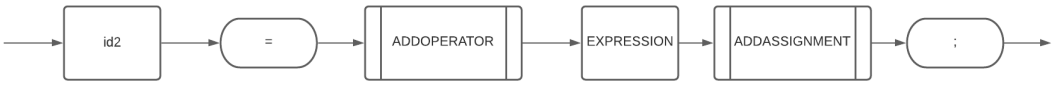
STATEMENTS



STATEMENT

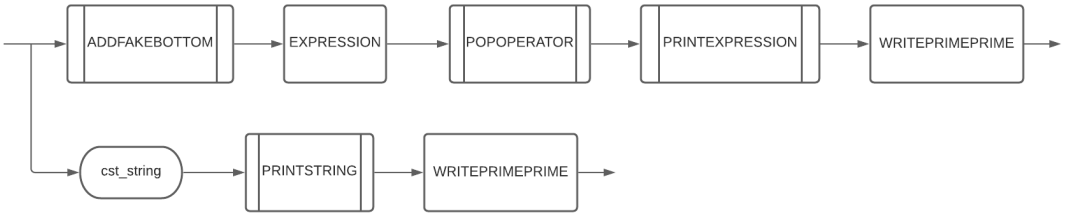


ASSIGNMENT

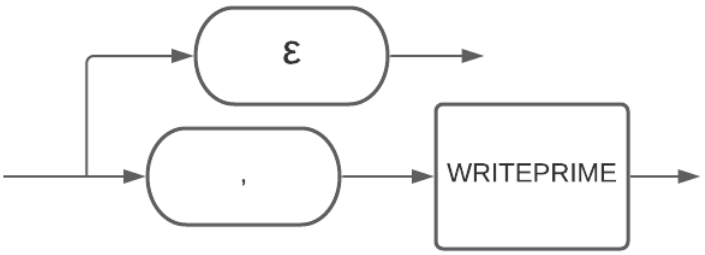


WRITE

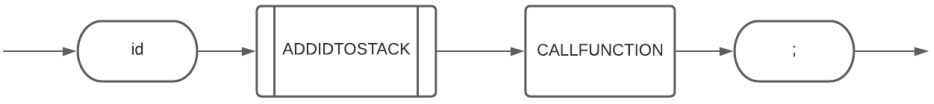
WRITEPRIME



WRITEPRIMEPRIME

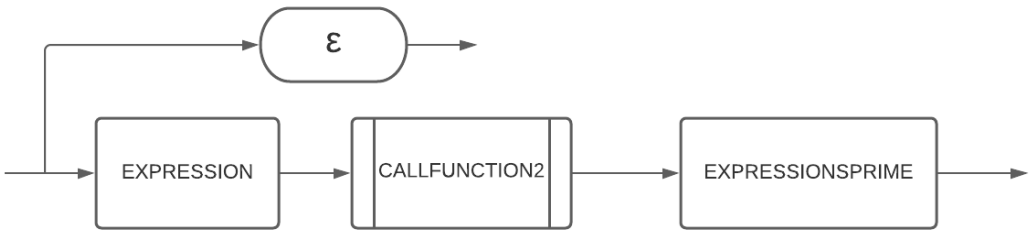


CALLVOIDF

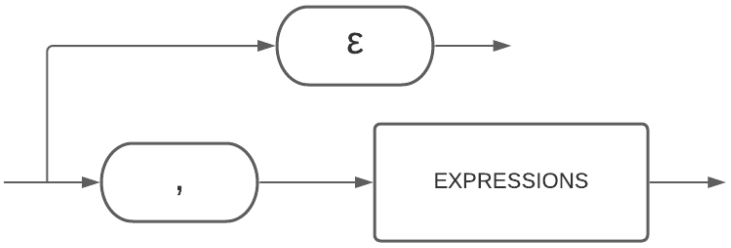


CALLFUNCTION

EXPRESSIONS

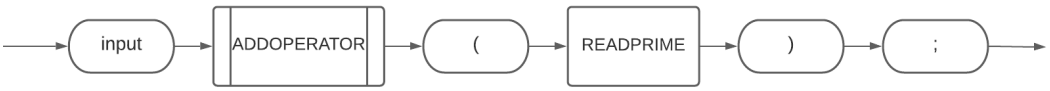


EXPRESSIONSPRIME

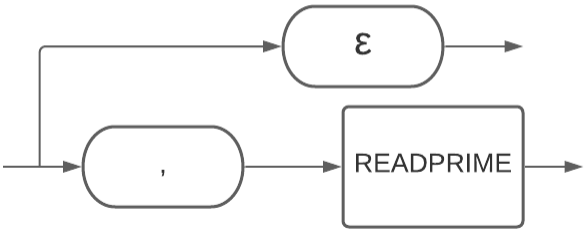


RETURN

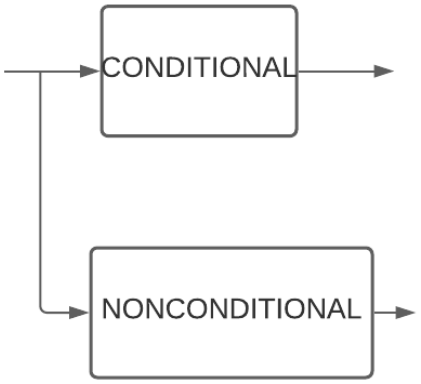
READ



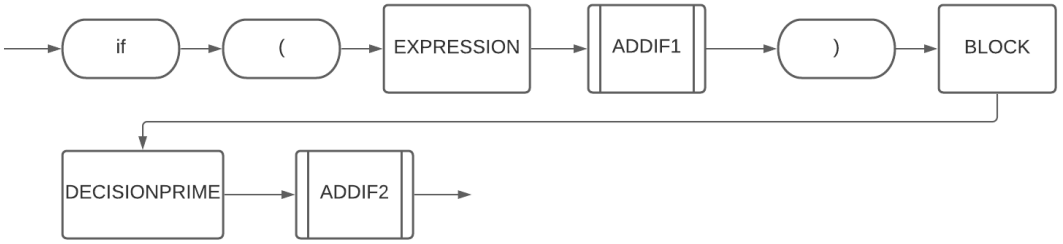
READPRIME



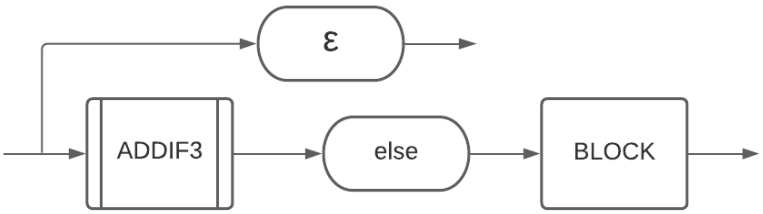
REPETITION



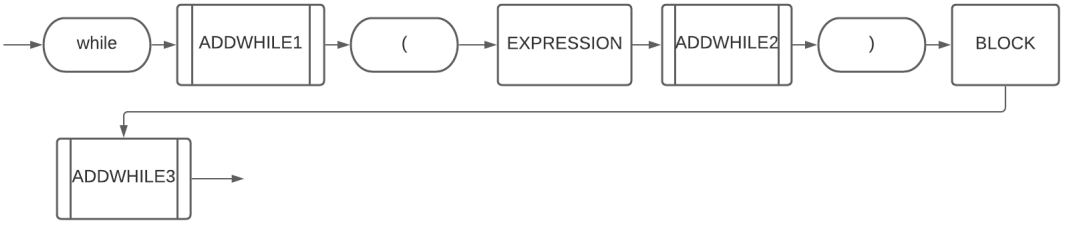
DECISION



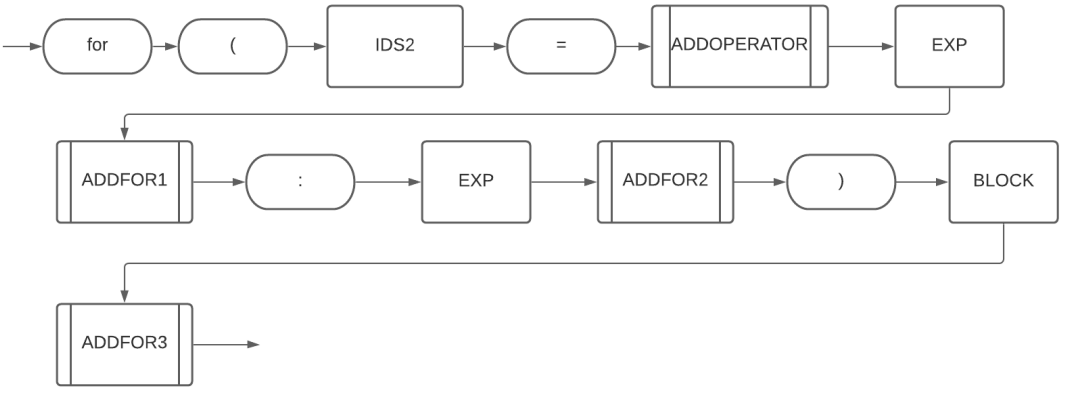
DECISIONPRIME



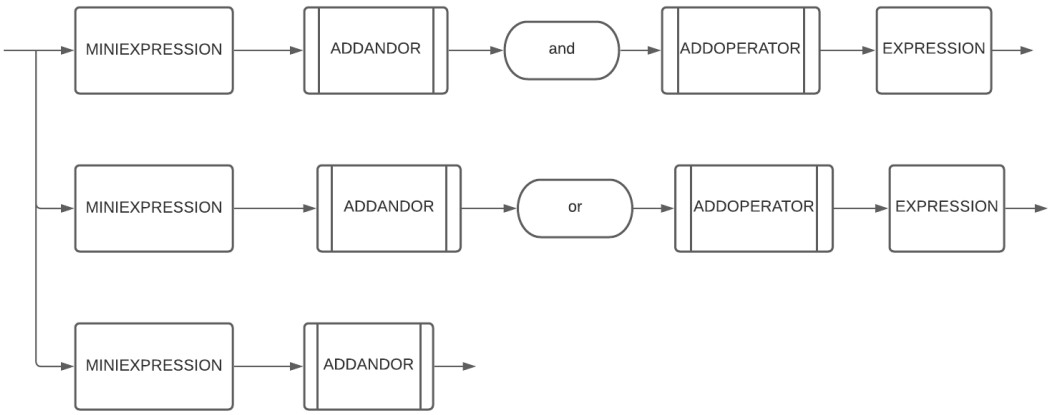
CONDITIONAL



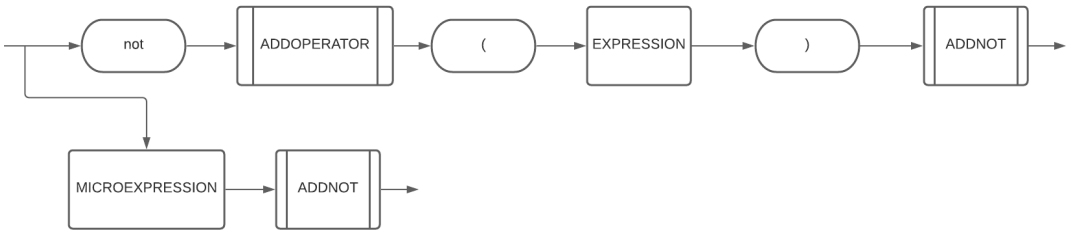
NONCONDITIONAL



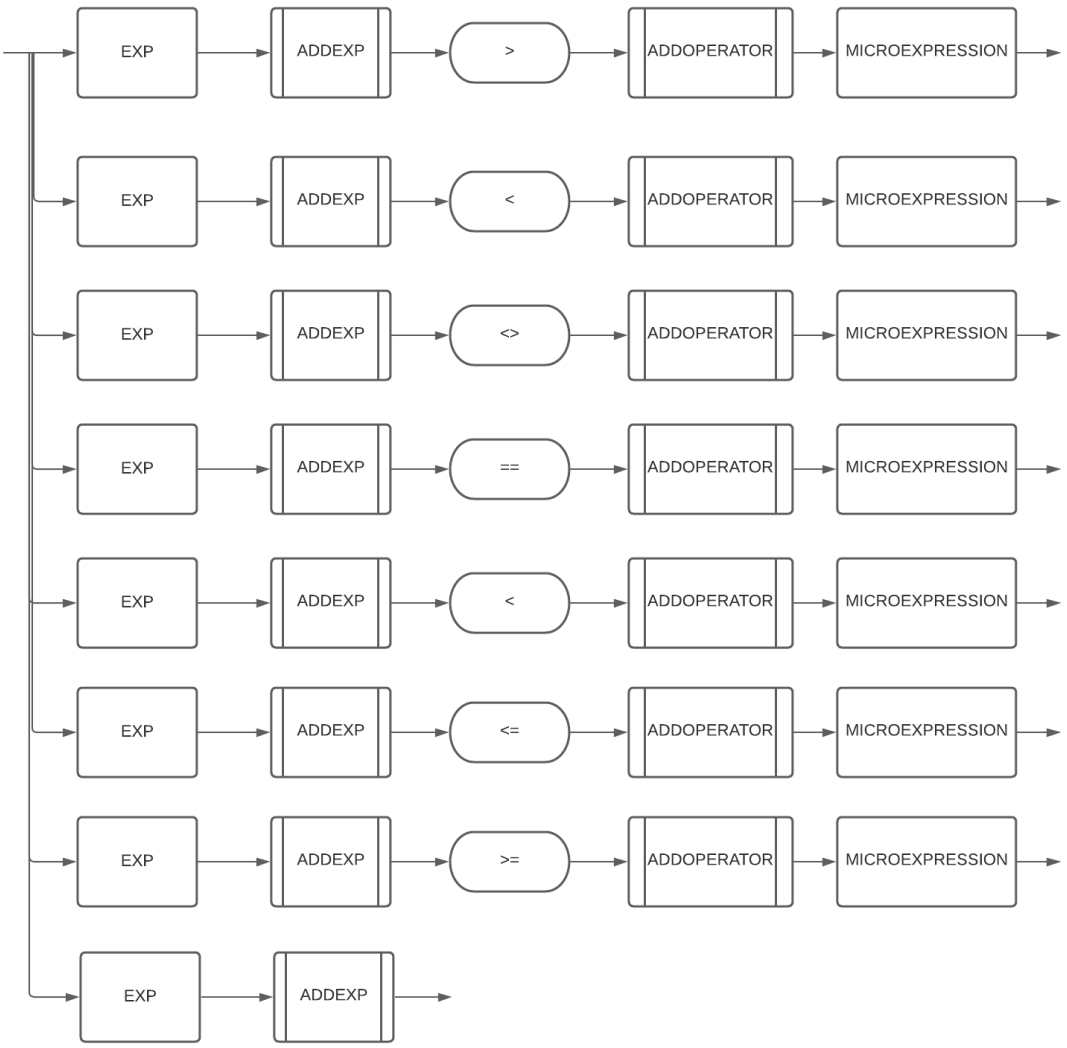
EXPRESSION



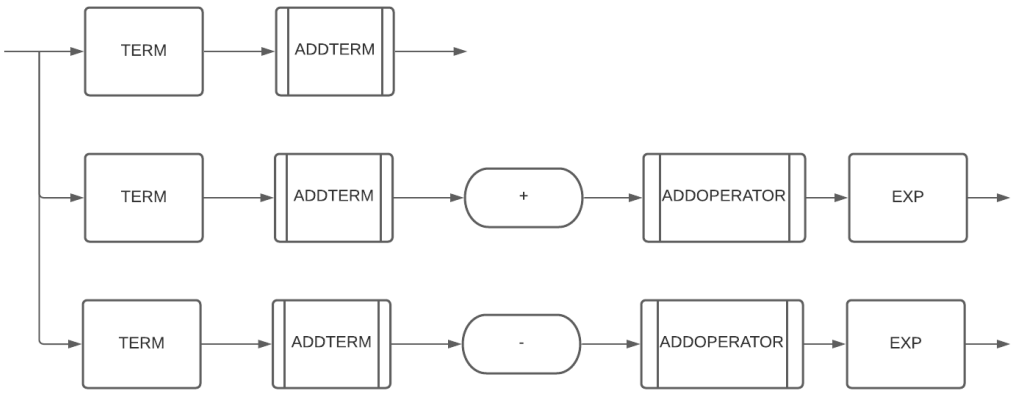
MINIEXPRESSION



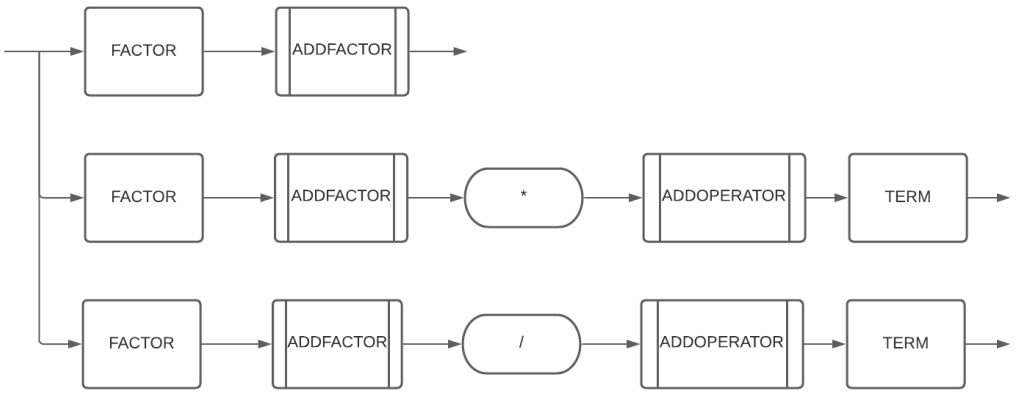
MICROEXPRESSION



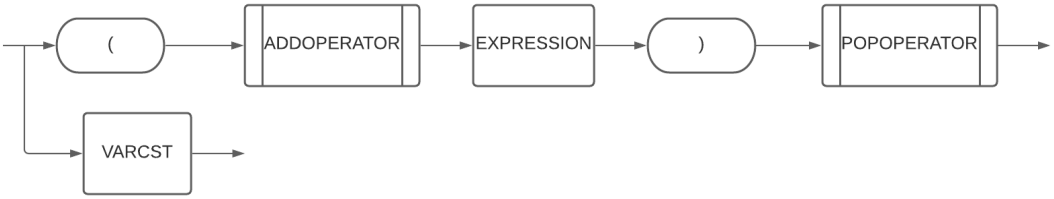
EXP



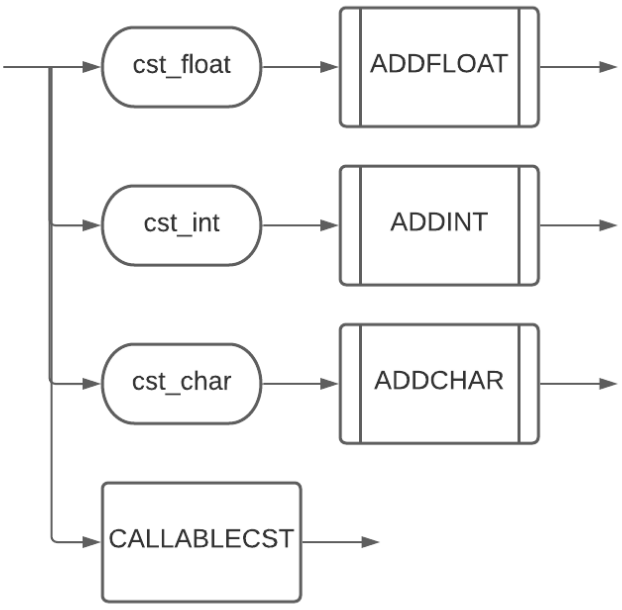
TERM

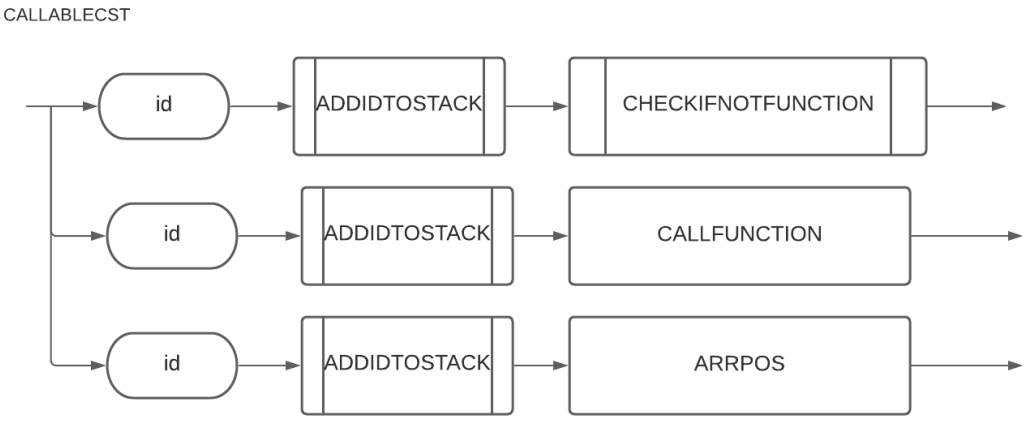


FACTOR



VARCST





Name	Description
mainStart	Indicates the position of the main function.
createGlobalTables	Initializes the global variables and dictionaries and generates the Goto Main Quadruple
addVars	Adds the variables in the queue to the variable table.
addFunction2	Adds local function variables to variable table.and has the same logic as addVars but with a paramTable.
addId1	Adds array with data to varIDs queue
addId2	Adds a second dimension and set its size, pass array as dict to differentiate arrays and variables
addId	Adds variable id to queue.
checkIfNotFunction	Checks if ID is not a function ID.
addIdToStack	Gets Id address and type and adds it to operator stack and type stack.
getArr1	Gets the id of an array and push a verify operation in an array [1st dimension and Pushes a fake botton

addArr3	Gets the id of an array and push a verify operation in an array [2nd dimension]
getArr2	Calculates addresses if it's a dimension=1 or dimension=2. Gets the memory addresses of the indexes and checks that the value passed is an Integer
addArr4	Pop fake bottom and verifies that the value passed is not out of bounds Appends quadruples addresses of the array/matrix
addArr5	Push a fake bottom in the addres to delimit that from that point until pop of fake bottom is part of the array/matrix
addFunction4	Sets the index of the quadruples length (at the moment) of the functionDirectory
addFunction3	Sets the parameters count for the functionDirectory
addFunction1	Initialize dictionaries, booleans. Sanity checks and delimits the function’s information
printExpression	Helps to print expressions e.g. print(x);
printString	Helps to print signs e.g. print("Hello World");
popOperator	Pop operator of the operatorStack
addFakeBottom	Add fake bottom to the operatorStack
callFunction1	Checks that the variable is a function, reset the boolean and push

	a Fake bottom Sets the paramCounter to 0 --> This helps for functions without parameters
callFunction2	Updates paramCounter and adds PARAMETER quadruple with param value. Checks that paramCounter is not bigger than the number of params.
callFunction3	Generates GOSUB quadruple. If non void function generates assignment quadruple and pushes result to stack.
readVar	Pushes input quadruple.
popOperator	Pop operator stack
addOperator	Push to the operator stack
addAssignment	Pushes assignment quadruple.
addAndOr	Push And/Or quadruple.
addNot	Add not quadruple.
addExp	Adds comparative operator quadruples.
addTerm	Adds +/- quadruples.
addFactor	Adds * or / quadruples.
addIf1	Adds GOTOF quadruples and adds current (starting) position to jumpstack.
addIf2	Tells quadruple in the jumpstack where the end of the if code is.
addIf3	Handles else statement.
addWhile1	Push start position to jumpstack
addWhile2	Check adds GOTOF with expression result.
addWhile3	Adds goto quad with start position added in addWhile1. Completes

	GOTOF quadruple with statements end position.
addFor1	Checks that expression type and the variable type are ints. Adds assignment quadruple. Sets up next for operations.
addFor2	Pushes <= comparation quadruple and GOTOF quadruple with the result of the <= operation as a condition.
addFor3	Adds countVar+1 quadruple and assignment quadruple. Adds GOTO start quadruple and fills GOTOF quadruple with end position.
addFloat	Add float constant to memory.
addInt	Add Int constant to memory
addChar	Add char constant to memory
addUMinus	Adds support for negative expression e.g (5 + (-6) = -1

Semantic Characteristics Tables

Addition, Subtraction and Multiplication

+, -, *	**int**	**float**	**char**
int	int	float	error
float	float	float	error
char	error	error	error

Division

/	**int**	**float**	**char**

int	float	float	error
float	float	float	error
char	error	error	error

Less Than, Greater Than

<, >	**int**	**float**	**char**
int	int	int	error
float	int	int	error
char	error	error	error

Not equal , Equal To

<>, ==	**int**	**float**	**char**
int	int	int	error
float	int	int	error
char	error	error	int

And/or

And, or	**int**	**float**	**char**
int	int	error	error
float	error	error	error
char	error	error	error

The and/or logic is used based on the same logic as our base programming language which is Python and it is employed with the tokens : “and” , “or”. It is used inside an if statement and we tested it with the following code :
For the boolean logic we used integer logic which means a 0 is a False and any number except 0 is True.

Test And expression	

program testAnd;
var x, y : int;
main(){
x = 1;
y = 2;
if(x == 1 and y == 2){
print("yes!");
}
}
	Output
	"yes!"
	FIN!

Test OR expression	

program testOr;
var x, y : int;
main(){
x = 1;
y = 2;
if(x == 1 or y <> 2){
print("yes!");
}
}
	Output
	"yes!"
	FIN!

Compile-time Memory Administration Description

In compile time, we use a very efficient data structure => dictionaries.
We also use stacks, queues and lists for keeping the instructions in the correct places.
We decided on using a dictionary as our main data structure to save the information in memory, because it is very efficient O(1) for searches and insertion (best case) and worst case is O(N), and because you can have multiple variables stored regarding its type
operatorStack: Stack that manages which operator is going first by its precedence
e.g 4 * 3 - 2 → Stack | - *
typeStack: Stack that manages the type to be assigned.
e.g. ‘int’, ‘float’

jumpStack: Stack that manages to which quadruple needs to jump used in conditions, loops and function calls.
e.g, a valid range number of quadruple
Quadruples: list containing the quadruples objects
e.g [{ operator, leftOperand, rightOperand, result}]
Function to get addresses during compile time.
varIds: Queue dequeued after all the variables are declared.
We used a dequeue because its more efficient

We use three different dictionaries to handle memory during compilation:

Variable Table

The variable table is a dictionary that contains the following structure:

```
{
	“Scope”: {
		“intVarName”: {
			“Type”: “int”,
			“IsArray”: False,
			“Address”: 100
		},
		“floatArrName”: {
			“Type”: “float”,
			“isArray”: True,
			“Address”: 1001,
			“Dim”: 1,	// One or two dimensions (Array or matrix)
			“Size”: 4	// Total size in memory
			“LIM”: 4	// Limit of first dimension.
		},
		“charMatrixName”: {
			“Type”: “char”,
			“isArray”: True”,
			“Address”: 1005,
			“Dim”: 2”,	// One or two dimensions (Array or matrix)
			“Size”: 10,	// Total size in memory
			“LIM”: 5,	// Limit of first dimension.
			“LIM2”: 2,	// Limit of second dimension.
		}
	}
}
```

Function directory

```
{
	'global': { // Main function
		'type': 'void',		// Function type
		'quadIndex': 5		 // Quadruple index
```

```
        },
        ‘voidFunctionName’: {
                'type': 'void',             // Function type
                'address': 13000,           // Address of global variable.
                'quadIndex': 1              // Index of function’s first quadruple.
        },
        ‘intFunctionName’: {
                'type': 'int',              // Function type
                'address': 1026,            // Address of global variable.
                'quadIndex': 3              // Index of function’s first quadruple.
        }
}
```

Param Table

```
{
        ‘oneParamFunction’: {
                'intVarName': {
                        'type': 'int'
                }
        },
        'NoParamsFunction': {
        }
}
```

Access function example:

Value = variableTable[existingKey]

Setter function example:

variableTable[newKey] = value

If we know the value of the address (constants). We will add them to the execution’s global memory.

```
# [Used in compile time]
# Returns next available address for a given variable type range
# If array offset will be array size
def getNextAddress(typeRange, offset=1, value=None, valType=None):
    # Stores current available address
    current_address = types[typeRange]
    # May pass value and type to store constants from compilation
    if valType != None and value != None:
        # parse value
        if valType == INT:
            value = int(value)
        elif valType == FLOAT:
            value = float(value)
        global_memory.setValue(current_address, value)
    # Updates next available address
    types[typeRange] = types[typeRange] + offset
    # Checks that memory addresses are not overflowed
    if types[typeRange] % 1000 != 0:
        return current_address
    else:
        # StackOverflow error
        raise StackOverflow
```

Virtual Machine Description

The Backend is hosted in Heroku using the buildpack heroku/python and gunicorn for Flask.

The Frontend is hosted in Netlify and we communicate with the server using a simple Bearer Token.

We are using a while loop that iterates over the number of quadruples handling conditions to know what to do for each quadruple operator

Command run in Server-Side to launch our Flask app.
web: gunicorn parser:app

Execution-time Memory Administration Description

For the execution we use a GlobalMemory Class and a LocalMemory Class

In the LocalMemory class we use a memory dictionary, and the parameters of it in a Queue. The local memory is instantiated when needed and discarded when unused. Since in the context of the program only 2 instances of memory can be active, the Local and Global (when a function call exists!), otherwise only one instance of memory is active in the context (The

Global Memory), we opted to create both these classes to manage that change of context and maintain access to the global context.

```
class LocalMemory():
    def __init__(self):
        self.memory = {}
        self.paramInts = Queue()
        self.paramFloats = Queue()
        self.paramChars = Queue()
```

We use a MemoryStack to manage the parameters recursion and we assign the correct parameters using a queue of each of the valid types: integers, floats and chars.

```
class GlobalMemory():
    def __init__(self):
        self.memory = {}
```

For the Global and LocalMemory we use a memory dictionary to manage the memory throughout the program with their respective setters and getters. Since it is a dictionary you can index it in any part but, we are verifying the address is valid before setting the value.

We are doing the validation using the getType() function to get the type of the variable (the ones from global to temporal local variables), and with the help of the function isLocal() we are verifying that the scope is Local. After a successful validation it sets the value in its correct position.

Functions to get the value of an address

```
# Function to check if its a local address
def isLocal(address):
    return (address >= 4000 and address < 7000) or (address >= 14000
                                                    and address <
17000)
# Returns value from current local memory/global memory
def getValue(address):
    # If pointer then get real address first
    if getType(address) == POINTER:
        address = global_memory.getValue(address)
    if isLocal(address):
        return local_memory_stack.top().getValue(address)
    else:
        return global_memory.getValue(address)
```

Class method to get the value of an address

```
# function to get the value of the address
    def getValue(self, address):
```

```
        return self.memory[address]
```

Test of how the dictionary looks for the cyclic factorial:
{'global': {'i': {'type': 'int', 'address': 1000, 'isArray': False}, 'ans': {'type': 'int', 'address': 1001, 'isArray': False}}}
{'global': {'type': 'void', 'quadIndex': 1}}

This is how the memory looks in the virtual machine :
{10000: 1, 10001: 1, 10002: 7, 10003: 1, 1001: 5040, 1000: 8, 7000: 0, 7001: 5040, 7002: 8}

Test of functionCalls memory
<u>In compilation</u>
Variable table:

```
{
        'global': {
                'i': {
                        'type': 'int',
                        'address': 1000,
                        'isArray': True,
                        'dim': 2,
                        'size': 25,
                        'lim': 5,
                        'lim2': 5
                  },
                'j': {
                        'type': 'int',
                        'address': 1025,
                        'isArray': False
                    },
                'intFunction': {
                        'type': 'int',
                        'address': 1026,
                        'isArray': False
                        }
        }
}
```

Function Directory:

```
{
        'global': {
```

```
            'type': 'void',
            'quadIndex':
            5
        },
        'voidFunction': {
            'type': 'void',
            'address': 13000,
            'quadIndex': 1
        },
        'intFunction': {
            'type': 'int',
            'address': 1026,
            'quadIndex': 3
        }
}
```

In execution:

Local

```
local
{4000: 1}
```

Global Memory

```
{
        10000: 1,
        10001: 5,
        10002: 0,
        10003: 1000,
        10004: 1,
        10005: 5,
        10006: 0,
        10007: 1000,
        10008: 2,
        10009: 1,
        1026: 1,
        7000: 1,
        7001: 5,
        7002: 5,
        17000: 1005,
        7003: 1,
        1005: 1,
        7004: 5,
        7005: 5,
```

```
        17001: 1005,
        7006: 1,
        7007: 1,
        1025: 4,
        7008: 1,
        7009: 3,
        7010: 0,
        7011: 4
}
```

Language Functionality Tests

Recursive Factorial

```
program recursiveFactorial;
var factorial, ans: int;

int function fact(m:int)
var i: int;
{
   if(m > 1)
   {
    return(m * fact(m - 1));
   } else
   {
     return(1);
   }
}

main()
{
  ans = 6;
  print(fact(ans));
}
```

Output

```
720
FIN!
```

Cyclic Factorial

```
program cyclicFactorial;
var i, ans: int;
main()
{
  ans = 1;
  for(i=1 : 7)
  {
    ans = ans * i;
  }
  print(ans);
}
```

Output

```
5040
FIN!
```

Recursive Fibonacci

```
program recursiveFibonacci;

int function fibo(num : int){
    if(num< 2){
      return(num);
    } else {
      return( fibo(num - 1) + fibo(num - 2) );
    }
}

main() {
  print(fibo(12));
}
```

Output

```
144
FIN!
```

Cyclic Fibonacci

```
program cyclicFibonacci;
var num, i, num1, num2 :  int;

main()
{
  num = 0;
  num1 = 0;
  num2 = 1;
  for(i=2 : 13)
  {
      num = num1 + num2;
      num1 = num2;
      num2 = num;
  }
  print(num);
}
```

Output

```
233
FIN!
```

*Operations in 2d arrays

```
program operations2dVector;
var vec[2][2], vec2[2][2], res[2][2], i, j,
find : int;


main(){
 vec[0][0] = 1;
 vec[0][1] = 2;
 vec[1][0] = 3;
 vec[1][1] = 4;
 vec2[0][0] = 4;
 vec2[0][1] = 5;
 vec2[1][0] = 6;
 vec2[1][1] = 7;
 find = 3;
 for(i = 0 : 1){
   for(j = 0 : 1){
      res[i][j] = vec[i][j] * vec2[i][j];
   }
```

Output

```
4
10
18
28
FIN!
```

```
 }
for(i = 0 : 1){
   for(j = 0 : 1){
       print(res[i][j]);
    }
 }
}
```

BubbleSort While loops

```
program bubbleSort;
var y[4], i, j, tmp: int;
main(){
y[0] = 4;
y[1] = 3;
y[2] = 2;
y[3] = 1;
i = 0;
j = 0;
while(i <> 4)
{
   j = 0;
   while(j <> 3)
   {
      if(y[j] > y[j+1])
      {
         tmp = y[j];
         y[j] = y[j+1];
         y[j + 1] = tmp;
      }
      j = j + 1;
   }
   i = i + 1;
}
print(y[0]);
print(y[1]);
print(y[2]);
print(y[3]);
}
```

Output

```
1
2
3
4
FIN!
```

BubbleSort For loops

```
program bubbleSort;

var y[4], i, j, tmp : int;
main(){
y[0] = 10;
y[1] = 6;
y[2] = 24;
y[3] = 3;
for(i=0:3)
{
        for(j=0:2)
        {
                if(y[j] > y[j+1])
                {
                        tmp = y[j];
                        y[j] = y[j+1];
                        y[j + 1] = tmp;
        }
        }
}

for(i=0 : 3){
  print(y[i]);
}
}
```

Output

```
3
6
10
24
FIN!
```

Find number in a matrix

```
program findNumber;
var vec[2][2], i, j, find : int;


main(){
 vec[0][0] = 1;
 vec[0][1] = 2;
 vec[1][0] = 3;
 vec[1][1] = 4;
 find = 3;
 for(i = 0 : 1){
   for(j = 0 : 1){
     if(find == vec[i][j]){
       print("found!");
     }
   }
 }
}
```

Output

```
"found!"
FIN!
```

functionCalls

```
program functionCalls;
var i[5][5], j: int;

void function pelos(i: int) {
    print(i);
}

int function pelambres() {
    return (1);
}

main() {
    i[pelambres()][0] = pelambres();
    print(i[1][0]);
    pelos(pelambres());
                        for(j=pelambres():
pelambres()+2) {
        print(j);
    }
```

Output

```
1
1
1
2
3
FIN!
```

}	

Project Files Documentation (Table , ModuleName/Details)

lexer.py	**Purpose:** Makes use of the PLY.LEX library to declare the tokens, reserved, regex for simple tokens, regex for tokens that require code, rule to manage whitespace line jumps, ignore empty spaces and comments, Error handling and lexer constructor **Used In:** parser.py
constants.py	**Purpose:** File to declare all the constant strings in the project to prevent typos **Used in:** datastructures.py memory.py parser.py vm.py
datastructures.py	**Purpose:** Declares the data structures used throughout the project: Arrays, Queues, and Quadruples. **Used In:** memory.py parser.py vm.py
errors.py	**Purpose:** Declares the types errors found throughout compilation and execution time.

	Used In: memory.py parser.py vm.py
memory.py	**Purpose:** Declares the GlobalMemory and LocalMemory classes to manage the compiler's memory and assign addresses in their correct positions. **Used in:** parser.py vm.py
parser.py	**Purpose:** Main document for compilation.In this code we verify that the grammar is correct and that all the tokens are in the right place. Otherwise we would generate a syntax error if the submitted code does not comply with the grammar or an Illegal Character error is thrown if an unknown character is used The Flask app is initialized here, and the quadruples are sent to the vm and the output generated to the backend and receive it in the frontend. **Used In:** lexer.py errors.py datastructures.py memory.py constants.py vm.py **Libraries used:** Flask Flask_cors ply.yacc
Procfile	**Purpose:** The Procfile is needed to run the

	necessary scripts to work in the live environment **Libraries used:** gunicorn **Used In:** Heroku deployment
vm.py	**Purpose:** Receives list of quadruples and optionally the currentQuad and inputValue Will receive the currentQuad and inputValue after vm stops for input response and start function returns list of outputs (PRINTS) **Used In :** parser.py
ui/(React) - for the frontend	**Purpose:** The UI folder contains all the necessary files to run the frontend part of the project. **Libraries used:** React **Used In:** **Netlify deployment**
requirements.txt	**Purpose:** The requirements file is needed to install the necessary libraries and python version to work in the live environment **Libraries used:** Flask==1.1.2 Flask-Compress==1.9.0 flask-core==2.9.0 Flask-Cors==3.0.10 gunicorn==20.1.0 json5==0.9.5 jsonschema==3.2.0 ply==3.11 Werkzeug==1.0.1 **Used In:** Heroku deployment

PyCoffee USER MANUAL

Link to video:
https://www.youtube.com/watch?v=32QlzYnb0NE

Structure of a program

```
program insertCoolName;

main(){

}
```

Declare a variable int/float/char

```
program insertCoolName;
var i : int; x : float; letter : char;
main(){


}
```

Declare comments

```
program insertCoolName;
main(){
%% This is a valid comment
%% The broth is more expensive than the meatballs
}
```

Assign a value to a declared variable

```
program insertCoolName;
var i : int; x : float; letter : char;
main(){
   i = 1;
   x = 1.5;
   letter = 'd';
}
```

Declare an array/matrix and assign values to them

```
program insertCoolName;
var i[2], y[2][2] : int;
main(){
   i[0] = 1;
   i[1] = 2;
   y[0][0] = 3;
   y[0][1] = 4;
   y[1][0] = 5;
   y[1][1] = 6;
}
```

Declare and call functions

```
program insertCoolName;
var i, j: int;

int function intFunction(i: int) {
    i = 4;
    return(i);
}
main() {
     j = intFunction(2);
     print(j);
}
```

Input and print values

```
program insertCoolName;
var i : int;
```

```
main(){
 input(i);
 print(i);
}
```

Use conditions

```
program insertCoolName;
var age : int;


main(){
 input(age);
 if(age >= 18){
    print("my daughter dances with the man");
  } else {
    print("my daughter does not dance with the man");
  }
}
```

Loops

```
program insertCoolName;
var to, it, from : int;


main(){
 input(to);
 input(from);
%%iterator increases by 1 automatically after each loop
 for(it=from:to){
  print(it);
 }
 it = 0;
%%iterator needs to be modified manually after each loop.
 while(it <> to+ 1){
    print(it);
    it = it + 1;
 }
```

```
}
```

Recursive functions

```
program insertCoolName;
var i[5][5], j: int;

void function voidFunction(i: int) {
    print(i);
}

int function intFunction() {
    return (1);
}

main() {
    i[intFunction()][0] = intFunction();
    print(i[1][0]);
    voidFunction(intFunction());
    for(j=intFunction(): intFunction()+2) {
        print(j);
    }
}
```