Tecnológico de Monterrey

Diseño de compiladores

MyRLike

Sergio Eduardo Vega Guzmán
A01194108

Índice

Propósito y Alcance

El presente proyecto tiene como propósito compilar y ejecutar un lenguaje de programación modelado a partir de R. Si bien es cierto que el lenguaje propuesto MyRLike tiene un parentesco considerable con R, su capacidad y rango de acciones disponibles para quien lo utilice no está cerca al nivel de manejo de complejidad que el original ofrece. Este es esencialmente un proyecto de nivel principiante desarrollado individualmente por un candidato a profesional con un gran espacio de mejora.

El lenguaje soporta la realización de operaciones básicas de tipo, suma, resta, división, multiplicación, llamado de funciones, manejo de arreglos de una y dos dimensiones, manejo de estatutos condicionales y estatutos cíclicos.

Al igual que con cualquier otro software de tamaño considerable, todo el set de posibles errores y “exploits” no ha sido descubierto en su totalidad, por lo que es muy probable que tras un determinado tiempo intentando con distintos inputs, el software falle de formas imprevistas.

Aquello que podría ser mejorado en futuras versiones tiene que ver principalmente con el manejo de los arreglos dimensionados, para hacer que puedan ser soportadas dimensiones por encima de R2 y una corrección holística de la gramática, puesto que debido a la construcción iterada sobre las versiones anteriores, muchos de los problemas que iban surgiendo eran solucionados de maneras precarias que terminaban pesando en etapas posteriores, dando como resultado una suerte de “spaghetti code” que necesitaría ser revisado desde sus fundamentos para reestructurarlo adecuadamente.

Análisis de Requerimientos

El programa debe compilar y ejecutar los siguiente:

Manejo de expresiones aritméticas básicas.

Manejo de operadores comparativos.

Manejo de tipos, entero, flotante y string.

Manejo de arreglos de 1 y 2 dimensiones

Manejo de funciones

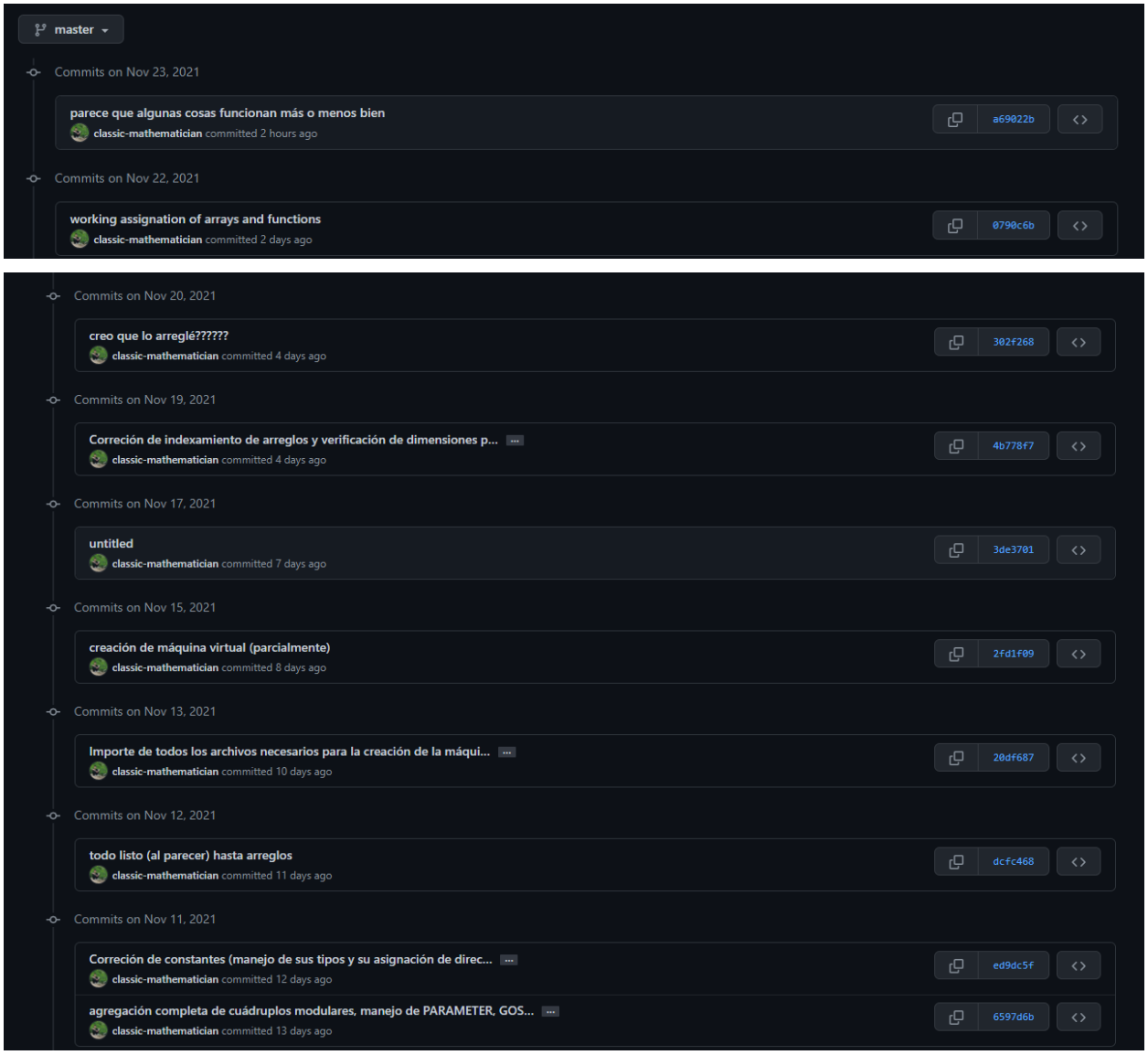
Manejo de estatutos condicionales

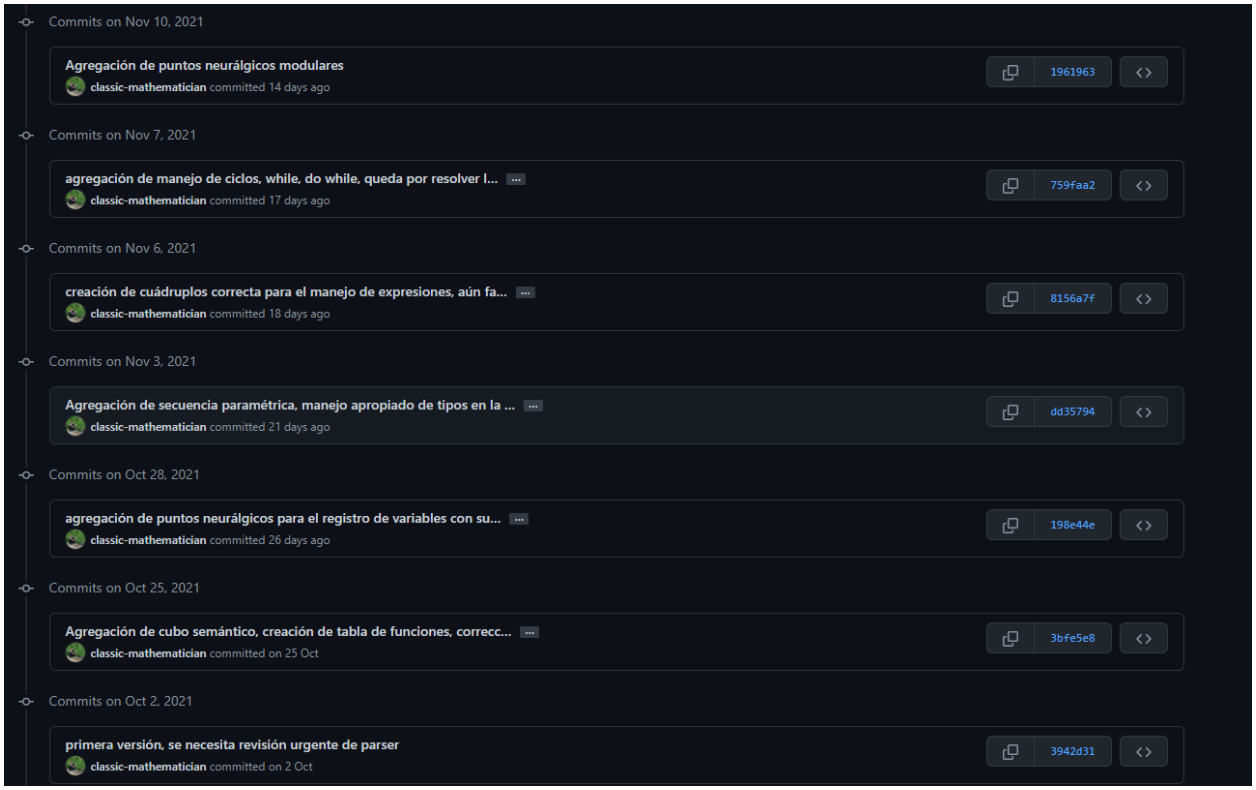
Manejo de estatutos cíclicos

El programa deberá ejecutarse de forma eficiente, sin desperdiciar memoria con estructuras de datos innecesarias.

Descripción del proyecto

A continuación se adjunta la evidencia de la lista de “commits” hechos al repositorio en el que se trabajó el proyecto. La lista está ordenado en forma descendente.




Commits on Nov 10, 2021
Agregación de puntos neurálgicos modulares
classic-mathematician committed 14 days ago
1961963
Commits on Nov 7, 2021
agregación de manejo de ciclos, while, do while, queda por resolver l...
classic-mathematician committed 17 days ago
759faa2
Commits on Nov 6, 2021
creación de cuádruplos correcta para el manejo de expresiones, aún fa...
classic-mathematician committed 18 days ago
8156a7f
Commits on Nov 3, 2021
Agregación de secuencia paramétrica, manejo apropiado de tipos en la ...
classic-mathematician committed 21 days ago
dd35794
Commits on Oct 28, 2021
agregación de puntos neurálgicos para el registro de variables con su...
classic-mathematician committed 26 days ago
198e44e
Commits on Oct 25, 2021
Agregación de cubo semántico, creación de tabla de funciones, correcc...
classic-mathematician committed on 25 Oct
3bfe5e8
Commits on Oct 2, 2021
primera versión, se necesita revisión urgente de parser
classic-mathematician committed on 2 Oct
3942d31

Reflexión individual

Creo que lo más importante que aprendí durante el proyecto es la importancia del desarrollo sólido. Durante el curso fui desarrollando el proyecto como mejor me di a entender, lo cual, aunado a las agregaciones incrementales terminaron por aumentar la dificultad de la tarea en cuestión puesto que ahora no solo tenía que lidiar con las necesidades lógicas que exigía el proyecto, sino que también tenía que hacerlo con las problemáticas ocasionadas por mi precaria arquitectura. El código se llenaba de “hot fixes” que si bien cumplían momentáneamente con la función de sostener pedazos lo que se estaban cayendo, eventualmente pesaban mucho más en etapas posteriores debido a que la ramificación del crecimiento hacía que las partes flojas del desarrollo se propagasen a otras partes y que estas al igual necesitaran arreglos precarios. Era esencialmente lo mismo que enfrentarse con el monstruo mitológico Hydra, cada que cortaba una cabeza, salían dos. Me parece pertinente mencionar los beneficios de trabajar constantemente en una sección específica del código hasta terminar su funcionalidad, pues en más de una ocasión cambiaba de tareas y cuando regresaba días o semanas después, batallaba para recordar e interpretar la lógica con la que había construído aquellas secciones.

El proyecto me enseñó como programador la importancia de cuidar hasta el último pedazo de código por más minúsculo que parezca, ya que de caer en la desidia y relegar la responsabilidad a quien o quienes desarrollen en un futuro, puede resultar de lo más detrimental y contraproducente. No le recomendaría a nadie escupir al aire con una mal praxis que pese diez veces más de lo que habría resultado hacer las cosas bien desde el inicio.

Descripción del lenguaje

Nombre: MyRLikeEdu

El lenguaje soporta el manejo de variables, atómicas y no atómicas, estatutos no lineales como condicionales de tipo if else y estatutos cíclicos como while y do while. Así mismo, este soporta el uso de funciones de tipo entero, flotante y void con la aceptación paramétrica de variables atómicas y constantes. Junto con retornos en cualquier parte de la función.
Las operaciones aceptadas por éste son las siguientes: suma, resta, multiplicación, división, mayor que, menor que, igual a, y, y ó.

Posibles errores

Los errores más comunes que podrían ocurrir son los siguientes:

- Confusión con la firma de función si esta se encuentra dentro de otra.
- Problemas en la asignación de las variables si estas se usan como parámetro de un arreglo.
- Se permite no escribir un return en una variable de tipo no-void así que en caso de que eso pasara, definitivamente rompería con la ejecución.
- Si hay cambios de contexto, podrían ocurrir problemas recursivos

Descripción del compilador

El compilador fue desarrollado en su totalidad en Python y funciona sin necesidad de poseer una arquitectura específica. La gramática fue escrita enteramente usando la herramienta PLY, la cual también se encargó de manejar el lexer y el parser que hicieron posible la fase de compilación.

Lista de tokens aceptados por el lenguaje: ID, PLUS, MINUS, TIMES, DIVIDE, EQUALS, SEMICOLON , INT, FLOAT, STRING, LT, GT, LTE, GTE, DOUBLEEQUAL, AND, OR, LPAREN, RPAREN, LBRACE, RBRACE, BLOCKSTART, BLOCKEND, COLON, COMMA y COMMENT.

Las expresiones regulares (con su código asociado) que se utilizaron fueron las siguientes:

```
t_PLUS = r'\+'
t_MINUS   = r'-'
t_TIMES = r'\*'
t_DIVIDE  = r'/'
t_LPAREN  = r'\('
t_RPAREN  = r'\)'
t_LBRACE = r'\['
t_RBRACE = r'\]'
t_BLOCKSTART = r'\{'
t_BLOCKEND = r'\}'
t_EQUALS = r'\='
t_SEMICOLON = r'\;'
t_GT = r'\>'
t_LT = r'\<'
t_LTE = r'\<\='
t_GTE = r'\>\='
t_DOUBLEEQUAL = r'\=\='
t_AND = r'\&\&'
t_OR = r'\|\|'
```

```
t_COMMENT = r'\%\%.*'
t_ignore  = ' \t'
t_COLON = r'\:'
t_COMMA = R'\,'
```

Código de operación para la generación de cuádruplos

El concepto abstracto de la memoria que diseñé es el siguiente:

global variables 1000-9999
- 1000-2999 int
- 3000- 5999 float
- 6000 - 9999 strings

local variables 10,000 - 19,9999
- 10,000 - 12,999 int
- 13,000 - 15,999 float
- 16,000 - 19,999 string

temporal variables 20,000 - 29,999
- 20,000 - 22,999 int
- 23,000 - 25999 float
- 26,000 - 29999 stringe

constantes 30,000 - 34,999
- 30,000 - 32,999 int
- 33,000 - 35999 float
- 36,000 - 39999 stringe

Tabla de consideraciones semánticas (cubo semántico)

Forma de interpretarse:

Para “tipo”,
“accion” : “tipo” produce: “tipo”

Para enteros

#operadores
"+" : "int" : "int", "float" : "float", "string" : "e",
"-" : "int" : "int", "float" : "float", "string" : "e",
"*" : "int" : "int", "float" : "float", "string" : "e",
"/" : "int" : "float", "float" : "float", "string" : "e",

#assignación
"=" : "int" : "int", "float" : "e", "string" : "e",

#comparaciones
"==" : "int" : "int", "float" : "e", "string" : "e" ,
">" : "int" : "int", "float" : "int", "string" : "e",
"<" : "int" : "int", "float" : "int", "string" : "e",
">=" : "int" : "int", "float" : "int", "string" : "e",
"<=" : "int" : "int", "float" : "int", "string" : "e",
"&&" : "int" : "int", "float" : "int", "string" : "e",
"||" : "int" : "int", "float" : "int", "string" : "e",

Para flotantes

#operadores
"+" : "int" : "float", "float" : "float", "string" : "e",
"-" : "int" : "float", "float" : "float", "string" : "e",
"*" : "int" : "float", "float" : "float", "string" : "e",
"/" : "int" : "float", "float" : "float", "string" : "e",

#asignación
"=" : "int" : "float", "float" : "float", "string" : "e" ,

#comparación
"==" : "int" : "e", "float" : "int", "string" : "e",
">" : "int" : "int", "float" : "int", "string" : "e",
"<" : "int" : "int", "float" : "int", "string" : "e",
">=" : "int" : "int", "float" : "int", "string" : "e",
"<=" : "int" : "int", "float" : "int", "string" : "e",
"&&" : "int" : "int", "float" : "int", "string" : "e" ,
"|" : "int" : "int", "float" : "int", "string" : "e" ,

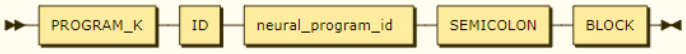
Para strings

#asignación
"=" : "int" : "e", "float" : "e", "string" : "string" ,

#comparadores
"==" : "int" : "e", "float" : "e", "string" : "int",
">" : "int" : "e", "float" : "e", "string" : "int",
"<" : "int" : "e", "float" : "e", "string" : "int",
">=" : "int" : "e", "float" : "e", "string" : "int",
"<=" : "int" : "e", "float" : "e", "string" : "int",
"&&": "int" : "int", "float" : "e", "string" : "int",
"|" : "int" : "int", "float" : "e", "string" : "int",

Diagramas de sintaxis

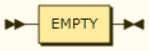
PROGRAM:




```
PROGRAM  ::= PROGRAM_K ID neural_program_id SEMICOLON BLOCK
```

no references

neural_program_id:




```
neural_program_id
        ::= EMPTY
```

referenced by:

- PROGRAM

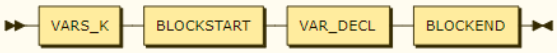
BLOCK:


```
BLOCK    ::= VAR_BLOCK PROC_BLOCK PRINCIPAL_BLOCK
```

referenced by:

- PROGRAM

VAR_BLOCK:

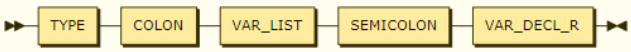



```
VAR_BLOCK
        ::= VARS_K BLOCKSTART VAR_DECL BLOCKEND
```

referenced by:

- BLOCK

VAR_DECL:

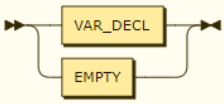



```
VAR_DECL ::= TYPE COLON VAR_LIST SEMICOLON VAR_DECL_R
```

referenced by:

- VAR_BLOCK
- VAR_DECL_R

VAR_DECL_R:

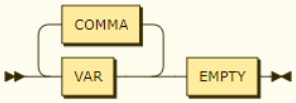



```
VAR_DECL_R
        ::= VAR_DECL
          | EMPTY
```

referenced by:

- VAR_DECL

VAR_LIST:

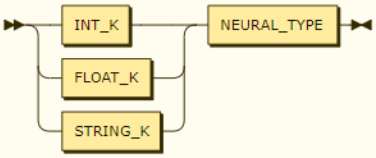



```
VAR_LIST ::= VAR ( COMMA VAR )* EMPTY
```

referenced by:

- VAR_DECL

TYPE:




```
TYPE    ::= ( INT_K | FLOAT_K | STRING_K ) NEURAL_TYPE
```

referenced by:

- LS_VARDECL
- PARAM_DECL
- PROC_DECL_RETURN
- VAR_DECL

NEURAL_TYPE:


```
NEURAL_TYPE
        ::= EMPTY
```

referenced by:

- TYPE

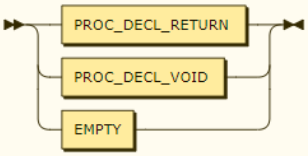
PROC_BLOCK:


```
PROC_BLOCK
        ::= PROC_DECL
```

referenced by:

- BLOCK

PROC_DECL:

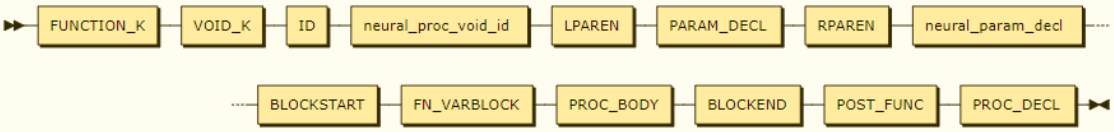



```
PROC_DECL
        ::= PROC_DECL_RETURN
          | PROC_DECL_VOID
          | EMPTY
```

referenced by:

- PROC_BLOCK
- PROC_DECL_RETURN
- PROC_DECL_VOID

PROC_DECL_VOID:

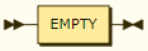



```
PROC_DECL_VOID
        ::= FUNCTION_K VOID_K ID neural_proc_void_id LPAREN PARAM_DECL RPAREN neural_param_decl BLOCKSTART FN_VARBLOCK PROC_BODY BLOCKEND POST_FUNC
            PROC_DECL
```

referenced by:

- PROC_DECL

neural_proc_void_id:

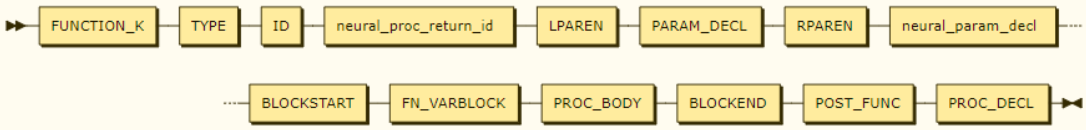



```
neural_proc_void_id
        ::= EMPTY
```

referenced by:

- PROC_DECL_VOID

PROC_DECL_RETURN:




```
PROC_DECL_RETURN
        ::= FUNCTION_K TYPE ID neural_proc_return_id LPAREN PARAM_DECL RPAREN neural_param_decl BLOCKSTART FN_VARBLOCK PROC_BODY BLOCKEND POST_FUNC
            PROC_DECL
```

referenced by:

- PROC_DECL

POST_FUNC:


```
POST_FUNC
        ::= EMPTY
```

referenced by:

- PROC_DECL_RETURN
- PROC_DECL_VOID

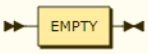
neural_proc_return_id:


```
neural_proc_return_id
        ::= EMPTY
```

referenced by:

- PROC_DECL_RETURN

neural_param_decl:

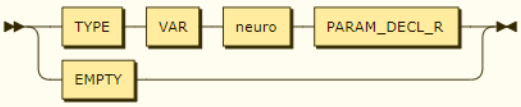



```
neural_param_decl
        ::= EMPTY
```

referenced by:

- PROC_DECL_RETURN
- PROC_DECL_VOID

PARAM_DECL:

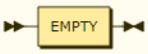



```
PARAM_DECL
        ::= TYPE VAR neuro PARAM_DECL_R
          | EMPTY
```

referenced by:

- PARAM_DECL_R
- PROC_DECL_RETURN
- PROC_DECL_VOID

neuro:

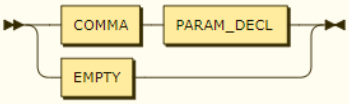



```
neuro    ::= EMPTY
```

referenced by:

- PARAM_DECL

PARAM_DECL_R:

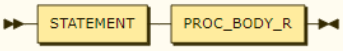



```
PARAM_DECL_R
        ::= COMMA PARAM_DECL
          | EMPTY
```

referenced by:

- PARAM_DECL

PROC_BODY:

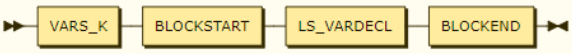



```
PROC_BODY
        ::= STATEMENT PROC_BODY_R
```

referenced by:

- PROC_BODY_R
- PROC_DECL_RETURN
- PROC_DECL_VOID

FN_VARBLOCK:

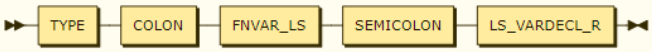



```
FN_VARBLOCK
        ::= VARS_K BLOCKSTART LS_VARDECL BLOCKEND
```

referenced by:

- PROC_DECL_RETURN
- PROC_DECL_VOID

LS_VARDECL:

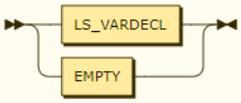



```
LS_VARDECL
        ::= TYPE COLON FNVAR_LS SEMICOLON LS_VARDECL_R
```

referenced by:

- FN_VARBLOCK
- LS_VARDECL_R

LS_VARDECL_R:

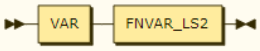



```
LS_VARDECL_R
         ::= LS_VARDECL
           | EMPTY
```

referenced by:

- LS_VARDECL

FNVAR_LS:

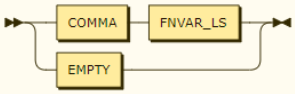



```
FNVAR_LS ::= VAR FNVAR_LS2
```

referenced by:

- FNVAR_LS2
- LS_VARDECL

FNVAR_LS2:

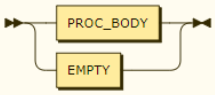



```
FNVAR_LS2
          ::= COMMA FNVAR_LS
            | EMPTY
```

referenced by:

- FNVAR_LS

PROC_BODY_R:

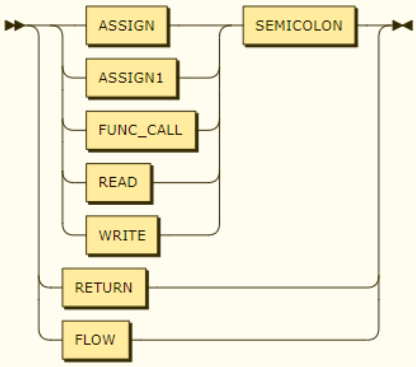



```
PROC_BODY_R
          ::= PROC_BODY
            | EMPTY
```

referenced by:

- PROC_BODY

STATEMENT:

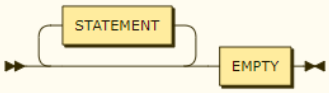



```
STATEMENT
        ::= ( ASSIGN | ASSIGN1 | FUNC_CALL | READ | WRITE ) SEMICOLON
          | RETURN
          | FLOW
```

referenced by:

- PRINCIPAL_BODY
- PROC_BODY
- STATEMENT_R

STATEMENT_R:

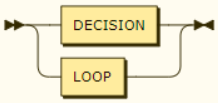



```
STATEMENT_R
        ::= STATEMENT* EMPTY
```

referenced by:

- DECISION
- DO_WHILE_LOOP
- ELSE
- FOR_LOOP
- WHILE_LOOP

FLOW:

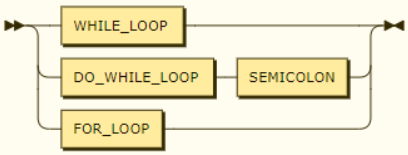



```
FLOW     ::= DECISION
           | LOOP
```

referenced by:

- STATEMENT

LOOP:

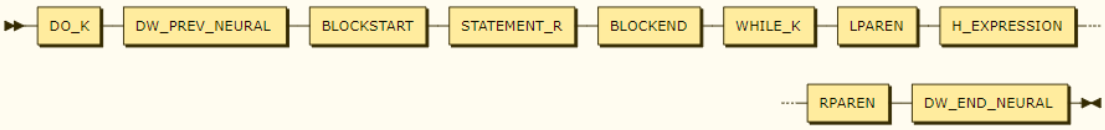



```
LOOP     ::= WHILE_LOOP
           | DO_WHILE_LOOP SEMICOLON
           | FOR_LOOP
```

referenced by:

- FLOW

DO_WHILE_LOOP:




```
DO_WHILE_LOOP
         ::= DO_K DW_PREV_NEURAL BLOCKSTART STATEMENT_R BLOCKEND WHILE_K LPAREN H_EXPRESSION RPAREN DW_END_NEURAL
```

referenced by:

- LOOP

DW_PREV_NEURAL:


```
DW_PREV_NEURAL
         ::= EMPTY
```

referenced by:

- DO_WHILE_LOOP

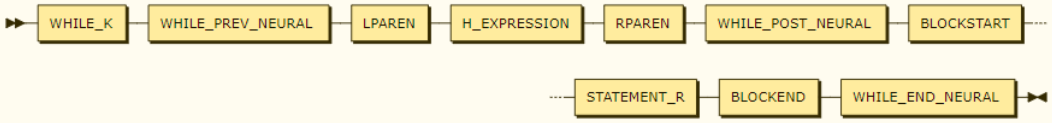
DW_END_NEURAL:


```
DW_END_NEURAL
         ::= EMPTY
```

referenced by:

- DO_WHILE_LOOP

WHILE_LOOP:




```
WHILE_LOOP
        ::= WHILE_K WHILE_PREV_NEURAL LPAREN H_EXPRESSION RPAREN WHILE_POST_NEURAL BLOCKSTART STATEMENT_R BLOCKEND WHILE_END_NEURAL
```

referenced by:

- LOOP

WHILE_PREV_NEURAL:


```
WHILE_PREV_NEURAL
        ::= EMPTY
```

referenced by:

- WHILE LOOP

WHILE_POST_NEURAL:


```
WHILE_POST_NEURAL
        ::= EMPTY
```

referenced by:

- WHILE LOOP

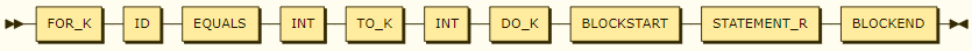
WHILE_END_NEURAL:


```
WHILE_END_NEURAL
        ::= EMPTY
```

referenced by:

- WHILE LOOP

FOR_LOOP:

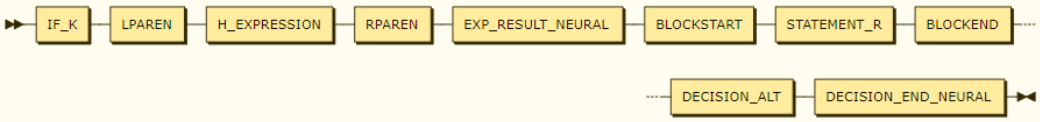



```
FOR_LOOP ::= FOR_K ID EQUALS INT TO_K INT DO_K BLOCKSTART STATEMENT_R BLOCKEND
```

referenced by:

- LOOP

DECISION:




```
DECISION ::= IF_K LPAREN H_EXPRESSION RPAREN EXP_RESULT_NEURAL BLOCKSTART STATEMENT_R BLOCKEND DECISION_ALT DECISION_END_NEURAL
```

referenced by:

- FLOW

EXP_RESULT_NEURAL:


```
EXP_RESULT_NEURAL
        ::= EMPTY
```

referenced by:

- DECISION

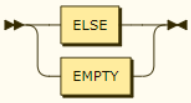
DECISION_END_NEURAL:


```
DECISION_END_NEURAL
        ::= EMPTY
```

referenced by:

- DECISION

DECISION_ALT:




```
DECISION_ALT
        ::= ELSE
          | EMPTY
```

referenced by:

- DECISION

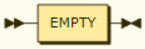
ELSE:


```
ELSE     ::= ELSE_NEURAL ELSE_K BLOCKSTART STATEMENT_R BLOCKEND
```

referenced by:

- DECISION_ALT

ELSE_NEURAL:

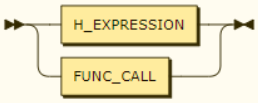



```
ELSE_NEURAL
         ::= EMPTY
```

referenced by:

- ELSE

RIGHT_ASSIGN:

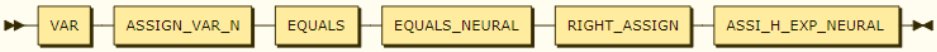



```
RIGHT_ASSIGN
         ::= H_EXPRESSION
           | FUNC_CALL
```

referenced by:

- ASSIGN
- ASSIGN1
- WRITE_LIST

ASSIGN:

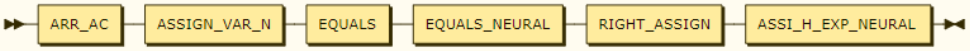



```
ASSIGN   ::= VAR ASSIGN_VAR_N EQUALS EQUALS_NEURAL RIGHT_ASSIGN ASSI_H_EXP_NEURAL
```

referenced by:

- STATEMENT

ASSIGN1:




```
ASSIGN1  ::= ARR_AC ASSIGN_VAR_N EQUALS EQUALS_NEURAL RIGHT_ASSIGN ASSI_H_EXP_NEURAL
```

referenced by:

- STATEMENT

N:


```
N        ::= EMPTY
```

no references

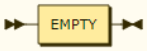
ASSI_H_EXP_NEURAL:


```
ASSI_H_EXP_NEURAL
         ::= EMPTY
```

referenced by:

- ASSIGN
- ASSIGN1

EQUALS_NEURAL:

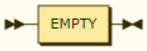



```
EQUALS_NEURAL
        ::= EMPTY
```

referenced by:

- ASSIGN
- ASSIGN1

ASSIGN_VAR_N:

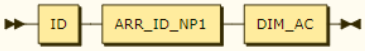



```
ASSIGN_VAR_N
        ::= EMPTY
```

referenced by:

- ASSIGN
- ASSIGN1

ARR_AC:




```
ARR_AC   ::= ID ARR_ID_NP1 DIM_AC
```

referenced by:

- ASSIGN1

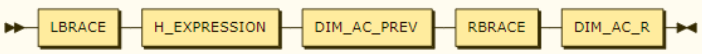
ARR_ID_NP1:


```
ARR_ID_NP1
        ::= EMPTY
```

referenced by:

- ARR_AC

DIM_AC:

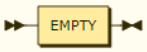



```
DIM_AC   ::= LBRACE H_EXPRESSION DIM_AC_PREV RBRACE DIM_AC_R
```

referenced by:

- ARR_AC
- DIM_AC_R

DIM_AC_PREV:

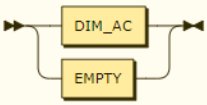



```
DIM_AC_PREV
        ::= EMPTY
```

referenced by:

- DIM_AC

DIM_AC_R:

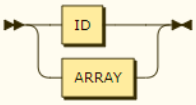



```
DIM_AC_R ::= DIM_AC
           | EMPTY
```

referenced by:

- DIM_AC

VAR:

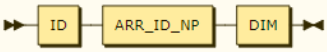



```
VAR      ::= ID
           | ARRAY
```

referenced by:

- ASSIGN
- FNVAR_LS
- PARAM_DECL
- VAR_LIST

ARRAY:




```
ARRAY    ::= ID ARR_ID_NP DIM
```

referenced by:

- VAR

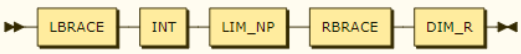
ARR_ID_NP:


```
ARR_ID_NP
        ::= EMPTY
```

referenced by:

- ARRAY

DIM:

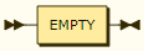



```
DIM     ::= LBRACE INT LIM_NP RBRACE DIM_R
```

referenced by:

- ARRAY
- DIM_R

LIM_NP:

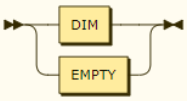



```
LIM_NP   ::= EMPTY
```

referenced by:

- DIM

DIM_R:

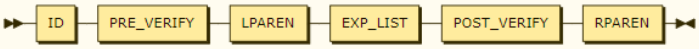



```
DIM_R    ::= DIM
           | EMPTY
```

referenced by:

- DIM

FUNC_CALL:




```
FUNC_CALL
        ::= ID PRE_VERIFY LPAREN EXP_LIST POST_VERIFY RPAREN
```

referenced by:

- RIGHT_ASSIGN
- STATEMENT

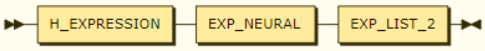
POST_VERIFY:


```
POST_VERIFY
        ::= EMPTY
```

referenced by:

- FUNC_CALL

EXP_LIST:




```
EXP_LIST ::= H_EXPRESSION EXP_NEURAL EXP_LIST_2
```

referenced by:

- EXP_LIST_2
- FUNC_CALL

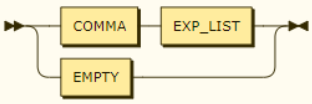
EXP_NEURAL:


```
EXP_NEURAL
        ::= EMPTY
```

referenced by:

- EXP_LIST

EXP_LIST_2:

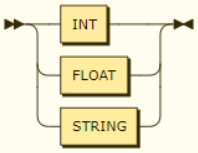



```
EXP_LIST_2
        ::= COMMA EXP_LIST
          | EMPTY
```

referenced by:

- EXP_LIST

CONSTANT:

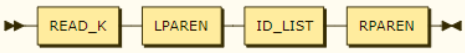



```
CONSTANT ::= INT
           | FLOAT
           | STRING
```

referenced by:

- FACTOR
- WRITE_LIST

READ:

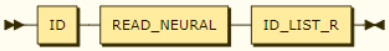



```
READ     ::= READ_K LPAREN ID_LIST RPAREN
```

referenced by:

- STATEMENT

ID_LIST:

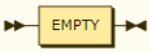



```
ID_LIST  ::= ID READ_NEURAL ID_LIST_R
```

referenced by:

- ID_LIST_R
- READ

READ_NEURAL:

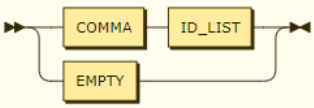



```
READ_NEURAL
        ::= EMPTY
```

referenced by:

- ID_LIST

ID_LIST_R:

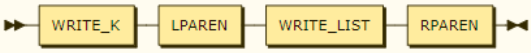



```
ID_LIST_R
        ::= COMMA ID_LIST
          | EMPTY
```

referenced by:

- ID_LIST

WRITE:

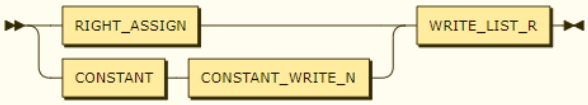



```
WRITE    ::= WRITE_K LPAREN WRITE_LIST RPAREN
```

referenced by:

- STATEMENT

WRITE_LIST:




```
WRITE_LIST
        ::= ( RIGHT_ASSIGN | CONSTANT CONSTANT_WRITE_N ) WRITE_LIST_R
```

referenced by:

- WRITE
- WRITE_LIST_R

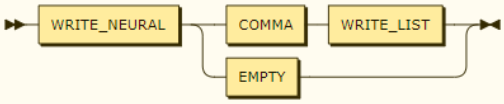
CONSTANT_WRITE_N:


```
CONSTANT_WRITE_N
        ::= EMPTY
```

referenced by:

- WRITE_LIST

WRITE_LIST_R:




```
WRITE_LIST_R
        ::= WRITE_NEURAL ( COMMA WRITE_LIST | EMPTY )
```

referenced by:

- WRITE_LIST

WRITE_NEURAL:


```
WRITE_NEURAL
        ::= EMPTY
```

referenced by:

- WRITE_LIST_R

RETURN:


```
RETURN   ::= RETURN_K LPAREN H_EXPRESSION RPAREN SEMICOLON
```

referenced by:

- STATEMENT

EXPRESSION:


```
EXPRESSION
        ::= TERM NEURAL_EXPRESSION EXPRESSION_R
```

referenced by:

- EXPRESSION_R
- S_EXPRESSION
- S_EXPRESSION_R

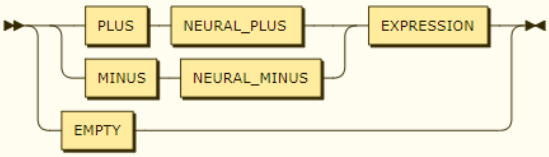
NEURAL_EXPRESSION:


```
NEURAL_EXPRESSION
        ::= EMPTY
```

referenced by:

- EXPRESSION

EXPRESSION_R:




```
EXPRESSION_R
        ::= ( PLUS NEURAL_PLUS | MINUS NEURAL_MINUS ) EXPRESSION
          | EMPTY
```

referenced by:

- EXPRESSION

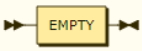
NEURAL_PLUS:


```
NEURAL_PLUS
        ::= EMPTY
```

referenced by:

- EXPRESSION_R

NEURAL_MINUS:

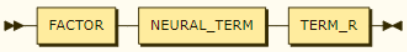



```
NEURAL_MINUS
        ::= EMPTY
```

referenced by:

- EXPRESSION_R

TERM:




```
TERM    ::= FACTOR NEURAL_TERM TERM_R
```

referenced by:

- EXPRESSION
- TERM_R

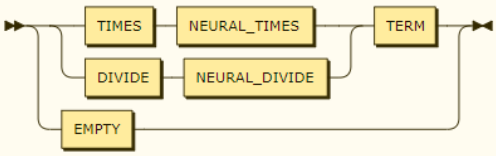
NEURAL_TERM:


```
NEURAL_TERM
        ::= EMPTY
```

referenced by:

- TERM

TERM_R:




```
TERM_R   ::= ( TIMES NEURAL_TIMES | DIVIDE NEURAL_DIVIDE ) TERM
           | EMPTY
```

referenced by:

- TERM

NEURAL_TIMES:


```
NEURAL_TIMES
        ::= EMPTY
```

referenced by:

- TERM_R

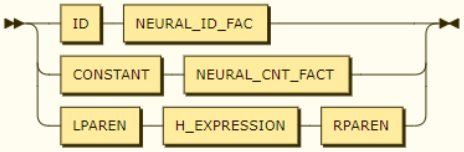
NEURAL_DIVIDE:


```
NEURAL_DIVIDE
        ::= EMPTY
```

referenced by:

- TERM_R

FACTOR:




```
FACTOR   ::= ID NEURAL_ID_FAC
           | CONSTANT NEURAL_CNT_FACT
           | LPAREN H_EXPRESSION RPAREN
```

referenced by:

- TERM

NEURAL_ID_FAC:


```
NEURAL_ID_FAC
         ::= EMPTY
```

referenced by:

- FACTOR

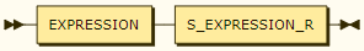
NEURAL_CNT_FACT:


```
NEURAL_CNT_FACT
         ::= EMPTY
```

referenced by:

- FACTOR

S_EXPRESSION:




```
S_EXPRESSION
        ::= EXPRESSION S_EXPRESSION_R
```

referenced by:

- H_EXPRESSION

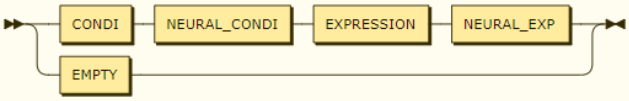
NEURAL_EXP:


```
NEURAL_EXP
        ::= EMPTY
```

referenced by:

- S_EXPRESSION_R

S_EXPRESSION_R:

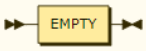



```
S_EXPRESSION_R
        ::= CONDI NEURAL_CONDI EXPRESSION NEURAL_EXP
          | EMPTY
```

referenced by:

- S_EXPRESSION

NEURAL_CONDI:

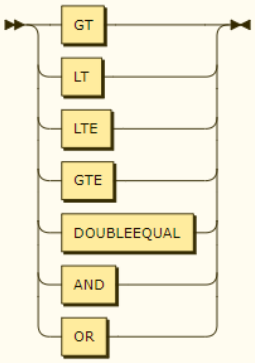



```
NEURAL_CONDI
        ::= EMPTY
```

referenced by:

- S_EXPRESSION_R

CONDI:

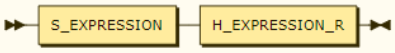



```
CONDI    ::= GT
           | LT
           | LTE
           | GTE
           | DOUBLEEQUAL
           | AND
           | OR
```

referenced by:

- S_EXPRESSION_R

H_EXPRESSION:

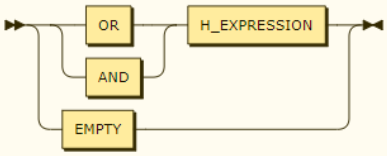



```
H_EXPRESSION
        ::= S_EXPRESSION H_EXPRESSION_R
```

referenced by:

- DECISION
- DIM_AC
- DO_WHILE_LOOP
- EXP_LIST
- FACTOR
- H_EXPRESSION_R
- RETURN
- RIGHT_ASSIGN
- WHILE_LOOP

H_EXPRESSION_R:

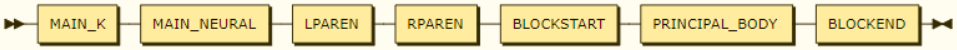



```
H_EXPRESSION_R
         ::= ( OR | AND ) H_EXPRESSION
           | EMPTY
```

referenced by:

- H_EXPRESSION

PRINCIPAL_BLOCK:




```
PRINCIPAL_BLOCK
        ::= MAIN_K MAIN_NEURAL LPAREN RPAREN BLOCKSTART PRINCIPAL_BODY BLOCKEND
```

referenced by:

- BLOCK

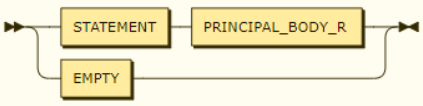
MAIN_NEURAL:


```
MAIN_NEURAL
        ::= EMPTY
```

referenced by:

- PRINCIPAL_BLOCK

PRINCIPAL_BODY:

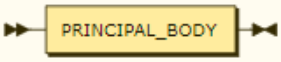



```
PRINCIPAL_BODY
         ::= STATEMENT PRINCIPAL_BODY_R
           | EMPTY
```

referenced by:

- PRINCIPAL_BLOCK
- PRINCIPAL_BODY_R

PRINCIPAL_BODY_R:




```
PRINCIPAL_BODY_R
        ::= PRINCIPAL_BODY
```

referenced by:

- PRINCIPAL_BODY

EMPTY:

```
EMPTY ::=
```

referenced by:

- ARR_ID_NP
- ARR_ID_NP1
- ASSIGN_VAR_N
- ASSI_H_EXP_NEURAL
- CONSTANT_WRITE_N
- DECISION_ALT
- DECISION_END_NEURAL
- DIM_AC_PREV
- DIM_AC_R
- DIM_R
- DW_END_NEURAL
- DW_PREV_NEURAL
- ELSE_NEURAL
- EQUALS_NEURAL
- EXPRESSION_R
- EXP_LIST_2
- EXP_NEURAL
- EXP_RESULT_NEURAL
- FNVAR_LS2
- H_EXPRESSION_R
- ID_LIST_R
- LIM_NP
- LS_VARDECL_R
- MAIN_NEURAL
- N
- NEURAL_CNT_FACT
- NEURAL_CONDI
- NEURAL_DIVIDE
- NEURAL_EXP
- NEURAL_EXPRESSION
- NEURAL_ID_FAC
- NEURAL_MINUS
- NEURAL_PLUS
- NEURAL_TERM
- NEURAL_TIMES
- NEURAL_TYPE
- O
- PARAM_DECL
- PARAM_DECL_R
- POST_FUNC
- POST_VERIFY
- PRE_VERIFY

Administración de memoria

La administración de la memoria fue realizada principalmente por un objeto de nombre VirtualMemory el cual se encargaba de la repartición de direcciones virtuales para una futura asignación “realista” dentro de la máquina virtual. Este objeto contaba con 4 funciones para realizar todas las operaciones.

```
class VirtualMemory(object):

    def __init__(self):
        self.global_variables = [1000, 3000, 6000]
        self.local_variables = [10000, 13000, 16000]
        self.temporal_variables = [20000, 23000, 26000]
        self.constants = [30000, 33000, 36000]
```

El constructor sin parámetros inicializaba la memoria con los rangos de direcciones virtuales que podían ser asignados dependiendo la exigencia.

```
def reset(self):
    self.global_variables = [1000, 3000, 6000]
    self.local_variables = [10000, 13000, 16000]
    self.temporal_variables = [20000, 23000, 26000]
```

Una función de reseteo que era invocada cada que se cambiaba de scope local y poder asignar nuevamente espacios de memoria virtual antes “utilizados” en otros niveles de la compilación.

```
def add_arr(self, scope, type, size):
    if (scope == 'g_scope'):
        if (type == 'int'):
            virtual_address = self.global_variables[0]
            self.global_variables[0] += size
            return virtual_address

        elif (type == 'float'):
            virtual_address = self.global_variables[1]
            self.global_variables[1] += size
            return virtual_address

        else:
            virtual_address = self.global_variables[2]
            self.global_variables[2] += size
            return virtual_address
```

Una función extensa (fragmento mostrado en fotografía) utilizada para la asignación de memoria a arreglos para no asignar las direcciones aledañas dadas en función de su tamaño.

```
def add(self, scope, type):
    if (scope == 'g_scope'):
        if (type == 'int'):
            virtual_address = self.global_variables[0]
            self.global_variables[0] += 1
            return virtual_address

        elif (type == 'float'):
            virtual_address = self.global_variables[1]
            self.global_variables[1] += 1
            return virtual_address

        else:
            virtual_address = self.global_variables[2]
            self.global_variables[2] += 1
            return virtual_address

    if (scope == 'l_scope'):
        if (type == 'int'):
            virtual_address = self.local_variables[0]
            self.local_variables[0] += 1
            return virtual_address
```

Y por último la piedra angular de la asignación, función que regresa el valor de memoria correspondiente dado un scope y tipo, “reservando” ese espacio volviendolo inaccesible para cualquier otra llamada dentro del mismo scope.

El directorio de funciones es un objeto que tiene la siguiente estructura:

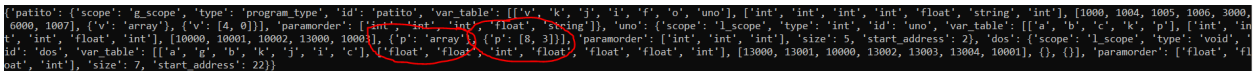
```
class FunctionDirectory(object):

    def __init__(self):
        self.functions = {}
        self.program_name = "program"


    def declare_function(self, id, type, scope):
        self.functions[id] = {"scope" : scope, "type" : type, "id" : id}
        self.functions[id]['var_table'] = [[],[],[],{}, {}]
        self.functions[id]['paramorder'] = []


    def add_var_table(self, id, var_table):
        self.functions[id]["var_table"]
```

Este consta principalmente de dos atributos, un diccionario de funciones y el nombre del programa (para referenciar el scope global).

Dentro del diccionario de funciones, cada función tiene distintos atributos como: scope, tipo y su id. La tabla de variable existe para cada una de las funciones dentro del programa y esta es una tabla de 3 atributos y 2 diccionarios, los primeros 3 atributos existen para delimitar las variables de 3 formas:

[‘var_table’][0] lidia con los nombres informales de las variables ‘v’, ‘x’, ‘p’, etc...
[‘var_table’][1] lidia con los tipos asociados de las variables, int, float y string
[var_table’[2] lidia con las direcciones virtuales asociadas a cada una de las variables.

Visualización del directorio de funciones



La posición 3 del directorio apunta a un diccionario que de lo único que se encarga de hacer es decir si una variable es de tipo atómico o array y por último, la posición 4 de la tabla de variables me indica qué dimensiones tienen los arreglos.

La memoria virtual es de lo más eficiente puesto que esta se asigna conforme se necesita, se utilizó la estructura de datos lista para poder hacerla crecer y seguir asignando direcciones hasta que estas se terminen de acuerdo a la arquitectura que se postuló.

A continuación muestro el objeto utilizado para el manejo de las direcciones virtuales.

```
class Memory(object):
    def __init__(self):
        self.integers = [[],[]]
        self.floats = [[],[]]
        self.strings = [[],[]]
        self.name = "unnamed"


    def print_memory(self):
        print("Memory Map", self.name)
        print("Integer memory: ", self.integers)
        print("Float memory: ", self.floats)
        print("String memory: ", self.strings)
```

Máquina Virtual

La máquina virtual, al igual que el compilador fue desarrollada enteramente en Python. La administración de la memoria en ejecución sigue el principio de un stack de “scopes” para aquellos que varían conforme se cambien los contextos (temporal y local).

```
class VirtualMachine(object):
    def __init__(self):
        self.IP = 1
        self.quads = []
        self.constants = []
        self.global_memory = Memory()
        self.local_memory = []
        self.temporal_memory = [Memory()]
        self.func_dir = None
        self.func_stack = []
        self.first_eq = False
        self.gosub_stack = []
        self.gosub_ip_stack = []
```

Cada que se llega a un nuevo registro de activación ‘ERA’, se agrega una memoria nueva al stack de memoria y cuando se llega al final de la función, se eliminan con un pop para que estos dejen de estar consumiendo recursos.

Cada que el cuádruplo ENDFunc leído por la máquina virtual se expulsan las memorias del stack para poder manejar los scopes.

```
elif action == 'ENDFunc':
    self.local_memory.pop()
    self.temporal_memory.pop()
    self.func_stack.pop()
    self.IP += 1
```

Cuando se recibe una asignación o cualquier otro cuádruplo en el que se referencíen direcciones virtuales se tienen que registrar estas haciendo uso de la función “register”.

```
elif action == '=':
    left_operand = int(quad[1])
    right_operand = int(quad[3])

    self.register(left_operand)
    self.register(right_operand)
```

```
elif action == '+':
    first_operand = int(quad[1])
    second_operand = int(quad[2])
    third_operand = int(quad[3])

    self.register(first_operand)
    self.register(second_operand)
    self.register(third_operand)
```

Cuando se reciben 3 direcciones en operaciones como la suma, la resta, multiplicación, división, etc… (operando 1, operando 2 y dirección del resultado) Simplemente se accede al valor que está dentro de la dirección de los primeros dos y se guarda en la dirección del tercero.

```
elif action == '-':
    first_operand = int(quad[1])
    second_operand = int(quad[2])
    third_operand = int(quad[3])

    self.register(first_operand)
    self.register(second_operand)
    self.register(third_operand)

    result = self.find_in_memory(first_operand) - self.find_in_memory(second_operand)
    self.allocate_in_memory(third_operand, result)
    self.IP += 1
```

La función allocate in memory se dedica a la almacenar en una dirección virtual dada cualquier item que se escriba como segundo parámetro. Dentro del scope en el que se halle la llamada.

La función find se dedica a extraer la información que se halla asociada a la dirección virtual especificada.

Pruebas de funcionamiento del lenguaje

Prueba 1

```
program patito;
vars{

  int : i;
  float : f;
  string : o;

}

function int uno (int a){
  vars {
          float : k;

  }


  return(5  * a);

}

main() {

  i = 2;
  o = "Hello world";

  write(i);


}
```

Resultado de la compilación y producción de cuádruplos.

```
C:\Users\A0119\Desktop\Compiler>new.py
>> Parsing test.txt...
20000
correct syntax
{'patito': {'scope': 'g_scope', 'type': 'program_type', 'id': 'patito', 'var_table': [['i', 'f', 'o', 'uno'], ['int', 'float', 'string', 'int'], [1000, 3000, 6000, 1001], {}, {}], 'paramorder': ['int', 'float',
'string']}, 'uno': {'scope': 'l_scope', 'type': 'int', 'id': 'uno', 'var_table': [['a', 'k'], ['int', 'float'], [10000, 13000], {}, {}], 'paramorder': ['int'], 'size': 2, 'start_address': 2}}
[[5, 2, '"Hello world"'], [30000, 30001, 36000]]
1 ['GOTO_MAIN', '_', '_', 5]
2 ['*', 30000, 10000, 20000]
3 ['RETURN', 20000, '_', '_']
4 ['ENDFunc', '_', '_', '_']
5 ['=', 30001, '_', 1000]
6 ['=', 36000, '_', 6000]
7 ['write', '_', '_', 1000]
8 ['write', '_', '_', 6000]
[[5, 2, '"Hello world"'], [30000, 30001, 36000]]
```

Resultado de la máquina virtual (ejecución).

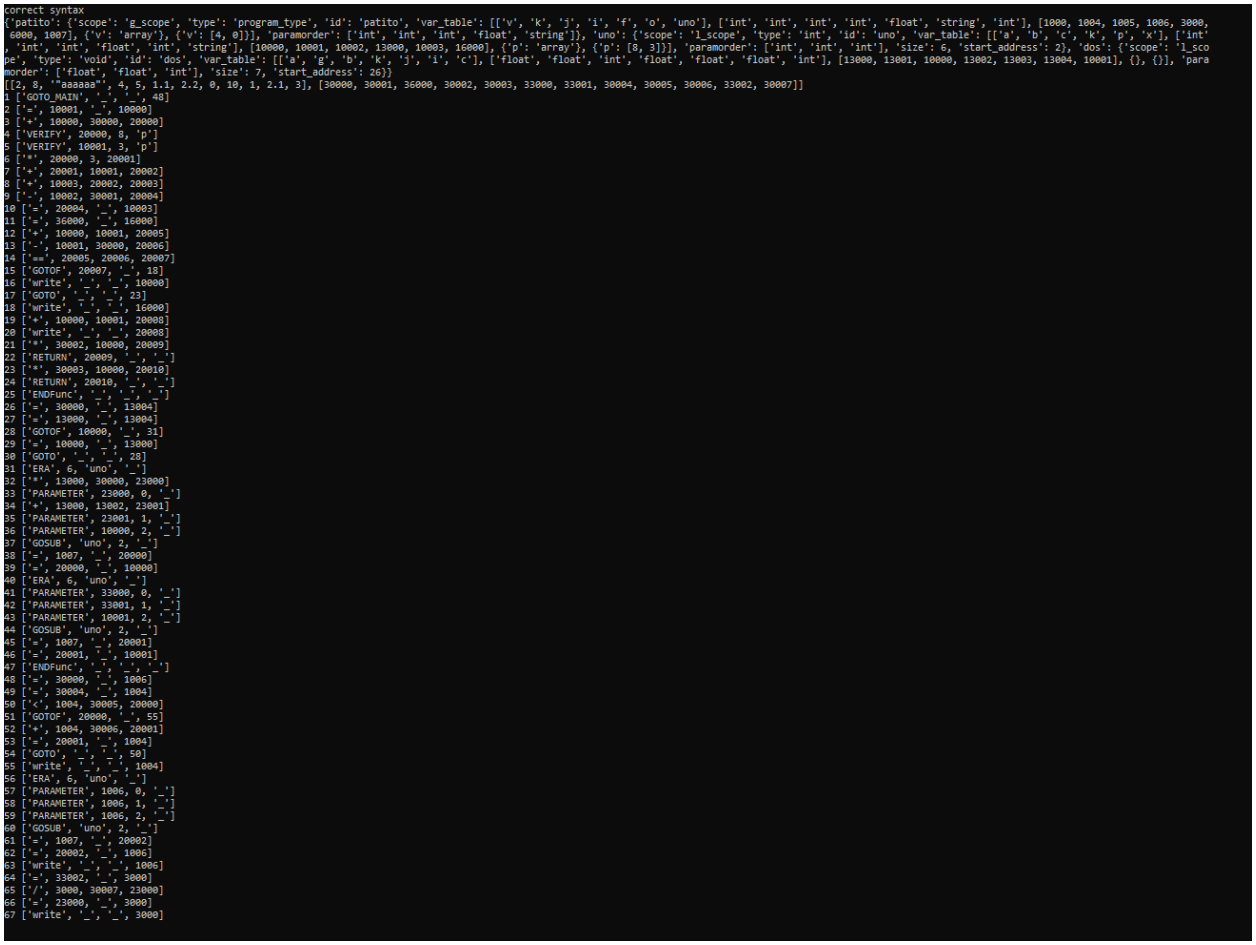
```
C:\Users\A0119\Desktop\Compiler>exec.py
patito
Memory Map global
Integer memory:  [[1000, 1001], [None, None]]
Float memory:  [[3000], [None]]
String memory:  [[6000], [None]]
1 ['GOTO_MAIN', '_', '_', 5]
2 ['*', 30000, 10000, 20000]
3 ['RETURN', 20000, '_', '_']
4 ['ENDFunc', '_', '_', '_']
5 ['=', 30001, '_', 1000]
6 ['=', 36000, '_', 6000]
7 ['write', '_', '_', 1000]
8 ['write', '_', '_', 6000]
[[30000, 30001, 36000], [5, 2, '"Hello world"']]
[[1000, 1001], [None, None]]
{'patito': {'scope': 'g_scope', 'type': 'program_type', 'id': 'patito', 'var_table': [['i', 'f', 'o', 'uno'], ['int', 'float', 'string', 'int'], [1000, 3000, 6000, 1001], {}, {}], 'paramorder': ['int', 'float',
'string']}, 'uno': {'scope': 'l_scope', 'type': 'int', 'id': 'uno', 'var_table': [['a', 'k'], ['int', 'float'], [10000, 13000], {}, {}], 'paramorder': ['int'], 'size': 2, 'start_address': 2}}
1 ['GOTO_MAIN', '_', '_', 5]
5 ['=', 30001, '_', 1000]
6 ['=', 36000, '_', 6000]
7 ['write', '_', '_', 1000]
2
8 ['write', '_', '_', 6000]
"Hello world"
```

Prueba 2

```
program patito;
vars{

  int : i, j, k, v[4];
  float : f;
  string : o;

}

function int uno (int a, int b, int c){
  vars {
        float : k;
        int : p[8][3];
        string : x;

  }
  a = b;
  p[a+2][b] = c - 8;

  x = "aaaaaa";
  if (a + b == b - 2){
        write(a);
  } else {
        write(x);
        write(a + b);
        return(4 * a);
  }



  return(5  * a);



}
```

```
main() {
  i = 2;

  k = 0;
  while(k < 10){
          k = k + 1;
  }
  write(k);

  i = uno(i, i, i);
  write(i);

  f = 2.10;
  f = f / 3;
  write(f);


}
```

Resultado de la compilación:



Resultado de la ejecución

```
51 ['GOTOF', 20000, '_', 55]
52 ['+', 1004, 30006, 20001]
53 ['=', 20001, '_', 1004]
54 ['GOTO', '_', '_', 50]
50 ['<', 1004, 30005, 20000]
51 ['GOTOF', 20000, '_', 55]
52 ['+', 1004, 30006, 20001]
53 ['=', 20001, '_', 1004]
54 ['GOTO', '_', '_', 50]
50 ['<', 1004, 30005, 20000]
51 ['GOTOF', 20000, '_', 55]
52 ['+', 1004, 30006, 20001]
53 ['=', 20001, '_', 1004]
54 ['GOTO', '_', '_', 50]
50 ['<', 1004, 30005, 20000]
51 ['GOTOF', 20000, '_', 55]
52 ['+', 1004, 30006, 20001]
53 ['=', 20001, '_', 1004]
54 ['GOTO', '_', '_', 50]
50 ['<', 1004, 30005, 20000]
51 ['GOTOF', 20000, '_', 55]
52 ['+', 1004, 30006, 20001]
53 ['=', 20001, '_', 1004]
54 ['GOTO', '_', '_', 50]
50 ['<', 1004, 30005, 20000]
51 ['GOTOF', 20000, '_', 55]
52 ['+', 1004, 30006, 20001]
53 ['=', 20001, '_', 1004]
54 ['GOTO', '_', '_', 50]
50 ['<', 1004, 30005, 20000]
51 ['GOTOF', 20000, '_', 55]
55 ['write', '_', '_', 1004]
10
56 ['ERA', 6, 'uno', '_']
57 ['PARAMETER', 1006, 0, '_']
58 ['PARAMETER', 1006, 1, '_']
59 ['PARAMETER', 1006, 2, '_']
60 ['GOSUB', 'uno', 2, '_']
2 ['=', 10001, '_', 10000]
3 ['+', 10000, 30000, 20000]
4 ['VERIFY', 20000, 8, 'p']
5 ['VERIFY', 10001, 3, 'p']
6 ['*', 20000, 3, 20001]
7 ['+', 20001, 10001, 20002]
8 ['+', 10003, 20002, 20003]
9 ['-', 10002, 30001, 20004]
10 ['=', 20004, '_', 10003]
11 ['=', 36000, '_', 16000]
12 ['+', 10000, 10001, 20005]
13 ['-', 10001, 30000, 20006]
14 ['==', 20005, 20006, 20007]
15 ['GOTOF', 20007, '_', 18]
18 ['write', '_', '_', 16000]
"aaaaaa"
19 ['+', 10000, 10001, 20008]
20 ['write', '_', '_', 20008]
4
21 ['*', 30002, 10000, 20009]
22 ['RETURN', 20009, '_', '_']
61 ['=', 1007, '_', 20002]
62 ['=', 20002, '_', 1006]
63 ['write', '_', '_', 1006]
8
64 ['=', 33002, '_', 3000]
65 ['/', 3000, 30007, 23000]
66 ['=', 23000, '_', 3000]
67 ['write', '_', '_', 3000]
0.7000000000000001
```

Prueba 3

```
program patito;
vars{

  int : i;


}


function int uno (int a){
  vars {
        float : b;
        int : p;
  }
  write(a);




  p = a * 3;
  return(a * 5);


}



main() {

  i = 1;

  i = uno(4);



  write(i);
}
```

Resultado de la compilación

```
correct syntax
{'patito': {'scope': 'g_scope', 'type': 'program_type', 'id': 'patito', 'var_table': [['i', 'uno'], ['int', 'int'], [1000, 1001], {}, {}], 'paramorder': ['int']}, 'uno': {'scope': 'l_scope', 'type': 'int', 'id':
 'uno', 'var_table': [['a', 'b', 'p'], ['int', 'float', 'int'], [10000, 13000, 10001], {}, {}], 'paramorder': ['int'], 'size': 3, 'start_address': 2}}
[[3, 5, 1, 4], [30000, 30001, 30002, 30003]]
1 ['GOTO_MAIN', '_', '_', 8]
2 ['write', '_', '_', 10000]
3 ['*', 10000, 30000, 20000]
4 ['=', 20000, '_', 10001]
5 ['*', 10000, 30001, 20001]
6 ['RETURN', 20001, '_', '_']
7 ['ENDFunc', '_', '_', '_']
8 ['=', 30002, '_', 1000]
9 ['ERA', 3, 'uno', '_']
10 ['PARAMETER', 30003, 0, '_']
11 ['GOSUB', 'uno', 2, '_']
12 ['=', 1001, '_', 20000]
13 ['=', 20000, '_', 1000]
14 ['write', '_', '_', 1000]
[[3, 5, 1, 4], [30000, 30001, 30002, 30003]]
```

Resultado de la ejecución

```
patito
Memory Map global
Integer memory:  [[1000, 1001], [None, None]]
Float memory:  [[], []]
String memory:  [[], []]
1 ['GOTO_MAIN', '_', '_', 8]
2 ['write', '_', '_', 10000]
3 ['*', 10000, 30000, 20000]
4 ['=', 20000, '_', 10001]
5 ['*', 10000, 30001, 20001]
6 ['RETURN', 20001, '_', '_']
7 ['ENDFunc', '_', '_', '_']
8 ['=', 30002, '_', 1000]
9 ['ERA', 3, 'uno', '_']
10 ['PARAMETER', 30003, 0, '_']
11 ['GOSUB', 'uno', 2, '_']
12 ['=', 1001, '_', 20000]
13 ['=', 20000, '_', 1000]
14 ['write', '_', '_', 1000]
[[30000, 30001, 30002, 30003], [3, 5, 1, 4]]
[[1000, 1001], [None, None]]
{'patito': {'scope': 'g_scope', 'type': 'program_type', 'id': 'patito', 'var_table': [['i', 'uno'], ['int', 'int'], [1000, 1001], {}, {}], 'paramorder': ['int']}, 'uno': {'scope': 'l_scope', 'type': 'int', 'id':
 'uno', 'var_table': [['a', 'b', 'p'], ['int', 'float', 'int'], [10000, 13000, 10001], {}, {}], 'paramorder': ['int'], 'size': 3, 'start_address': 2}}
1 ['GOTO_MAIN', '_', '_', 8]
8 ['=', 30002, '_', 1000]
9 ['ERA', 3, 'uno', '_']
10 ['PARAMETER', 30003, 0, '_']
11 ['GOSUB', 'uno', 2, '_']
2 ['write', '_', '_', 10000]
4
3 ['*', 10000, 30000, 20000]
4 ['=', 20000, '_', 10001]
5 ['*', 10000, 30001, 20001]
6 ['RETURN', 20001, '_', '_']
12 ['=', 1001, '_', 20000]
13 ['=', 20000, '_', 1000]
14 ['write', '_', '_', 1000]
20
```

Manual de usuario

Todo programa escrito en MyRLikeE, necesita iniciar de la siguiente forma:

program nombre;

Donde program es una palabra reservada, nombre es el nombre que se le asigne al código y el punto y coma es el delimitador imperativo.

Después vendrá un bloque de declaración de variables con la siguiente sintaxis:

vars {

}

Dentro de este, se harán las declaraciones de cualquier tipo de variable que se requiera dentro de los estatutos del lenguaje de la siguiente manera:

tipo : nombreVar, nombreVar2;

Ejemplo;

Int : a, b, c;

Siendo en conjunto así.

vars {
Int : a, b, c;

}

Después de la sección de variables globales, viene la sección de declaración de procedimientos ya sea de tipo entero, flotante, string o void.

Estos poseen la siguiente sintaxis:

```
function tipo foo (tipo a, tipo b) {


}
```

La palabra function es una palabra reservada y es necesaria para cualquier declaración de procedimientos.

Los parámetros únicamente deben contener el tipo y el nombre de la variable asignada.

La variables se declaran de la misma forma que con las globales, dentro de un estatuto vars y siguiendo las mismas reglas.

El lenguaje es permisivo con el uso de los returns, es decir, estos pueden no ser escritos en las funciones no void y no diría nada hasta la ejecución en donde nada funcionaría como debe ser.

Se añade una función write para escribir en la consola cualquier tipo de variable de la siguiente forma.

```
write(var);
```

El manejo de arreglos funciona de la misma forma que en C, estos empiezan en índice 0 y se declaran de la misma forma en las secciones de declaración de variables.

```
int : i, a, b, c[3];
```

Y por último, la llamada a funciones se realiza de la siguiente manera:

Si es de tipo no void:

```
a = foo(a,b,c);
```

En donde a,b,c son los parámetros que requiere.

Si la memoria resulta ser de tipo void, esta puede ser invocada sin necesidad de ser asignada a ninguna variable.