

DataScript

**08/Mayo/2019**

**Diseño de Compiladores**

**Alejandro De la Cruz Tarín**

**A00816503**

# 

[a) Descripción del proyecto 3](#_Toc8177847)

[a.1) Propósito, Objetivos y Alcance del Proyecto 3](#_Toc8177848)

[a.1.1) Propósito 3](#_Toc8177849)

[a.1.2) Objetivo principal 3](#_Toc8177850)

[a.1.3) Alcance del Proyecto 3](#_Toc8177851)

[a.2) Análisis de Requerimientos y Casos de Uso generales. 3](#_Toc8177852)

[a.2.1) Casos de uso 3](#_Toc8177853)

[a.3) Descripción de los principales test cases. 4](#_Toc8177854)

[a.4) Descripción del Proceso 5](#_Toc8177855)

[a.4.1) Bitácoras: 5](#_Toc8177856)

[a.4.2) Aprendizajes logrados 6](#_Toc8177857)

[b) Descripción del lenguaje 7](#_Toc8177858)

[b.1 ) Nombre del lenguaje: Datascript 7](#_Toc8177859)

[b.2) Principales características del lenguaje 7](#_Toc8177860)

[b.3) Listado de los errores compilación y ejecución 7](#_Toc8177861)

[c) Descripción del compilador 8](#_Toc8177862)

[c.1) Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto. 8](#_Toc8177863)

[c.2) Descripción del Análisis de Léxico. Debe incluir: 8](#_Toc8177864)

[Patrones de Construcción (expresados con Expresiones Regulares) de los elementos principales. 8](#_Toc8177865)

[c.3) Descripción del Análisis de Sintaxis. 10](#_Toc8177866)

[c.4) Descripción de Generación de Código Intermedio y Análisis Semántico. 12](#_Toc8177867)

[c.4.1) Código de operación y direcciones virtuales asociadas a los elementos del código. 12](#_Toc8177868)

[c.4.2) Diagramas de Sintaxis con las acciones correspondientes. TODO 13](#_Toc8177869)

[c.4.4) Tabla de consideraciones semánticas 20](#_Toc8177870)

[c.5) Descripción detallada del proceso de Administración de Memoria usado en la compilación. 22](#_Toc8177871)

[d). Descripción de la máquina virtual 23](#_Toc8177872)

[d.2) Descripción detallada del proceso de Administración de Memoria en ejecución (Arquitectura). 23](#_Toc8177873)

[e) Pruebas del funcionamiento del lenguaje 24](#_Toc8177874)

[e.1) Cálculo de Factorial en versión cíclica usando un ciclo de tipo while. 24](#_Toc8177875)

[e.2) Cálculo de Factorial en versión recursiva usando una función que se llama a si misma. 25](#_Toc8177876)

[e.3) Cálculo de Fibonacci en versión cíclica usando un ciclo while. 25](#_Toc8177877)

[e.4) Cálculo de Fibonacci en versión recursiva usando una función que se llama a si misma. 25](#_Toc8177878)

[e.5) Sort de un vector: Un bubble sort de un arreglo de números. 26](#_Toc8177879)

[e.6) Find en un vector: Una búsqueda de un número en un arreglo de números. 27](#_Toc8177880)

[e.7) Una multiplicación de dos matrices de tres por tres guardadas en una matriz, el resultado debe de quedar guardado en una tercera matriz. 0](#_Toc8177881)

[e.8) Obtener media, varianza, máximo, mínimo, desviación estándar y rango de un vector de datos. 2](#_Toc8177882)

[e.9) Obtener la probabilidad y la probabilidad acumulada normal de un vector de datos. 2](#_Toc8177883)

[e.10) Obtener la probabilidad y la probabilidad acumulada binomial a partir de los parámetros p = 0.2 y n = 4 y x = 2. 3](#_Toc8177884)

[e.11) Obtener la probabilidad y la probabilidad acumulada uniforme a partir del rango inferior = 7 y superior = 13 para una variable x = 10.5f. 3](#_Toc8177885)

[e.12) Generar una gráfica de líneas a partir de un array de datos y un array de etiquetas. 4](#_Toc8177886)

[e.13) Generar una gráfica de barras a partir de un array de datos y un array de etiquetas. 4](#_Toc8177887)

[f) Listados documentados del proyecto 5](#_Toc8177888)

# 

# a) Descripción del proyecto

## a.1) Propósito, Objetivos y Alcance del Proyecto

### a.1.1) Propósito

El propósito del proyecto es poner en practica el proceso de compilación y ejecución visto en la clase de diseño de compiladores, incluyendo algoritmos y estructuras de datos para su implementación en lenguajes de programación imperativos de alto nivel.

### a.1.2) Objetivo principal

El objetivo del proyecto es diseñar un pequeño lenguaje de programación, y desarrollar el análisis del léxico, de la sintaxis y de semántica correspondiente, así como generar el código intermedio de éste y finalmente realizar una máquina virtual que sea capaz de interpretar el código intermedio y procesarlo.

### a.1.3) Alcance del Proyecto

El lenguaje debe tener las funcionalidades de un lenguaje clásico imperativo, entre las que incluyen expresiones aritméticas, estatutos secuenciales para asignación, lectura, escritura, estatutos condicionales para condicionales y ciclos, funciones y tipos estructurados (arreglos y matrices).

En el tiempo de desarrollo destinado al proyecto de diseño de compiladores, el lenguaje se

limitará a operaciones estadísticas básicas: media, varianza, máximo, mínimo, desviación estándar y rango.

Además, Datascript ofrecerá funciones de probabilidad y funciones de probabilidad acumulada para distribuciones típicas. El alcance del proyecto abarca la distribución normal, uniforme y binomial.

Finalmente, se ofrecerán herramientas básicas para graficar, en el proyecto se abordarán las funciones para gráficas lineales y de barra.

# a.2) Análisis de Requerimientos y Casos de Uso generales.

## a.2.1) Casos de uso

1. Declarar un programa.
2. Declarar una variable de tipo int, float, string o bool.
3. Declarar un array de tipo int, float, string o bool.
4. Asignar constantes a variables.
5. Asignar constantes a un dirección de un arreglo.
6. Asignar constantes a una dirección de una matriz.
7. Realizar operaciones aritméticas básicas (suma, resta, multiplicación, división, módulo) y asignarlo a una variable.
8. Realiza operaciones lógicas ( >, <, <=, >=, ===, ==, !=, !==) y asignarlos a una variable.
9. Imprimir el valor de una variable.
10. Asignar el valor de una variable a partir de entrada por consola.
11. Generar un estatuto condicional simple (if).
12. Generar un estatuto condicional “if else”.
13. Generar un ciclo.
14. Ejecutar una función estadísticas básica (media, varianza, máximo, mínimo, desviación estándar y rango) y asignar el valor a una variable.
15. Ejecutar una función de distribución normal, binomial o uniforme y asignar el valor a una variable.
16. Crear una gráfica de barras.
17. Crear una gráfica de línea.
18. Crear una función,
19. Crear una función con parámetros.
20. Crear una función que use recursión.
21. Ejecutar una función creada.

# a.3) Descripción de los principales test cases.

1. Cálculo de Factorial en versión cíclica usando un ciclo de tipo while.
2. Cálculo de Factorial en versión recursiva usando una función que se llama a si misma.
3. Cálculo de Fibonacci en versión cíclica usando un ciclo de tipo while.
4. Cálculo de Fibonacci en versión recursiva usando una función que se llama a si misma.
5. Sort de un vector: Un bubble sort de un arreglo de números.
6. Find en un vector: Una búsqueda de un número en un arreglo de números..
7. Una multiplicación de dos matrices de tres por tres guardadas en una matriz, el resultado debe de quedar guardado en una tercer matriz.
8. Obtener media, varianza, máximo, mínimo, desviación estándar y rango de un vector de datos.
9. Obtener la probabilidad y la probabilidad acumulada normal de un vector de datos.
10. Obtener la probabilidad y la probabilidad acumulada binomial a partir de los parámetros p y n.
11. Obtener la probabilidad y la probabilidad acumulada uniforme a partir del rango inferior y superior.
12. Generar una gráfica de líneas a partir de un array de datos y un array de etiquetas.
13. Generar una gráfica de barras a partir de un array de datos y un array de etiquetas.

# a.4) Descripción del Proceso

Para el desarrollo del proyecto se trató de seguir la rúbrica dada por la profesora al principio del semestre y desarrollar cada parte acorde al plan, o lo menor atrasado posible.

En el proceso se trató de implementar lo visto en clase, el uso de códigos y guías entregadas por la maestra como base para la implementación del lenguaje, con las alteraciones correspondientes para la sintaxis y las funcionalidades propias del lenguaje.

## a.4.1) Bitácoras:

**Semana 1**

- Bases para el análisis léxico y semántico en JISON.

- Desarrollo de las gramáticas básicas del lenguaje "patito".

**Estado:** Atrasado.

**Semana 2**

- Cambio de las gramáticas y el léxico en base al feedback ofrecido.

- Añadida funciones para symbol table.

- Symbol table no completa.

- Procedure table no completo.

**Estado:** Atrasado

**Semana 3**

- Generación de cubo semántico y generación de expresiones aritméticas.

- Procedure table desarrolladas.

- Implementación básica de cuádruplos.

**Estado:** Atrasado

**Semana 4**

No se desarrolló nada esta semana.

**Estado**: Atrasado

**Semana 5**

- Generación de código para ciclos.

- Primera implementación de funciones, desarrollada la declaración de funciones.

- Cambios en la tabla de direcciones de funciones

- Symbol table y cuadruplos cambiados a funciones virtuales

**Estado:** Atrasado

**Semana 6**

- Generado código de funciones, calls y parámetros.

- Cambios en el sistema de memoria para manejar adecuadamente scope global y local.

- Planeación y primera implementación básica de memoria virtual.

**Estado:** Atrasado

**Semana 7**

- Cambio en el mapa de memoria.

- Implementación de memoria virtual para estatutos secuenciales.

- Implementacion de codigo de arreglos, declaración y lectura.

- Estatutos condicionales en VM pendientes.

- Módulos en VM pendientes.

- Código para aplicación particular en VM pendiente.

- Documentación pendiente.

**Estado:** Atrasado

**Semana 8**

- Cambio en el mapa de memoria para añadir apuntadores.

- Implementación de estatutos condicionales.

- Implementación de módulos en VM pendientes.

- Implementación de arreglos y matrices.

- Código para aplicación particular en VM. (En progreso).

- Documentación. (En progreso).

**Estado:** Atrasado

## a.4.2) Aprendizajes logrados

El proyecto de la materia de diseño de compiladores ha sido un reto, muchas dificultades he tenido en el desarrollo, desde dificultades con mis herramientas elegidas para el compilador hasta retos en comprender los temas vistos en clase y aplicarlos a mi caso en particular, fue un reto diseñar mi propio compilador.

Sin embargo, creo que fue esencial para conocer en profundidad el de diseño de un lenguaje y el proceso de compilación y ejecución.

Además, que me permitió tener una comprensión más profunda tanto de la teoría vista en clase y aplicar temas vistos en otras clases como el proceso de léxico, semántica y sintaxis.

Este proyecto me ha hecho ver los lenguajes de programación y su proceso de compilación y ejecución de otra forma y comprender más a profundidad en que se cimientan las bases de estos y la lógica y matemáticas detrás de ellos.

Alejandro De la Cruz Tarín

# b) Descripción del lenguaje

## b.1 ) Nombre del lenguaje: Datascript

## b.2) Principales características del lenguaje

Como lenguaje destinado a la estadística, DataScript ofrece funciones nativas para cálculos estadísticos a la vez de funciones para modelación estadística. A la vez de tener funciones para modelar y gráficos.

DataScript es un lenguaje que compila a Javascript, y su uso es posible en ambientes Javascript. Para declarar variables, tendría un tipado fuerte para mejor manejo de los datos. Las variables tienen que ser declaradas con su tipo antes de usarse.

Datascript debe tener las funcionalidades de un lenguaje clásico imperativo, entre las que incluyen expresiones aritméticas, estatutos secuenciales para asignación, lectura, escritura, estatutos condicionales para condicionales y ciclos, funciones y tipos estructurados (arreglos y matrices).

En el tiempo de desarrollo destinado al proyecto de diseño de compiladores, el lenguaje se

limitará a operaciones estadísticas básicas: media, varianza, máximo, mínimo, desviación estándar y rango.

Además, Datascript ofrecerá funciones de probabilidad y funciones de probabilidad acumulada para distribuciones típicas. El alcance del proyecto abarca la distribución normal, uniforme y binomial.

Finalmente, se ofrecerán herramientas básicas para graficar, en el proyecto se abordarán las funciones para gráficas lineales y de barra.

## b.3) Listado de los errores compilación y ejecución

Errores requeridos:

* Type mismatch.
* Errores de sintaxis.
* Errores de variables indefinidas.
* Errores de funciones indefinidas.
* Errores de rango en variables estructuradas.

Posibles errores indeseados:

* Errores en prioridad de expresiones aritméticas.
* Errores en operaciones con arreglos o matrices dadas por los errores en expresiones aritméticas.

# c) Descripción del compilador

## c.1) Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto.

**Equipo de computo:** Dell Precision M3520.

**Sistema Operativo:** Windows 10.

**Lenguaje:** Javascript, en un ambiente NodeJS.

**Utilerias especiales usadas:**

* Jison: Implementación de Bison para javascript.
* Node-chartist: Herramienta para crear gráficos en javascript.
* Prompt-sync: Herramienta para manejar entradas síncronas en NodeJS.
* Jstat: Herramienta para implementar cálculos estadísticos.

## c.2) Descripción del Análisis de Léxico. Debe incluir:

### Patrones de Construcción (expresados con Expresiones Regulares) de los elementos principales.

|  |  |
| --- | --- |
| Token / Keyword | Expresión Regular |
| ID | [a-zA-Z][a-zA-Z0-9]\* |
| CTEI | [0-9]+ |
| CTEF | [0-9]+("."[0-9]+)?”f” |
| CTES | \"([^\\\"]|\\.)\*\" |
| CTEB | true|false |
| Skip | \s+  \n+  \t+ |
| PROGRAM | program |
| END | end |
| INT | int |
| FLOAT | float |
| BOOL | bool |
| STRING | string |
| FUNTION | function |
| WHILE | while |
| IF | if |
| ELSE | else |
| READLINE | readline |
| PRINT | print |
| VOID | void |
| TRUE | true |
| FALSE | false |
| NAN | NaN |
| MAIN | main |
| STDEV | stdev |
| RANGE | range |
| MIN | min |
| MAX | max |
| MEAN | mean |
| VARIANCE | variance |
| DNORMP | dNormPdf |
| DBINOMIALP | dBinomialPdf |
| DNORMC | dNormCdf |
| DBINOMIALC | dBinomialCdf |
| DUNIFORMP | dUniformPdf |
| DUNIFORMC | dUniformCdf |
| BARPLOT | barPlot |
| LINEPLOT | linePlot |
| COLON | : |
| SEMICOLON | ; |
| COMMA | , |
| LBRACKET | [ |
| RBRACKET | ] |
| LPAREN | ( |
| RPAREN | ) |
| LBRACE | { |
| RBRACE | } |
| TIMES | \* |
| DIVIDE | / |
| PLUS | + |
| DIVIDE | - |
| REST | % |
| LESSER | > |
| GREATER | < |
| DIFF | != |
| DEEP\_DIFF | !== |
| EQUAL | == |
| DEEP\_EQUAL | === |
| GREATER\_EQUAL | >= |
| LEASER\_EQUAL | <= |
| ASSIGN | = |

## c.3) Descripción del Análisis de Sintaxis.

program: PROGRAM ID COLON var modules MAIN block END EOF;

var : VAR var-recursive var-follow | %empty;

var-types

: ID

| ID LBRACKET CTEI RBRACKET

| ID LBRACKET CTEI RBRACKET LBRACKET CTEI RBRACKET;

var-recursive: var-types | var-types COMMA var-recursive | %empty;

var-follow: COLON type SEMICOLON var ;

modules

: FUNCTION ID LPAREN params RPAREN COLON module-type block-vars modules

| FUNCTION ID LPAREN RPAREN COLON module-type block-vars modules

| %empty ;

module-type: type | VOID ;

params : ID COLON type params-recursive;

params-recursive: COMMA params | %empty;

type: INT | FLOAT | BOOL | STRING ;

block: LBRACE block-inside RBRACE;

block-vars: LBRACE var block-inside RBRACE;

block-inside: statement block-inside | %empty;

statement : assignation | condition | cycle | print | read | call SEMICOLON | native-functions SEMICOLON ;

assignation: assignation-destination ASSIGN and-or-expression SEMICOLON;

assignation-destination: id | array | matrix;

expression: exp expression-recursive;

expression-recursive: expression-op expression | %empty;

and-or-expression: expression and-or-expression-recursive;

and-or-expression-recursive: and-or-expression-op and-or-expression | %empty;

expression-op

: GREATER

| GREATER\_EQUAL

| LESSER

| LESSER\_EQUAL

| DIFF

| DEEP\_DIFF

| EQUAL

| DEEP\_EQUAL ;

and-or-expression-op: AND | OR;

exp: term exp-recursive;

exp-recursive: exp-op exp | %empty ;

exp-op: MINUS | PLUS;

term: factor term-recursive;

term-op: TIMES | REST | DIVIDE;

term-recursive: term-op term | %empty;

factor : LPAREN expression RPAREN | factor-op var-cte | var-cte;

factor-op : PLUS | MINUS | %empty;

var-cte

: id | array | matrix

| CTEI | CTEF | CTES

| TRUE | FALSE | call

| native-functions;

id: ID;

array: ID LBRACKET exp RBRACKET ;

matrix: ID LBRACKET exp RBRACKET LBRACKET exp RBRACKET ;

call: ID LPAREN call-exp RPAREN | ID LPAREN RPAREN;

call-exp: exp COMMA call-exp | exp ;

read: READLINE LPAREN exp COMMA ID RPAREN SEMICOLON;

print: PRINT LPAREN exp RPAREN SEMICOLON;

condition: IF LPAREN and-or-expression RPAREN block condition-else;

condition-else: ELSE block | %empty;

cycle: cycle-while;

cycle-while: WHILE LPAREN and-or-expression RPAREN block;

native-functions : statistics | distributions | plot;

statistics : stdev | range | min | max | variance | mean ;

stdev: STDEV LPAREN ID RPAREN ;

range: RANGE LPAREN ID RPAREN ;

min: MIN LPAREN ID RPAREN ;

max: MAX LPAREN ID RPAREN ;

variance: VARIANCE LPAREN ID RPAREN ;

mean: MEAN LPAREN ID RPAREN ;

distributions

: dNormP | dNormC

| dBinomialP | dBinomialC

| dUniformP | dUniformC;

dNormP: DNORMP LPAREN exp COMMA ID RPAREN;

dNormC: DNORMC LPAREN exp COMMA ID RPAREN;

dBinomialP: DBINOMIALP LPAREN exp COMMA exp COMMA exp RPAREN ;

dBinomialC: DBINOMIALC LPAREN exp COMMA exp COMMA exp RPAREN ;

dUniformP: DUNIFORMP LPAREN exp COMMA exp COMMA exp RPAREN ;

dUniformC: DUNIFORMC LPAREN exp COMMA exp COMMA exp RPAREN ;

plot: barPlot | linePlot;

barPlot: BARPLOT LPAREN ID COMMA ID COMMA exp RPAREN;

linePlot: LINEPLOT LPAREN ID COMMA ID COMMA exp RPAREN ;

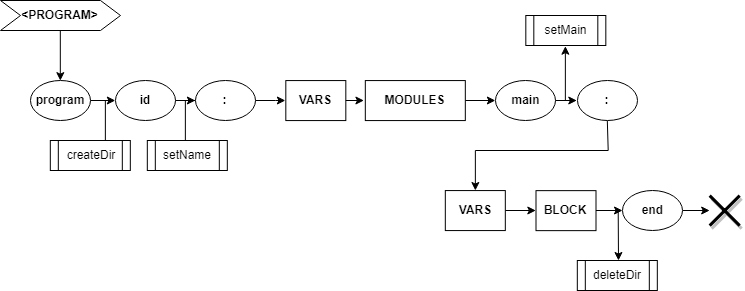
## c.4) Descripción de Generación de Código Intermedio y Análisis Semántico.

### c.4.1) Código de operación y direcciones virtuales asociadas a los elementos del código.

|  |  |
| --- | --- |
| Tipos de datos | Dirección de memoria inicial |
| global int | 0 |
| global float | 2500 |
| global bool | 5000 |
| global string | 7500 |
| local int | 10000 |
| local float | 12500 |
| local bool | 15000 |
| local string | 17500 |
| constant int | 20000 |
| constant float | 22500 |
| constant string | 25000 |
| constant bool | 27500 |
| Pointer | 30000 |

|  |  |
| --- | --- |
| Código de operación | ID |
| + | 0 |
| - | 1 |
| \* | 2 |
| / | 3 |
| % | 4 |
| = | 5 |
| === | 6 |
| == | 7 |
| !== | 8 |
| != | 9 |
| > | 10 |
| >= | 11 |
| < | 12 |
| <= | 13 |
| && | 14 |
| || | 15 |
| GOTO | 16 |
| GOTOF | 17 |
| ENDPROC | 18 |
| ERA | 20 |
| GOSUB | 21 |
| PARAMETER | 22 |
| VER | 23 |
| STDEV | 30 |
| MAX | 31 |
| MIN | 32 |
| RANGE | 33 |
| VARIANCE | 34 |
| MEAN | 35 |
| SETPARAM | 39 |
| SETVECTOR | 40 |
| NORMALPDF | 41 |
| NORMALCDF | 42 |
| BINOMIALPDF | 43 |
| BINOMIALCDF | 44 |
| UNIFORMPDF | 45 |
| UNIFORMCDF | 46 |
| PLOT | 47 |
| PRINT | 51 |
| READLINE | 52 |

### c.4.2) Diagramas de Sintaxis con las acciones correspondientes. TODO

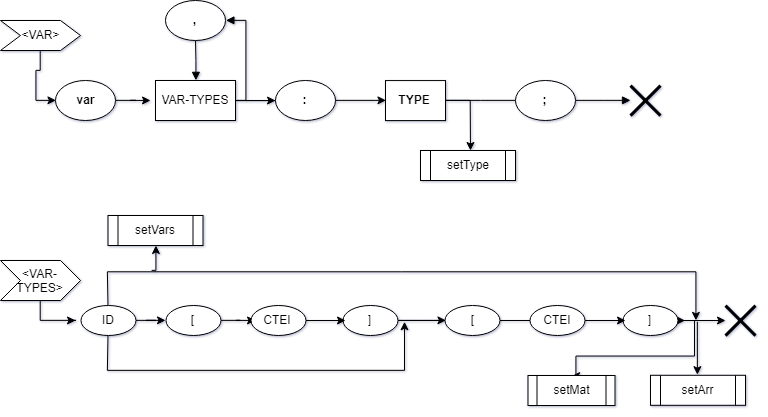


createDir: Crea el objeto para almacenar los datos del programa.

setName: Añade el nombre de la variable al objeto del programa.

setMain: Añade el cuadruplo GOTO para ir al main al principio de la lista.

deleteDir: Corre la máquina virtual con los cuadruplos y la memoria de compilación. Elimina las estructuras usadas en compilación.

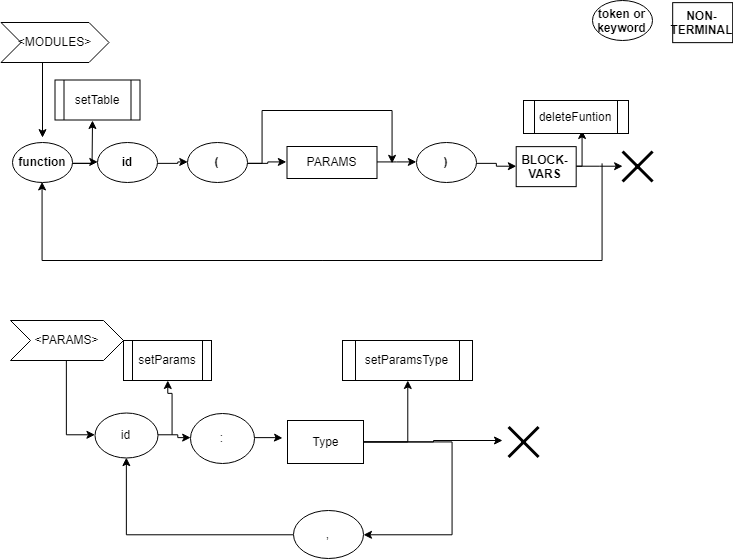


setType: Para cada una de las variables encontradas, define su tipo y las mete en memoria.

setVars: Valida que la variable no esté definida. *Guarda una variable simple en la pila de variables.*

*setArr:* *Valida que la variable no esté definida. Guarda una matriz con sus dimensiones en la pila de memoria*

setMat: Valida que la variable no esté definida. Guarda una matriz con sus dimensiones en la pila de variables.

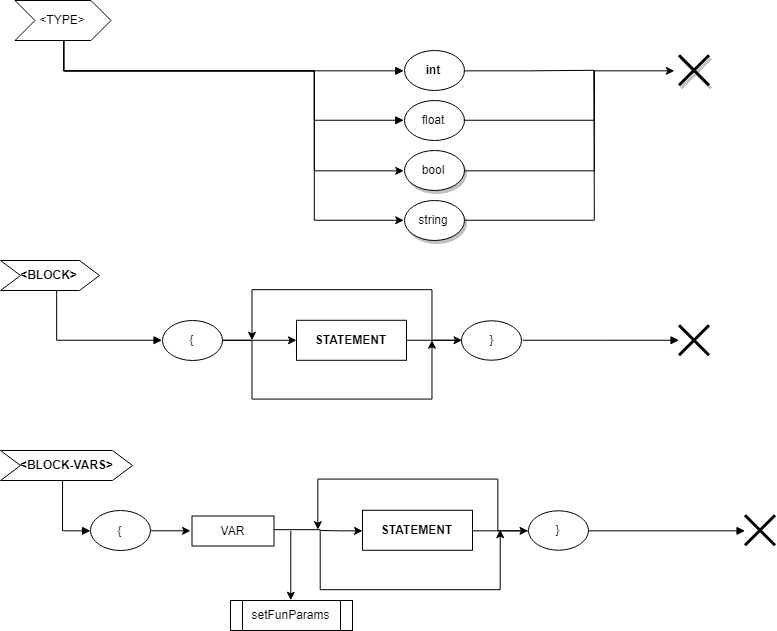
**

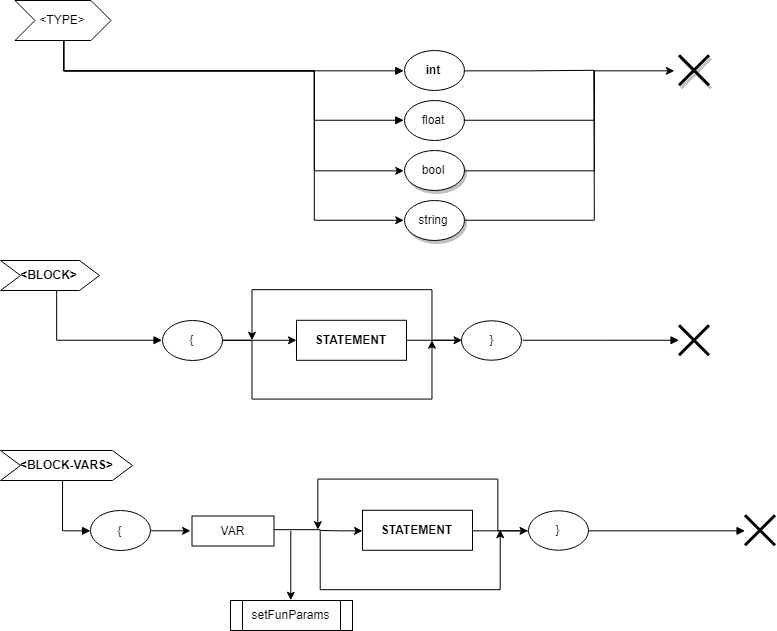
SetTable: Añade la función al directorio de variables.

deleteFunction: Restablece la memoria local y añade el cuádruplo ENDPROC.

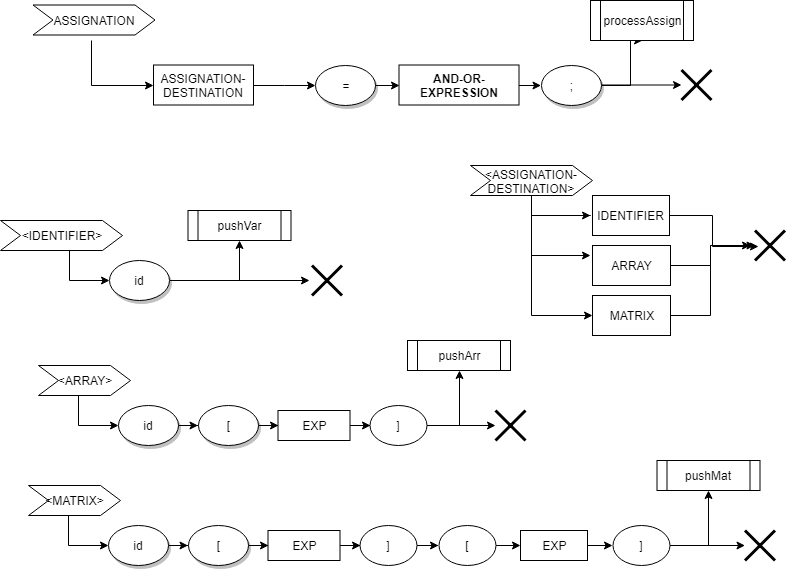
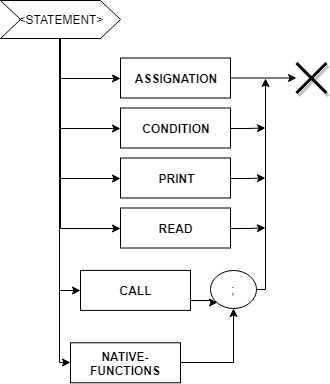
setParams: Guarda los nombres de variables de parámetros para la función actual en memoria.

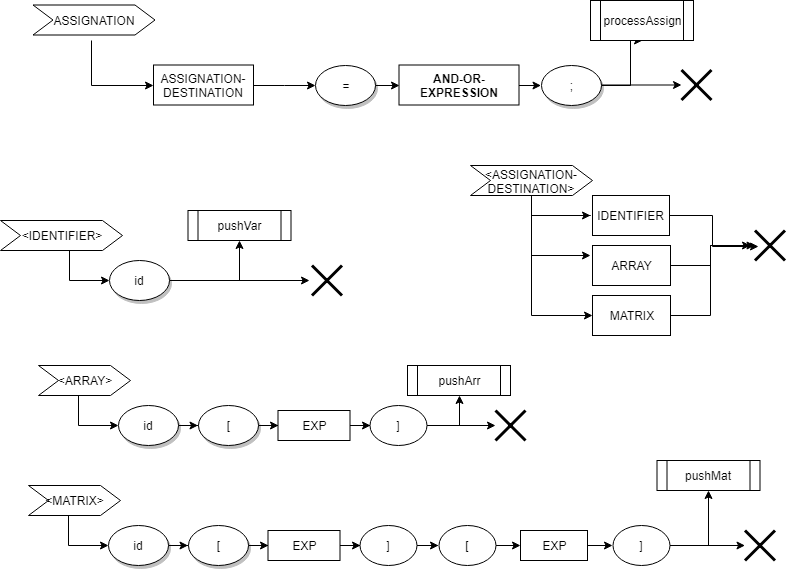
setParamsType: Para cada una de las variables locales, define su tipo y las mete en memoria.





*setFunParams: Establece guarda el contador de parámetros, variables y el cuádruplo de la función en el directorio de funciones.*

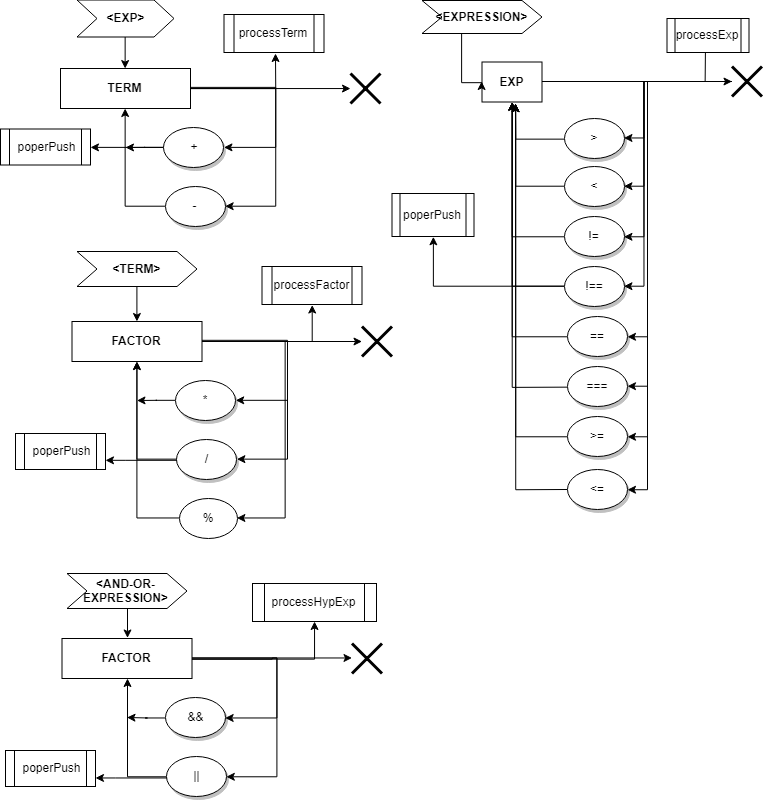


processAssign: Crea el cuádruplo de asignación. Valida el tipo de las expresiones que conforman el cuádruplo.

pushVar: Valida que la variable exista, si existe lo mete a la pila de tipos y operadores.

pushArr: Valida que la variable exista, si existe crea los cuádruplos para calcular la dirección del array y mete el pointer a la pila de operadores con su tipo.

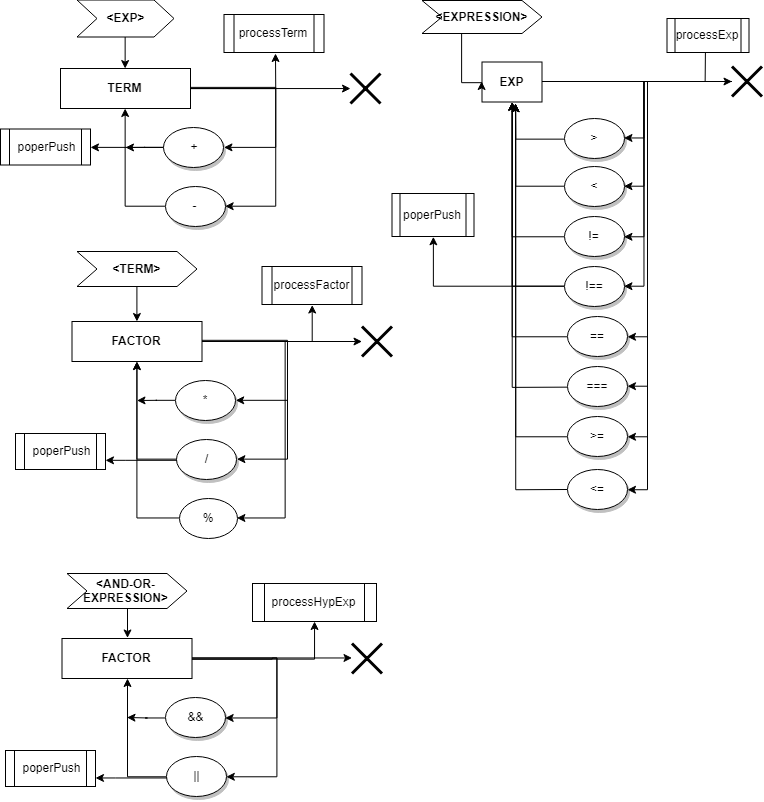
pushMat: Valida que la variable exista, si existe crea los cuádruplos para calcular la dirección de la matriz y mete el pointer a la pila de operadores con su tipo.

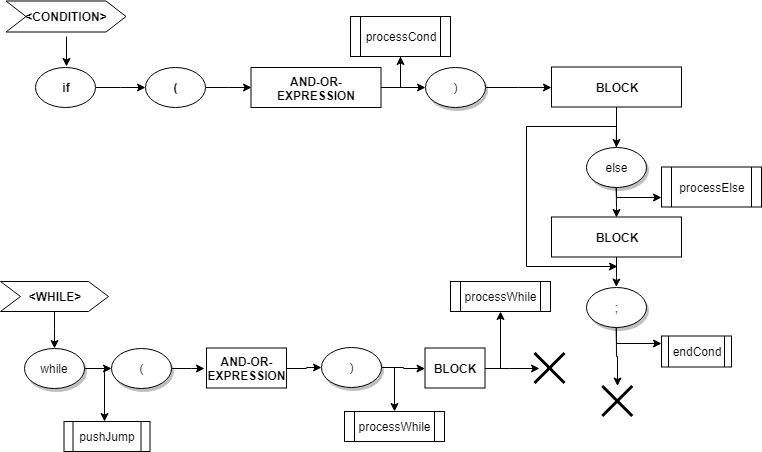


*processTerm: Si llega una suma, resta, división, multiplicación o modulo* *genera el nuevo cuádruplo y mete el temporal con el resultado en la pila de operadores y tipos.*

*processExp: procesa una expresión si llega una operación lógica o aritmética que no sea AND o OR genera el cuadruplo con el codigo del operador.*

*processFactor: Evalúa las operaciones multiplicación, división o módulo, genera el nuevo cuádruplo y mete el temporal con el resultado en la pila de operadores y tipos.*

**

*processHypExp:* Si llega *AND, OR, o cualquier otra expresión lógica o aritmética,* v*alida los tipos, genera el cuádruplo y mete el resultado en la pila de operadores y tipos.*

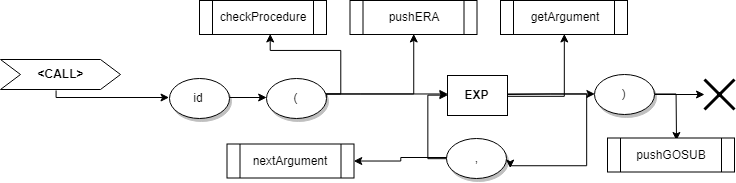
*processCond*: *Procesa una condición a partir de una expresión* l*a expresión debe de ser de tipo booleano, de ser así, genera el cuádruplo con el resultado y mete el salto en la pila.*

processWhile: *Procesa una condición a partir de una expresión* l*a expresión debe de ser de tipo booleano, de ser así, genera el cuádruplo con el resultado y mete el salto en la pila.*

*processElse:* Genera el cuádruplo de salto para un else, a la vez, mete el índice de cuádruplo actual a la pila de saltos y rellena el cuádruplo del if anterior.

endCond: Al término de una condición, rellena el cuádruplo de salto de dicha condición.

pushJump: Guarda el incide del cuádruplo actual en la pila de saltos.

endWhile: G*enera el GOTO de retorno al principio del ciclo* y *rellena a su vez el cuádruplo de salto al que se metió al principio del ciclo.*

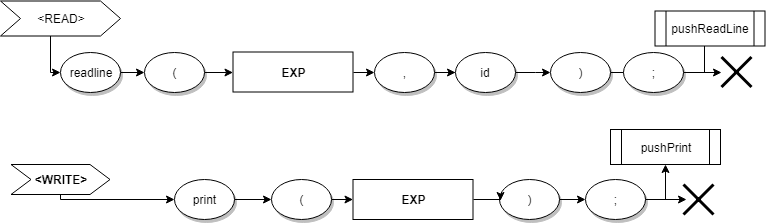
checkProcedure: Valida que el módulo exista.

pushEra: Genera el cuádruplo de ERA para módulo llamado.

getArgument: obtiene de una llamada de función un argumento, lo valida con la tabla de procedimientos y genera el cuádruplo de parámetro.

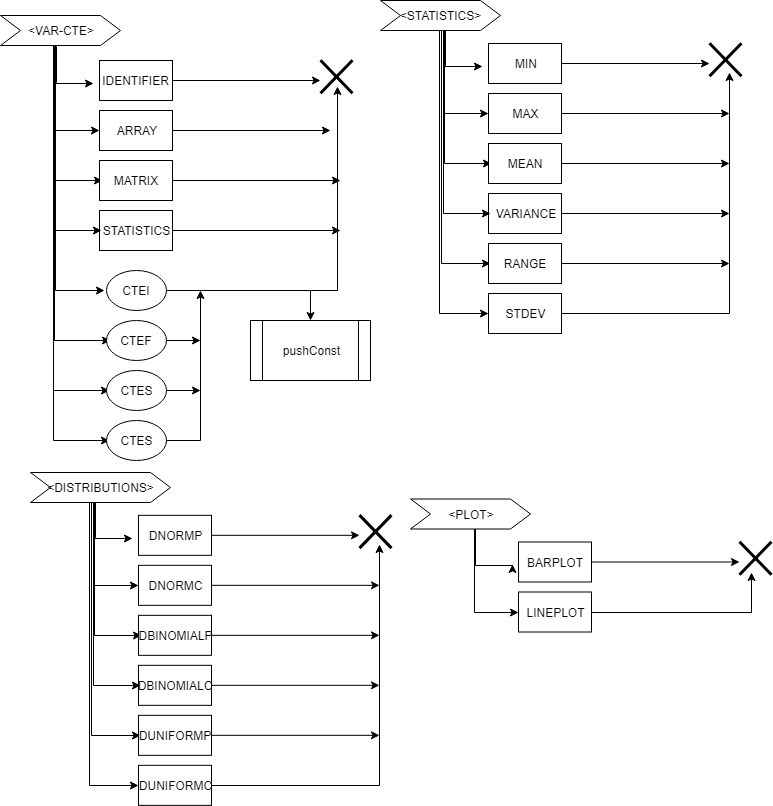
nextArgument: Aumenta el contador de argumentos.

pushGOSUB: Genera el cuadruplo de GOSUB para el procedimiento actual.

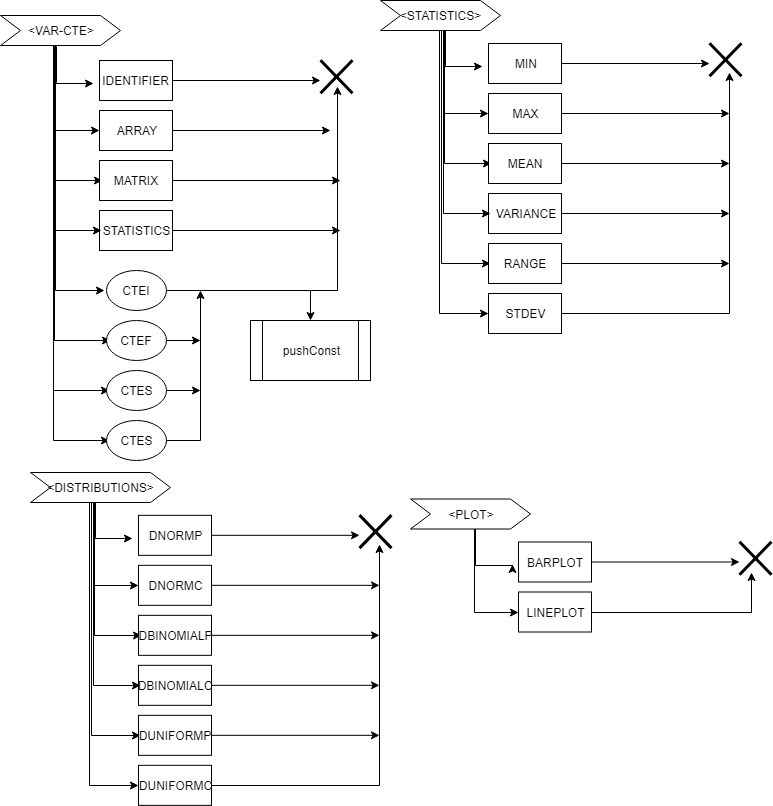


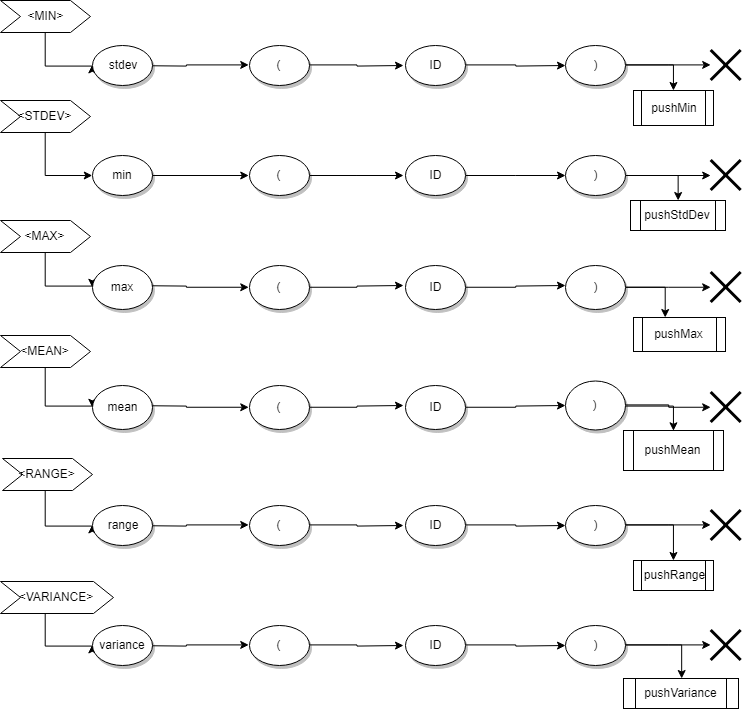
pushReadLine: Valida que el ID de destino sea válido, si lo es, genera el cuádruplo de lectura.

pushPrint: Genera el cuádruplo de salida para la expresión definida.

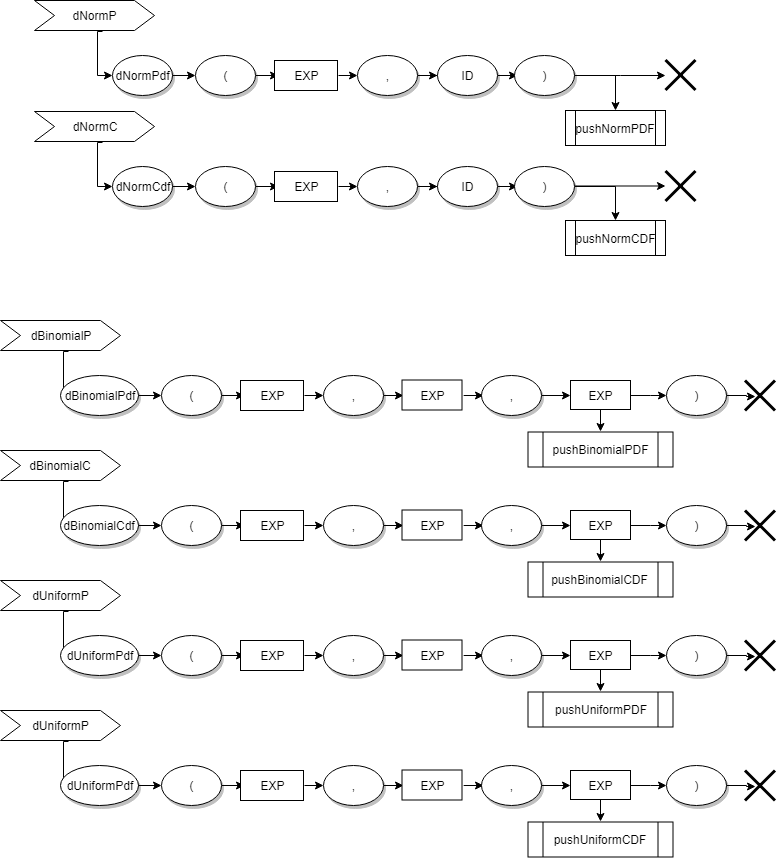


pushConst: Genera la memoria para una constante y mete la constante con su valor a la tabla de constantes.



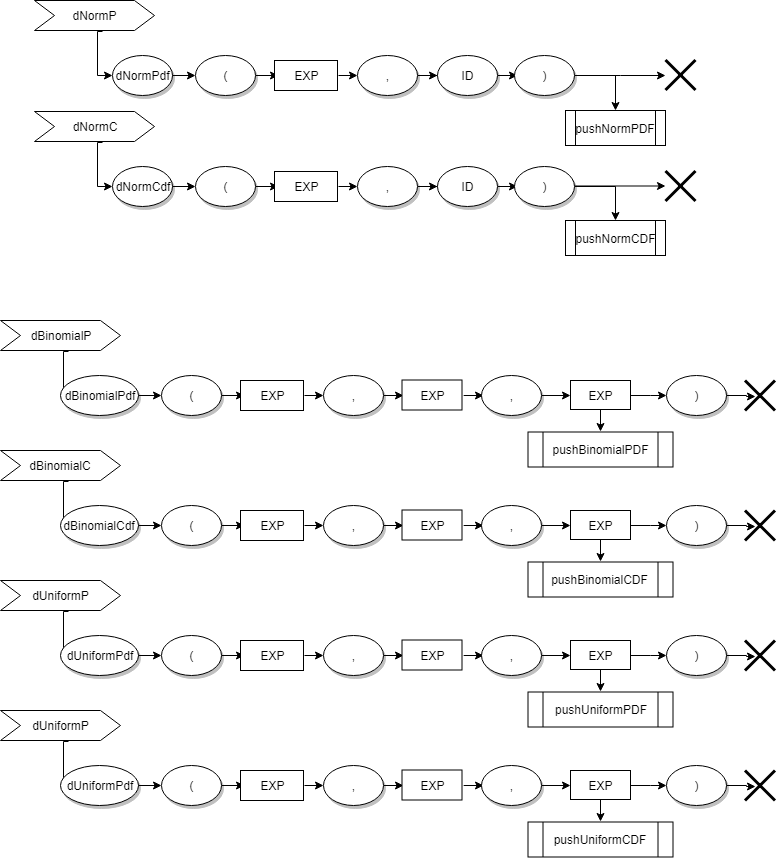


Las acciones semánticas pushMin, pushStdDev, pushMax, pushMean, pushRange y pushVariance generan un cuádruplo para establecer el vector como parámetro y otro cuádruplo con el código de operación correspondiente.



pushNormPDF: Generan con el código de operación para calcular la función de distribución normal, dado un valor y un vector de números.

pushNormCDF: Generan un cuádruplo con el código para calcular la función de distribución acumulada normal, dado un valor y un vector de números.

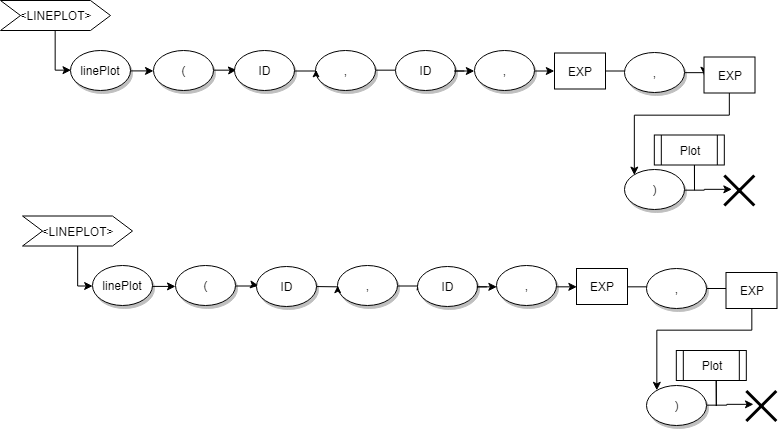


pushUniformPDF: Generan un cuádruplo con el código de operación para calcular la función de distribución uniforme, dados los valores a y b en expresiones.

pushUniformCDF: Generan un cuádruplo con el código de operación para calcular la función de distribución acumulada uniforme, dados los valores a y b en expresiones.

pushBinomialPDF: Generan un cuádruplo con el código de operación para calcular la función de distribución uniforme dados los valores P y N en expresiones.

pushBinomialCDF: Generan un cuádruplo con el código de operación para calcular la función de distribución acumulada uniforme, dados los valores P y N.



Plot: Genera el cuádruplo para crear una gráfica de línea o de barras a partir de dos variables y de el nombre del archivo HTML que se usa para crear la gráfica.

### c.4.4) Tabla de consideraciones semánticas

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| op1 | op 2 | + | - | \* | / | > | < | == | === | != | !== | >= | <= | Notas: |
| int | Int | int | int | int | int | bool | bool | bool | bool | bool | bool | bool | bool |  |
| int | float | float | float | float | float | bool | bool | bool | bool | bool | bool | bool | bool |  |
| int | string | string | X | X | X | X | X | bool | bool | bool | bool | X | X | Concatenate Strings with + |
| int | bool | X | X | X | X | X | X | bool | bool | bool | bool | X | X | 0 == false |
| int | undefined | NaN | NaN | NaN | NaN | bool | bool | bool | bool | bool | bool | bool | bool | Operaciones con undefined son siempre falsas |
| int | NaN | NaN | NaN | NaN | NaN | bool | bool | bool | bool | bool | bool | bool | bool | Operaciones logicas con NaN siempre son falsas |
| float | string | string | X | X | X | X | X | bool | bool | bool | bool | X | X | Concatenate Strings with + |
| float | bool | X | X | X | X | X | X | bool | bool | bool | bool | X | X | 0 == false |
| float | undefined | NaN | NaN | NaN | NaN | bool | bool | bool | bool | bool | bool | bool | bool | Operaciones lógicas con NaN y números siempre son falsas. |
| float | NaN | NaN | NaN | NaN | NaN | bool | bool | bool | bool | bool | bool | bool | bool | Operaciones lógicas con NaN y números siempre son falsas. |
| string | bool | X | X | X | X | X | X | bool | bool | bool | bool | X | X |  |
| string | undefined | string | string | string | String | X | X | bool | bool | bool | bool | X | X | Operaciones lógicas con undefined son siempre falsas. |
| string | NaN | string | string | string | string | X | X | bool | bool | bool | bool | X | X | Operaciones lógicas con NaN son siempre falsas |
| bool | undefined | X | X | X | X | X | X | bool | bool | bool | bool | X | X | Operaciones lógicas con undefined son siempre falsas. |
| bool | NaN | X | X | X | X | X | X | bool | bool | bool | bool | X | X | Operaciones lógicas con NaN son siempre falsas |
| undefined | NaN | X | X | X | X | X | X | bool | bool | bool | bool | X | X | Operaciones lógicas con undefined son siempre falsas. |

## c.5) Descripción detallada del proceso de Administración de Memoria usado en la compilación.

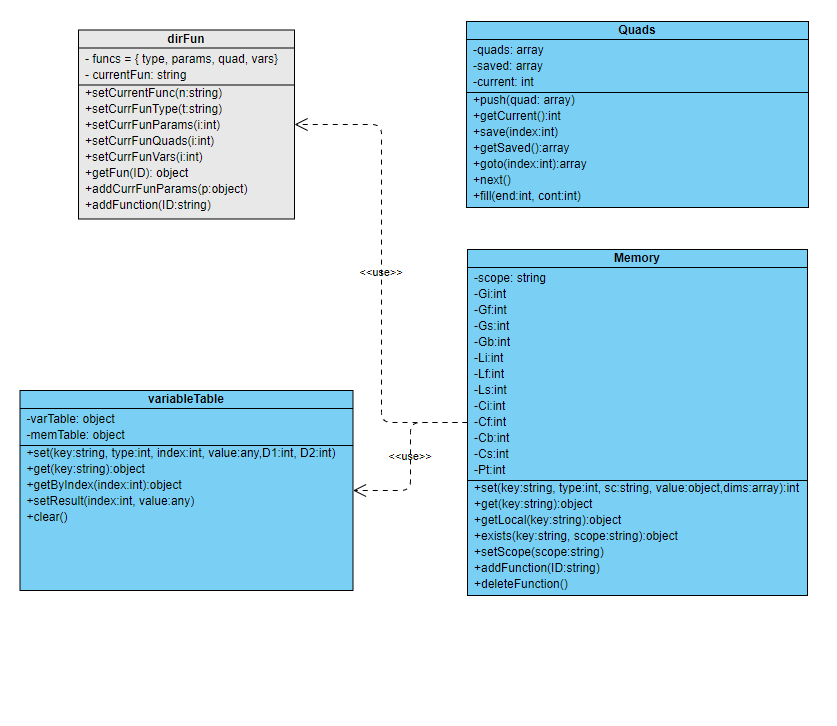
En compilación la memoria virtual se manejaba a partir de la estructura de datos de memory, como administra la unidad virtual para cada nueva variable creada, para ello utiliza variables enteras para cada uno de los tipos de memoria que representan la siguiente unidad de memoria disponible por tipo. La memoria maneja el scope, que se establece por otros métodos de la sintaxis.

La estructura memory maneja el directorio de funciones por la estructura dirFun, que cuenta con un objeto que almacena las funciones con sus respectivos parámetros y cuya llave es el nombre de la función. La estructura dirFun cuenta con sus respectivos módulos de acceso y modificación de variables.

Al llamar al módulo set, se calcula la memoria necesaria a partir del tipo, scope, las dimensiones de la variable y se guardan en una estructura de variableTable. El módulo regresa el índice de la variable en la tabla.

Hay una variableTable para cada uno de los scopes (global, local, constantes y pointers). Cada variableTable maneja dos objetos o arrays asociativos para almacenar la memoria. Uno cuya llave es el nombre de la variable(varTable) y el otro cuya llave es la dirección de memoría. Esto para eficientizar la búsqueda de memoria por llave o por tipo. En el proceso de la compilación solo se guardan los valores de las variables constantes.

La estructura de quads maneja los cuádruplos, que se almacenan en una lista de arrays de dimensión de 4 ([operador, left\_op, right\_op, result]). Durante el proceso de compilación se van agregando a la lista los cuádruplos, usando los índices de memoria creados.



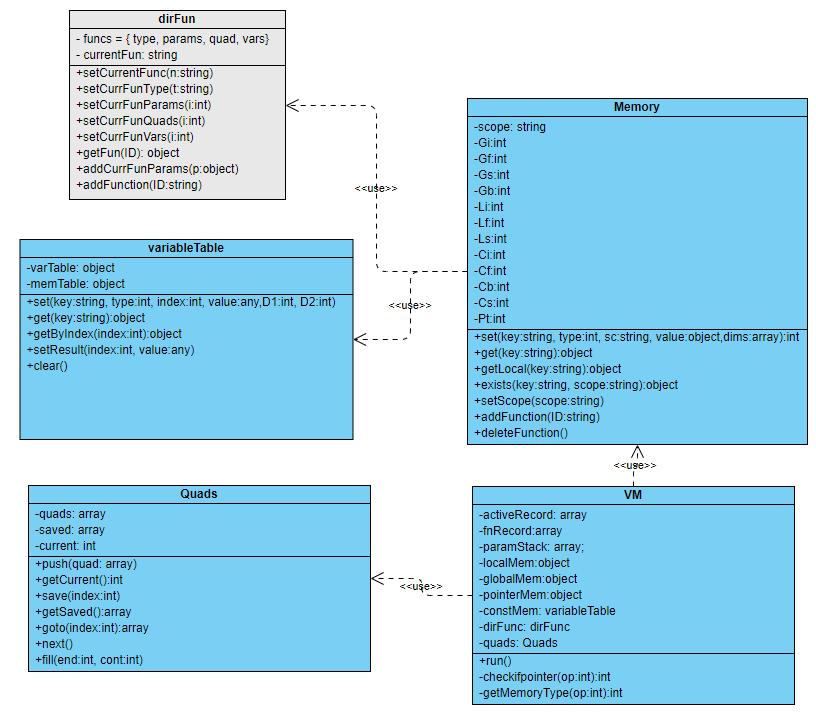
# d). Descripción de la máquina virtual

## d.2) Descripción detallada del proceso de Administración de Memoria en ejecución (Arquitectura).

En ejecución la memoria se manejaba a partir de diferentes objetos de javascript o arrays asociativos para almacenar cada uno de los scopes, cuyo índice en el objeto es la dirección de memoria establecida en compilación.

De las variables de compilación solo se rescatan las variables constantes, con sus valores correspondientes, establecidas previamente en el proceso de compilación. Las demás variables se calculan a partir de las constantes durante la ejecución.

La máquina virtual además guarda de compilación el directorio de funciones (dirFun) para manejar los saltos, y validaciones referentes a módulos. Para manejo de scope, se utiliza el stack activeRecord para guardar la memoria “dormida” de módulos.



# e) Pruebas del funcionamiento del lenguaje

## e.1) Cálculo de Factorial en versión cíclica usando un ciclo de tipo while.

program myProgram:

var n, resultado: int;

main {

readline("N:", n);

resultado = 1;

while(n>0){

resultado = resultado \* n;

n = n - 1;

}

print(resultado);

}

End

1 [ 16, null, null, 1 ]

2 [ 52, 25000, null, 0 ]

3 [ 5, 20000, null, 1 ]

4 [ 10, 0, 20001, 17500 ]

5 [ 17, 17500, null, 10 ]

6 [ 2, 1, 0, 10000 ]

7 [ 5, 10000, null, 1 ]

8 [ 1, 0, 20002, 10001 ]

9 [ 5, 10001, null, 0 ]

10 [ 16, null, null, 3 ]

11 [ 51, 1, null, null ]

Resultado:

>> N: 10

3628800

## e.2) Cálculo de Factorial en versión recursiva usando una función que se llama a si misma.

Código:

function factorial(n: int) {

resultado = resultado \* n;

n = n-1;

if(n>0){

factorial(n);

}

}

main {

resultado = 1;

factorial(10);

print(resultado);

}

end

Cuádruplos:

1 [ 16, null, null, 11 ]

2 [ 2, 0, 10000, 10001 ]

3 [ 5, 10001, null, 0 ]

4 [ 1, 10000, 20000, 10002 ]

5 [ 5, 10002, null, 10000 ]

6 [ 10, 10000, 20001, 17500 ]

7 [ 17, 17500, null, 10 ]

8 [ 20, null, null, 'factorial' ]

9 [ 22, 10000, 0, 'factorial' ]

10 [ 21, null, null, 'factorial' ]

11 [ 18, null, null, null ]

12 [ 5, 20002, null, 0 ]

13 [ 20, null, null, 'factorial' ]

14 [ 22, 20003, 0, 'factorial' ]

15 [ 21, null, null, 'factorial' ]

16 [ 51, 0, null, null ]

Resultado:

3628800

## e.3) Cálculo de Fibonacci en versión cíclica usando un ciclo while.

program myProgram:

var n, actual, anterior, temp, cont: int;

main {

readline("N:", n);

actual = 0;

anterior = 0;

temp = 0;

cont = 0;

while (cont < n) {

if (actual == 0) {

actual = 1;

}

else {

temp = actual;

actual = actual + anterior;

anterior = temp;

}

print(actual);

cont = cont + 1;

}

}

end

1 [ 16, null, null, 1 ]

2 [ 52, 25000, null, 0 ]

3 [ 5, 20000, null, 1 ]

4 [ 5, 20001, null, 2 ]

5 [ 5, 20002, null, 3 ]

6 [ 5, 20003, null, 4 ]

7 [ 12, 4, 0, 17500 ]

8 [ 17, 17500, null, 19 ]

9 [ 7, 1, 20004, 17501 ]

10 [ 17, 17501, null, 12 ]

11 [ 5, 20005, null, 1 ]

12 [ 16, null, null, 16 ]

13 [ 5, 1, null, 3 ]

14 [ 0, 1, 2, 10000 ]

15 [ 5, 10000, null, 1 ]

16 [ 5, 3, null, 2 ]

17 [ 0, 4, 20006, 10001 ]

18 [ 5, 10001, null, 4 ]

19 [ 16, null, null, 6 ]

20 [ 51, 1, null, null ]

>> N: 10

55

## e.4) Cálculo de Fibonacci en versión recursiva usando una función que se llama a si misma.

program myProgram:

var n, actual: int;

function fibonacci(cont:int, temp: int, anterior: int){

if (actual === 0) {

actual = 1;

}

else {

temp = actual;

actual = actual + anterior;

anterior = temp;

}

cont = cont + 1;

if(cont < n){

fibonacci(cont, temp, anterior);

}

}

main {

readline("N:", n);

actual = 0;

fibonacci(0, 0 , 0);

print("Resultado:" + actual);

}

End

Cuádruplos:

1 [ 16, null, null, 19 ]

2 [ 6, 1, 20000, 17500 ]

3 [ 17, 17500, null, 5 ]

4 [ 5, 20001, null, 1 ]

5 [ 16, null, null, 9 ]

6 [ 5, 1, null, 10001 ]

7 [ 0, 1, 10002, 10003 ]

8 [ 5, 10003, null, 1 ]

9 [ 5, 10001, null, 10002 ]

10 [ 0, 10000, 20002, 10004 ]

11 [ 5, 10004, null, 10000 ]

12 [ 12, 10000, 0, 17501 ]

13 [ 17, 17501, null, 18 ]

14 [ 20, null, null, 'fibonacci' ]

15 [ 22, 10000, 0, 'fibonacci' ]

16 [ 22, 10001, 1, 'fibonacci' ]

17 [ 22, 10002, 2, 'fibonacci' ]

18 [ 21, null, null, 'fibonacci' ]

19 [ 18, null, null, null ]

20 [ 52, 25000, null, 0 ]

21 [ 5, 20003, null, 1 ]

22 [ 20, null, null, 'fibonacci' ]

23 [ 22, 20004, 0, 'fibonacci' ]

24 [ 22, 20005, 1, 'fibonacci' ]

25 [ 22, 20006, 2, 'fibonacci' ]

26 [ 21, null, null, 'fibonacci' ]

27 [ 0, 25001, 1, 15000 ]

28 [ 51, 15000, null, null ]

Resultados:

>> N: 10

Resultado:55

## e.5) Sort de un vector: Un bubble sort de un arreglo de números.

program myProgram:

var a, b, lenght, swap, i[5]: int;

var swapped:bool;

main {

i[0] = 3;

i[1] = 1;

i[2] = 2;

i[3] = 2;

i[4] = 5;

lenght = 5;

swapped = true;

while (swapped) {

a = 0;

swapped=false;

while (a < lenght - 1) {

b = a + 1;

if (i[a] > i[b]) {

swap = i[a];

i[a] = i[b];

i[b] = swap;

swapped=true;

}

a = a + 1;

}

}

print("sorted:");

print(i[0]);

print(i[1]);

print(i[2]);

print(i[3]);

print(i[4]);

}

end

Cuadruplos:

1 [ 16, null, null, 1 ]

2 [ 23, 20000, 0, 5 ]

3 [ 0, 20000, 20001, 10000 ]

4 [ 5, 20002, null, '(10000)' ]

5 [ 23, 20003, 0, 5 ]

6 [ 0, 20003, 20004, 10001 ]

7 [ 5, 20005, null, '(10001)' ]

8 [ 23, 20006, 0, 5 ]

9 [ 0, 20006, 20007, 10002 ]

10 [ 5, 20008, null, '(10002)' ]

11 [ 23, 20009, 0, 5 ]

12 [ 0, 20009, 20010, 10003 ]

13 [ 5, 20011, null, '(10003)' ]

14 [ 23, 20012, 0, 5 ]

15 [ 0, 20012, 20013, 10004 ]

16 [ 5, 20014, null, '(10004)' ]

17 [ 5, 20015, null, 2 ]

18 [ 5, 27500, null, 7500 ]

19 [ 17, 7500, null, 48 ]

20 [ 5, 20016, null, 0 ]

21 [ 5, 27501, null, 7500 ]

22 [ 1, 2, 20017, 10005 ]

23 [ 12, 0, 10005, 17500 ]

24 [ 17, 17500, null, 47 ]

25 [ 0, 0, 20018, 10006 ]

26 [ 5, 10006, null, 1 ]

27 [ 23, 0, 0, 5 ]

28 [ 0, 0, 20019, 10007 ]

29 [ 23, 1, 0, 5 ]

30 [ 0, 1, 20020, 10008 ]

31 [ 10, '(10007)', '(10008)', 17501 ]

32 [ 17, 17501, null, 44 ]

33 [ 23, 0, 0, 5 ]

34 [ 0, 0, 20021, 10009 ]

35 [ 5, '(10009)', null, 3 ]

36 [ 23, 0, 0, 5 ]

37 [ 0, 0, 20022, 10010 ]

38 [ 23, 1, 0, 5 ]

39 [ 0, 1, 20023, 10011 ]

40 [ 5, '(10011)', null, '(10010)' ]

41 [ 23, 1, 0, 5 ]

42 [ 0, 1, 20024, 10012 ]

43 [ 5, 3, null, '(10012)' ]

44 [ 5, 27502, null, 7500 ]

45 [ 0, 0, 20025, 10013 ]

46 [ 5, 10013, null, 0 ]

47 [ 16, null, null, 21 ]

48 [ 16, null, null, 18 ]

49 [ 51, 25000, null, null ]

50 [ 23, 20026, 0, 5 ]

51 [ 0, 20026, 20027, 10014 ]

52 [ 51, '(10014)', null, null ]

53 [ 23, 20028, 0, 5 ]

54 [ 0, 20028, 20029, 10015 ]

55 [ 51, '(10015)', null, null ]

56 [ 23, 20030, 0, 5 ]

57 [ 0, 20030, 20031, 10016 ]

58 [ 51, '(10016)', null, null ]

59 [ 23, 20032, 0, 5 ]

60 [ 0, 20032, 20033, 10017 ]

61 [ 51, '(10017)', null, null ]

62 [ 23, 20034, 0, 5 ]

63 [ 0, 20034, 20035, 10018 ]

64 [ 51, '(10018)', null, null ]

Resultados:

sorted:

1

2

2

3

5

## e.6) Find en un vector: Una búsqueda de un número en un arreglo de números.

program myProgram:

var a[5], lenght: int;

var swapped:bool;

function find(b:int){

var i:int;

i = 0;

while (i < lenght - 1) {

if (a[i] == b) {

print("index find: " + i);

}

i = i + 1;

}

}

main {

a[0] = 3;

a[1] = 1;

a[2] = 2;

a[3] = 3;

a[4] = 5;

lenght = 5;

find(3);

}

End

Cuádruplos:

1 [ 16, null, null, 15 ]

2 [ 5, 20000, null, 10001 ]

3 [ 1, 5, 20001, 10002 ]

4 [ 12, 10001, 10002, 17500 ]

5 [ 17, 17500, null, 14 ]

6 [ 23, 10001, 0, 5 ]

7 [ 0, 10001, 20002, 10003 ]

8 [ 7, '(10003)', 10000, 17501 ]

9 [ 17, 17501, null, 11 ]

10 [ 0, 25000, 10001, 15000 ]

11 [ 51, 15000, null, null ]

12 [ 0, 10001, 20003, 10004 ]

13 [ 5, 10004, null, 10001 ]

14 [ 16, null, null, 2 ]

15 [ 18, null, null, null ]

16 [ 23, 20004, 0, 5 ]

17 [ 0, 20004, 20005, 10000 ]

18 [ 5, 20006, null, '(10000)' ]

19 [ 23, 20007, 0, 5 ]

20 [ 0, 20007, 20008, 10001 ]

21 [ 5, 20009, null, '(10001)' ]

22 [ 23, 20010, 0, 5 ]

23 [ 0, 20010, 20011, 10002 ]

24 [ 5, 20012, null, '(10002)' ]

25 [ 23, 20013, 0, 5 ]

26 [ 0, 20013, 20014, 10003 ]

27 [ 5, 20015, null, '(10003)' ]

28 [ 23, 20016, 0, 5 ]

29 [ 0, 20016, 20017, 10004 ]

30 [ 5, 20018, null, '(10004)' ]

31 [ 5, 20019, null, 5 ]

32 [ 20, null, null, 'find' ]

33 [ 22, 20020, 0, 'find' ]

34 [ 21, null, null, 'find' ]

Resultados:

index find: 0

index find: 3

## e.7) Una multiplicación de dos matrices de tres por tres guardadas en una matriz, el resultado debe de quedar guardado en una tercera matriz.

program myProgram:

var a1[3][3], a2[3][3], a3[3][3]:int;

var i,j,k, aux1, aux2, aux3, aux4 ,lenght: int;

main {

a1[0][0] = 2;

a1[0][1] = 0;

a1[0][2] = 1;

a1[1][0] = 3;

a1[1][1] = 0;

a1[1][2] = 0;

a1[2][0] = 5;

a1[2][1] = 1;

a1[2][2] = 1;

a2[0][0] = 1;

a2[0][1] = 0;

a2[0][2] = 1;

a2[1][0] = 1;

a2[1][1] = 2;

a2[1][2] = 1;

a2[2][0] = 1;

a2[2][1] = 1;

a2[2][2] = 0;

a3[0][0] = 0;

a3[0][1] = 0;

a3[0][2] = 0;

a3[1][0] = 0;

a3[1][1] = 0;

a3[1][2] = 0;

a3[2][0] = 0;

a3[2][1] = 0;

a3[2][2] = 0;

lenght = 3;

i = 0;

while(i < lenght) {

j = 0;

while(j < lenght) {

k = 0;

while(k < lenght) {

aux1 = a1[i][k];

aux2 = a2[k][j];

aux3 = a3[i][j];

aux3 = aux1 \* aux2 + aux3;

a3[i][j] = aux3;

k = k+1;

}

j = j+1;

}

i = i+1;

}

i = 0;

while(i < lenght) {

j = 0;

while(j < lenght) {

print(a3[i][j]);

j = j+1;

}

i = i+1;

}

}

End

1 [ 16, null, null, 1 ]

2 [ 23, 20000, 0, 3 ]

3 [ 2, 20000, 20002, 10000 ]

4 [ 23, 20001, 0, 3 ]

5 [ 0, 10000, 20001, 10001 ]

6 [ 0, 10001, 20003, 10002 ]

7 [ 5, 20004, null, '(10002)' ]

8 [ 23, 20005, 0, 3 ]

9 [ 2, 20005, 20007, 10003 ]

10 [ 23, 20006, 0, 3 ]

11 [ 0, 10003, 20006, 10004 ]

12 [ 0, 10004, 20008, 10005 ]

13 [ 5, 20009, null, '(10005)' ]

14 [ 23, 20010, 0, 3 ]

15 [ 2, 20010, 20012, 10006 ]

16 [ 23, 20011, 0, 3 ]

17 [ 0, 10006, 20011, 10007 ]

18 [ 0, 10007, 20013, 10008 ]

19 [ 5, 20014, null, '(10008)' ]

20 [ 23, 20015, 0, 3 ]

21 [ 2, 20015, 20017, 10009 ]

22 [ 23, 20016, 0, 3 ]

23 [ 0, 10009, 20016, 10010 ]

24 [ 0, 10010, 20018, 10011 ]

25 [ 5, 20019, null, '(10011)' ]

26 [ 23, 20020, 0, 3 ]

27 [ 2, 20020, 20022, 10012 ]

28 [ 23, 20021, 0, 3 ]

29 [ 0, 10012, 20021, 10013 ]

30 [ 0, 10013, 20023, 10014 ]

31 [ 5, 20024, null, '(10014)' ]

32 [ 23, 20025, 0, 3 ]

33 [ 2, 20025, 20027, 10015 ]

34 [ 23, 20026, 0, 3 ]

35 [ 0, 10015, 20026, 10016 ]

36 [ 0, 10016, 20028, 10017 ]

37 [ 5, 20029, null, '(10017)' ]

38 [ 23, 20030, 0, 3 ]

39 [ 2, 20030, 20032, 10018 ]

40 [ 23, 20031, 0, 3 ]

41 [ 0, 10018, 20031, 10019 ]

42 [ 0, 10019, 20033, 10020 ]

43 [ 5, 20034, null, '(10020)' ]

44 [ 23, 20035, 0, 3 ]

45 [ 2, 20035, 20037, 10021 ]

46 [ 23, 20036, 0, 3 ]

47 [ 0, 10021, 20036, 10022 ]

48 [ 0, 10022, 20038, 10023 ]

49 [ 5, 20039, null, '(10023)' ]

50 [ 23, 20040, 0, 3 ]

51 [ 2, 20040, 20042, 10024 ]

52 [ 23, 20041, 0, 3 ]

53 [ 0, 10024, 20041, 10025 ]

54 [ 0, 10025, 20043, 10026 ]

55 [ 5, 20044, null, '(10026)' ]

56 [ 23, 20045, 0, 3 ]

57 [ 2, 20045, 20047, 10027 ]

58 [ 23, 20046, 0, 3 ]

59 [ 0, 10027, 20046, 10028 ]

60 [ 0, 10028, 20048, 10029 ]

61 [ 5, 20049, null, '(10029)' ]

62 [ 23, 20050, 0, 3 ]

63 [ 2, 20050, 20052, 10030 ]

64 [ 23, 20051, 0, 3 ]

65 [ 0, 10030, 20051, 10031 ]

66 [ 0, 10031, 20053, 10032 ]

67 [ 5, 20054, null, '(10032)' ]

68 [ 23, 20055, 0, 3 ]

69 [ 2, 20055, 20057, 10033 ]

70 [ 23, 20056, 0, 3 ]

71 [ 0, 10033, 20056, 10034 ]

72 [ 0, 10034, 20058, 10035 ]

73 [ 5, 20059, null, '(10035)' ]

74 [ 23, 20060, 0, 3 ]

75 [ 2, 20060, 20062, 10036 ]

76 [ 23, 20061, 0, 3 ]

77 [ 0, 10036, 20061, 10037 ]

78 [ 0, 10037, 20063, 10038 ]

79 [ 5, 20064, null, '(10038)' ]

80 [ 23, 20065, 0, 3 ]

81 [ 2, 20065, 20067, 10039 ]

82 [ 23, 20066, 0, 3 ]

83 [ 0, 10039, 20066, 10040 ]

84 [ 0, 10040, 20068, 10041 ]

85 [ 5, 20069, null, '(10041)' ]

86 [ 23, 20070, 0, 3 ]

87 [ 2, 20070, 20072, 10042 ]

88 [ 23, 20071, 0, 3 ]

89 [ 0, 10042, 20071, 10043 ]

90 [ 0, 10043, 20073, 10044 ]

91 [ 5, 20074, null, '(10044)' ]

92 [ 23, 20075, 0, 3 ]

93 [ 2, 20075, 20077, 10045 ]

94 [ 23, 20076, 0, 3 ]

95 [ 0, 10045, 20076, 10046 ]

96 [ 0, 10046, 20078, 10047 ]

97 [ 5, 20079, null, '(10047)' ]

98 [ 23, 20080, 0, 3 ]

99 [ 2, 20080, 20082, 10048 ]

100 [ 23, 20081, 0, 3 ]

101 [ 0, 10048, 20081, 10049 ]

102 [ 0, 10049, 20083, 10050 ]

103 [ 5, 20084, null, '(10050)' ]

104 [ 23, 20085, 0, 3 ]

105 [ 2, 20085, 20087, 10051 ]

106 [ 23, 20086, 0, 3 ]

107 [ 0, 10051, 20086, 10052 ]

108 [ 0, 10052, 20088, 10053 ]

109 [ 5, 20089, null, '(10053)' ]

110 [ 23, 20090, 0, 3 ]

111 [ 2, 20090, 20092, 10054 ]

112 [ 23, 20091, 0, 3 ]

113 [ 0, 10054, 20091, 10055 ]

114 [ 0, 10055, 20093, 10056 ]

115 [ 5, 20094, null, '(10056)' ]

116 [ 23, 20095, 0, 3 ]

117 [ 2, 20095, 20097, 10057 ]

118 [ 23, 20096, 0, 3 ]

119 [ 0, 10057, 20096, 10058 ]

120 [ 0, 10058, 20098, 10059 ]

121 [ 5, 20099, null, '(10059)' ]

122 [ 23, 20100, 0, 3 ]

123 [ 2, 20100, 20102, 10060 ]

124 [ 23, 20101, 0, 3 ]

125 [ 0, 10060, 20101, 10061 ]

126 [ 0, 10061, 20103, 10062 ]

127 [ 5, 20104, null, '(10062)' ]

128 [ 23, 20105, 0, 3 ]

129 [ 2, 20105, 20107, 10063 ]

130 [ 23, 20106, 0, 3 ]

131 [ 0, 10063, 20106, 10064 ]

132 [ 0, 10064, 20108, 10065 ]

133 [ 5, 20109, null, '(10065)' ]

134 [ 23, 20110, 0, 3 ]

135 [ 2, 20110, 20112, 10066 ]

136 [ 23, 20111, 0, 3 ]

137 [ 0, 10066, 20111, 10067 ]

138 [ 0, 10067, 20113, 10068 ]

139 [ 5, 20114, null, '(10068)' ]

140 [ 23, 20115, 0, 3 ]

141 [ 2, 20115, 20117, 10069 ]

142 [ 23, 20116, 0, 3 ]

143 [ 0, 10069, 20116, 10070 ]

144 [ 0, 10070, 20118, 10071 ]

145 [ 5, 20119, null, '(10071)' ]

146 [ 23, 20120, 0, 3 ]

147 [ 2, 20120, 20122, 10072 ]

148 [ 23, 20121, 0, 3 ]

149 [ 0, 10072, 20121, 10073 ]

150 [ 0, 10073, 20123, 10074 ]

151 [ 5, 20124, null, '(10074)' ]

152 [ 23, 20125, 0, 3 ]

153 [ 2, 20125, 20127, 10075 ]

154 [ 23, 20126, 0, 3 ]

155 [ 0, 10075, 20126, 10076 ]

156 [ 0, 10076, 20128, 10077 ]

157 [ 5, 20129, null, '(10077)' ]

158 [ 23, 20130, 0, 3 ]

159 [ 2, 20130, 20132, 10078 ]

160 [ 23, 20131, 0, 3 ]

161 [ 0, 10078, 20131, 10079 ]

162 [ 0, 10079, 20133, 10080 ]

163 [ 5, 20134, null, '(10080)' ]

164 [ 5, 20135, null, 34 ]

165 [ 5, 20136, null, 27 ]

166 [ 12, 27, 34, 17500 ]

167 [ 17, 17500, null, 209 ]

168 [ 5, 20137, null, 28 ]

169 [ 12, 28, 34, 17501 ]

170 [ 17, 17501, null, 206 ]

171 [ 5, 20138, null, 29 ]

172 [ 12, 29, 34, 17502 ]

173 [ 17, 17502, null, 203 ]

174 [ 23, 27, 0, 3 ]

175 [ 2, 27, 20139, 10081 ]

176 [ 23, 29, 0, 3 ]

177 [ 0, 10081, 29, 10082 ]

178 [ 0, 10082, 20140, 10083 ]

179 [ 5, '(10083)', null, 30 ]

180 [ 23, 29, 0, 3 ]

181 [ 2, 29, 20141, 10084 ]

182 [ 23, 28, 0, 3 ]

183 [ 0, 10084, 28, 10085 ]

184 [ 0, 10085, 20142, 10086 ]

185 [ 5, '(10086)', null, 31 ]

186 [ 23, 27, 0, 3 ]

187 [ 2, 27, 20143, 10087 ]

188 [ 23, 28, 0, 3 ]

189 [ 0, 10087, 28, 10088 ]

190 [ 0, 10088, 20144, 10089 ]

191 [ 5, '(10089)', null, 32 ]

192 [ 2, 30, 31, 10090 ]

193 [ 0, 10090, 32, 10091 ]

194 [ 5, 10091, null, 32 ]

195 [ 23, 27, 0, 3 ]

196 [ 2, 27, 20145, 10092 ]

197 [ 23, 28, 0, 3 ]

198 [ 0, 10092, 28, 10093 ]

199 [ 0, 10093, 20146, 10094 ]

200 [ 5, 32, null, '(10094)' ]

201 [ 0, 29, 20147, 10095 ]

202 [ 5, 10095, null, 29 ]

203 [ 16, null, null, 171 ]

204 [ 0, 28, 20148, 10096 ]

205 [ 5, 10096, null, 28 ]

206 [ 16, null, null, 168 ]

207 [ 0, 27, 20149, 10097 ]

208 [ 5, 10097, null, 27 ]

209 [ 16, null, null, 165 ]

210 [ 5, 20150, null, 27 ]

211 [ 12, 27, 34, 17503 ]

212 [ 17, 17503, null, 227 ]

213 [ 5, 20151, null, 28 ]

214 [ 12, 28, 34, 17504 ]

215 [ 17, 17504, null, 224 ]

216 [ 23, 27, 0, 3 ]

217 [ 2, 27, 20152, 10098 ]

218 [ 23, 28, 0, 3 ]

219 [ 0, 10098, 28, 10099 ]

220 [ 0, 10099, 20153, 10100 ]

221 [ 51, '(10100)', null, null ]

222 [ 0, 28, 20154, 10101 ]

223 [ 5, 10101, null, 28 ]

224 [ 16, null, null, 213 ]

225 [ 0, 27, 20155, 10102 ]

226 [ 5, 10102, null, 27 ]

227 [ 16, null, null, 210 ]

Resultado:

3

1

2

3

0

3

7

3

6

## e.8) Obtener media, varianza, máximo, mínimo, desviación estándar y rango de un vector de datos.

**Código:**

program myProgram:

var a1[3][3], b:int;

main {

a1[0][0] = 20;

a1[0][1] = 10;

a1[1][0] = 30;

b = max(a1);

print("max: " + b);

b = min(a1);

print("min: " + b);

b = mean(a1);

print("mean: " + b);

b = variance(a1);

print("variance: " + b);

b = stdev(a1);

print("stdev: " + b);

b = range(a1);

print("range: " + b);

}

End

**Cuádruplos:**

1 [ 16, null, null, 1 ]

2 [ 23, 20000, 0, 3 ]

3 [ 2, 20000, 20002, 10000 ]

4 [ 23, 20001, 0, 3 ]

5 [ 0, 10000, 20001, 10001 ]

6 [ 0, 10001, 20003, 10002 ]

7 [ 5, 20004, null, '(10002)' ]

8 [ 23, 20005, 0, 3 ]

9 [ 2, 20005, 20007, 10003 ]

10 [ 23, 20006, 0, 3 ]

11 [ 0, 10003, 20006, 10004 ]

12 [ 0, 10004, 20008, 10005 ]

13 [ 5, 20009, null, '(10005)' ]

14 [ 23, 20010, 0, 3 ]

15 [ 2, 20010, 20012, 10006 ]

16 [ 23, 20011, 0, 3 ]

17 [ 0, 10006, 20011, 10007 ]

18 [ 0, 10007, 20013, 10008 ]

19 [ 5, 20014, null, '(10008)' ]

20 [ 40, 0, 9, null ]

21 [ 31, 0, null, 12500 ]

22 [ 5, 12500, null, 9 ]

23 [ 0, 25000, 9, 15000 ]

24 [ 51, 15000, null, null ]

25 [ 40, 0, 9, null ]

26 [ 32, 0, null, 12501 ]

27 [ 5, 12501, null, 9 ]

28 [ 0, 25001, 9, 15001 ]

29 [ 51, 15001, null, null ]

30 [ 40, 0, 9, null ]

31 [ 35, 0, null, 12502 ]

32 [ 5, 12502, null, 9 ]

33 [ 0, 25002, 9, 15002 ]

34 [ 51, 15002, null, null ]

35 [ 40, 0, 9, null ]

36 [ 34, 0, null, 12503 ]

37 [ 5, 12503, null, 9 ]

38 [ 0, 25003, 9, 15003 ]

39 [ 51, 15003, null, null ]

40 [ 40, 0, 9, null ]

41 [ 30, 0, null, 12504 ]

42 [ 5, 12504, null, 9 ]

43 [ 0, 25004, 9, 15004 ]

44 [ 51, 15004, null, null ]

45 [ 40, 0, 9, null ]

46 [ 33, 0, null, 12505 ]

47 [ 5, 12505, null, 9 ]

48 [ 0, 25005, 9, 15005 ]

49 [ 51, 15005, null, null ]

**Resultados:**

max: 30

min: 10

mean: 17.5

variance: 51.171875

stdev: 15.237817619837484

range: 20

## e.9) Obtener la probabilidad y la probabilidad acumulada normal de un vector de datos.

program myProgram:

var a1[3], b:int;

main {

a1[0] = 20;

a1[2] = 10;

a1[1] = 30;

b = dNormPdf(20,a1);

print("PDF: " + b);

b = dNormCdf(20,a1);

print("CDF: " + b);

}

end

Cuádruplos:

1 [ 16, null, null, 1 ]

2 [ 23, 20000, 0, 3 ]

3 [ 0, 20000, 20001, 10000 ]

4 [ 5, 20002, null, '(10000)' ]

5 [ 23, 20003, 0, 3 ]

6 [ 0, 20003, 20004, 10001 ]

7 [ 5, 20005, null, '(10001)' ]

8 [ 23, 20006, 0, 3 ]

9 [ 0, 20006, 20007, 10002 ]

10 [ 5, 20008, null, '(10002)' ]

11 [ 40, 0, 3, null ]

12 [ 41, 0, 20009, 12500 ]

13 [ 5, 12500, null, 3 ]

14 [ 0, 25000, 3, 15000 ]

15 [ 51, 15000, null, null ]

16 [ 40, 0, 3, null ]

17 [ 42, 0, 20010, 12501 ]

18 [ 5, 12501, null, 3 ]

19 [ 0, 25001, 3, 15001 ]

20 [ 51, 15001, null, null ]

Resultados:

PDF: 0.04886025119029199

CDF: 0.6724814597526316

## e.10) Obtener la probabilidad y la probabilidad acumulada binomial a partir de los parámetros p = 0.2 y n = 4 y x = 2.

program myProgram:

var a1[3], b:int;

main {

a1[0] = 20;

a1[2] = 10;

a1[1] = 30;

b = dBinomialPdf(2, 4, 0.2f);

print("PDF: " + b);

b = dBinomialCdf(2 ,4, 0.2f);

print("CDF: " + b);

}

end

1 [ 16, null, null, 1 ]

2 [ 23, 20000, 0, 3 ]

3 [ 0, 20000, 20001, 10000 ]

4 [ 5, 20002, null, '(10000)' ]

5 [ 23, 20003, 0, 3 ]

6 [ 0, 20003, 20004, 10001 ]

7 [ 5, 20005, null, '(10001)' ]

8 [ 23, 20006, 0, 3 ]

9 [ 0, 20006, 20007, 10002 ]

10 [ 5, 20008, null, '(10002)' ]

11 [ 39, null, null, 22500 ]

12 [ 39, null, null, 20010 ]

13 [ 39, null, null, 20009 ]

14 [ 45, null, null, 12500 ]

15 [ 5, 12500, null, 3 ]

16 [ 0, 25000, 3, 15000 ]

17 [ 51, 15000, null, null ]

18 [ 39, null, null, 22501 ]

19 [ 39, null, null, 20012 ]

20 [ 39, null, null, 20011 ]

21 [ 46, null, null, 12501 ]

22 [ 5, 12501, null, 3 ]

23 [ 0, 25001, 3, 15001 ]

24 [ 51, 15001, null, null ]

Resultados:

PDF: 0.15360000000000007

CDF: 0.9728000000000003

## e.11) Obtener la probabilidad y la probabilidad acumulada uniforme a partir del rango inferior = 7 y superior = 13 para una variable x = 10.5f.

program myProgram:

var a1[3], b:int;

main {

a1[0] = 20;

a1[2] = 10;

a1[1] = 30;

b = dUniformPdf(10.5f, 7,13);

print("PDF: " + b);

b = dUniformCdf(10.4f ,7,13);

print("CDF: " + b);

}

End

Cuádruplos:

1 [ 16, null, null, 1 ]

2 [ 23, 20000, 0, 3 ]

3 [ 0, 20000, 20001, 10000 ]

4 [ 5, 20002, null, '(10000)' ]

5 [ 23, 20003, 0, 3 ]

6 [ 0, 20003, 20004, 10001 ]

7 [ 5, 20005, null, '(10001)' ]

8 [ 23, 20006, 0, 3 ]

9 [ 0, 20006, 20007, 10002 ]

10 [ 5, 20008, null, '(10002)' ]

11 [ 39, null, null, 20010 ]

12 [ 39, null, null, 20009 ]

13 [ 39, null, null, 22500 ]

14 [ 43, null, null, 12500 ]

15 [ 5, 12500, null, 3 ]

16 [ 0, 25000, 3, 15000 ]

17 [ 51, 15000, null, null ]

18 [ 39, null, null, 20012 ]

19 [ 39, null, null, 20011 ]

20 [ 39, null, null, 22501 ]

21 [ 44, null, null, 12501 ]

22 [ 5, 12501, null, 3 ]

23 [ 0, 25001, 3, 15001 ]

24 [ 51, 15001, null, null ]

Resultados:

PDF: 0.16666666666666666

CDF: 0.5666666666666668

## e.12) Generar una gráfica de líneas a partir de un array de datos y un array de etiquetas.

program myProgram:

var series[3], b:int;

var labels[3]: string;

main {

labels[0] = "primero";

labels[2] = "segundo";

labels[1] = "tercero";

series[0] = 10;

series[1] = 20;

series[2] = 30;

barPlot(series,labels, "index.html",”X Axis(units)”, “Y Axis(units)”);

}

End

1 [ 16, null, null, 1 ]

2 [ 23, 20000, 0, 3 ]

3 [ 0, 20000, 20001, 15000 ]

4 [ 5, 25000, null, '(15000)' ]

5 [ 23, 20002, 0, 3 ]

6 [ 0, 20002, 20003, 15001 ]

7 [ 5, 25001, null, '(15001)' ]

8 [ 23, 20004, 0, 3 ]

9 [ 0, 20004, 20005, 15002 ]

10 [ 5, 25002, null, '(15002)' ]

11 [ 23, 20006, 0, 3 ]

12 [ 0, 20006, 20007, 10000 ]

13 [ 5, 20008, null, '(10000)' ]

14 [ 23, 20009, 0, 3 ]

15 [ 0, 20009, 20010, 10001 ]

16 [ 5, 20011, null, '(10001)' ]

17 [ 23, 20012, 0, 3 ]

18 [ 0, 20012, 20013, 10002 ]

19 [ 5, 20014, null, '(10002)' ]

20 [ 40, 0, 3, null ]

21 [ 40, 5000, 3, null ]

22 [ 39, null, null, 25005 ]

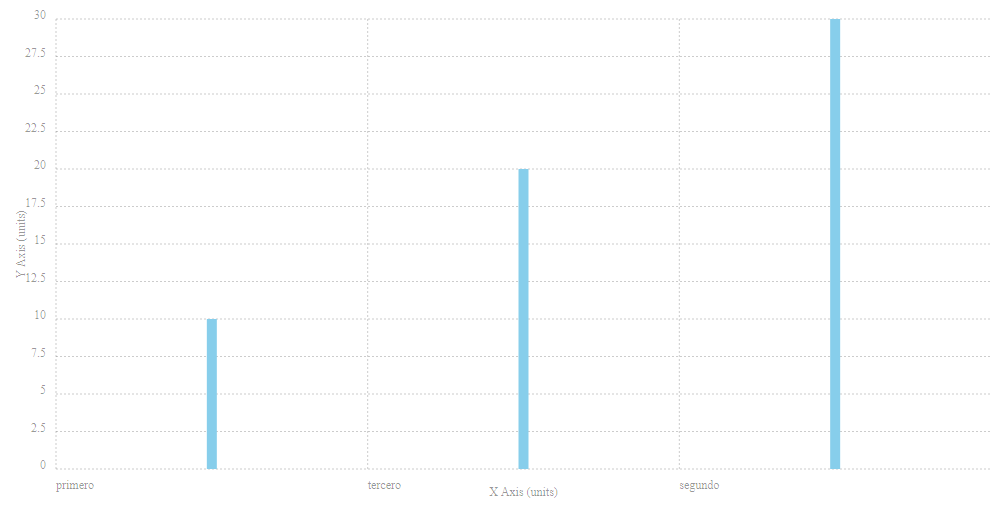
23 [ 39, null, null, 25004 ]

24 [ 39, null, null, 25003 ]

25 [ 47, null, null, 'bar' ]

26 [ 'end', null, null, null ]

Chart runing on port 8081...



### e.13) Generar una gráfica de barras a partir de un array de datos y un array de etiquetas.

program myProgram:

var series[3], b:int;

var labels[3]: string;

main {

labels[0] = "primero";

labels[2] = "segundo";

labels[1] = "tercero";

series[0] = 10;

series[1] = 13;

series[2] = 4;

linePlot(series,labels, ",”X Axis(units)”, “Y Axis(units)”);

}

End

1 [ 16, null, null, 1 ]

2 [ 23, 20000, 0, 3 ]

3 [ 0, 20000, 20001, 15000 ]

4 [ 5, 25000, null, '(15000)' ]

5 [ 23, 20002, 0, 3 ]

6 [ 0, 20002, 20003, 15001 ]

7 [ 5, 25001, null, '(15001)' ]

8 [ 23, 20004, 0, 3 ]

9 [ 0, 20004, 20005, 15002 ]

10 [ 5, 25002, null, '(15002)' ]

11 [ 23, 20006, 0, 3 ]

12 [ 0, 20006, 20007, 10000 ]

13 [ 5, 20008, null, '(10000)' ]

14 [ 23, 20009, 0, 3 ]

15 [ 0, 20009, 20010, 10001 ]

16 [ 5, 20011, null, '(10001)' ]

17 [ 23, 20012, 0, 3 ]

18 [ 0, 20012, 20013, 10002 ]

19 [ 5, 20014, null, '(10002)' ]

20 [ 40, 0, 3, null ]

21 [ 40, 5000, 3, null ]

22 [ 39, null, null, 25005 ]

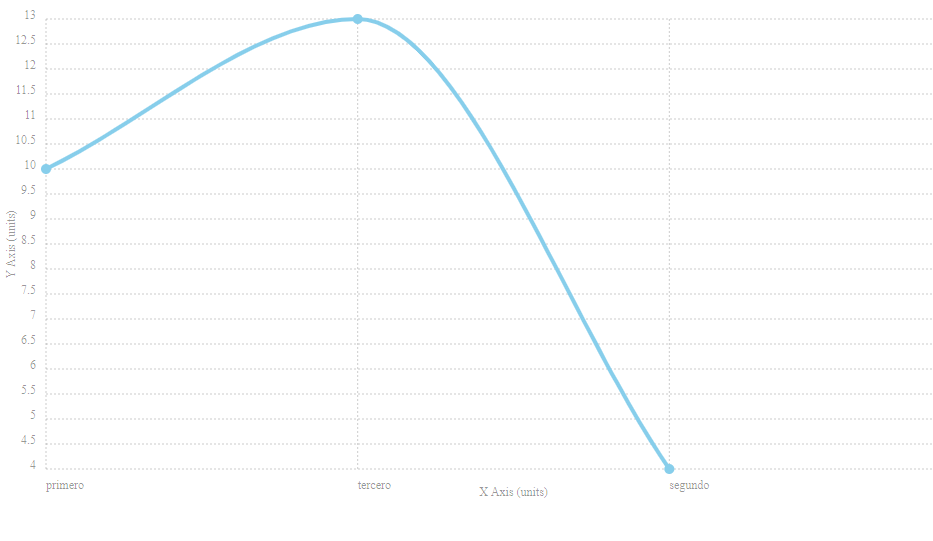
23 [ 39, null, null, 25004 ]

24 [ 39, null, null, 25003 ]

25 [ 47, null, null, 'line' ]

26 [ 'end', null, null, null ]

Chart runing on port 8081...



# f) Listados documentados del proyecto

memory.js

var variableTable = require('./varTable.js').variableTable;

var dirFun = require('./dirFun.js').dirFun;

class Memory {

constructor(

scope = 'global',

gi = 0,

gf = 2500,

gs = 5000,

gb = 7500,

li = 10000,

lf = 12500,

ls = 15000,

lb = 17500,

ci = 20000,

cf = 22500,

cs = 25000,

cb = 27500) {

this.scope = scope;

this.varTables = {

global: new variableTable(),

local: new variableTable(),

const: new variableTable(),

pointer: new variableTable()

}

this.Gi = gi;

this.Gf = gf;

this.Gs = gs;

this.Gb = gb;

this.Li = li;

this.Lf = lf;

this.Ls = ls;

this.Lb = lb;

this.Ci = ci;

this.Cf = cf;

this.Cs = cs;

this.Cb = cb;

this.size = cs + gf;

this.dirFun = new dirFun();

}

/\*\*

\* Funcion que añade una variable con sus datos a memoria virtual

\* Calcula sus parametros para variables dimensionadas

\* @param {string} key Nombre de la variable

\* @param {int} type Código del tipo de variable

\* @param {string} sc Scope de memoria donde alocar la variable

\* @param {any} value Valor de la variable

\* @param {array} dims Array de dimensiones, default de 1

\* @returns {int} Retorna el indice de la variable en memoria

\*/

set({key, type, sc, value, dims: [D1 = 1, D2 = 1] = []}) {

let scope = sc || this.scope;

let memory = 0;

let R = D1 \* D2;

switch (scope) {

case 'global':

switch (type) {

case 0:

memory = this.Gi;

this.Gi += R;

break;

case 1:

memory = this.Gf;

this.Gf += R;

break;

case 2:

memory = this.Gs;

this.Gs += R;

break;

case 3:

memory = this.Gb;

this.Gb += R;

break;

}

break;

case 'local':

switch (type) {

case 0:

memory = this.Li;

this.Li += R;

break;

case 1:

memory = this.Lf;

this.Lf += R;

break;

case 2:

memory = this.Ls;

this.Ls += R;

break;

case 3:

memory = this.Lb;

this.Lb += R;

break;

}

break;

case 'const':

switch (type) {

case 0:

memory = this.Ci;

this.Ci += R;

break;

case 1:

memory = this.Cf;

this.Cf += R;

break;

case 2:

memory = this.Cs;

this.Cs += R;

break;

case 3:

memory = this.Cb;

this.Cb += R;

break;

}

break;

}

this.varTables[scope].set({key, type, scope, index:memory, value, R, D1, D2});

return memory;

}

/\*\*

\* Funcion que retorna las variables en memoria de existir

\* @param {string} KEY

\* @returns {object} Variable

\*/

get(KEY) {

return this.varTables[this.scope].get(KEY) || this.varTables.global.get(KEY)

}

/\*\*

\* Funcion para establecer el scope de la memoria

\* @param {string} scope

\*/

setScope(scope) {

this.scope = scope;

}

/\*\*

\* Funcion que añade una nueva funcion al directorio de memoria, de hacerlo cambia el scope a local y renueva la memoria local

\* @param {string} ID

\*/

addFunction(ID) {

if(this.dirFun.addFunction(ID)) {

this.setScope('local');

this.varTables['local'].clear();

}

}

/\*\*

\* Funcion que al terminar una función, renueva la memoria local y cambia a memoria global.

\*/

deleteFunction() {

this.varTables['local'] = new variableTable();

this.setScope('global');

this.Li = 10000;

this.Lf = 12500;

this.Ls = 15000;

this.Lb = 17500;

}

}

exports.Memory = Memory;

quads.js

class Quads {

constructor( ) {

this.quads = [[16, null, null, undefined]];

this.current = 0;

this.saved = [];

}

/\*\*

\* Añade un nuevo cuadruplo a la pila

\* @param {array} quad

\*/

push(quad) {

this.quads.push(quad);

}

/\*\*

\* Obtiene el indice del arreglo presente

\*/

getCurrent() {

return this.current;

}

/\*\*

\* Guarda el indice de un arreglo en la pila de guardados

\* @param {int} index

\*/

save(index) {

index = index ? index : this.current;

this.saved.push(index || this.current);

}

/\*\*

\* Regresa el ultimo arreglo guardado

\*/

getSaved() {

return this.saved.pop();

}

/\*\*

\* Regresa el cuadruplo con el indice pasado, cambia el indice de los cuadruplos al nuevo indice

\* @param {int} index

\*/

goto(index){

this.current = index - 1;

return this.quads[index];

}

/\*\*

\* Aumenta la variable current, para ver el siguiente cuadruplo

\*/

next() {

this.current++;

}

/\*\* Rellena el primer cuadruplo (Main) con el valor actual de cuadruplos \*/

main(){

this.quads[0].splice(3, 1, this.length());

}

/\*\*

\* Regresa el valor del cuadruplo actual

\*/

now() {

return this.quads[this.current];

}

/\*\*

\* Regresa el numero de cuadruplos

\*/

length() {

return this.quads.length

}

/\*\*

\* Usado en saltos, rellena el resultado del cuadruplo con el nuevo valor pasado

\* @param {int} end

\* @param {int} count

\*/

fill(end, count){

this.quads[end].splice(3, 1, count)

}

/\*\*

\* Imprime todos los cuadruplos, usado en pruebas

\*/

print() {

this.quads.forEach((quad,index) => {

console.log(index + 1 , quad)

})

}

}

exports.Quads = Quads;

index.js

*/\*\**

*\* Corre la máquina virtual con los cuadruplos y la memoria de compilación*

*\* Elimina las estructuras usadas en compilación.*

*\*/*

deleteDir = *\_* => {

let vm = new VM(quads, mm);

vm.run();

delete mm;

delete quads;

delete p;

}

*/\*\**

*\* Funcion para guardar los nombres de variables*

*\* Params: ID de variable a guardar*

*\*/*

setVars = *ID* => {

try {

if (pVars[ID]) {

throw "ERROR: Duplicated variable: " + ID;

} else {

pVars[ID] = { name: ID };

varCount++;

}

} catch (error) {

console.error(error);

process.exit();

}

}

*/\*\**

*\* Funcion que mete a memoria las variables en memoria junto con su tipo*

*\* Params: Tipo de las variables a guardar*

*\*/*

setType = *TYPE* => {

try {

Object.values(pVars).forEach( *v* => {

mm.set({key: v.name, type: findType(TYPE), dims: v.dims})

})

pVars = [];

} catch (error) {

console.error(error);

process.exit();

}

}

*/\*\**

*\* Funcion que guarda un arreglo con sus dimensiones en la pila de memoria.*

*\* Primero valida que la variable no haya sido definida en el scope.*

*\*/*

setArr = (*ID*, *D1*) => {

try {

if (pVars[ID]) {

throw "ERROR: Duplicated variable: " + ID + pVars[ID];

} else {

pVars[ID] = { name: ID, dims:[parseInt(D1)] };

varCount+=D1;

}

} catch (error) {

console.error(error);

process.exit();

}

}

*/\*\**

*\* Funcion que guarda una matriz con sus dimensiones en la pila de memoria.*

*\* Primero valida que la variable no haya sido definida en el scope.*

*\*/*

setMat = (*ID*, *D1*, *D2*) => {

try {

if (pVars[ID]) {

throw "ERROR: Duplicated variable: " + ID;

} else {

pVars[ID] = { name: ID, dims:[parseInt(D1), parseInt(D2)] };

varCount+=D1;

}

} catch (error) {

console.error(error);

process.exit();

}

}

*/\*\**

*\* Añade la función al directorio de variables. Reinicia el contador de variables.*

*\*/*

setTable = *ID* => {

varCount = 0;

mm.addFunction(ID);

}

*/\*\**

*\* Funcion que reestablece la memoria local y añade el cuadruplo ENDPROC*

*\*/*

deleteFunction = *\_* => {

mm.deleteFunction();

quads.push([18, null , null, null])

}

*/\*\**

*\* Establece guarda el contador de parametros, variables*

*\* y el cuadruplo de la función en el directorío de funciones*

*\*/*

setFunParams = *\_* => {

mm.setCurrFunParams(paramCount);

mm.setCurrFunVars(varCount);

mm.setCurrFunQuads(quads.length());

varCount = 0;

paramCount = 0;

}

*/\*\**

*\* Valida que la variable exista, si existe lo mete a stack de tipos y operadores*

*\*/*

pushVar = *ID* => {

try {

const {type, index} = mm.get(ID) || {index: -1};

if (index >= 0) {

pTypes.push(type)

pOp.push(index)

} else {

throw `ERROR VAR: Undefined variable ${ID}`

}

} catch (e){

console.error(e);

process.exit();

}

}

*/\*\**

*\* Función que valida que la variable exista y que sea dimensionado,*

*\* si existe crea los cuádruplos para calcular la dirección*

*\* del arreglo y mete el pointer a la pila de operadores con su tipo.*

*\*/*

pushArr = *ID* => {

try {

let S1\_type = pTypes.pop();

let S1 = pOp.pop();

*// Validar que la dimensión sea un integer*

if(S1\_type !== 0) {

throw `ERROR ARRAY: ${ID} invalid dimension type ${S1\_type}`

}

*// Validar que la variable exista y traer sus datos.*

const {type, index, D1} = mm.get(ID) || {index: -1}

if (index >= 0) {

*//Validar que sea una variable dimensionada*

if (D1 === 1) {

throw `ERROR ARRAY: ${ID} is not an array`

}

*// Crear memoria para la temporal del array y la base constante*

let pointer = mm.set({type, sc:'local'});

let base = mm.set({type: 0, sc:'const', value: index});

*// VER la dimension S1, si esta en el rango 0 al limite Superior*

quads.push([23 , S1, 0, D1 ]);

*// Sumar la dimension S1 a la base y guardarlo en un temporal*

quads.push([0, S1, base, pointer]);

*// Push del temporal y su tipo*

pTypes.push(type);

pOp.push(`(${pointer})`);

} else {

throw `ERROR ARRAY: Undefined variable ${ID}`

}

} catch (e){

console.error(e);

process.exit();

}

}

*/\*\**

*\* Valida que la variable exista y que sea dimensionado,*

*\* si existe crea los cuádruplos para calcular la dirección de la matriz y*

*\* mete el pointer a la pila de operadores con su tipo.*

*\*/*

pushMat = *ID* => {

try {

let S2\_type = pTypes.pop();

let S2 = pOp.pop();

let S1\_type = pTypes.pop();

let S1 = pOp.pop();

if(S1\_type !== 0 || S2\_type !== 0) {

throw `ERROR MATRIX ${ID}, invalid dimension type`

}

const {type, index, D1, D2} = mm.get(ID) || {index: -1}

if (index >= 0) {

if (D1 === 1 && D2 === 1) {

throw `ERROR MATRIX: ${ID} is not a matrix`

}

*// Crear dos temporales para operaciones y un temporal para el resultado*

let temp = mm.set({type: 0, sc:'local'});

let temp2 = mm.set({type: 0, sc:'local'});

let pointer = mm.set({type, sc:'local'});

*// Crear direcciones para la dimensión D2 y la base*

let dim2 = mm.set({type: 0, sc:'const', value: D2});

let base = mm.set({type: 0, sc:'const', value: index});

*// Validar el rango para S1*

quads.push([23 , S1 , 0 , D1 ]);

*// S1 \* d2*

quads.push([2 , S1 , dim2, temp ]);

*// Validar el rango para S2*

quads.push([23 , S2 , 0 , D2 ]);

*// S1 \* d2 + S2*

quads.push([0 , temp , S2 , temp2 ]);

*// S1 \* d2 + S2 + Base*

quads.push([0 , temp2, base , pointer]);

pTypes.push(type);

pOp.push(`(${pointer})`);

} else {

throw `ERROR ARR: Undefined variable ${ID}`

}

} catch (e){

console.error(e);

process.exit();

}

}

*/\*\**

*\* Función que mete a memoria*

*\* y la pila de operadores una constante con su valor*

*\*/*

pushConst = (*DATA*, *TYPE*) => {

switch(TYPE){

case 'int':

DATA = parseInt(DATA); break;

case 'float':

DATA = parseFloat(DATA); break;

case 'string':

DATA = DATA.substr(1).slice(0, -1);

break;

case 'bool':

DATA = DATA === 'true'; break

}

pTypes.push(findType(TYPE))

pOp.push(mm.set({type: findType(TYPE), sc:'const', value: DATA}))

}

*/\*\**

*\* Función que crea el el cuadruplo de asignación,*

*\* validando previamente los tipos de las expresiones.*

*\*/*

processAssign = *ID* => {

try {

let right\_op = pOp.pop();

let right\_type = pTypes.pop();

let left\_op = pOp.pop();

let left\_type = pTypes.pop();

let result\_type = findCube(left\_type, right\_type, 5);

if (result\_type === 6) throw `Type mismatch trying to assing in ${ID}`;

quads.push([5, right\_op ,null, left\_op])

} catch (e) {

console.error(e);

process.exit();

}

}

*/\*\**

*\* processTerm: Función que procesa un temino en una expresión*

*\* Si llega una suma, resta, división, multiplicación o modulo*

*\* Genera el nuevo cuadruplo y mete el temporal con el resultado*

*\* en la pila de operadores y tipos*

*\*/*

processTerm = *\_* => {

try {

if (poper[0] >= 0 && poper[0] <= 4){

let right\_op = pOp.pop();

let right\_type = pTypes.pop();

let left\_op = pOp.pop();

let left\_type = pTypes.pop();

let operator = poper.pop();

let result\_type = findCube(left\_type, right\_type, operator);

if (result\_type == 6) {

throw "Type mismatc in sum/res"

} else {

let temp = mm.set({type: result\_type, sc: 'local'});

pOp.push(temp);

quads.push([operator, left\_op, right\_op, temp])

pTypes.push(result\_type);

}

}

} catch (error) {

console.error(error);

process.exit();

}

}

*/\*\**

*/\*\**

*\* processCond*

*\* Función que procesa una condición a partir de una expresión*

*\* La expresión debe de ser de tipo booleano*

*\* De ser así, genera el cuadruplo con el resultado y mete el salto en la pila*

*\*/*

processCond = *\_* => {

try {

*// Verificar que el tipo sea booleano, sino lanzar error*

if(pTypes.pop() !== 3 ) throw "Type mismatch, boolean expected in condition"

else {

*// De ser booleano*

let temp = pOp.pop();

*// Crear cuadruplo con la temporal que se evalua e indefinido para el salto*

quads.push([17, temp ,null, undefined]);

*// Guardar el contador de quadruplos - 1*

pJumps.push(quads.length() - 1);

}

} catch (e) {

console.error(e);

process.exit();

}

}

*/\*\**

*\* Función que al termino de una condición, rellena el cuadruplo de salto*

*\*/*

endCond = *\_* => {

let end = pJumps.pop();

quads.fill(end, quads.length());

}

*/\*\**

*\* processElse:*

*\* Función que genera el cuadruplo de salto para un else*

*\* A la vez, mete el indice de cuadruplo actual a la pila de saltos*

*\* y rellena el cuadruplo del If anterior*

*\*/*

processElse = *\_* => {

quads.push([16, null, null, undefined])

let f = pJumps.pop();

pJumps.push(quads.length() - 1);

quads.fill(f, quads.length());

}

*/\*\**

*\* Función que al final del while genera el GOTO de retorno al principio del ciclo*

*\* Y rellena a su vez el cuadruplo de salto al que se metio al principio del ciclo*

*\*/*

endWhile = *\_* => {

let end = pJumps.pop();

let ret = pJumps.pop();

quads.push([16, null ,null, ret])

quads.fill(end, quads.length());

}

*/\*\* Función que valida si una función ya existe al llamarla.*

*\* Si existe, la pone como el procedimiento actual*

*\* \*/*

checkProcedure = *ID* => {

try {

if(mm.getFunc(ID) === undefined) throw `Undefined function ${ID} called`;

currentProcedure = ID;

} catch (error) {

console.error(error);

process.exit();

}

}

*/\*\**

*\* Función que obtiene de un call un argumento,*

*\* lo valida con la tabla de procedimientos y*

*\* genera el cuadruplo de parametro*

*\*/*

getArgument = *\_* => {

try {

let arg = pOp.pop();

pTypes.pop();

let { paramTable } = mm.getFunc(currentProcedure);

if(paramTable) {

quads.push([22, arg, paramCount, currentProcedure]);

}

} catch (e) {

console.error(e);

process.exit();

}

}

*/\*\* Genera el cuadruplo de GOSUB para el procedimiento actual \*/*

pushGOSUB = *\_* => {

quads.push([21,null, null, currentProcedure]);

}

*/\*\* Genera el cuadruplo de print para la expresión contenida \*/*

pushPrint = *\_* => {

let arg = pOp.pop();

pTypes.pop();

quads.push([51, arg, null, null]);

}

*/\*\**

*\* Función que genera el cuadruplo de lectura*

*\* Valida si la función destinada existe.*

*\*/*

pushReadLine = *ID* => {

let arg = pOp.pop();

pTypes.pop();

let obj = mm.get(ID);

if (obj) {

quads.push([52, arg, null, obj.index]);

}

}

*/\*\**

*\* Funcion general para las funciones estadisticas basicas,*

*\* Las funciones estadisticas basicas son (Mean, Max, Min, Variance, Range, StDev)*

*\* Tiene los mismos parametros, por tanto se uso esta formula general*

*\* Params: ID: vector(numerico), nombre de la funcion, Operador*

*\* Usos:*

*\* pushStdDev*

*\* pushMax*

*\* pushMin*

*\* pushRange*

*\* pushVariance*

*\* pushMean*

*\*\*/*

pushStadistics = (*ID*, *fnName*, *Op*) => {

try {

*// Obtener de la memoria virtual el ID*

let memory = mm.get(ID);

*// Verificar que existe (undefined es considerado falso)*

if (memory){

*// Crear un nuevo temporal*

let temp = mm.set({type: 1, sc: 'local'});

*// Verificar que el ID*

if (memory.type !== 0 && memory.type !== 1)

throw `Error ${fnName} only accepts an integer or float as parameter`;

*// Cuadruplo para guardar un vector como parametro: [SetVector, ID = index, tamaño = R, null]*

quads.push([40, memory.index, memory.R, null])

*// Crear cuadruplo de operacion*

*// [Codigo de operacion, ID = index , null, temporal a guardar]*

quads.push([Op, memory.index, null , temp]);

*// Guardar el indice de memoria y el tipo (todas las funciones estadisticas son float = 1)*

pOp.push(temp);

pTypes.push(1);

} else { *// Sino existe marcar un error y terminar el programa*

throw `Error ${fnName}: ${ID} is not defined`

}

} catch (e) {

console.error(e);

process.exit();

}

}

*/\*\* Funcion general para las distribuciones normales PDF y CDF.*

*\* Params: ID: vector(numerico), nombre de funcion, Operador*

*\* Params indirectos: right\_op (numerico)*

*\* Usos:*

*\* pushNormPDF*

*\* pushNormCDF*

*\*/*

pushNormalDistributions = (*ID*, *Op*, *fnName*) => {

try {

*// Obtener los valores de la variable de memoria*

let memory = mm.get(ID);

if (memory){

*// Obtener la expresion a evaluar*

let right\_op = pOp.pop();

let right\_type = pTypes.pop();

*// Evaluar los tipos de los parametros de la funcion, tienen que ser numbericos*

*// Sino lanzar un error*

if (right\_type !== 0 && right\_type !== 1) {

throw `Error ${fnName}: primer parametro debe de ser un valor numerico`;

}

if (memory.type !== 0 && memory.type !== 1 || memory.R <= 1) {

throw `Error ${fnName}: segundo parametro debe ser un vector de numeros`;

}

*// Crear variable temporal*

let temp = mm.set({type: 1, sc: 'local'});

*// Generar cuadruplo para guardar un vector como parametro: [SetVector, ID = index, tamaño = R, null]*

quads.push([40, memory.index, memory.R, null]);

*// Generar cuadruplo propio de la operacion pasando el valor restante*

quads.push([Op, memory.index, right\_op, temp]);

*// Meter el resultado a la pila de operandos y a la pila de tipos 1 = float*

pOp.push(temp);

pTypes.push(1);

} else {

*// Si el vecto no existe, marcar error*

throw `Error ${fnName}: ${ID} is not defined`

}

} catch (e) {

console.error(e);

process.exit();

}

}

*/\*\* Funcion general para las distribuciones uniformes y binomiales.*

*\* Params: nombre de funcion, Operador*

*\* Params indirectos: tres expresiones (numericas)*

*\* Usos:*

*\* pushUniformPDF*

*\* pushUniformCDF*

*\* pushBinomialPDF*

*\* pushBinomialCDF*

*\*/*

pushDistributions = (*Op*, *fnName*) => {

try {

*// Obtener las expresiones a evaluar*

let first\_op = pOp.pop();

let first\_type = pTypes.pop();

let second\_op = pOp.pop();

let second\_type = pTypes.pop();

let third\_op = pOp.pop();

let third\_type = pTypes.pop();

*// Evaluar los tipos de los parametros de la funcion, tienen que ser numbericos*

*// Sino lanzar un error*

if (first\_type !== 0 && first\_type !== 1) {

throw `Error ${fnName} only accepts an integer or float at first parameter`

}

if (second\_type !== 0 && second\_type !== 1) {

throw `Error ${fnName} only accepts an integer or float at second parameter`

}

if (third\_type !== 0 && third\_type !== 1) {

throw `Error ${fnName} only accepts an integer or float at third parameter`

}

let temp = mm.set({type: 1, sc: 'local'});

*// Meter todos los parametros a memoria Op:39 => guarda parametros*

quads.push([39, null, null, first\_op]);

quads.push([39, null, null, second\_op]);

quads.push([39, null, null, third\_op]);

*// Cuadruplo especifico de la distribucion con el temporal donde se guarda el resultado*

quads.push([Op, null, null, temp]);

*// Meter a la pila el resultado y su tipo (flotante)*

pOp.push(temp);

pTypes.push(1);

} catch (e) {

console.error(e);

process.exit();

}

}

*/\*\* Funcion general para las distribuciones uniformes y binomiales.*

*\* Params: Series: Vector<number>, Label: Vector<number>, nombre de funcion*

*\* Params indirectos: expresion numerica*

*\* Usos:*

*\* pushLinePlot*

*\* pushBarPlot*

*\*/*

Plot = (*X*,*Y*, *fnName*) => {

try {

*// Obtener los vectores para series y labels*

let series = mm.get(X);

let labels = mm.get(Y);

*// Evaluar si los dos existen*

if (series && labels){

*// Obtener la expresion a evaluar*

let third\_op = pOp.pop();

let third\_type = pTypes.pop();

*// Evaluar las dos variables que deben de ser vectores (R > 1) y numericos*

*// Y evaluar la tercera expresion que sea string, contiene el nombre del archivo*

if (series.type !== 0 && series.type !== 1 || series.R <= 1) {

throw `Error ${fnName} only accepts an integer or float at first parameter`

}

if (labels.type !== 0 && labels.type !== 1 && labels.type !== 2 || labels.R <= 1) {

throw `Error ${fnName} only accepts an integer or float at second parameter`

}

if (third\_type !== 2) {

throw `Error ${fnName} only accepts an integer or float at third parameter`

}

*// Crear un temporal*

let temp = mm.set({type: 1, sc: 'local'});

*// Cuadruplos para meter los dos vectores como parametros*

quads.push([40, series.index, series.R, null]);

quads.push([40, labels.index, labels.R, null]);

*// Cuadruplo para meter el tercer parametro*

quads.push([39, null, null, third\_op]);

*// Cuadruplo para graficas*

quads.push([47, null, null, fnName]);

*// Meter a la pila el resultado y su tipo (flotante)*

pOp.push(temp);

pTypes.push(1);

} else { *// Sino marcar error*

throw `Error ${fnName}: ${X} or ${Y} is not defined`

}

} catch (e) {

console.error(e);

process.exit();

}

}

case 'Ls':

case 'Lb':

left\_value = this.localMem[left\_op];

break;

case 'Ci':

case 'Cf':

case 'Cs':

case 'Cb':

left\_value = this.constMem[left\_op].value;

break;

default:

throw 'Error: operator is not valid in print'

}

*// PRINT, no remover*

console.log(left\_value);

break;