

**Proyecto Final**

**Diseño de Compiladores**

\_\_\_\_\_\_\_\_\_

~~\_~~

\_\_\_\_\_\_\_\_\_\_\_

Iván Alejandro Anguiano Leal

A00817460

22 de Noviembre del 2022

Monterrey, Nuevo León

ÍNDICE

[Descripción del proyecto 3](#_Toc105440975)

[Visión 3](#_Toc105440976)

[Objetivo 3](#_Toc105440977)

[Alcance 3](#_Toc105440978)

[Requerimientos 4](#_Toc105440979)

[Requerimientos Funcionales 4](#_Toc105440980)

[Requerimientos No Funcionales 4](#_Toc105440981)

[Descripción de casos de uso 4](#_Toc105440982)

[Proceso del desarrollo del proyecto 5](#_Toc105440983)

[Commits 5](#_Toc105440984)

[Reflexión 16](#_Toc105440985)

[Descripción del Lenguaje 17](#_Toc105440986)

[Nombre del Lenguaje 17](#_Toc105440987)

[Descripción de las principales características del lenguaje 17](#_Toc105440988)

[Errores 17](#_Toc105440989)

[Descripción del Compilador 18](#_Toc105440990)

[Equipo de Cómputo, Lenguaje y Utilerías especiales 18](#_Toc105440991)

[Descripción del Análisis de Léxico 19](#_Toc105440992)

[Descripción del Análisis de Sintáxis 20](#_Toc105440993)

[Descripción de Generación de Código Intermedio y Análisis Semántico 24](#_Toc105440994)

[Diagramas de Sintáxis 25](#_Toc105440995)

[Acciones semánticas 28](#_Toc105440998)

[Tablas de Consideraciones Semánticas 31](#_Toc105440999)

[Descripción de Administración de Memoria en Compilación 32](#_Toc105441000)

[Descripción de la Máquina Virtual 34](#_Toc105441001)

[Equipo de cómputo, lenguaje y utilerías especiales usadas 34](#_Toc105441002)

[Descripción del proceso de Administración de Memoria en ejecución 34](#_Toc105441003)

[Pruebas de Funcionamiento del Lenguaje 35](#_Toc105441004)

[Documentación de archivos 43](#_Toc105441005)

[Documentación de Código del Proyecto 45](#_Toc105441006)

# Descripción del proyecto

## Visión

El propósito de este proyecto es hacer uso de los conocimientos del área de Computer Science a lo largo de la carrera como: Estructuras de Datos, Lenguajes de Programación, Matemáticas Computacionales; así como hacer uso de los conceptos básicos del proceso de compilación como: Análisis Léxico, Análisis Sintáctico, Análisis Semántico, Traducción y Generación de código intermedio para crear un compilador y una máquina virtual .

## Objetivo

Diseñar y crear un lenguaje de programación en español que sea fácil de aprender y utilizar para programadores hispanohablantes principiantes. El lenguaje contiene operaciones aritméticas básicas, operaciones booleanas, algunas operaciones matemáticas así como operaciones con arreglos unidimensionales y bidimensionales.

## Alcance

El lenguaje contiene todos los elementos básicos de un lenguaje de programación, tales como:

* Declaración de Variables
* Declaración de Funciones/Módulos
* Llamadas de Funciones Void
* Expresiones de asignación
* Retorno de funciones
* Lectura de inputs
* Impresión de outputs
* Estatutos condicionales (si)
* Estatutos cíclicos (para, mientras)
* Expresiones booleanas
* Multiplicacion de matrices
* Operaciones y funciones matemáticas (raizcuadrada, fórmula general, pow, log, gamma, etc.)

# Requerimientos

## Requerimientos Funcionales

* Puede declarar funciones.
* Puede llamar funciones.
* Puede leer valores desde la línea de comandos.
* Puede imprimir valores a la consola.
* Recibe código inicializado con la palabra “programa”.
* Se pueden generar arreglos/matrices y hacer operaciones (suma, resta, division, multiplicación) con las mismas.
* Errores deben ser desplegados cuando sea necesario.

## Requerimientos No Funcionales

* La sintáxis es fácil de entender para un principiante.
* El código se lee desde archivos .txt

# Descripción de casos de uso

|  |  |
| --- | --- |
| **Nombre** | **Descripción** |
| Factorial cíclico | Versión cíclica del cálculo de factorial. |
| Factorial recursivo | Versión recursiva del cálculo factorial utilizando módulos. |
| Fibonacci cíclico | Versión cíclica de calcular el numero N en una secuencia Fibonacci. |
| Fibonacci recursivo | Versión recursiva de calcular el numero N en una secuencia Fibonacci utilizando módulos. |
| Ordenamiento de burbuja (bubble sort) | Ordenamiento burbuja tradicional |
| Búsqueda en Arreglo | Encuentra un elemento en específico en un Arreglo. |
| Multiplicación de Matrices | Calcula la matriz resultante de la multiplicación entre 2 matrices. |
| Otras Operaciones y Funciones matemáticas | Realiza operaciones y funciones matemáticas como raíz cuadrada, potencia de un número, fórmula general, función gamma, logaritmo, seno, coseno, etc.. |

# Proceso del desarrollo del proyecto

Se utilizó **Github** para control de versiones. Se subían los avances semanales a **Canvas** con la descripción de lo que se había hecho hasta esa fecha.

## Commits

|  |
| --- |
| commit 203f5982e866d27c30c16b29f92a23e470b9e07f (HEAD -> main, origin/main, origin/HEAD)  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Thu Nov 17 16:27:06 2022 -0600  Typos  commit a22b20249d0b93ad9c63a934e771ed605242465c  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Thu Nov 17 16:17:11 2022 -0600  Actualizacion correcta documentacion  commit 1d79620bd97a17af84489477cbf79a6e01d195d0  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Thu Nov 17 16:16:53 2022 -0600  Actualizacion algunos errores y documentacion  commit 11bb5dd728a3afb17bab15cf6d2f49c37b82bedf  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Thu Nov 17 13:31:09 2022 -0600  Primera version Documentacion  commit 1efc24b52b08342881c9c78b5cedcecb4ed8c2ba  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Wed Nov 16 21:07:04 2022 -0600  Ultimas funciones matematicas  commit 7c7993708a82488f637469dfc701da2509420587  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Wed Nov 16 18:41:31 2022 -0600  Arregle error en archivo de prueba array sort  commit 317dc0f890c87a3843d9816165267780e3e057d3  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Mon Nov 14 00:46:15 2022 -0600  Elimine tokens no utilizados y actualice codigo de pruebas  commit a15d2980ebb27de3f3a31d674c2985d9a62e4fbd  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Sat Nov 12 23:13:59 2022 -0600  Elimine codigo que no se va a utilizar  commit 28cd66b2e974ab63c6acfcb17d166cffd120a395  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Tue Nov 8 19:47:01 2022 -0600  Generacion de codigo de arreglos  commit 8ed92597ef2f5b8ac78c34aef46fa0552243aa92  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Mon Nov 7 05:09:02 2022 -0600  Mas operaciones matematicas  commit 290845bd2137c48196abb9d4b58c5277da8ea50a  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Mon Nov 7 04:12:07 2022 -0600  Operaciones circulo y cuadrado  commit 7ea9dd32564534f29ea79b50172331fba2db3240  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Mon Nov 7 02:44:49 2022 -0600  Agregue librerias para estadisticas  commit 92783ce50749f0b3d7584f5cb5b80b4c9a884f98  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Mon Nov 7 02:41:11 2022 -0600  Mas operaciones matematicas y programa de prueba  commit 581e952a74c56dbacd4c60f118b4dfdb7405bbab  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Sun Nov 6 05:10:58 2022 -0600  Nuevas operaciones matematicas    Se agregaron raiz cuadrada y ecuacion cuadratica (formula general). Generan sus cuadruplos y maquina virtual hace las operaciones  commit 4a6e33a0ff62f4bd7d5dc732af50da2010e871d1  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Sun Nov 6 02:22:44 2022 -0600  Read me actualizacion  commit cbb60087bb342c5a21e27395e468be6534ba351d  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Thu Nov 3 17:31:02 2022 -0600  Ejecucion de Estatutos secuenciales y condicionales, ejecucion de modulos.  commit f4d6ba645a21f0f5cb77d6be85e9060ee86776d3  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Fri Oct 28 02:14:04 2022 -0500  Maquina virtual operaciones aritmeticas basicas  commit 807de48c2a2351d97b58775fea7e1eb9841427aa  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Wed Oct 26 19:25:05 2022 -0500  Agregue comentarios  commit c54456b38856355db35fdcfd1f1431a0684d4f85  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Wed Oct 26 17:28:48 2022 -0500  Direcciones de Memoria  commit e5b8ec0fbb134190bda512c22fe59c9f2920cc87  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Mon Oct 24 01:46:52 2022 -0500  Update read me  commit 29643434f144f8e637aded6bb693dbf745916438  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Tue Oct 18 18:58:32 2022 -0500  Funciones  commit aeec7e6661093c55336cd2fea723148223ab337d  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Sun Oct 16 00:38:20 2022 -0500  README actualizacion  commit d71a3f47a6238f0594251b1a6e722a22884d0edb  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Fri Oct 14 02:33:08 2022 -0500  Generacion de codigo estatutos condicionales (ciclos)  commit cba2340084d5874aad12fd1b6c8de498a256786a  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Thu Oct 13 15:26:59 2022 -0500  Agregue comentarios  commit f73f35b79dd067782cb547497b2049e7b150c20d  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Mon Oct 10 18:31:24 2022 -0500  Avance semana 2    Generacion de codigo de expresiones aritmeticas y estatutos secuenciales (lectura, asignacion) y generacion de codigo de estatutos condicionales: Decisiones (IF e IF else)  commit 306b13cdb3cc60f652302b87bed655ef03364170  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Wed Oct 5 01:10:32 2022 -0500  Update comments  commit 067ca7335c9cdc290387a87e45fe87bc9d361fd5  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Wed Oct 5 01:02:54 2022 -0500  Generacion de codigo estatutos condicionales: decisiones  commit fc5fb83b26273d886b5b7061dc94c4714af4c2e8  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Sun Oct 2 23:04:47 2022 -0500  Update errors  commit 566fb5c8f2fd3f12a191dc2145cb0bcbca31157e  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Sun Oct 2 03:07:47 2022 -0500  Agregue errores  commit eb8de0c892e3fa65dfa55741ae75c5e4ae6c4555  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Sun Oct 2 00:16:48 2022 -0500  Lexer, cubo semantico  commit e37628b4c82ef13e4f608a6974c1684e1342f823  Author: iaal96 <ivan.anguiano17@hotmail.com>  Date: Sun Oct 2 00:14:45 2022 -0500  Initial commit |

## Reflexión

\_\_\_\_\_\_\_\_\_

~~\_~~

\_\_\_\_\_\_\_\_\_\_\_

Iván Alejandro Anguiano Leal

# Descripción del Lenguaje

## Nombre del Lenguaje

TLD

## Descripción de las principales características del lenguaje

Es un lenguaje de programación que contiene operaciones aritméticas , operaciones booleanas simples, funciones matemáticas y también puede hacer uso de arreglos de una dimensión y bidimensionales para hacer operaciones aritméticas básicas con ellos.

Se puede usar para aprender sobre programación con usos básicos de almacenamiento de memoria temporal e input y output de resultados.

## Errores

|  |  |
| --- | --- |
| **Compilación** | |
| Error sintáctico. | Token inesperado en alguna línea en específico |
| Type mismatch. | Type mismatch en la asignación de una variable. |
| Type mismatch en operación. | Operandos en una operación aritmética no son compatibles. |
| Type mismatch en condición. | Operandos en una operación condicional no son del mismo tipo. |
| Variable indefinida. | Se usó una variable indefinida. |
| Redefinición de variable. | Se define una Variable con un ID que ya fue usado y ya no puede volver a usarse. |
| Número de argumentos inesperados. | Argumentos en el uso del Módulo no coinciden con los que se usaron en la declaración del Módulo. |
| Type mismatch Módulo. | Tipo del Módulo y la variable asignada no son tipos compatibles. |
| Regresa en Función Void. | Hay un Regresa() en una Función tipo Void. |
| No hay Retorno en tipo Función. | Un tipo Función no tiene valor de Retorno. |
| Matriz accesada como Arreglo. | Una variable de tipo Matriz es llamada con sólo 1 índice. |
| Type mismatch en Index. | Índice usado en llamada de Arreglo no es Int. |
| Variable no subindicada como Matriz. | Una variable que no es matriz es llamada con 2 o más índices. |
| Variable no subindicada como Arreglo. | Una variable simple es llamada con un índice. |
| Parametro de Array en llamada a Módulo. | Se llama a un Módulo con un Arreglo como parámetro. |
| Print inválido en variable de Arreglo. | Un Arreglo es enviado a un operador print como parámetro. |
| Operador inválido en Arreglos. | Un Arreglo es usado como operando por un operador que no acepta arreglos como operandos. |
| Operación inválida. | Cualquier tipo de operación inválida. |
| Dimensiones no coinciden. | Se llama a una operación entre variables dimensionadas pero las dimensiones de ambas no coinciden. |
| Asignación inválida a variable de Arreglo. | Se le asigna una variable inválida a la variable de un Arreglo. |
| Tamaño del Arreglo debe ser positivo. | En la declaración del Arreglo, el tamaño es negativo. |

|  |  |
| --- | --- |
| **Ejecución** | |
| Índice fuera de los límites (out of bounds) | El acceso al índice del Arreglo o Matriz está fuera del rango de memoria de la variable. |

# Descripción del Compilador

## Equipo de Cómputo, Lenguaje y Utilerías especiales

**Marca:** HP

**Modelo:** 16-c0011dx

**Sistema Operativo:** Windows 11 Home

**Lenguaje utilizado:** Python 3.10

**Analizador Léxico y Sintáctico:** PLY

## Descripción del Análisis de Léxico

|  |  |
| --- | --- |
|  |  |
|  |  |

## Descripción del Análisis de Sintáxis

|  |
| --- |
| 'program : PROGRAMA ID globalTable PUNTOYCOMA declaration programFunc main'  'globalTable : '  'error : '  'main : mainTable PRINCIPAL LEFTPAR RIGHTPAR LEFTBRACE declaration statement RIGHTBRACE'  'mainTable : '  '''programFunc : function programFunc  | '''  '''assignment : ID dimArray IGUAL hyperExpression PUNTOYCOMA’’’  '''declaration : VAR declarationPrim  | '''  '''declarationPrim : primitive vars PUNTOYCOMA declarationPrim  | '''  '''primitive : INT  | FLOAT  | CHAR '''  'return : REGRESA LEFTPAR hyperExpression RIGHTPAR PUNTOYCOMA'  'if : SI LEFTPAR hyperExpression RIGHTPAR createJQif ENTONCES LEFTBRACE statement RIGHTBRACE ifElse updateJQ'  'createJQif : '  'updateJQ : '  '''ifElse : SINO createJQelse LEFTBRACE statement RIGHTBRACE  | '''  'createJQelse : '  'for : PARA forAssignment HASTA pushJumpFor hyperExpression createQuadFor LEFTBRACE statement RIGHTBRACE updateQuadFor'  'pushJumpFor : '  'createQuadFor : '  'updateQuadFor : '  'forAssignment : ID IGUAL CST\_INT addTypeInt'  'pushLoop : '  'startLoop : '  'endLoop : '  'comment : COMMENT\_TEXT'  'while : MIENTRAS pushLoop LEFTPAR hyperExpression RIGHTPAR startLoop LEFTBRACE statement RIGHTBRACE endLoop'  'vars : ID addVarsToTable varsArray varsComa'  'addVarsToTable : '  '''varsComa : COMA vars  | '''  '''varsMatrix : LEFTBRACK CST\_INT addTypeInt RIGHTBRACK setCols  | '''  '''varsArray : LEFTBRACK CST\_INT addTypeInt RIGHTBRACK setRows varsMatrix  | '''  'setRows : '  'setCols : '  'function : functionType ID addFuncToDir LEFTPAR param RIGHTPAR setParamLength LEFTBRACE declaration statement RIGHTBRACE'  '''param : primitive ID addFuncParams functionParam  | '''  '''functionParam : COMA param  | '''  'addFuncParams : '  'setParamLength : '  '''functionType : FUNCION primitive  | FUNCION VOID setVoidType'''  '''cst\_primitive : CST\_INT addTypeInt  | CST\_FLOAT addTypeFloat  | CST\_CHAR addTypeChar'''  'addTypeInt : '  'addTypeFloat : '  'addTypeChar : '  'addFuncToDir : '  '''hyperExpression : superExpression evaluateHyperExp opHyperExpression  | superExpression opMatrix evaluateOpMatrix  | superExpression evaluateHyperExp'''  'evaluateOpMatrix : '  'evaluateHyperExp : '  '''opHyperExpression : AND addOperator  | OR addOperator'''  '''superExpression : exp evaluateSuperExp opSuperExpression exp evaluateSuperExp  | exp evaluateSuperExp'''  '''opSuperExpression : MAYOR\_QUE addOperator  | MENOR\_QUE addOperator  | MAYOR\_IGUAL addOperator  | MENOR\_IGUAL addOperator  | DIFERENTE\_A addOperator  | IGUAL\_A addOperator'''  'evaluateSuperExp : '  '''opMatrix : addOperator'''  '''exp : term evaluateTerm expFunction  | term evaluateTerm '''  'evaluateTerm : '  '''expFunction : MAS addOperator exp  | MENOS addOperator exp '''  'setVoidType : '  '''term : factor evaluateFactor termFunction  | factor evaluateFactor'''  'evaluateFactor : '  '''termFunction : MULTIPLICA addOperator term  | DIVIDE addOperator term '''  'addOperator : '  '''factor : LEFTPAR addFF hyperExpression RIGHTPAR removeFF  | cst\_primitive  | module  | ID dimArray'''  'addFF : '  'removeFF : '  'read : LEE LEFTPAR id\_list RIGHTPAR PUNTOYCOMA'  '''id\_list : ID dimArray addRead id\_listFunction'''  '''id\_listFunction : COMA id\_list  | '''  'addRead : '  '''print : IMPRIME LEFTPAR printFunction RIGHTPAR PUNTOYCOMA  '''printFunction : print\_param COMA printFunction2  | print\_param '''  'printFunction2 : printFunction'  '''print\_param : hyperExpression addPrint  | CST\_STRING addPrintString '''  'addPrintString : '  'addPrint : '  'module : ID checkFunctionExists generateERASize LEFTPAR moduleFunction nullParam RIGHTPAR generateGosub'  'checkFunctionExists : '  'generateERASize : '  'nullParam : '  'generateGosub : '  'generateParam : '  'nextParam : '  '''dimArray : addOperandId addTypeId LEFTBRACK readIDType hyperExpression verifyRows RIGHTBRACK dimMatrix  | addOperandId addTypeId '''  'addOperandId : '  'addTypeId : '  'readIDType : '  'verifyRows : '  '''dimMatrix : LEFTBRACK hyperExpression verifyCols RIGHTBRACK  | checkMatAsArray '''  'verifyCols : '  'checkMatAsArray : '  '''statement : return checkVoidType  | if statement  | comment statement  | read statement  | print statement  | assignment statement  | module PUNTOYCOMA statement  | for statement  | checkNonVoidType'''  'checkVoidType : '  'checkNonVoidType : ' |

## Descripción de Generación de Código Intermedio y Análisis Semántico

Las operaciones se realizan mediante cuádruplos generados usando el formato que vimos en clase:

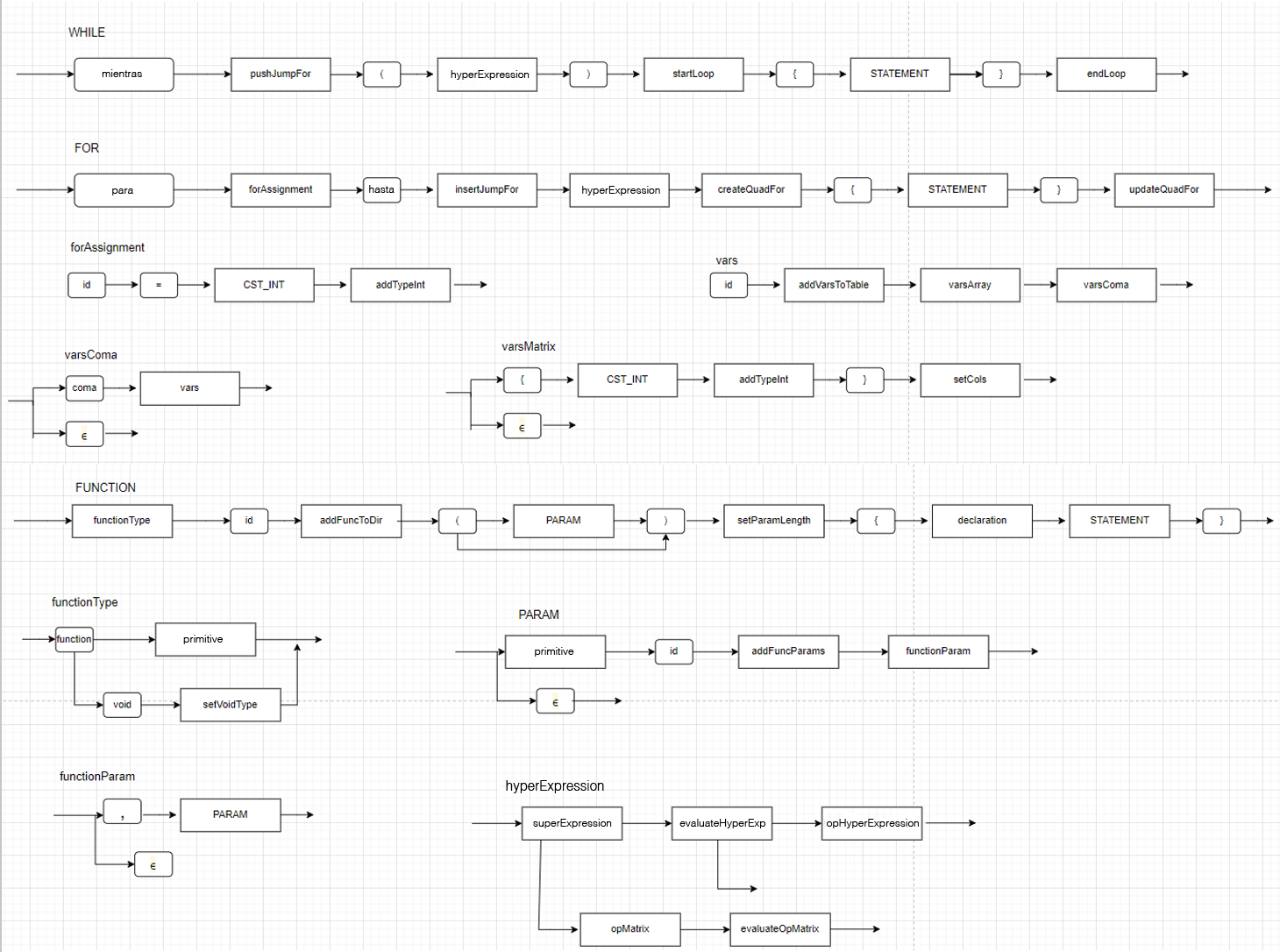
Cuadruplo = Qn (operador, operando\_izquierdo, operando\_derecho, resultado)

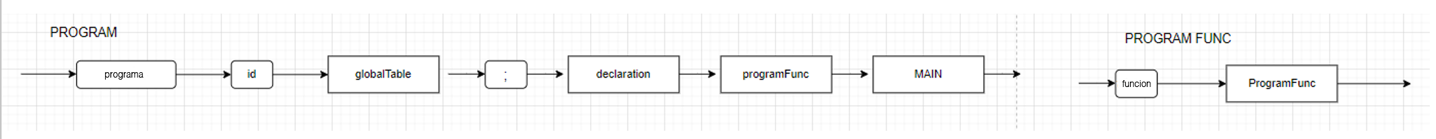
En este formato, el operador puede ser cualquier operador que se encuentre dentro del rango de operadores del lenguaje, sean lógicos o matemáticos, también algunos operadores especiales como el GOTO y GOTOF que se usan en ciclos y condiciones, así como operadores que se usan para el manejo de llamadas de módulos y cambio de contexto como GOSUB y ERA, y el VERIFICA para variables dimensionadas.

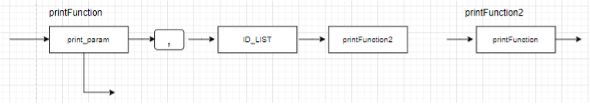
Los operandos y el resultado son “direcciones” de memoria dependiendo su rango de valores de dirección. Los valores de las direcciones para cada tipo de variable o constante se establecieron de la siguiente manera:

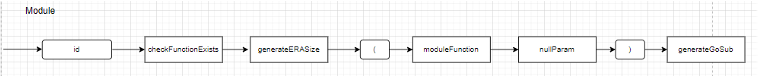
* Int global: 0-999
* Float global: 1000-1999
* Char global: 2000-2999
* Int local: 3000-3999
* Float local: 4000-4999
* Char local: 5000-5999
* Int temporal: 6000-6999
* Float temporal: 7000-7999
* Char temporal: 8000-8999
* Int constante: 9000-9999
* Float constante: 10000-10999
* Char constante: 11000-11999
* Temporal pointer: 12000-12999
* Void: 13000-13999

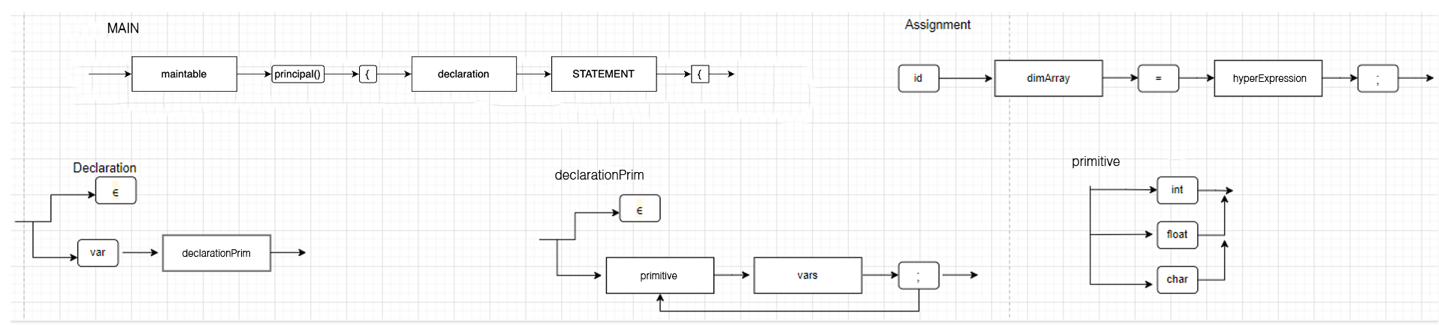
## Diagramas de Sintáxis

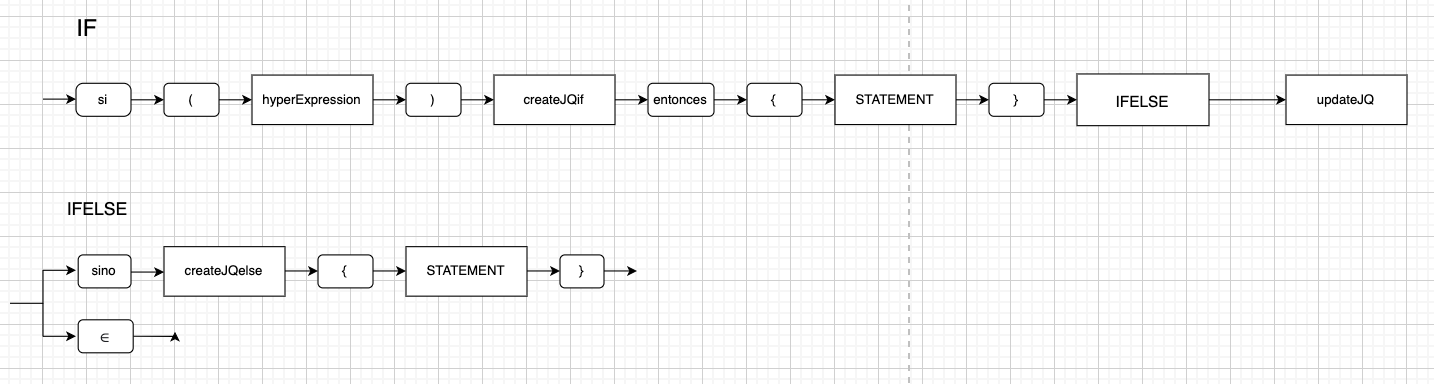


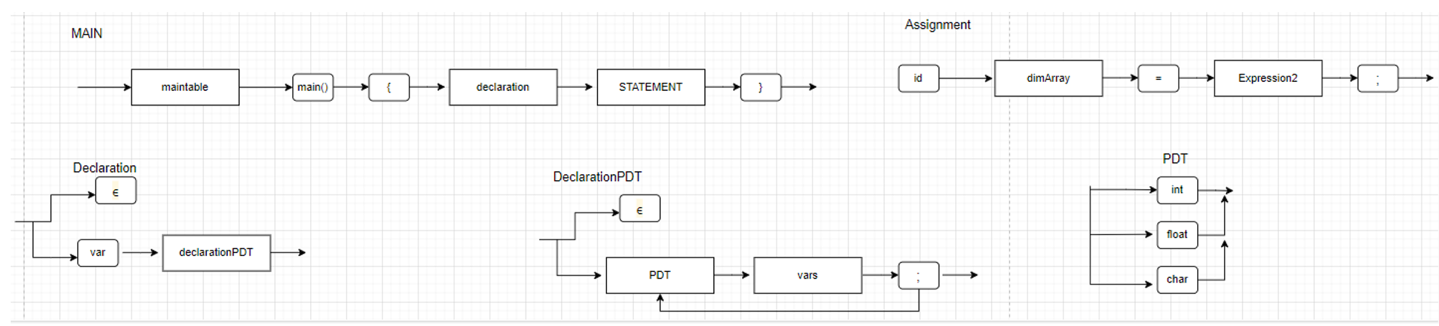


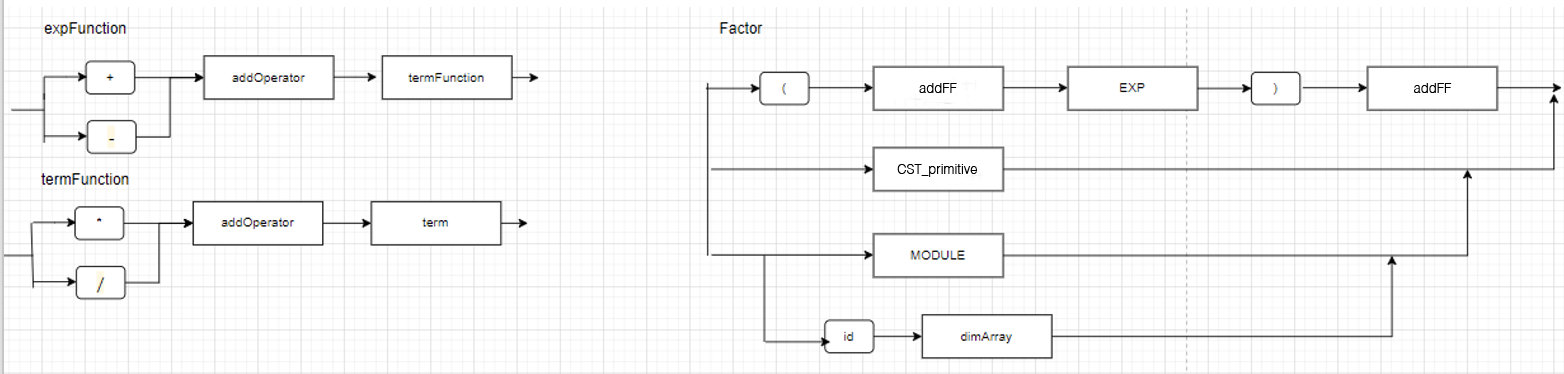


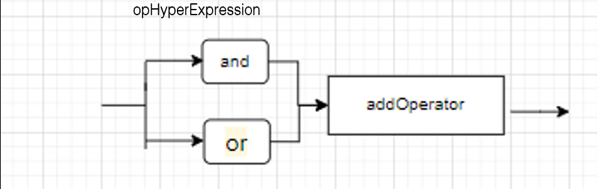


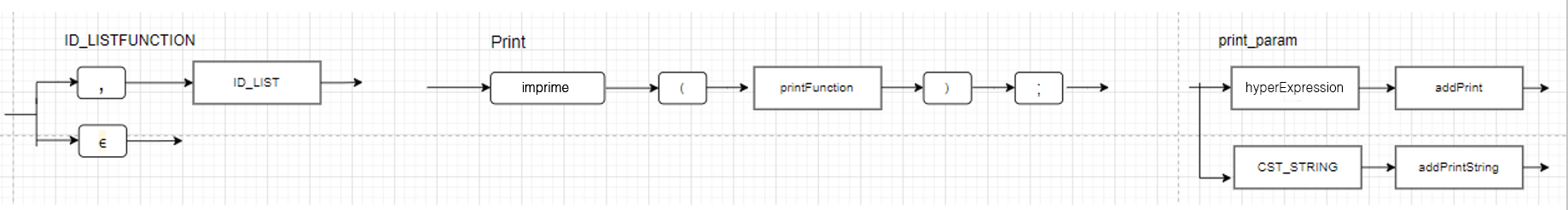


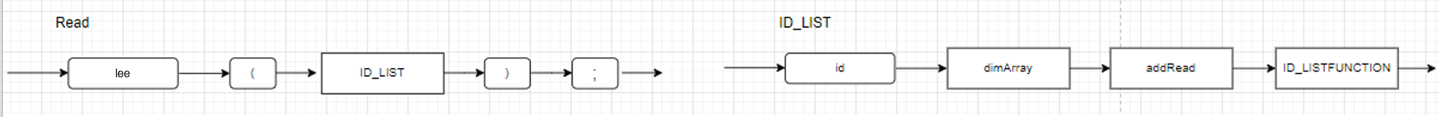




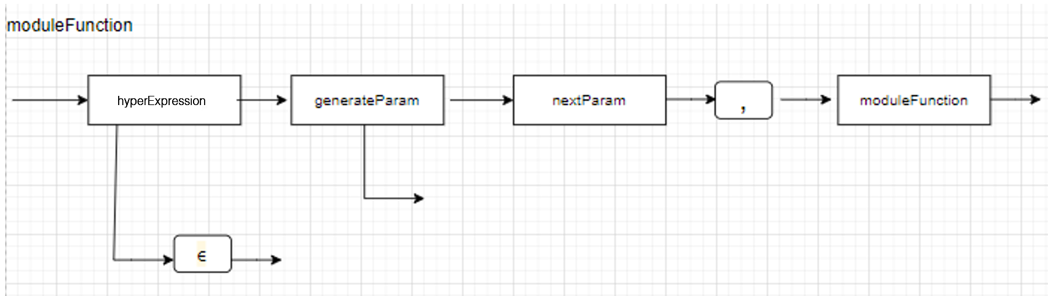


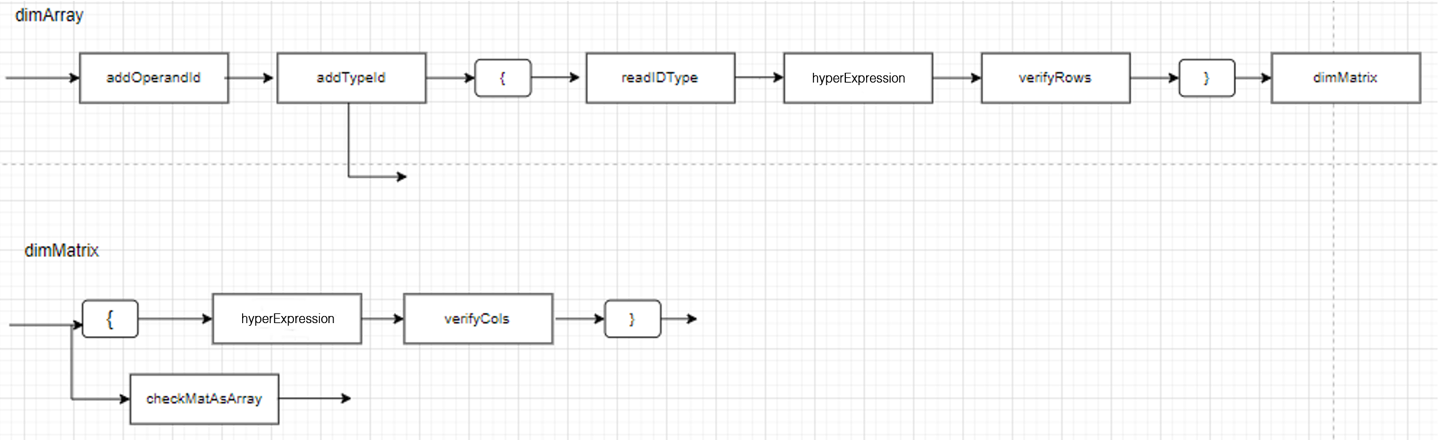


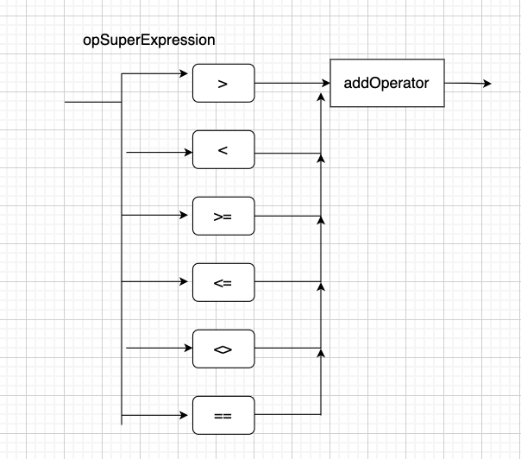




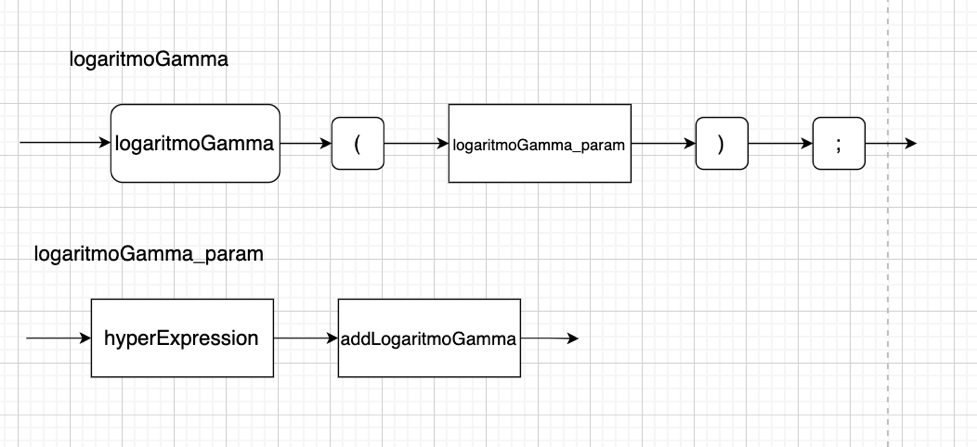
## 

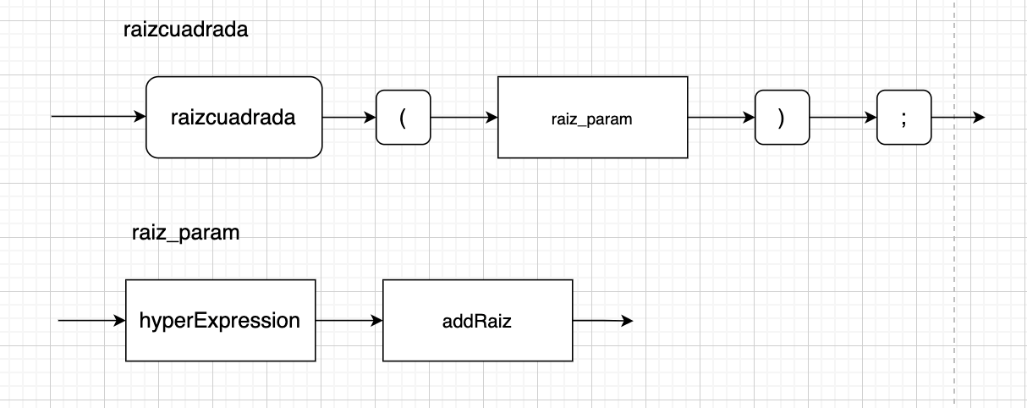


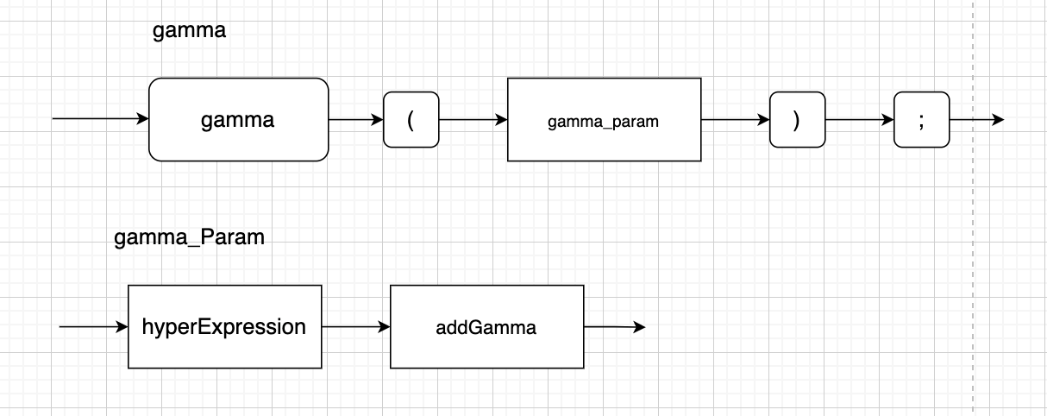




## 







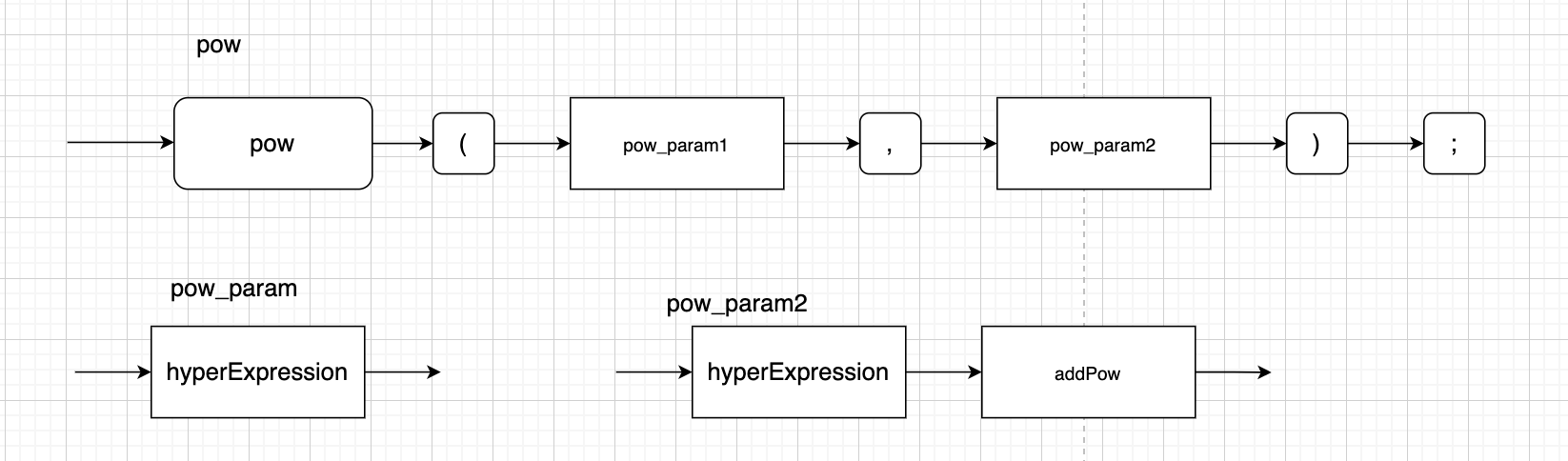
## 

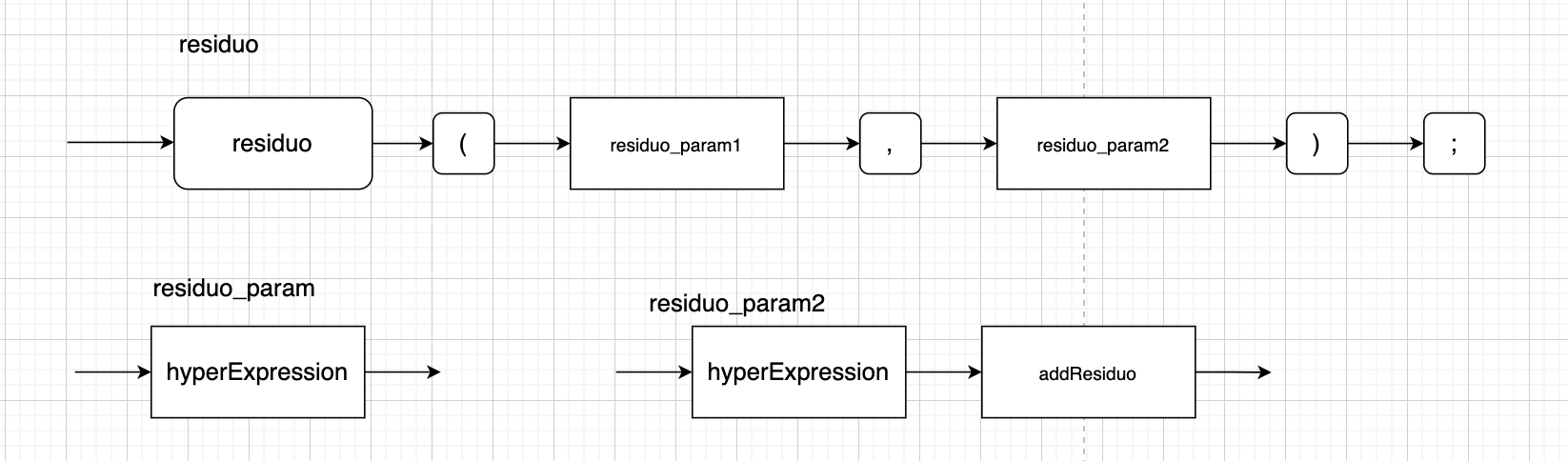
## 

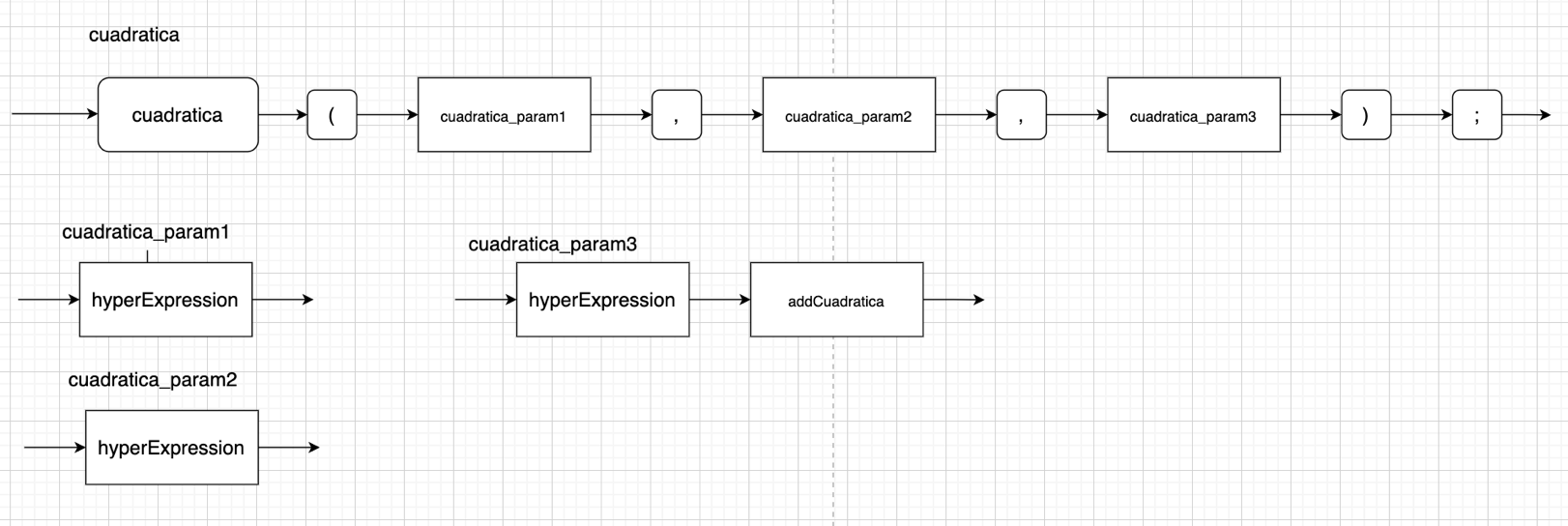
## 

## 

## 







## Acciones semánticas

|  |  |
| --- | --- |
| **Nombre** | **Acción** |
| globalTable | Inicializar programa y crear tabla de variables. |
| mainTable | Agregar main a varTable e inicializar propiedades de la función main.  Actualizar el cuádruplo main para saltar al inicio del programa. |
| assignment | Generar cuádruplo en la varTable correspondiente. |
| declaration | Definir cuádruplo start para una función. |
| primitive | Cambiar el tipo actual por una declaración. |
| createJQif | Checar tipo y valor de expresión y generar cuádruplo de salto. |
| updateJQ | Actualizar cuádruplo de salto con el id del cuádruplo al que se va a saltar. |
| createJQelse | Crear cuádruplo de salto para estatuto else. |
| pushLoop | Hacer push al id del cuádruplo a la pila de saltos. |
| startLoop | Checar el tipo del resultado de la expresión, generar cuádruplo y hacer push al id a la pila de saltos. |
| endLoop | Generar cuádruplo una vez que el while termine y actualizar GOTOF con el id al final del cuádruplo del loop. |
| pushJumpFor | Push al id del cuádruplo al cual saltar saltar a la pila de saltos. |
| createQuadFor | Agregar GOTOF a cuádruplos. |
| updateQuadFor | Actualizar el cuádruplo GOTOF con el ID del cuádruplo al cual saltar para FOR. |
| forAssignment | Agregar iterador a la tabla de constantes y crear variable iterativa. |
| addVarsToTable | Agrega ID actual y su tipo a la tabla de variables. |
| varsArray | Sólo para declaración de arreglos, guarda la dirección base en las constantes de la tabla de variables. |
| setRows | Definir la cantidad de filas de una variable dimensionada. |
| setCols | Definir la cantidad de columnas de una variable dimensionada. |
| function | Crea cuádruplo ENDFUNC y define tabla de variable local. |
| addFuncToDir | Verificar tipo de función e insertar función en el directorio de funciones con tipo, tabla de variables y parámetros. |
| setVoidType | Definir tipo actual de función como Void. |
| addFuncParams | Agregar una lista de tipos de parámetros al scope de la función. |
| setParamLength | Definir la cantidad de parámetros en la función. |
| addTypeInt | Guardar int en tabla de constantes y hacer push al operando a la pila de operandos. |
| addTypeFloat | Guardar Float en tabla de constantes y hacer push al operando a la pila de operandos. |
| addTypeChar | Guardar Char en tabla de constantes y hacer push al operando a la pila de operandos. |
| evaluateOpMatrix | Evalúa operador y operandos de la operación de una variable dimensionada. |
| evaluateHyperExp | Evalúa operador y operandos de expresiones booleanas del tipo AND y OR. |
| evaluateSuperexp | Evalúa operador y operandos de expresiones booleanas del tipo >, <, ==, <>, <= y >=. |
| evaluateTerm | Evalúa operador y operandos del tipo + y – para variables y variables dimensionadas. |
| evaluateFactor | Evalúa operadores y operandos del tipo \* y / para variables y variables dimensionadas (en dimensionadas solo \*) |
| addOperator | Hace push a un operador read a la pila de operadores. |
| addFF | Hace push a un paréntesis al stack de operadores como fondo falso. |
| removeFF | Hace pop a al paréntesis del stack de operadores. |
| addRead | Genera un cuádruplo READ y le hace push a la lista de cuádruplos. |
| addPrint | Genera un cuádruplo PRINT y le hace push a la lista de cuádruplos. |
| addPrintString | Lee un string y lo guarda en la tabla de constantes para después ser impreso por el operador PRINT. |
| checkVoidType | Lanza un error si hay un REGRESAen una función Void. |
| checkNonVoidType | Lanza un error si no hay REGRESA en una función que NO es Void. |
| checkFunctionExists | Verifica que una función existe en el directorio de funciones y le hace push al operador del módulo a la pila de operadores. |
| generateERASize | Crea el cuádruplo ERA con la dirección de la función que será llamada. |
| nullParam | Lanza un error si falta un parámetro en la llamada de una función. |
| generateGoSub | Genera el cuádruplo GoSub con la dirección de la función a llamar y guarda el resultado en una dirección temporal si NO es void. |
| generateParam | Genera el cuádruplo PARAM con el operando que está siendo leído. |
| nextParam | Suma 1 al iterador de param. |
| dimArray | Hace pop al id y scope de la matriz o arreglo a usar. |
| addOpperandId | Hace push al id del arreglo a la pila de ID de arreglos y el scope a la pila de scope. |
| AddTypeID | Hace push a los tipos de la matriz a la pila de tipos. |
| readIDType | Checa tipos y operandos y lanza error si hay un mismatch. |
| verifyRows | Genera el cuádruplo Verify del índice que está siendo usado para ver si está dentro del rango correcto de número de filas. |
| dimMatrix | Genera el cuádruplo para sumar la dirección base y la constante del índice que está siendo usada para accesar al espacio de memoria correcto. |
| verifyCols | Genera el cuádruplo verify del segundo índice que está siendo usado para ver si está dentro del rango correcto del número de filas. |
| checkMatAsArray | Lanza un error si sólo se está usando un índice en una Matriz. |
| addLogaritmo | Genera cuádruplo logaritmo y lo mete a la lista de cuádruplos. |
| addLogaritmoGamma | Genera cuádruplo logaritmoGamma y lo mete a la lista de cuádruplos. |
| addRaiz | Genera cuádruplo raizcuadrada y lo mete a la lista de cuádruplos. |
| addGamma | Genera cuádruplo gamma y lo mete a la lista de cuádruplos. |
| addRadianes | Genera cuádruplo radianes y lo mete a la lista de cuádruplos. |
| addGrados | Genera cuádruplo grados y lo mete a la lista de cuádruplos. |
| addSeno | Genera cuádruplo seno y lo mete a la lista de cuádruplos. |
| addCoseno | Genera cuádruplo coseno y lo mete a la lista de cuádruplos. |
| addTangente | Genera cuádruplo tangente y lo mete a la lista de cuádruplos. |
| addRedondearArriba | Genera cuádruplo redondearArriba y lo mete a la lista de cuádruplos. |
| addRedondearAbajo | Genera cuádruplo redondearAbajo y lo mete a la lista de cuádruplos. |
| addExponencial | Genera cuádruplo exponencial y lo mete a la lista de cuádruplos. |
| addPow | Genera cuádruplo pow y lo mete a la lista de cuádruplos. |
| addResiduo | Genera cuádruplo residuo y lo mete a la lista de cuádruplos. |
| addCuadratica | Genera cuádruplo cuadratica y lo mete a la lista de cuádruplos. |

## Tablas de Consideraciones Semánticas

Suma, Resta y Multiplicación

|  |  |  |  |
| --- | --- | --- | --- |
| **+, -, \*** | **int** | **float** | **char** |
| **int** | int | float | ERROR |
| **float** | float | float | ERROR |
| **char** | ERROR | ERROR | ERROR |

Menor que, Igual que, Mayor o Igual, Menor o Igual

|  |  |  |  |
| --- | --- | --- | --- |
| **< , >, <=, >=** | **int** | **float** | **char** |
| **int** | int | int | ERROR |
| **float** | int | int | ERROR |
| **char** | ERROR | ERROR | ERROR |

División

|  |  |  |  |
| --- | --- | --- | --- |
| **/** | **int** | **float** | **char** |
| **int** | float | float | ERROR |
| **float** | float | float | ERROR |
| **char** | ERROR | ERROR | ERROR |
|  |  |  |  |

Diferente a, igual a

|  |  |  |  |
| --- | --- | --- | --- |
| **<>, ==** | **int** | **float** | **char** |
| **int** | int | int | ERROR |
| **float** | int | int | ERROR |
| **char** | ERROR | ERROR | int |

And, Or

Los operadores & y | funcionan igual que en Python, donde el valor del operando izquierdo se toma en una operación OR , y el valor del operador derecho se toma en una operación AND (por ejemplo, la operación "x" | 1 da "x", mientras que la operación “x” & 1 da 1, sin embargo, si hay un 0, que es falso, en el lado izquierdo de una operación OR se obtendrá el lado derecho, y si tiene un 0 en el lado derecho de una operación AND, se obtendrá un 0.

## Descripción de Administración de Memoria en Compilación

En compilación, se depende en gran parte de la estructura de datos “Dictionary” de Python para guardar toda la información de las variables y funciones. Esta estructura se usa para almacenar grupos de objetos. Consiste en un mapeo de pares clave-valor, donde cada clave está asociada a un valor. Puede contener datos con tipos de datos iguales o diferentes, no está ordenado y es mutable.

# Ejemplo de dirFunc

    "global":

        "type": "void"

        "vars": variableTable["global"] -> "i":

                                                "type": "int"

                                                "value": 1

                                                "address": 0

    "main":

        "type": "void",

        "vars": variableTable["main"] -> "c":

                                              "type": "char"

                                              "value": "y"

                                              "address": 2000

    "funcion\_uno":

        "type": "int",

        "params": Queue[int, int, float]

        "paramsLength": len(params)

        "vars": variableTable["funcion\_uno"] -> "x":

                                            "type": "int"

                                            "value": 1

“address”: 5000

Se usan nombres de funciones o scopes como claves en la hashtable del directorio de funciones.

Por ejemplo, “funcion\_uno”, functionDir[“funcion\_uno”] nos indica que es una función de tipo int y que cuenta con 3 parámetros de tipo int, int y float.

Si accedemos a functionDir[“funcion\_uno”][“vars”], obtendríamos la tabla de variables de esta función, que es una referencia a la tabla de variables de la función y contiene todas las variables con sus respectivas direcciones y tipos, las cuales se asignan durante el proceso de compilación.

En variableTable[“constants”], se guardan las constantes identificadas en el proceso de parseo. Las claves son los valores en sí y almacenan sus direcciones. Por ejemplo, variableTable[“constants”][“x”] tendría la dirección 11000, que es para chars constantes.

Como ya se había mencionado, la estructura para los cuádruplos es la siguiente:

**(operador, operando\_izquierdo, operando\_derecho, resultado)**

Estos se construyen utilizando el constructor de clases “Quadruple” y después se almacenan en una clase “Quadruples” que almacena todos estos objetos “Quadruple”.

# Descripción de la Máquina Virtual

## Equipo de cómputo, lenguaje y utilerías especiales usadas

**Marca:** HP

**Modelo:** 16-c0011dx

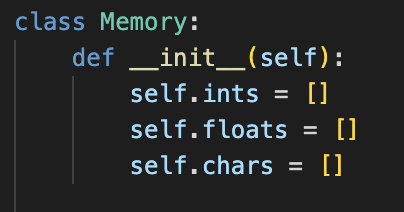
**Sistema Operativo:** Windows 11 Home

**Lenguaje utilizado:** Python 3.10

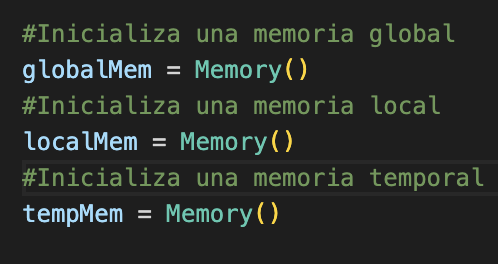
**Analizador Léxico y Sintáctico:** PLY

## Descripción del proceso de Administración de Memoria en ejecución

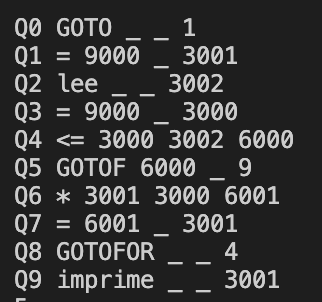
Durante la ejecución, se depende de una clase “Memory” que tiene una lista de variables de tipo int, float y char.



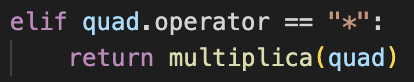
La máquina virtual inicializa una memoria global, una local y otra temporal usando el constructor de la clase “Memory”.

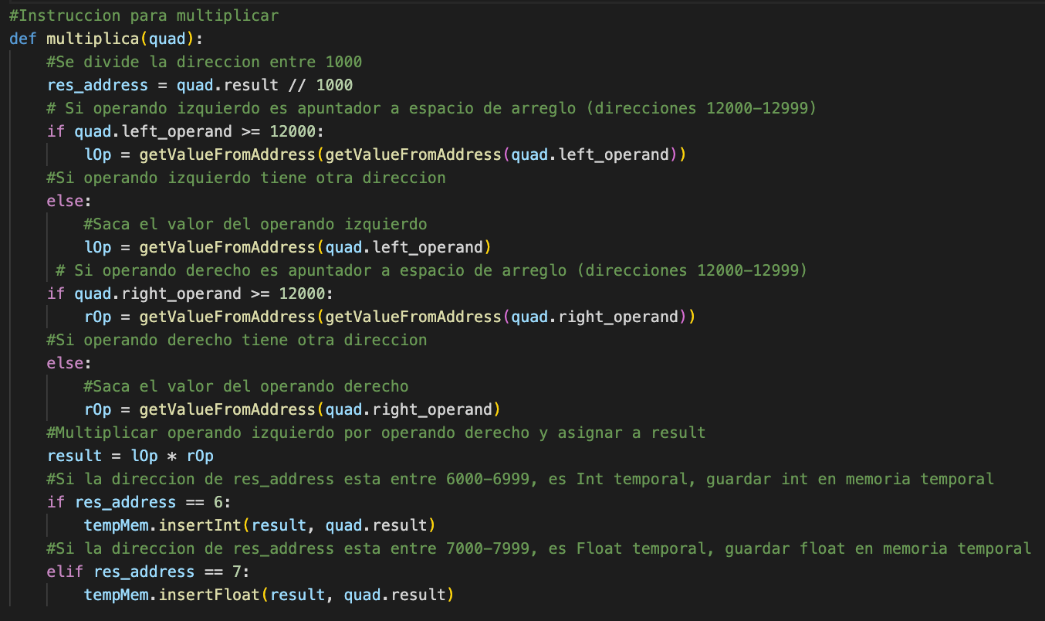


El resultado de esto es un objeto de memoria global, memoria local y memoria temporal, cada uno de estos objetos tendrá una lista de int, float y char. También hacemos uso de la tabla de constantes obtenida durante la compilación, aunque invertimos las claves con las direcciones dentro de ellas para tener en su lugar las direcciones como clave y usar esas direcciones para obtener el valor real de la constante, ya que lo que recibimos con los cuádruplos son las direcciones, no los valores como tal:



La máquina virtual va leyendo los cuádruplos y dependiendo del operador es la instrucción que ejecuta, por ejemplo, en la imágen de arriba en el cuádruplo 6 se tiene un \* como operador por lo que al llegar a ese cuádruplo pasará a ejecutar la instrucción de multiplica.





Lo que hace este fragmento de código es sacar los valores de los operandos izquierdo y derecho utilizando las direcciones de “memoria” que el cuádruplo contiene, los multiplica y “guarda” el resultado en la dirección de “memoria” temporal que viene en el cuádruplo como resultado.

# Pruebas de Funcionamiento del Lenguaje

|  |  |  |
| --- | --- | --- |
| **Factorial Cíclico** | **Cuádruplos** | **Output** |
| programa factorialCic;  principal() {  var int c, resultado, num;  resultado = 1;  lee(num);  para c = 1 hasta c <= num {  resultado = resultado \* c;  }  imprime(resultado);  }  Input: 7 | Q0 GOTO \_ \_ 1  Q1 = 9000 \_ 3001  Q2 lee \_ \_ 3002  Q3 = 9000 \_ 3000  Q4 <= 3000 3002 6000  Q5 GOTOF 6000 \_ 9  Q6 \* 3001 3000 6001  Q7 = 6001 \_ 3001  Q8 GOTOFOR \_ \_ 4  Q9 imprime \_ \_ 3001 | **5040** |

|  |  |  |
| --- | --- | --- |
| **Factorial Recursivo** | **Cuádruplos** | **Output** |
| programa factorialRec;  funcion int factorial(int x) {  si (x > 1) entonces {  regresa(x \* factorial(x - 1));  }  regresa(1);  }  principal() {  var int x, y;  lee(y);  x = factorial(y);  imprime(x);  }  Input: 6 | Q1 > 3000 9000 6000  Q2 GOTOF 6000 \_ 10  Q3 ERA 0 \_ \_  Q4 - 3000 9000 6001  Q5 PARAM 6001 \_ 3000  Q6 GOSUB 0 \_ 1  Q7 = 0 \_ 6002  Q8 \* 3000 6002 6003  Q9 REGRESA \_ \_ 6003  Q10 REGRESA \_ \_ 9000  Q11 ENDFUNC \_ \_ \_  Q12 lee \_ \_ 3001  Q13 ERA 0 \_ \_  Q14 PARAM 3001 \_ 3000  Q15 GOSUB 0 \_ 1  Q16 = 0 \_ 6003  Q17 = 6003 \_ 3000  Q18 imprime \_ \_ 3000 | **720** |

|  |  |  |
| --- | --- | --- |
| **Fibonacci Cíclico** | **Cuádruplos** | **Output** |
| programa fibonacciCic;    principal() {  var int nTermino, primero, segundo, resultado, x;  primero= 0;  segundo= 1;  imprime("Inserta numero: ");  lee(nTermino);  para x = 2 hasta x <= nTermino {  resultado = primero + segundo;  primero = segundo;  segundo = resultado;  }  imprime(resultado);  }  Input: 8 | Q0 GOTO \_ \_ 1  Q1 = 9000 \_ 3001  Q2 = 9001 \_ 3002  Q3 imprime \_ \_ 11000  Q4 lee \_ \_ 3000  Q5 = 9002 \_ 3004  Q6 <= 3004 3000 6000  Q7 GOTOF 6000 \_ 13  Q8 + 3001 3002 6001  Q9 = 6001 \_ 3003  Q10 = 3002 \_ 3001  Q11 = 3003 \_ 3002  Q12 GOTOFOR \_ \_ 6  Q13 imprime \_ \_ 3003 | **21** |

|  |  |  |
| --- | --- | --- |
| **Fibonacci Recursivo** | **Cuádruplos** | **Output** |
| programa fibonacciRecursivo;  funcion int fibonacci(int n) {  var int a,b;  si(n <= 1) entonces {  regresa(n);  }  a = fibonacci(n-1);  b = fibonacci(n-2);  regresa(a+b);  }  principal() {  var int x;  imprime("Valor de x: ");  lee(x);  imprime(fibonacci(fibonacci(x)));  }  Input: 4 | Q0 GOTO \_ \_ 19  Q1 <= 3000 9000 6000  Q2 GOTOF 6000 \_ 4  Q3 REGRESA \_ \_ 3000  Q4 ERA 0 \_ \_  Q5 - 3000 9000 6001  Q6 PARAM 6001 \_ 3000  Q7 GOSUB 0 \_ 1  Q8 = 0 \_ 6002  Q9 = 6002 \_ 3001  Q10 ERA 0 \_ \_  Q11 - 3000 9001 6003  Q12 PARAM 6003 \_ 3000  Q13 GOSUB 0 \_ 1  Q14 = 0 \_ 6004  Q15 = 6004 \_ 3002  Q16 + 3001 3002 6005  Q17 REGRESA \_ \_ 6005  Q18 ENDFUNC \_ \_ \_  Q19 imprime \_ \_ 11000  Q20 lee \_ \_ 3000  Q21 ERA 0 \_ \_  Q22 ERA 0 \_ \_  Q23 PARAM 3000 \_ 3000  Q24 GOSUB 0 \_ 1  Q25 = 0 \_ 6006  Q26 PARAM 6006 \_ 3000  Q27 GOSUB 0 \_ 1  Q28 = 0 \_ 6007  Q29 imprime \_ \_ 6007 | **2**  **(el ejemplo saca fibonacci del fibonacci)** |

|  |  |  |
| --- | --- | --- |
| **Ordenamiento Burbuja (bubble sort)** | **Cuádruplos** | **Output** |
| programa bubbleSort;  var int arreglo[5];  principal() {  var int i, x, aux ;  x = 0;  i = 0;  aux = 0;  //asignar arreglo  arreglo[0] = 5;  arreglo[1] = 9;  arreglo[2] = 14;  arreglo[3] = 8;  arreglo[4] = 0;  para i = 0 hasta i <= 4 {  para x = 0 hasta x <= 3 {  si (arreglo[i] < arreglo[x]) entonces {  aux = arreglo[i];  arreglo[i] = arreglo[x];  arreglo[x] = aux;  }  }  }  para i = 0 hasta i <= 4 {  imprime(arreglo[i]);  }  } | Q0 GOTO \_ \_ 1  Q1 = 9002 \_ 3001  Q2 = 9002 \_ 3000  Q3 = 9002 \_ 3002  Q4 VERIFICA 9002 0 4  Q5 + 9001 9002 12000  Q6 = 9000 \_ 12000  Q7 VERIFICA 9003 0 4  Q8 + 9001 9003 12001  Q9 = 9004 \_ 12001  Q10 VERIFICA 9005 0 4  Q11 + 9001 9005 12002  Q12 = 9006 \_ 12002  Q13 VERIFICA 9007 0 4  Q14 + 9001 9007 12003  Q15 = 9008 \_ 12003  Q16 VERIFICA 9009 0 4  Q17 + 9001 9009 12004  Q18 = 9002 \_ 12004  Q19 = 9002 \_ 3000  Q20 <= 3000 9009 6000  Q21 GOTOF 6000 \_ 44  Q22 = 9002 \_ 3001  Q23 <= 3001 9007 6001  Q24 GOTOF 6001 \_ 43  Q25 VERIFICA 3000 0 4  Q26 + 9001 3000 12005  Q27 VERIFICA 3001 0 4  Q28 + 9001 3001 12006  Q29 < 12005 12006 6002  Q30 GOTOF 6002 \_ 42  Q31 VERIFICA 3000 0 4  Q32 + 9001 3000 12007  Q33 = 12007 \_ 3002  Q34 VERIFICA 3000 0 4  Q35 + 9001 3000 12008  Q36 VERIFICA 3001 0 4  Q37 + 9001 3001 12009  Q38 = 12009 \_ 12008  Q39 VERIFICA 3001 0 4  Q40 + 9001 3001 12010  Q41 = 3002 \_ 12010  Q42 GOTOFOR \_ \_ 23  Q43 GOTOFOR \_ \_ 20  Q44 = 9002 \_ 3000  Q45 <= 3000 9009 6003  Q46 GOTOF 6003 \_ 51  Q47 VERIFICA 3000 0 4  Q48 + 9001 3000 12011  Q49 imprime \_ \_ 12011  Q50 GOTOFOR \_ \_ 45 | **0**  **5**  **8**  **9**  **14** |

|  |  |  |
| --- | --- | --- |
| **Búsqueda en Arreglo** | **Cuádruplos** | **Output** |
| programa busquedaArreglo;  var int arreglo[6];  funcion int find(int i, int j) {  si (j < 0) entonces {  regresa(0-1);  }  si (arreglo[j] == i) entonces {  regresa(j);  }  regresa(find(i, j - 1));  }  principal() {  var int resultado,x;    arreglo[0] = 4;  arreglo[1] = 9;  arreglo[2] = 10;  arreglo[3] = 3;  arreglo[4] = 8;  arreglo[5] = 6;  imprime("Teclea el valor a buscar: ");  lee(x);  resultado = find(x, 5);  si(resultado < 0) entonces{  imprime("Valor no encontrado");  }  sino{  imprime(resultado);  }  }  Input: 8 | Q0 GOTO \_ \_ 18  Q1 < 3001 9002 6000  Q2 GOTOF 6000 \_ 5  Q3 - 9002 9003 6001  Q4 REGRESA \_ \_ 6001  Q5 VERIFICA 3001 0 5  Q6 + 9001 3001 12000  Q7 == 12000 3000 6002  Q8 GOTOF 6002 \_ 10  Q9 REGRESA \_ \_ 3001  Q10 ERA 6 \_ \_  Q11 PARAM 3000 \_ 3000  Q12 - 3001 9003 6003  Q13 PARAM 6003 \_ 3001  Q14 GOSUB 6 \_ 1  Q15 = 6 \_ 6004  Q16 REGRESA \_ \_ 6004  Q17 ENDFUNC \_ \_ \_  Q18 VERIFICA 9002 0 5  Q19 + 9001 9002 12001  Q20 = 9004 \_ 12001  Q21 VERIFICA 9003 0 5  Q22 + 9001 9003 12002  Q23 = 9005 \_ 12002  Q24 VERIFICA 9006 0 5  Q25 + 9001 9006 12003  Q26 = 9007 \_ 12003  Q27 VERIFICA 9008 0 5  Q28 + 9001 9008 12004  Q29 = 9008 \_ 12004  Q30 VERIFICA 9004 0 5  Q31 + 9001 9004 12005  Q32 = 9009 \_ 12005  Q33 VERIFICA 9010 0 5  Q34 + 9001 9010 12006  Q35 = 9000 \_ 12006  Q36 imprime \_ \_ 11000  Q37 lee \_ \_ 3001  Q38 ERA 6 \_ \_  Q39 PARAM 3001 \_ 3000  Q40 PARAM 9010 \_ 3001  Q41 GOSUB 6 \_ 1  Q42 = 6 \_ 6005  Q43 = 6005 \_ 3000  Q44 < 3000 9002 6006  Q45 GOTOF 6006 \_ 48  Q46 imprime \_ \_ 11001  Q47 GOTO \_ \_ 49  Q48 imprime \_ \_ 3000 | **4** |

|  |  |  |
| --- | --- | --- |
| **Multiplicación de Matrices** | **Cuádruplos** | **Output** |
| programa multiplicacionMatrices;  principal() {  var int matriz1[3][2], matriz2[2][3], resultado[3][3], i, j;  matriz1[0][0] = 1;  matriz1[0][1] = 2;  matriz1[1][0] = 3;  matriz1[1][1] = 4;  matriz1[2][0] = 5;  matriz1[2][1] = 6;  matriz2[0][0] = 1;  matriz2[0][1] = 2;  matriz2[0][2] = 3;  matriz2[1][0] = 3;  matriz2[1][1] = 4;  matriz2[1][2] = 5;  resultado = matriz1 \* matriz2;  para j = 0 hasta j < 3 {  para i = 0 hasta i < 3 {  imprime(resultado[i][j]);  }  }  } |  | **7**  **15**  **23**  **10**  **22**  **34**  **13**  **29**  **45** |

|  |  |  |
| --- | --- | --- |
| **Funciones Matemáticas** | **Cuádruplos** | **Output** |
| programa matematicas;  principal() {  imprime(raizcuadrada(pow(raizcuadrada(16),3)));  imprime(exponencial(residuo(5,2)));  imprime(redondear.arriba(raizcuadrada(5)));  imprime(redondear.abajo(grados(4.50)));  imprime(gamma(pow(2,5)));  imprime(residuo(224,125));  imprime(radianes(100.03));  imprime(grados(8.90));  imprime(seno(coseno(3.141528)));  imprime(coseno(tangente(90)));  imprime(tangente(90));  imprime(logaritmo(raizcuadrada(pow(5,2))));  imprime(logaritmo.gamma(7));  cuadratica(2,9,10);  } | Q0 GOTO \_ \_ 1  Q1 raizcuadrada 9000 \_ 6000  Q2 pow 6000 9001 6001  Q3 raizcuadrada 6001 \_ 6002  Q4 imprime \_ \_ 6002  Q5 residuo 9002 9003 6003  Q6 exponencial 6003 \_ 6004  Q7 imprime \_ \_ 6004  Q8 raizcuadrada 9002 \_ 6005  Q9 redondearArriba 6005 \_ 6006  Q10 imprime \_ \_ 6006  Q11 grados 10000 \_ 7000  Q12 redondearAbajo 7000 \_ 7001  Q13 imprime \_ \_ 7001  Q14 pow 9003 9002 6007  Q15 gamma 6007 \_ 6008  Q16 imprime \_ \_ 6008  Q17 residuo 9004 9005 6009  Q18 imprime \_ \_ 6009  Q19 radianes 10001 \_ 7002  Q20 imprime \_ \_ 7002  Q21 grados 10002 \_ 7003  Q22 imprime \_ \_ 7003  Q23 coseno 10003 \_ 7004  Q24 seno 7004 \_ 7005  Q25 imprime \_ \_ 7005  Q26 tangente 9006 \_ 6010  Q27 coseno 6010 \_ 6011  Q28 imprime \_ \_ 6011  Q29 tangente 9006 \_ 6012  Q30 imprime \_ \_ 6012  Q31 pow 9002 9003 6013  Q32 raizcuadrada 6013 \_ 6014  Q33 logaritmo 6014 \_ 6015  Q34 imprime \_ \_ 6015  Q35 logaritmoGamma 9007 \_ 6016  Q36 imprime \_ \_ 6016  Q37 cuadratica 9003 9008 9009 | **8.0**  **2.718281828459045**  **3**  **257**  **8.222838654177925e+33**  **-26.0**  **1.7458528507699278**  **509.9324376664327**  **-0.8414709836786413**  **-0.41177780729510316**  **-1.995200412208242**  **1.6094379124341003**  **6.579251212010102**  **-2.0**  **-2.5** |

# Documentación de archivos

|  |  |
| --- | --- |
| **Nombre** | **Detalles** |
| **EstructurasDatos.py** | Declara e inicializa las estructuras de datos que se usan en el proyecto tales como:   * Directorio de Funciones (DirFunc) * Tabla de Variables (varTable) * Cubo Semántico * Pila de Operadores * Pila de Operandos * Pila de Tipos * Pila de Operandos de Arreglos y Matrices * Diccionario de Asignación de Direcciones a los Tipos * Lista de Operadores * Diccionario de IDs para tipos   Crea los objetos Stack() y Queue()  Se usa en:   * **parser.py**: importa objetos inicializados. * **cuadruplos.py**: importa la estructura de datos Stack. * **maquinavirtual.py**: importa la tabla de variables. |
| **errores.py** | Declara y exporta una clase Error() que centraliza los displays de errores.  A todos los errores en tiempo de compilación se les pasa como argumento el número de línea donde ocurre el error para mostrar dónde ocurrió el error, mientras que los errores en tiempo de ejecución sólo muestran el tipo de error.  Se usa en:   * **parser.py:** importa la clase Error() y las usa en las funciones de sintáxis y gramática. |
| **lexer.py** | Hace uso del módulo Lex de PLY. Declara las palabras reservadas del lenguaje en un diccionario.  Enlista todos los tokens para simbolizar todos los operadores del lenguaje.  Declara las expresiones regulares para cada token. Este lexer luego se pasa al parser, que utiliza este análisis léxico para realizar su análisis de sintaxis.  Se usa en:  **parser.py**: Importa el lexer para hacer uso de los tokens y números de línea para pasárselos a la clase de Error. |
| **cuadruplos.py** | Declara las clases Quadruple() y Quadruples().  La clase Quadruple es capaz de crear un objeto con operador, operando izquierdo, operando derecho y resultado.  La clase Quadruples guarda la lista de cuadruplos, la pila de saltos y es capaz de manipularlas para luego pasársela a la máquina virtual.  Usado en:  **parser.py**: importa cuadruplos para crearlos en sus respectivas acciones  **maquinavirtual.py**: importa la lista de cuádruplos para iterarla y ejecutar el código. |
| **maquinavirtual.py** | Declara el método maquina\_virtual() que está a cargo de iterar a través de toda la lista de cuádruplos.  Con cada cuádruplo que lee, ejecuta la instrucción relacionada con su operador.  La máquina virtual también es responsable de la creación de las instancias de la clase Memory(), la pila de memoria local, la pila de apuntadores y el mapa de memoria de constantes.  Usado en:  **parser.py**: Importa el método maquina\_virtual() y lo corre después de que todo el código ha sido analizado y parseado para ejecutarlo. |
| **numpy** | Librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos.  Incorpora una nueva clase de objetos llamados **arrays** que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación.  Usado en:  **maquinavirtual.py**: Una vez que los arreglos/matrices son procesados se usan las funciones de numpy para hacer las operaciones entre las matrices. |
| **parser.py** | El propósito principal del parser es transformar el código escrito en el lenguaje a código intermedio (cuádruplos). Hace uso del módulo YACC de PLY.  Es el archivo que debe correrse para compilar el código del lenguaje.  Parser.py importa:   * **lexer.py** * **Yacc** * **EstructurasDatos.py** * **Quadruples.py** * **Errores.py** * **Maquinavirtual.py** |

# Documentación de Código del Proyecto

#addVarsToTable: Agrega ID actual (y su tipo) a varTable

def p\_addVarsToTable(t):

    'addVarsToTable : '

    #Si el ID ya existe en el scope o global, dar error redefinicion de variable

    if t[-1] in variableTable[currentScope]:

        Error.redefinition\_of\_variable(t[-1], t.lexer.lineno)

    else:

        # Si no existe, agregar ID a variableTable[scope]

        variableTable[currentScope][t[-1]] = {"type": currentType}

        #Si el scope es global, el tipo de direccion va a ser global

        address\_type = "global"

        #Si no es global, va a ser de tipo local

        if currentScope != "global":

            address\_type = "local"

        #Si el tipo es entero, se le asigna la variable va a ser tipo+entero (localInt o globalInt)

        if currentType == "int":

            address\_type += "Int"

        #Si el tipo es float, se le asigna la variable va a ser tipo+entero (localFloat o globalFloat)

        elif currentType == "float":

            address\_type += "Float"

        #Si el tipo es char, se le asigna la variable va a ser tipo+entero (localChar o globalChar)

        else:

            address\_type += "Char"

        #Se le asigna la direccion a la variable

        variableTable[currentScope][t[-1]]["address"] = addresses[address\_type]

        #Se le suma 1 para darselo a la siguiente variable que este dentro del scope

        addresses[address\_type] += 1

        global arrMatId

        arrMatId = Stack()

        arrMatId.push(t[-1])

#Guarda la direccion base de una variable tipo arreglo en las constantes de la tabla de variables

def p\_varsArray(t):

    '''varsArray : LEFTBRACK CST\_INT addTypeInt RIGHTBRACK setRows varsMatrix

                 | '''

    #Asigna tipo de direccion a global

    address\_type = "global"

    const\_address = "constant"

    #Si currentScope no es global, cambia el address type a local

    if currentScope != "global":

        address\_type = "local"

    #Si tipo es entero, sera localInt y asigna constantInt a const\_address

    if currentType == "int":

        address\_type += "Int"

        const\_address += "Int"

    #Si tipo es float, sera localFloat y asigna constantFloat a const\_address

    if currentType == "float":

        address\_type += "Float"

        const\_address += "Float"

    #Si tipo es char, sera localChar y asigna constantChar a const\_address

    if currentType == "char":

        address\_type += "Char"

        const\_address += "Char"

    global arrMatId

    #Asigna direccion al arreglo

    arrMatAddress = variableTable[currentScope][arrMatId.peek()]["address"]

    #Si el arreglo es de 1 dimension

    if "rows" in variableTable[currentScope][arrMatId.peek()] and "cols" not in variableTable[currentScope][arrMatId.peek()]:

        #Asigna el numero de filas a rows

        rows = variableTable[currentScope][arrMatId.peek()]["rows"]

        #Asigna como direccion la direccion actual mas la cantidad de filas (o elementos) - 1.

        addresses[address\_type] += rows - 1

        #Mete la direccion en la tabla de variables

        variableTable["constants"][arrMatAddress] = {"address": addresses[const\_address], "type": "int"}

        #Se le suma 1 para darselo a la siguiente variable de ese tipo que este dentro del scope

        addresses[const\_address] += 1

    #Si es de 2 dimensiones

    if "cols" in variableTable[currentScope][arrMatId.peek()]:

        #Asigna el numero de filas a rows

        rows = variableTable[currentScope][arrMatId.peek()]["rows"]

        #Asigna el numero de columnas a cols

        cols = variableTable[currentScope][arrMatId.peek()]["cols"]

        #Asigna como direccion la direccion actual mas (rows \* cols) - 1.

        addresses[address\_type] += rows \* cols - 1

        #Mete la direccion en la tabla de variables

        variableTable["constants"][arrMatAddress] = {"address": addresses[const\_address], "type": "int"}

        #Se le suma 1 para darselo a la siguiente variable de ese tipo que este dentro del scope

        addresses[const\_address] += 1

    #Saca el Id de la pila

    arrMatId.pop()

#generateGosub: Crea el cuadruplo Gosub

def p\_generateGosub(t):

    'generateGosub : '

    global funcName

    #Generar cuadruplo GOSUB

    tmp\_quad = Quadruple("GOSUB", variableTable["global"][funcName]["address"], "\_", functionDir[funcName]["start"])

    #Hacer push del cuadruplo a la lista de cuadruplos

    Quadruples.push\_quad(tmp\_quad)

    #Si el tipo de la funcion no es void

    if functionDir[funcName]["type"] != "void":

        #Si es entero

        if functionDir[funcName]["type"] == "int":

            #La direccion va a ser tipo temporal entero

            tmpAddress = addresses["temporalInt"]

            #Se le suma 1 para darselo a la siguiente variable de ese tipo que este dentro del scope

            addresses["temporalInt"] += 1

        #Si es float

        if functionDir[funcName]["type"] == "float":

            #La direccion va a ser tipo temporal float

            tmpAddress = addresses["temporalFloat"]

            #Se le suma 1 para darselo a la siguiente variable de ese tipo que este dentro del scope

            addresses["temporalFloat"] += 1

        #Si es char

        if functionDir[funcName]["type"] == "char":

            #La direccion va a ser tipo temporal char

            tmpAddress = addresses["temporalChar"]

            #Se le suma 1 para darselo a la siguiente variable de ese tipo que este dentro del scope

            addresses["temporalChar"] += 1

    #PARCHE GUADALUPANO

        #Genera cuadruplo con la direccion de la funcion y tmpAddress

        tmp\_quad = Quadruple("=", variableTable["global"][funcName]["address"], "\_", tmpAddress)

        #Se le hace push al cuadruplo a la lista de cuadruplos

        Quadruples.push\_quad(tmp\_quad)

        #Se le hace push a direccion temporal a la pila de operandos

        operands.push(tmpAddress)

        #Se le hace push al tipo de la funcion a la pila de tipos

        types.push(variableTable["global"][funcName]["type"])

    #Se saca el operador de la pila de operadores

    operators.pop()