

**Instituto Tecnológico y de Estudios Superiores de Monterrey**

**Campus Monterrey**

**Proyecto Final**

**Diseño De Compiladores**



**FoodLOOP - Food Language Object Oriented Programming**

**Profesores: Ph. D. Héctor Gibrán Ceballos Cancino**

**Ing. Elda G. Quiroga, M.C.**

**21 de Noviembre del 2018**

**Elaborada por:**

**Integrantes:**

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

**Victor Hugo Elizalde Muñoz Oscar Michel Herrera A00817586 A00817658**

## Contenido

**FoodLOOP**

**FoodLOOP**

[**Contenido**](https://docs.google.com/document/d/1PY3pPMODVPODeN4C37LnEJ25lg59gVjLVRYef8dyWZE/edit#heading=h.6tyk02x0tgu6)

**Descripción del Proyecto 3**

**Propósito 3**

**Objetivos 3**

**Alcance del Proyecto 3**

**Análisis de requerimientos 3**

**Proceso General de Desarrollo 4**

**Bitacoras 4**

**Reflexiones 6**

**Descripción del Lenguaje 6**

**Nombre del Lenguaje 6**

**Características del Lenguaje 7**

**Errores que Pueden Ocurrir 7**

**Descripción del Compilador 7**

**Equipo de Cómputo, Lenguaje y Utilerías 7**

**Descripción del Análisis de Léxico 8**

**Patrones de Construcción 8**

**Tokens y Código Asociado. 9**

**Descripción del Análisis de Sintaxis 10**

**Gramática Formal 10**

**Descripción Código Intermedio y Análisis Semántico 10**

**Diagramas de Sintaxis con las acciones correspondientes. 10**

**Breve descripción de cada una de las acciones semánticas y de código**

**Tabla de consideraciones semánticas 19**

**Descripción detallada del proceso de Administración de Memoria usado en la compilación. 20**

**Especificación gráfica de CADA estructura de datos usada (Dir.Funciones, Tablas de Var's, Cuádruplos, etc...) 20**

**Descripción De La Máquina Virtual 20**

**Descripción detallada del proceso de Administración de Memoria en ejecución (Arquitectura).**

**Especificación gráfica de CADA estructura de datos usada para manejo de scopes (Memoria Local, global, etc..) 21**

**Asociación hecha entre las direcciones virtuales (compilación) y las reales (ejecución).**

**Pruebas Del Funcionamiento 21**

**Manual de Usuario 45**

**Descripción del Proyecto**

Correspondiente a la clase de diseño de compiladores el siguiente proyecto precisamente consta del diseño, creación e implementación de un compilador con un alcance “básico” que permita aplicar los fundamentos básicos de programación y el paradigma de la programación de objetos.

**Propósito**

Como ya se ha mencionado en el preámbulo del proyecto, este proyecto tiene como propósito desarrollar un compilador para un lenguaje de programación con los conceptos básicos para la implementación de diagramas de flujo así como la inclusión del paradigma de la programación orientada a objetos.

Este lenguaje de programación usa una sintaxis y semántica fuera de lo convencional donde toda su gramática es una referencia/analogía directa a la cocina/comida, y que con estos “conceptos básicos y comunes” sea fácil de entender y divertido de utilizar para cualquier persona sin conocimientos básicos de programación. La población objetivo es cualquier persona que no sepa nada de programación por lo que aplica para personas de todas las edades.

**Objetivo**

Implementar un lenguaje de programación básico que trate de englobar todos los conceptos básicos de programación en analogías generales de la cocina/comida que todo el mundo pueda comprender fácilmente. Esta idea nace de la premisa común que cuando recién nos enseñan que es un programa informático nos dan la analogía con una receta de cocina.

**Alcance del Proyecto**

Este proyecto se centra en el diseño y desarrollo de un compilador para un lenguaje de programación con soporte de objetos para el aprendizaje especialmente infantil y adolescente aunque esto no limita su uso para cualquier principiante o aficionado de los lenguajes de programación.

**Categoría Específica:**

Un lenguaje básico orientado a objetos que produzca mecanismos para definir clases, atributos, métodos e itinerancia simple.

**Análisis de requerimientos**

**Características de la Semántica**

* Legibilidad por la sintaxis similar a la preparación de una receta de cocina
* Usabilidad para poder introducir a cualquier persona a la programación
* Facilidad de uso por la lógica simple del lenguaje enfocado a recetas.

**Funciones**

* Clases: El lenguaje permite crear nuevos tipos de variables.
* Atributos: Variables específicas a una clase.
* Métodos: Funciones específicas a una clase.
* Herencia: El lenguaje permite que las clases obtengan métodos y atributos una clase padre.

**Tipos de Datos**

* Integer: Número enteros
* Decimal: Número flotante
* Boolean: Booleano
* String: Cadena de caracteres

**Descripción de los principales Test Cases**

**Proceso General de Desarrollo**

**Bitacoras**

1. En primer lugar una vez ya tomada la decisión sobre el los alcances y propósito, así como las herramientas para el desarrollo, se procedió a diseñar lo que vendría a ser el lenguaje final y se crearon varios ejemplos dummies de como quedaría la gramática final, con esto comenzamos una investigación sobre la documentación de la herramienta que utilizaríamos (principalmente http://antlr.ohboyohboyohboy.org) para definir los tokens a forma de desarrollar el Lexer y después realizamos los diagramas de la gramática para codificar esto en lo que sería la base del Parser sobre el cual estaríamos trabajando el resto del proyecto.
2. Se realizó testeo y correcciones de ambigüedad en la gramática hasta que los primeros archivos de prueba fueron aceptados y bien validados léxica y sintácticamente.
3. Enseguida procedimos a definir los contextos o directorios que comprenden todas las instancia de los objetos y estructuras que se estarían accediendo según el momento de ejecución de la gramática. Con esto se definieron contextos para las clases, las funciones y las variables que vendrían a ser Hashes de los ids de cada cosa hacia objetos de cada tipo que contienen toda la información necesaria para las operaciones y dependencias lógicas durante el desarrollo. En el caso del directorio de clases contiene objetos con el nombre de la clase, si tienen o no herencia y su contexto de procedencia, para las funciones se almacena la información del nombre de la función, su tipo de retorno, sus parámetros y el número del cuádruplo donde comienza esta función y para las variables se guarda su nombre, su tipo, su valor, si este tiene dimensiones y los objetos de esta dimensión y su tamaño, su dirección de memoria y un bit para saber discernir si esta “variable” en su forma de declaración se declaró como una variable dinámica o estática (constante).
4. Se crearon los tipos y contextos para las variables dentro del main y funciones dejando para después el contexto de las clases para la definición de objetos.
5. Se crean las estructuras genéricas correspondientes para cuadruplos y un manager para la gestión de la creación de estos y accesos a los distintos contextos para obtener toda la información que se insertará en estos cuadruplos.
6. Se crean los primeros cuadruplos para el procesamiento de expresiones aritméticas teniendo que definir y desarrollar el cubo semántico y validar el match de cada tipo entre sus operaciones y salidas.
7. Se definen las operaciones GOTO y GOTOF para los cuádruplos para el soporte de los estatutos condicionales, el goto al main del principio el estatuto de ciclo WHILE.
8. Se crea cuadruplos para el soporte de OUTPUTS (prints) e INPUTS (reads), posterior a ello los cuádruplos para el esqueleto de las funciones (operaciones GOSUB y ERA, ENDPROC) y finalmente la asignación de los valores a las variables definidas.
9. Se añaden comprobaciones para los tipos de las variables (que la variable tenga que existir para su uso, la compatibilidad entre tipos, que respeten sus contextos de definición etc.) para los retornos de las funciones y que los parámetros de envío correspondan a la cantidad y al tipo de los encabezados.
10. Creación de los cuádruplos y contextos faltantes para el soporte de clases las cuales contienen variables y funciones.
11. Desarrollo de estructuras de almacenamiento de datos útiles para la implementación de variables dimensionadas y acoplamiento con la asignación actual de las variables no-dimensionadas. Se crea las funciones para la creación de cuádruplos de variables dimensionadas (operación VERIFICA)
12. Implementación para la resolución de recursividad definiendo un stack de contextos de llamadas que se contengan a sí mismo y estos devuelvan su valor al contexto que los ha llamado.
13. Se agrego soporte de comentarios y constantes a la gramática.

**Reflexiones**

**Oscar Michel Herrera**

Durante el proceso de desarrollo he logrado captar de manera práctica todo lo teórico comprendido en clase durante el semestre y con este nuevo aprendizaje tengo ahora mismo una percepción aún más amplias de lo que es aquello que utilizamos todos los días, los lenguajes de programación. Con este proyecto entendí más a fondo sobre el funcionamiento interno de los lenguajes modernos con el paradigma de la orientación de objetos así sobre cómo se logro la evolución de las primeras codificaciones en binario pasando por hexadecimal, ensamblador y hasta lo que tenemos hoy día. Me llama mucho la atención como con esto ahora podemos ser creadores o innovadores no solo en el desarrollo de software sino en el paradigma tal cual de la abstracción de la realidad en programas de código. Me gustaría recalcar que me he llevado también una lección de disciplina para no postergar este proyecto pues realmente no se tuvo mucho avance sino hasta el último mes del semestre y ni hablar de la última semana previo a su entrega.

**Victor Elizalde**

En este proyecto aprendí demasiadas cosas acerca de las formas en las que están hechas los lenguajes, yo siempre me preguntaba porque C++ usaba puntos y comas que porque no le hacían como ruby que no usaban punto y comas, a la hora de hacer este proyecto me di cuenta el porque, tanto así que hasta a mi lenguaje le puse punto y coma. Muchos detalles de cómo están construidos los lenguajes ahora me hacen click y la verdad es el proyecto más retador que he tenido ya que hace mucho uso de arquitectura del software y tmb de algoritmos, es la mezcla perfecta de un programador completo, tuve un amor odio por este proyecto porque me estresé y desvelé mucho pero cuando solucionaba el error me sentía omnipotente. Aprendí a no dejar todo hasta el último y como se hacen las máquinas virtuales y código intermedio.

**Descripción del Lenguaje**

**Nombre del Lenguaje**

FoodLOOP - Food Lenguage Object Oriented Programming

El nombre fue inspirado en una de las analogías más común en el mundo de la informática cuando a una persona se le trata de explicar por primera vez que es un algoritmo y un programa (conjunto de indicaciones) haciendo referencia a que estos son como una receta de comida la cual sigue una serie de pasos ordenadas de tal manera que haga algún platillo en particular y que así mismo ocurre con los programas básicos en el mundo de la informática.

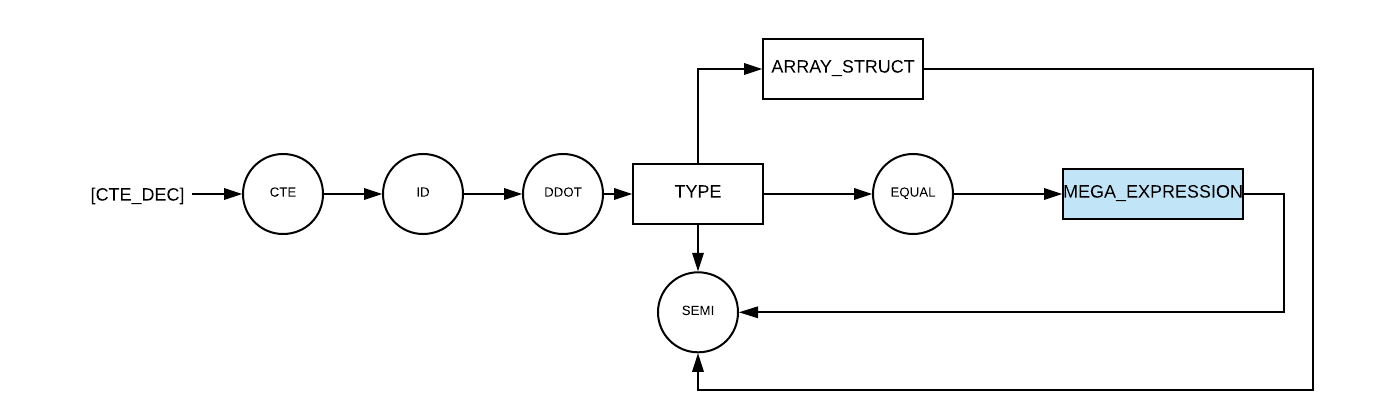
**Características del Lenguaje**

Como ya se ha venido mencionando, las características principales y de diferenciación respecto a otro dentro del mismo ámbito podemos destacar:

* Una gramática muy sencilla y la vez con propiedades comunes entre muchos de los más populares y robustos lenguajes de programación en la actualidad.
* Palabras reservadas y en español con total referencia y analogía hacia una receta de comida

**Errores que Pueden Ocurrir**

Se realizaron múltiples casos de prueba y como un ’bug’ muy calcado que encontramos, reside en algo que tiene que ver con la creación de constantes pues si sintácticamente el lenguaje para la declaración de una constante sólo debería permitir el siguiente esquema:



sin embargo el lenguaje está permitiendo sentencias de declaraciones sin ID como:

CTE : TYPE = MEGA\_EXPRESSION;

**Descripción del Compilador**

**Equipo de Cómputo, Lenguaje y Utilerías**

Utilizamos el lenguaje de programación Ruby, el ambiente de desarrollo fue MacOS y Windows. Para el desarrollo del scanner y parser se uso la herramienta antlr en su versión 3 que da soporte de exportación para Ruby mediante una gema(librería) .



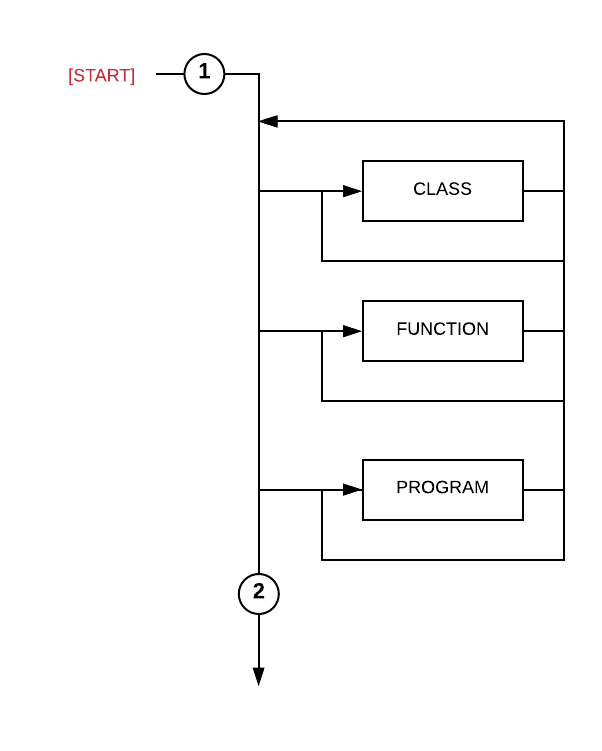
**Descripción del Análisis de Léxico**

**Patrones de Construcción**

**Tokens y Código Asociado.**

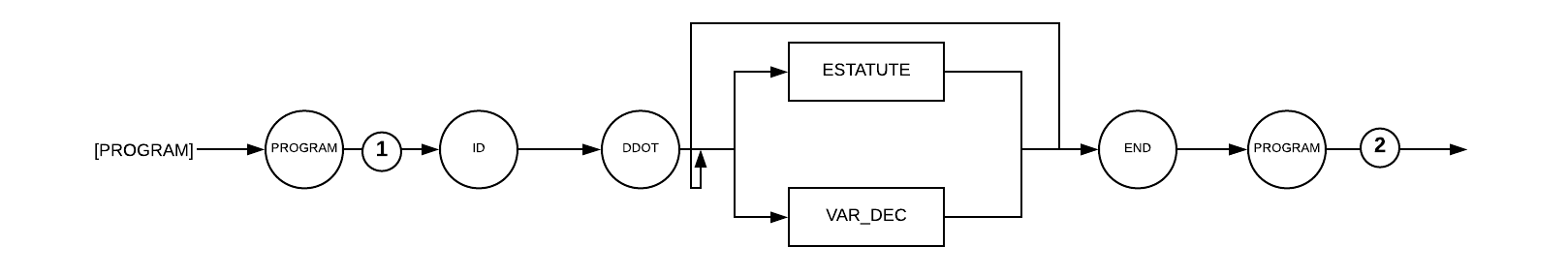
| **Token** | **Lexema** | **Expresión Regular** |
| --- | --- | --- |
| **PROGRAM** | **menu** | **[menu]** |
| **PRINT** | **mostrar** | **[mostrar]** |
| **VAR** | **ingrediente** | **[ingrediente]** |
| **CTE** | **utensilio** | **[utensilio]** |
| **IF** | **si** | **[si]** |
| **ELSE** | **sino** | **[sino]** |
| **AND** | **&&** | **[&&]** |
| **OR** | **||** | **[||]** |
| **GT** | **>** | **[>]** |
| **GTE** | **>=** | **[>=]** |
| **LT** | **<** | **[<]** |
| **LTE** | **<=** | **[<=]** |
| **INTEGER** | **nocortable** | **[nocortable]** |
| **FLOAT** | **cortable** | **[cortable]** |
| **STRING** | **nombre** | **[nombre]** |
| **BOOLEAN** | **estado** | **[estado]** |
| **VOID** | **proceso** | **[proceso]** |
| **FUNCTION** | **receta** | **[receta]** |
| **END** | **terminar** | **[terminar]** |
| **RETURN** | **servir** | **[servir]** |
| **INPUT** | **traer** | **[traer]** |
| **WHILE** | **mientras** | **[mientras]** |
| **CLASS** | **platillo** | **[platillo]** |
| **WORD** | **“Alpha, Beta”...** | **[".\*"]** |
| **NUM\_I** | **1, 2, 3...** | **[+-]?[1-9][0-9]\*** |
| **NUM\_F** | **1.5, 1.9898, 5.656, ...** | **[+-]?[1-9][0-9]\*(\.[0-9]+)?** |
| **BOOL** | **verdadero, falso** | **(true|false)** |
| **ID** | **Plato, Lonche1, Licuadora** | **[A-Za-z][A-Za-z0-9]\*** |
| **LPAREN** | **(** | **[(]** |
| **RPAREN** | **)** | **[)]** |
| **LKEY** | **{** | **[{]** |
| **RKEY** | **}** | **[}]** |
| **EQUAL** | **=** | **[=]** |
| **DIF** | **<>** | **[!=]** |
| **SIMILAR** | **==** | **[==]** |
| **COMMA** | **,** | **[,]** |
| **DDOT** | **:** | **[:]** |
| **SEMI** | **;** | **[;]** |
| **PLUS** | **+** | **[+]** |
| **SUB** | **-** | **[-]** |
| **DIV** | **/** | **[/]** |
| **MUL** | **\*** | **[\*]** |
| **WHITESPACE** | **‘ ‘** | **[\s]** |
| **INHERITS** | **<<** | **[<<]** |
| **DOT** | **.** | **[.]** |
| **NEWLINE** | **\n** | **(‘\n’ | ‘\r’) + { $channel = HIDDEN };** |

**Descripción del Análisis de Sintaxis**

****

1. Se crea el primer cuadrupolo con la indicación de goto al main

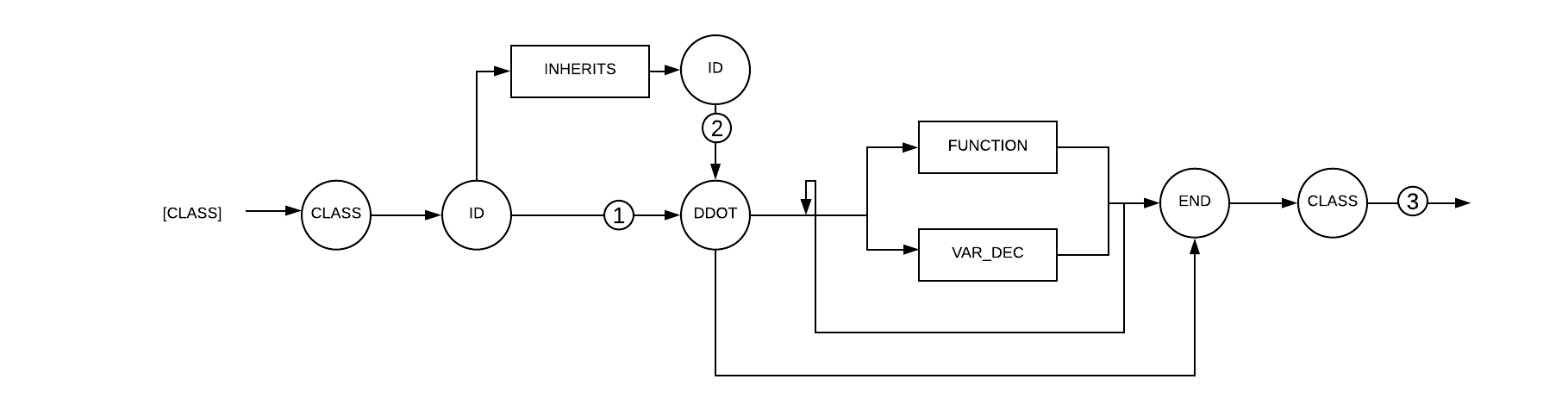
2. Se termina toda la ejecución en memorias, contextos, etc. del programa.

**Program: Patrón para el main de ejecución del programa **

1. Realiza el set del cuádruplo donde comienza al GOTO main del principio

2. Se le indica al administrador de recursos que pase los cuádruplos generados a la máquina virtual para su ejecución y opcionalmente imprime los cuádruplos.

**Class: Estructura o patrón de las clases del programa**

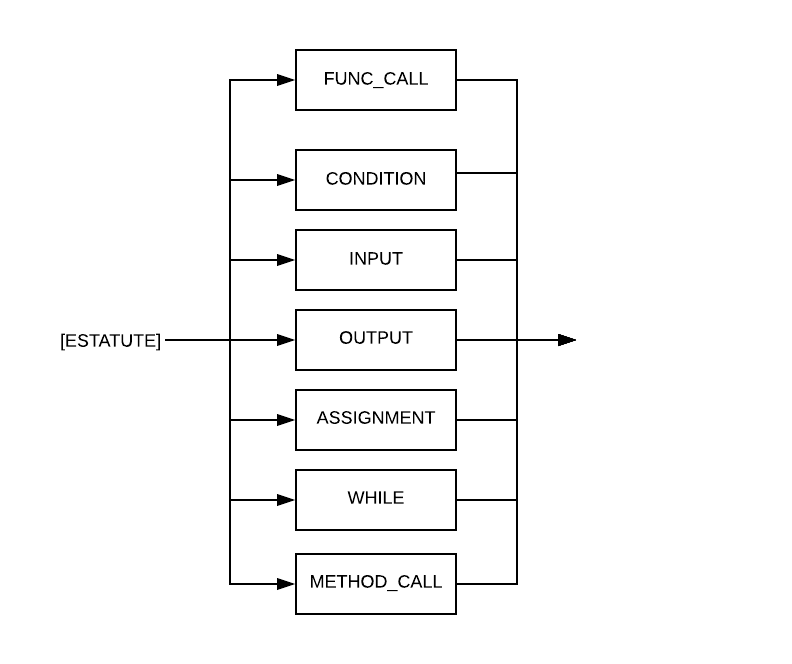


1. Verifica que no exista otra clase con el mismo ID

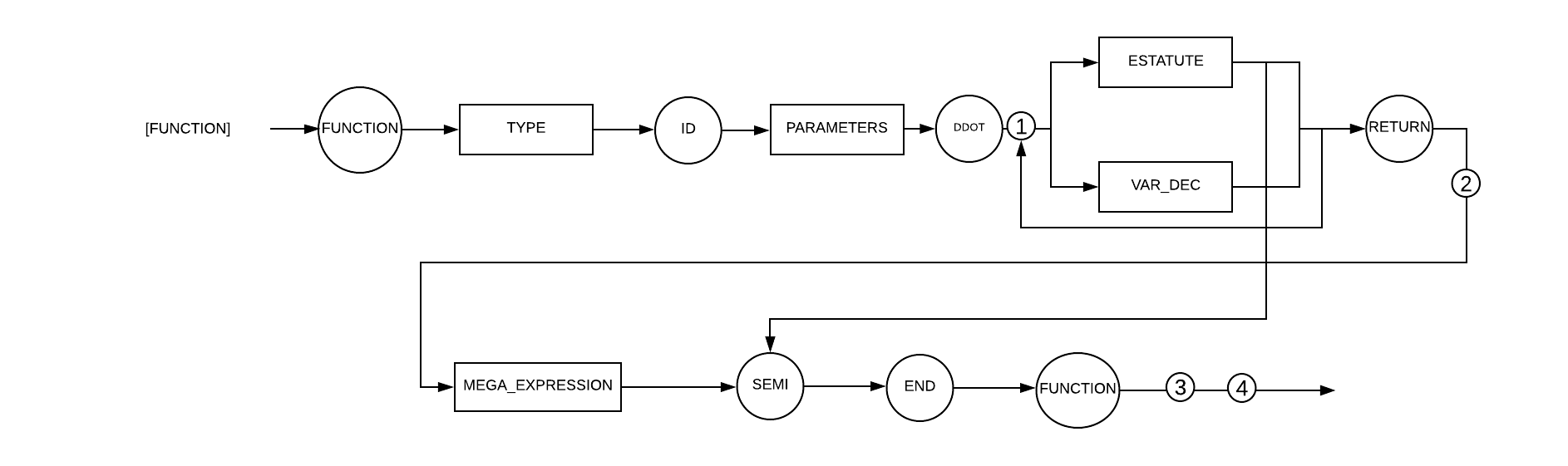
2. Revisa que exista la padre de la que va a heredar

3. Se hace un reinicio del contexto para la clase

**Estatute: Estatutos permitidos dentro del main del programa**

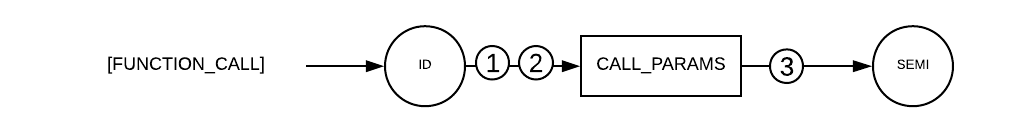
****

**Function: Estructura general de una función**

****

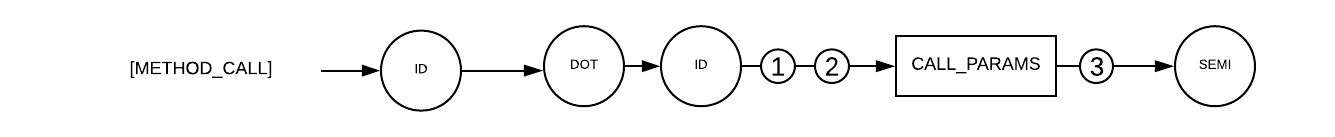
1. Verifica que la función no exista y la agrega al contexto/directorio
2. Si el tipo es distinto a void se verifica que exista un return del mismo tipo
3. Se agrega el cuádruplo con la operación ENDPROC (ENDFUNC en nuestro caso).
4. Se hace un reinicio del contexto de la función.

**Function Call: Patrón para realizar la llamada de una función**



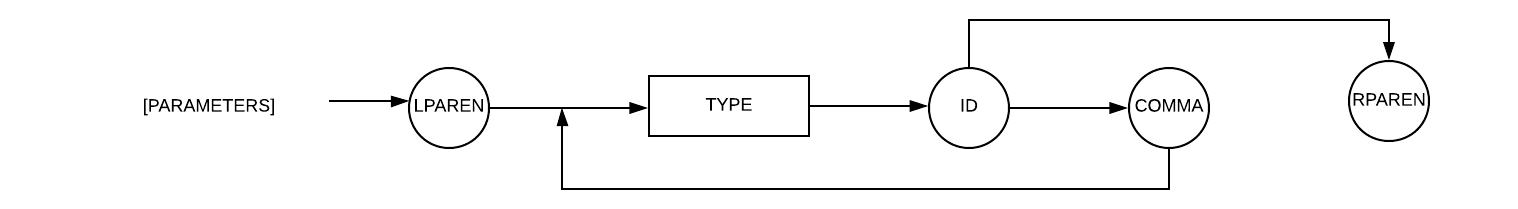
1. Se verifica que exista tanto el id del objeto como el de la función
2. Se genera el cuádruplo con el procedimiento ERA
3. Si existe algún retorno se devuelve.

**Method Call: Patrón para realizar una llamada a una función de un objeto**

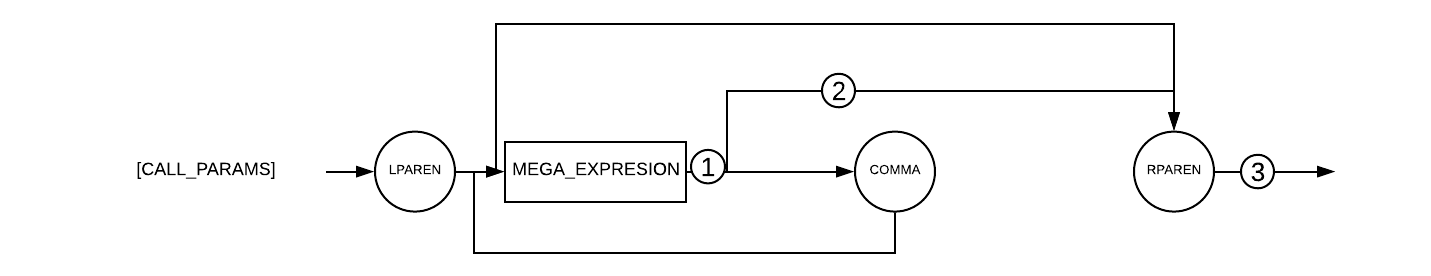
****

1. Se verifica que exista tanto el id del objeto como el de la función
2. Se genera el cuádruplo con el procedimiento ERA
3. Si existe algún retorno se devuelve.

**Parameters: Utilizado en los encabezados de las funciones**



**Call\_Params: Parámetros pasados a la función desde sus llamadas**

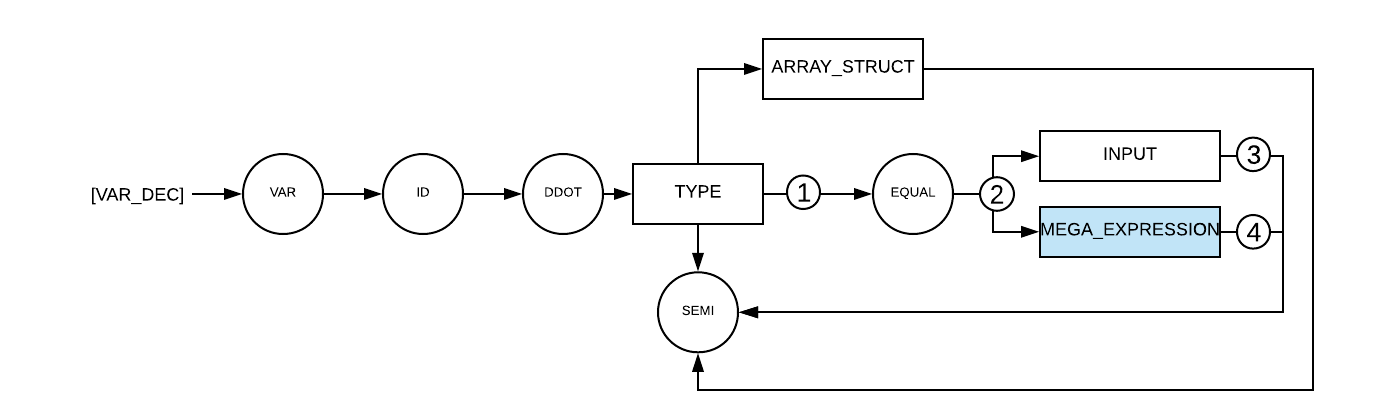
****

1. Se genera el cuádruplo de PARAMETER que pasa los valores a la función

2. Verifica que la cantidad y tipos de parámetros hagan match con los de la función

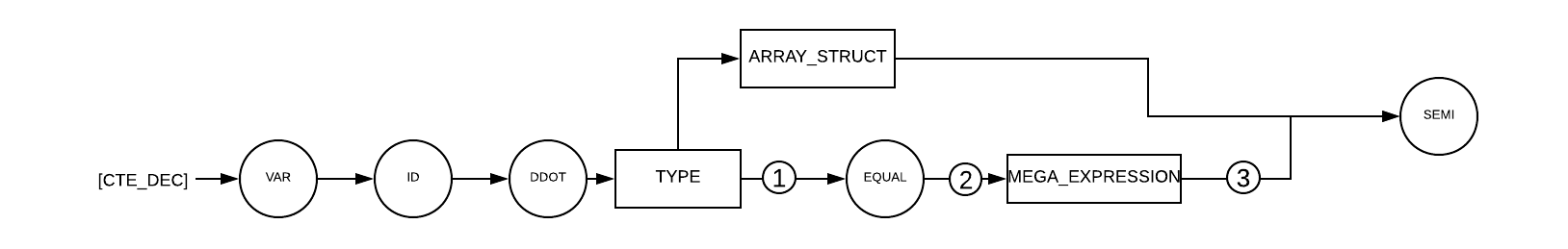
3. Se genera el cuádruplo con la operación GOSUB

**Var\_Dec: Declaracion de variables**

****

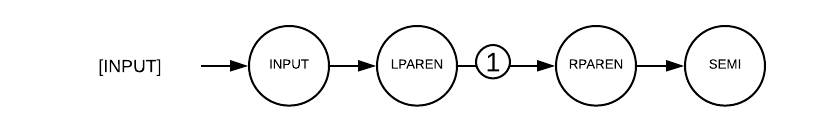
1. Se comprueba que no exista una declaración previa y se agrega su información al directorio de variables (stack de ids y su operador al stack de operadores)
2. Se agrega el cuádruplo con la operación de asignación
3. Se agrega el cuádruplo con la operación de Read
4. Se agrega el cuádruplo con la evaluación de mega\_expresions (operaciones aritmeticas, acceso a variables, llamada a funciones, etc.

**Cte\_Dec: Declaracion de constantes**



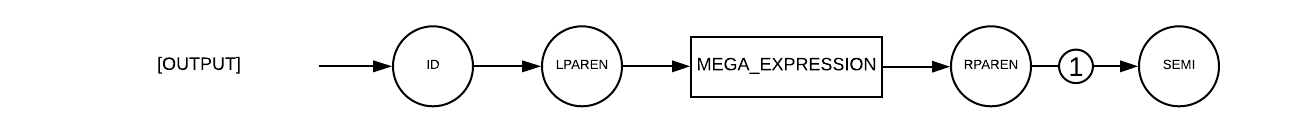
1. Se comprueba que no exista una declaración previa y se agrega su información al directorio de variables (stack de ids y su operador al stack de operadores)
2. Se agrega el cuádruplo con la operación de asignación
3. Se agrega el cuádruplo con la evaluación de mega\_expresions (operaciones aritmeticas, acceso a variables, llamada a funciones, etc.

**Input: Lectura desde el teclado**

****

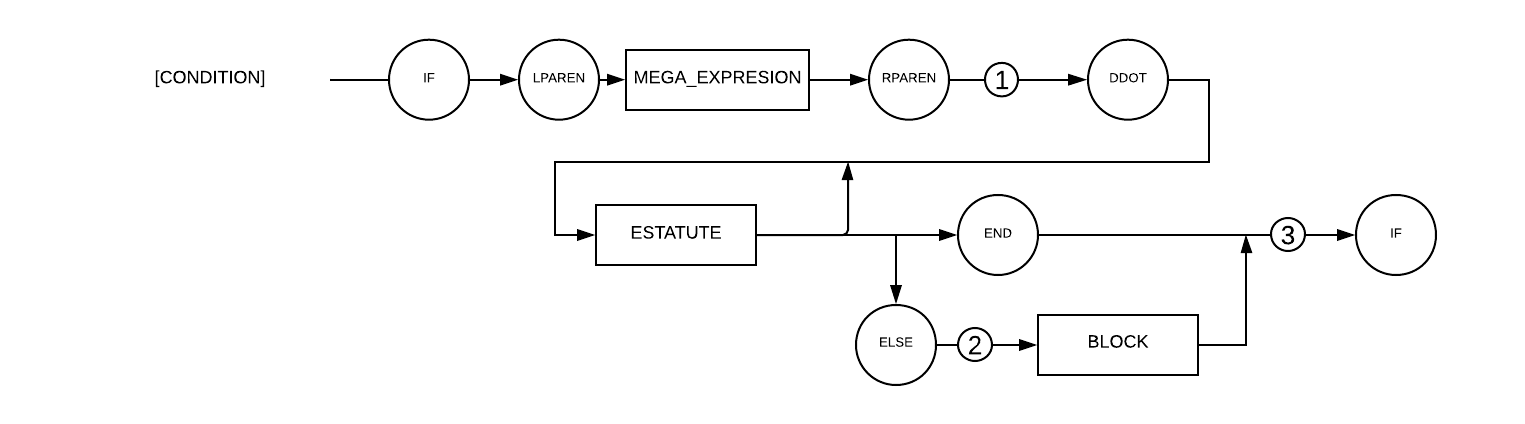
1. Se genera el cuádruplo con la operación Read que indica que se recibirá input desde el teclado

**Output: Print en consola**



1. Se obtiene la última id de la pila de ids y se genera el cuadruplo con la operación PRINT

**Condition: Estatutos para flujo de datos**

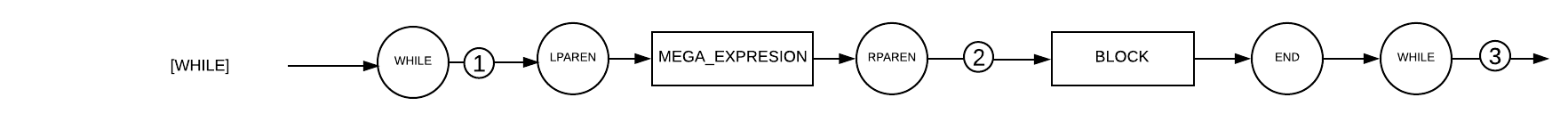


1. Verifica el tipo de valor (si es boolean). Genera cuádruplo con la operación GOTOF y el contador se agrega a la pila de saltos.

2. Genera el cuádruplo con la operacion GOTO utiizando los valores con el contador actual + 1

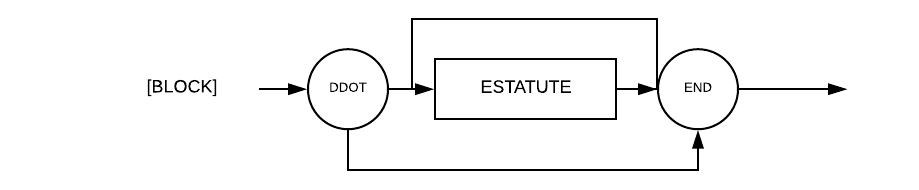
3. Pop a la pila de saltos y rellena el cuádruplo correspondiente con el contador actual.

**While : Patrón del loop while**

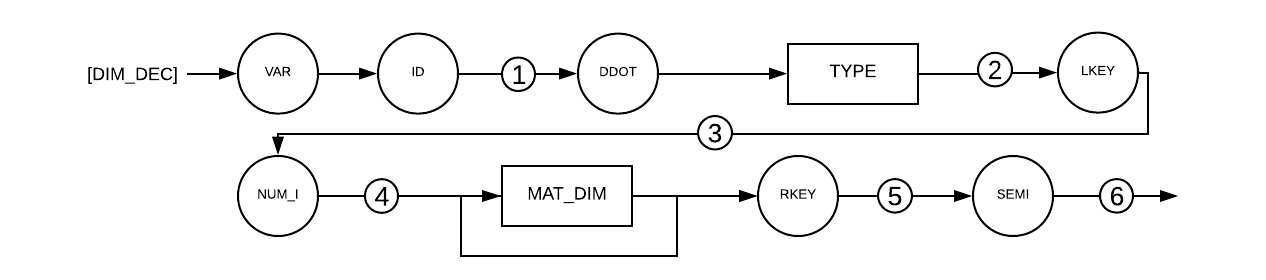
****

1. Al igual que en el estatuto de condición se agrega el contador a la pila de saltos
2. Se genera el cuádruplo con la operación GOTOF sacando el ultimo valor o contador de la pila de saltos utilizando estos datos para esta generación
3. Finaliza y se agrega el cuádruplo del while.

**Block: Patrón “repetitivo” para estatutos de condición y ciclos**



**Dim\_Dec: Declaración de variables dimensionadas**

****

1. Se almacena en una variable el id correspondiente de la declaración y se comprueba que no exista

2. Por medio del administrador de tareas se agrega al directorio de variables la susodicha variable

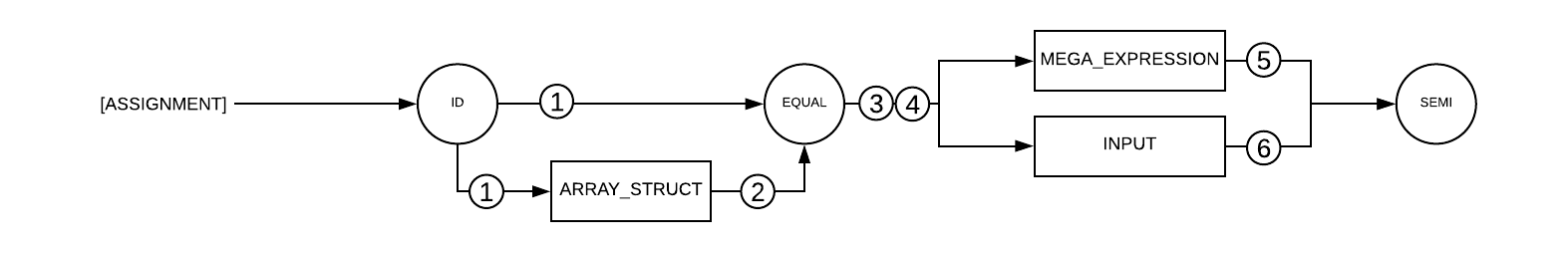
3. Se crea la estructura de la dimensión que albergará la información útil de esta.

4. Se registra el tamaño o los límites de la variable dimensionada

5. Con la información previa se calcula M y se separan los espacios contiguos de memoria para esta variable

6. Se reinicia R

**Assigment: Forma para asignar valores a las variables**

****

1. Se captura el id de la variable para encontrarlo en el contexto y modificarlo

2. Si la variables dimensionada se procede a comprobar que se reciba un número entero (index) con un límite válido.

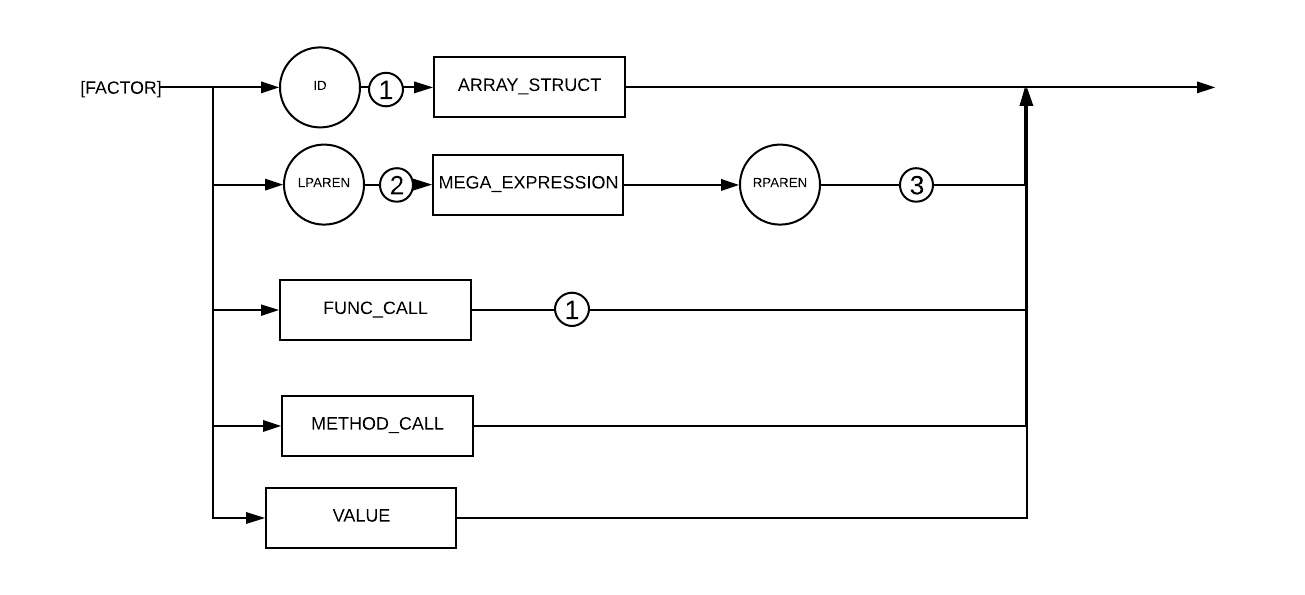
3. Se agrega el operador al stack de operadores.

4. Solo se realizara el assign si la variable existe por lo que se comprueba si y ya está declarada

5. Se agrega el cuádruplo para mega\_expressions con el operador de assign ‘=’

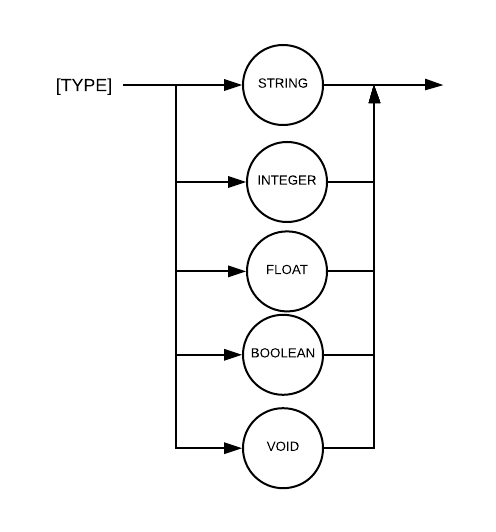
6. Se agrega el cuádruplo para inputs con el operador de assign ‘=’ haciendo uso del procedimiento Read.

**Factor**

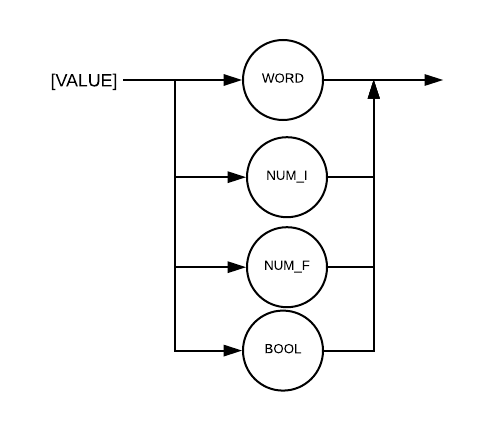


1. Push id a la pila de ids.
2. Se agrega un fondo falso a la pila de operadores.
3. Se quita fondo falso de la pila de operadores.

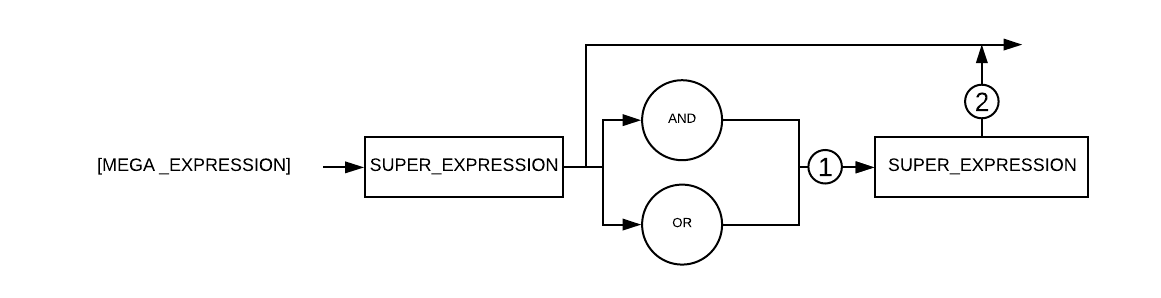
**Type: Tipo de variables permitidas dentro del lenguaje**

****

**Value: Valores aritméticos y booleanos que pueden asignársele a cada tipo de dato**

****

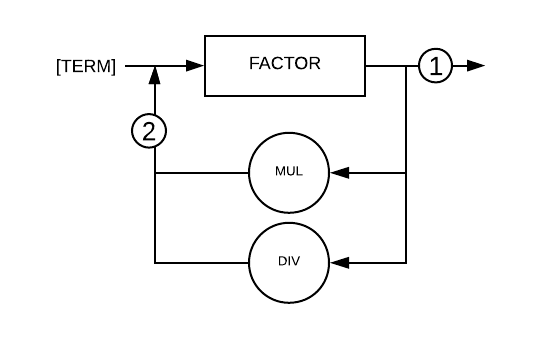
**Mega Expression: Es la que permite realizar casi cualquier llamada para operaciones aritméticas, lógicas y funcionales**

****

1.Agrega el operador que llegue, a la pila de operadores en el generador de cuádruplos

2. Verificación operaciones o llamadas pendientes. Si lo hay genera un cuádruplo con los últimos dos ids de la pila de ids y con el último operador de la pila de operadores

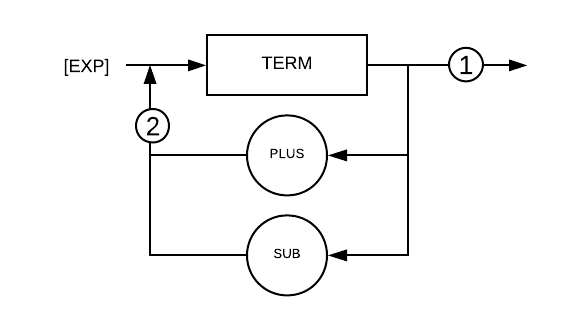
**Term: Operaciones de division y multiplicacion**



1. Verificación de operaciones pendientes y generación de cuádruplo con los últimos ids de la pila y el último operador de la pila de operadores.

2. Agrega operador a la pila de operadores

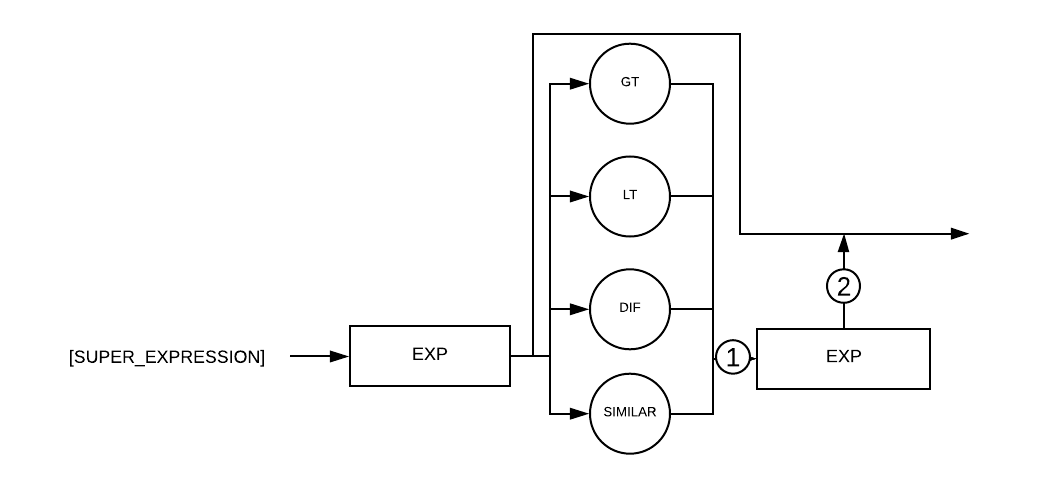
**Exp: Operaciones de suma y resta**

****

1. Verificación de operaciones pendientes y generación de cuádruplo con los últimos ids de la pila y el último operador de la pila de operadores.

2. Agrega operador a la pila de operadores

**Super Expression: Contenida en Mega Expression esta realiza o no operaciones de comparación:**

****

1. Agrega el operador de llegada a su pila

2. Al igual que en mega expression se verifica no haya operaciones pendientes y si las hay se generan los cuádruplos con los ids de su respectiva pila y su operador

**Cubo Semántico**

Nuestro cubo semántico es un hash con arreglos de 3 posiciones donde metemos el operador y los 2 operandos y esa es la key que regresa el tipo de valor.

{["nocortable", "nocortable", "<"]=>"estado",

["nocortable", "nocortable", ">"]=>"estado",

["nocortable", "nocortable", "<="]=>"estado",

["nocortable", "nocortable", ">="]=>"estado",

["nocortable", "nocortable", "=="]=>"estado",

["nocortable", "nocortable", "<>"]=>"estado",

["nocortable", "nocortable", "&&"]=>"estado",

["nocortable", "nocortable", "||"]=>"estado",

["nocortable", "nocortable", "="]=>"nocortable",

["cortable", "cortable", "<"]=>"estado",

["cortable", "cortable", ">"]=>"estado",

["cortable", "cortable", "<="]=>"estado",

["cortable", "cortable", ">="]=>"estado",

["cortable", "cortable", "=="]=>"estado",

["cortable", "cortable", "<>"]=>"estado",

["cortable", "cortable", "&&"]=>"estado",

["cortable", "cortable", "||"]=>"estado",

["cortable", "cortable", "="]=>"cortable",

["nombre", "nombre", "<"]=>"estado",

["nombre", "nombre", ">"]=>"estado",

["nombre", "nombre", "<="]=>"estado",

["nombre", "nombre", ">="]=>"estado",

["nombre", "nombre", "=="]=>"estado",

["nombre", "nombre", "<>"]=>"estado",

["nombre", "nombre", "&&"]=>"estado",

["nombre", "nombre", "||"]=>"estado",

["nombre", "nombre", "="]=>"nombre",

["estado", "estado", "<"]=>"estado",

["estado", "estado", ">"]=>"estado",

["estado", "estado", "<="]=>"estado",

["estado", "estado", ">="]=>"estado",

["estado", "estado", "=="]=>"estado",

["estado", "estado", "<>"]=>"estado",

["estado", "estado", "&&"]=>"estado",

["estado", "estado", "||"]=>"estado",

["estado", "estado", "="]=>"estado",

["proceso", "proceso", "<"]=>"estado",

["proceso", "proceso", ">"]=>"estado",

["proceso", "proceso", "<="]=>"estado",

["proceso", "proceso", ">="]=>"estado",

["proceso", "proceso", "=="]=>"estado",

["proceso", "proceso", "<>"]=>"estado",

["proceso", "proceso", "&&"]=>"estado",

["proceso", "proceso", "||"]=>"estado",

["proceso", "proceso", "="]=>"proceso",

["nocortable", "nocortable", "+"]=>"nocortable",

["nocortable", "nocortable", "-"]=>"nocortable",

["nocortable", "nocortable", ""]=>"nocortable",

["nocortable", "cortable", "+"]=>"cortable",

["nocortable", "cortable", "-"]=>"cortable",

["nocortable", "cortable", ""]=>"cortable",

["nocortable", "nocortable", "/"]=>"cortable",

["cortable", "cortable", "+"]=>"cortable",

["cortable", "cortable", "-"]=>"cortable",

["cortable", "cortable", ""]=>"cortable",

["cortable", "nocortable", "+"]=>"cortable",

["cortable", "nocortable", "-"]=>"cortable",

["cortable", "nocortable", ""]=>"cortable",

["cortable", "cortable", "/"]=>"cortable",

["nocortable", "cortable", "="]=>"cortable",

["nombre", "nombre", "+"]=>"nombre"}

**Descripción de la máquina virtual**

**d.1) Equipo de cómputo, lenguaje y utilerías usadas en el proyecto**

De equipo de cómputo se usó una Macbook Pro y una Windows, el lenguaje utilizado fue Ruby y no usamos ninguna herramienta extra para la máquina virtual.

**d.2) Descripción del proceso de administración de memoria**

Nosotros utilizamos una pila de hashes que guarda la dirección virtual como llave y su valor, la llave también puede ser un String y esto significa que el valor es un parámetro o el valor de retorno de una función.

Estructura:

{

5001 => 20,

5002 => 12.3,

“factorial” => 5,

5003 => verdadero,

param1 => 21

}

Nosotros cuando compilamos le asignamos una dirección de memoria directa a los cuádruplos, no usamos temps, de esta forma la máquina virtual solo tiene que leer los cuádruplos de manera directa y el valor se le asigna a las direcciones virtuales. También para distinguir una constante se le agrega un “@” antes para que la máquina virtual sepa que eso es un valor directo y no una dirección de memoria.

**Pruebas del funcionamiento del lenguaje**

1. **bubble sort.fl**

**Código:**

menu bubble\_sort:

ingrediente arr : nocortable[5];

ingrediente n : nocortable = 5 ;

ingrediente i : nocortable = 0 ;

ingrediente j : nocortable = 0 ;

ingrediente aux : nocortable;

arr[0] = 12 ;

arr[1] = 9 ;

arr[2] = 32 ;

arr[3] = 1 ;

arr[4] = 20 ;

mientras(i < n - 1):

j = 0 ;

mientras(j < n - i - 1):

si (arr[j] > arr[j+1]):

aux = arr[j];

arr[j] = arr[j+1];

arr[j+1] = aux;

terminar si

j = j + 1 ;

terminar mientras

i = i + 1 ;

terminar mientras

i = 0 ;

mostrar("Array:");

mientras(i < n):

mostrar(arr[i]);

i = i + 1 ;

terminar mientras

terminar menu

**Resultado:**

0|GOTO|||1

1|=|@5||5005

2|=|@0||5006

3|=|@0||5007

4|VER|@0|0|4

5|+|@0|@5000|5010

6|=|@12||(5010)

7|VER|@1|0|4

8|+|@1|@5000|5011

9|=|@9||(5011)

10|VER|@2|0|4

11|+|@2|@5000|5012

12|=|@32||(5012)

13|VER|@3|0|4

14|+|@3|@5000|5013

15|=|@1||(5013)

16|VER|@4|0|4

17|+|@4|@5000|5014

18|=|@20||(5014)

19|-|5005|@1|5015

20|<|5006|5015|5016

21|GOTOF|5016||53

22|=|@0||5007

23|-|5005|5006|5017

24|-|5017|@1|5018

25|<|5007|5018|5019

26|GOTOF|5019||50

27|VER|5007|0|4

28|+|5007|@5000|5020

29|+|5007|@1|5021

30|VER|5021|0|4

31|+|5021|@5000|5022

32|>|(5020)|(5022)|5023

33|GOTOF|5023||47

34|VER|5007|0|4

35|+|5007|@5000|5024

36|=|(5024)||5008

37|VER|5007|0|4

38|+|5007|@5000|5025

39|+|5007|@1|5026

40|VER|5026|0|4

41|+|5026|@5000|5027

42|=|(5027)||(5025)

43|+|5007|@1|5028

44|VER|5028|0|4

45|+|5028|@5000|5029

46|=|5008||(5029)

47|+|5007|@1|5030

48|=|5030||5007

49|GOTO|||23

50|+|5006|@1|5031

51|=|5031||5006

52|GOTO|||19

53|=|@0||5006

54|print|@"Array:"||

55|<|5006|5005|5032

56|GOTOF|5032||63

57|VER|5006|0|4

58|+|5006|@5000|5033

59|print|(5033)||

60|+|5006|@1|5034

61|=|5034||5006

62|GOTO|||55

Ingredients retrieved succesfully.

Cooking menu...

"Array:"

1

9

12

20

32

Menu cooked successfully.

1. **chef\_analysis.fl**

**Código:**

platillo Pechuga\_Pollo:

ingrediente sal : nombre;

receta nombre getTipoSal():

servir sal;

terminar receta

receta proceso tipoSal(nombre sal\_name):

sal = sal\_name;

terminar receta

terminar platillo

platillo Pechuga\_Cordon\_Bleu << Pechuga\_Pollo:

ingrediente queso : cortable;

receta cortable getQuesoCant():

servir queso;

terminar receta

receta proceso ponerPimienta(cortable cantQueso):

queso = cantQueso;

terminar receta

receta proceso cocinar(cortable cantQueso, nombre sal\_name):

queso = cantQueso;

sal = sal\_name;

terminar receta

terminar platillo

menu Kitchen:

ingrediente restaurante1 : Pechuga\_Cordon\_Bleu;

restaurante1.tipoSal("sal de mar");

restaurante1.ponerPimienta(7.8);

ingrediente restaurante2 : Pechuga\_Cordon\_Bleu;

restaurante2.tipoSal("sal yodada");

restaurante2.ponerPimienta(9.2);

ingrediente restaurante3 : Pechuga\_Cordon\_Bleu;

restaurante3.cocinar(3.1416,"sal de jamaica");

mostrar(restaurante1.getTipoSal());

mostrar(restaurante1.getQuesoCant());

mostrar(restaurante2.getTipoSal());

mostrar(restaurante2.getQuesoCant());

mostrar(restaurante3.getTipoSal());

mostrar(restaurante3.getQuesoCant());

terminar menu

**Resultado:**

0|GOTO|||16

1|return|5000||

2|ENDFUNC|||

3|=|param0||5002

4|=|5002||5000

5|ENDFUNC|||

6|return|5002||

7|ENDFUNC|||

8|=|param0||5004

9|=|5004||5002

10|ENDFUNC|||

11|=|param0||5005

12|=|param1||5006

13|=|5005||5002

14|=|5006||5000

15|ENDFUNC|||

16|ERA|restaurante1||tipoSal

17|PARAM|@"sal de mar"||param0

18|GOSUB|tipoSal||3

19|=|tipoSal||5008

20|ERA|restaurante1||ponerPimienta

21|PARAM|@7.8||param0

22|GOSUB|ponerPimienta||8

23|=|ponerPimienta||5009

24|ERA|restaurante2||tipoSal

25|PARAM|@"sal yodada"||param0

26|GOSUB|tipoSal||3

27|=|tipoSal||5011

28|ERA|restaurante2||ponerPimienta

29|PARAM|@9.2||param0

30|GOSUB|ponerPimienta||8

31|=|ponerPimienta||5012

32|ERA|restaurante3||cocinar

33|PARAM|@3.1416||param0

34|PARAM|@"sal de jamaica"||param1

35|GOSUB|cocinar||11

36|=|cocinar||5014

37|ERA|restaurante1||getTipoSal

38|GOSUB|getTipoSal||1

39|=|getTipoSal||5015

40|print|5015||

41|ERA|restaurante1||getQuesoCant

42|GOSUB|getQuesoCant||6

43|=|getQuesoCant||5016

44|print|5016||

45|ERA|restaurante2||getTipoSal

46|GOSUB|getTipoSal||1

47|=|getTipoSal||5017

48|print|5017||

49|ERA|restaurante2||getQuesoCant

50|GOSUB|getQuesoCant||6

51|=|getQuesoCant||5018

52|print|5018||

53|ERA|restaurante3||getTipoSal

54|GOSUB|getTipoSal||1

55|=|getTipoSal||5019

56|print|5019||

57|ERA|restaurante3||getQuesoCant

58|GOSUB|getQuesoCant||6

59|=|getQuesoCant||5020

60|print|5020||

Ingredients retrieved succesfully.

Cooking menu...

"sal de mar"

7.8

"sal yodada"

9.2

"sal de jamaica"

3.1416

Menu cooked successfully.

1. **factorial\_recursive.fl**

**Código:**

receta nocortable factorial(nocortable n):

ingrediente res : nocortable;

si(n == 0):

res = 1 ;

sino:

res = n \* factorial(n - 1);

terminar si

servir res;

terminar receta

menu recursiveFactorial:

ingrediente x : nocortable;

x = factorial(8);

mostrar(x);

terminar menu

**Resultado:**

0|GOTO|||15

1|=|param0||5001

2|==|5001|@0|5003

3|GOTOF|5003||6

4|=|@1||5001

5|GOTO|||13

6|ERA|||factorial

7|-|5001|@1|5004

8|PARAM|5004||param0

9|GOSUB|factorial||1

10|=|factorial||5005

11|\*|5001|5005|5006

12|=|5006||5001

13|return|5001||

14|ENDFUNC|||

15|ERA|||factorial

16|PARAM|@8||param0

17|GOSUB|factorial||1

18|=|factorial||5008

19|=|5008||5006

20|print|5006||

Ingredients retrieved succesfully.

Cooking menu...

40320

Menu cooked successfully.

1. **factorial.fl**

**Código:**

menu factorial:

ingrediente n : nocortable = 8 ;

ingrediente factorial : nocortable = 1 ;

ingrediente i : nocortable = 1 ;

mientras(i <= n):

factorial = factorial \* i;

i = i + 1 ;

terminar mientras

mostrar(factorial);

terminar menu

**Resultado:**

0|GOTO|||1

1|=|@8||5000

2|=|@1||5001

3|=|@1||5002

4|<=|5002|5000|5004

5|GOTOF|5004||11

6|\*|5001|5002|5005

7|=|5005||5001

8|+|5002|@1|5006

9|=|5006||5002

10|GOTO|||4

11|print|5001||

Ingredients retrieved succesfully.

Cooking menu...

40320

Menu cooked successfully.

1. **fibonacci\_recursive.fl**

**Código:**

receta nocortable fibonacci(nocortable num):

si(num <= 2):

num = 1 ;

sino:

num = fibonacci(num - 2) + fibonacci(num - 1);

terminar si

servir num;

terminar receta

menu recursiveFibonacci:

ingrediente res : nocortable;

res = fibonacci(20);

mostrar(res);

terminar menu

**Resultado:**

0|GOTO|||20

1|=|param0||5001

2|<=|5001|@2|5002

3|GOTOF|5002||6

4|=|@1||5001

5|GOTO|||18

6|ERA|||fibonacci

7|-|5001|@2|5003

8|PARAM|5003||param0

9|GOSUB|fibonacci||1

10|=|fibonacci||5004

11|ERA|||fibonacci

12|-|5001|@1|5005

13|PARAM|5005||param0

14|GOSUB|fibonacci||1

15|=|fibonacci||5006

16|+|5004|5006|5007

17|=|5007||5001

18|return|5001||

19|ENDFUNC|||

20|ERA|||fibonacci

21|PARAM|@20||param0

22|GOSUB|fibonacci||1

23|=|fibonacci||5009

24|=|5009||5007

25|print|5007||

Ingredients retrieved succesfully.

Cooking menu...

6765

Menu cooked successfully.

1. **fibonacci.fl**

**Código:**

menu fibonacci:

ingrediente i : nocortable = 2 ;

ingrediente x : nocortable = 1 ;

ingrediente y : nocortable = 1 ;

ingrediente aux : nocortable = 0 ;

mostrar(x);

mostrar(y);

mientras(i < 20):

aux = x + y;

x = y;

y = aux;

mostrar(aux);

i = i + 1 ;

terminar mientras

terminar menu

**Resultado:**

0|GOTO|||1

1|=|@2||5000

2|=|@1||5001

3|=|@1||5002

4|=|@0||5003

5|<|5000|@20|5005

6|GOTOF|5005||14

7|+|5001|5002|5006

8|=|5006||5003

9|=|5002||5001

10|=|5003||5002

11|+|5000|@1|5007

12|=|5007||5000

13|GOTO|||5

14|print|5003||

Ingredients retrieved succesfully.

Cooking menu...

6765

Menu cooked successfully.

1. **find.fl**

**Código:**

menu find:

ingrediente arr : nocortable[5];

ingrediente size : nocortable = 5 ;

ingrediente i : nocortable = 0 ;

ingrediente objectiveNumber : nocortable;

ingrediente position : nocortable;

ingrediente numberFound : estado = falso;

arr[0] = 12 ;

arr[1] = 9 ;

arr[2] = 32 ;

arr[3] = 1 ;

arr[4] = 20 ;

mostrar("¿Qué número quieres encontrar?");

objectiveNumber = traer();

mientras (i < size):

si (arr[i] == objectiveNumber):

numberFound = verdadero;

position = i;

terminar si

i = i + 1 ;

terminar mientras

si(numberFound):

mostrar("El número esta en la posición:");

mostrar(position);

sino:

mostrar("Número no existe");

terminar si

terminar menu

**Resultado:**

0|GOTO|||1

1|=|@5||5005

2|=|@0||5006

3|=|@false||5009

4|VER|@0|0|4

5|+|@0|@5000|5011

6|=|@12||(5011)

7|VER|@1|0|4

8|+|@1|@5000|5012

9|=|@9||(5012)

10|VER|@2|0|4

11|+|@2|@5000|5013

12|=|@32||(5013)

13|VER|@3|0|4

14|+|@3|@5000|5014

15|=|@1||(5014)

16|VER|@4|0|4

17|+|@4|@5000|5015

18|=|@20||(5015)

19|print|@"¿Qué número quieres encontrar?"||

20|read||5007|nocortable

21|=|5007||5007

22|<|5006|5005|5016

23|GOTOF|5016||33

24|VER|5006|0|4

25|+|5006|@5000|5017

26|==|(5017)|5007|5018

27|GOTOF|5018||30

28|=|@true||5009

29|=|5006||5008

30|+|5006|@1|5019

31|=|5019||5006

32|GOTO|||22

33|GOTOF|5009||37

34|print|@"El número esta en la posición:"||

35|print|5008||

36|GOTO|||38

37|print|@"Número no existe"||

Ingredients retrieved succesfully.

Cooking menu...

"¿Qué número quieres encontrar?"

20

"El número esta en la posición:"

4

Menu cooked successfully.

1. **matrixMult.fl**

**Código:**

menu matrixMultiplication:

ingrediente mat1 : nocortable[3,3];

ingrediente mat2 : nocortable[3,3];

ingrediente result : nocortable[3,3];

ingrediente i : nocortable = 0 ;

ingrediente j : nocortable = 0 ;

ingrediente k : nocortable = 0 ;

mat1[0,0] = 1 ;

mat1[0,1] = 2 ;

mat1[0,2] = 3 ;

mat1[1,0] = 4 ;

mat1[1,1] = 5 ;

mat1[1,2] = 6 ;

mat1[2,0] = 7 ;

mat1[2,1] = 8 ;

mat1[2,2] = 9 ;

mat2[0,0] = 1 ;

mat2[0,1] = 2 ;

mat2[0,2] = 3 ;

mat2[1,0] = 4 ;

mat2[1,1] = 5 ;

mat2[1,2] = 6 ;

mat2[2,0] = 7 ;

mat2[2,1] = 8 ;

mat2[2,2] = 9 ;

mientras(i < 3):

j = 0 ;

mientras(j < 3):

k = 0 ;

result[i,j] = 0 ;

mientras(k < 3):

result[i,j] = result[i,j] + (mat1[i,k] \* mat2[k,j]);

k = k + 1 ;

terminar mientras

j = j + 1 ;

terminar mientras

i = i + 1 ;

terminar mientras

i = 0 ;

mientras(i < 3):

j = 0 ;

mientras(j < 3):

mostrar(result[i,j]);

j = j + 1 ;

terminar mientras

i = i + 1 ;

terminar mientras

terminar menu

**Resultado:**

0|GOTO|||1

1|=|@0||5027

2|=|@0||5028

3|=|@0||5029

4|VER|@0|0|2

5|\*|@0|@3|5031

6|VER|@0|0|2

7|+|5031|@0|5032

8|+|5032|@5000|5033

9|=|@1||(5033)

10|VER|@0|0|2

11|\*|@0|@3|5034

12|VER|@1|0|2

13|+|5034|@1|5035

14|+|5035|@5000|5036

15|=|@2||(5036)

16|VER|@0|0|2

17|\*|@0|@3|5037

18|VER|@2|0|2

19|+|5037|@2|5038

20|+|5038|@5000|5039

21|=|@3||(5039)

22|VER|@1|0|2

23|\*|@1|@3|5040

24|VER|@0|0|2

25|+|5040|@0|5041

26|+|5041|@5000|5042

27|=|@4||(5042)

28|VER|@1|0|2

29|\*|@1|@3|5043

30|VER|@1|0|2

31|+|5043|@1|5044

32|+|5044|@5000|5045

33|=|@5||(5045)

34|VER|@1|0|2

35|\*|@1|@3|5046

36|VER|@2|0|2

37|+|5046|@2|5047

38|+|5047|@5000|5048

39|=|@6||(5048)

40|VER|@2|0|2

41|\*|@2|@3|5049

42|VER|@0|0|2

43|+|5049|@0|5050

44|+|5050|@5000|5051

45|=|@7||(5051)

46|VER|@2|0|2

47|\*|@2|@3|5052

48|VER|@1|0|2

49|+|5052|@1|5053

50|+|5053|@5000|5054

51|=|@8||(5054)

52|VER|@2|0|2

53|\*|@2|@3|5055

54|VER|@2|0|2

55|+|5055|@2|5056

56|+|5056|@5000|5057

57|=|@9||(5057)

58|VER|@0|0|2

59|\*|@0|@3|5058

60|VER|@0|0|2

61|+|5058|@0|5059

62|+|5059|@5009|5060

63|=|@1||(5060)

64|VER|@0|0|2

65|\*|@0|@3|5061

66|VER|@1|0|2

67|+|5061|@1|5062

68|+|5062|@5009|5063

69|=|@2||(5063)

70|VER|@0|0|2

71|\*|@0|@3|5064

72|VER|@2|0|2

73|+|5064|@2|5065

74|+|5065|@5009|5066

75|=|@3||(5066)

76|VER|@1|0|2

77|\*|@1|@3|5067

78|VER|@0|0|2

79|+|5067|@0|5068

80|+|5068|@5009|5069

81|=|@4||(5069)

82|VER|@1|0|2

83|\*|@1|@3|5070

84|VER|@1|0|2

85|+|5070|@1|5071

86|+|5071|@5009|5072

87|=|@5||(5072)

88|VER|@1|0|2

89|\*|@1|@3|5073

90|VER|@2|0|2

91|+|5073|@2|5074

92|+|5074|@5009|5075

93|=|@6||(5075)

94|VER|@2|0|2

95|\*|@2|@3|5076

96|VER|@0|0|2

97|+|5076|@0|5077

98|+|5077|@5009|5078

99|=|@7||(5078)

100|VER|@2|0|2

101|\*|@2|@3|5079

102|VER|@1|0|2

103|+|5079|@1|5080

104|+|5080|@5009|5081

105|=|@8||(5081)

106|VER|@2|0|2

107|\*|@2|@3|5082

108|VER|@2|0|2

109|+|5082|@2|5083

110|+|5083|@5009|5084

111|=|@9||(5084)

112|<|5027|@3|5085

113|GOTOF|5085||158

114|=|@0||5028

115|<|5028|@3|5086

116|GOTOF|5086||155

117|=|@0||5029

118|VER|5027|0|2

119|\*|5027|@3|5087

120|VER|5028|0|2

121|+|5087|5028|5088

122|+|5088|@5018|5089

123|=|@0||(5089)

124|<|5029|@3|5090

125|GOTOF|5090||152

126|VER|5027|0|2

127|\*|5027|@3|5091

128|VER|5028|0|2

129|+|5091|5028|5092

130|+|5092|@5018|5093

131|VER|5027|0|2

132|\*|5027|@3|5094

133|VER|5028|0|2

134|+|5094|5028|5095

135|+|5095|@5018|5096

136|VER|5027|0|2

137|\*|5027|@3|5097

138|VER|5029|0|2

139|+|5097|5029|5098

140|+|5098|@5000|5099

141|VER|5029|0|2

142|\*|5029|@3|5100

143|VER|5028|0|2

144|+|5100|5028|5101

145|+|5101|@5009|5102

146|\*|(5099)|(5102)|5103

147|+|(5096)|5103|5104

148|=|5104||(5093)

149|+|5029|@1|5105

150|=|5105||5029

151|GOTO|||124

152|+|5028|@1|5106

153|=|5106||5028

154|GOTO|||115

155|+|5027|@1|5107

156|=|5107||5027

157|GOTO|||112

158|=|@0||5027

159|<|5027|@3|5108

160|GOTOF|5108||176

161|=|@0||5028

162|<|5028|@3|5109

163|GOTOF|5109||173

164|VER|5027|0|2

165|\*|5027|@3|5110

166|VER|5028|0|2

167|+|5110|5028|5111

168|+|5111|@5018|5112

169|print|(5112)||

170|+|5028|@1|5113

171|=|5113||5028

172|GOTO|||162

173|+|5027|@1|5114

174|=|5114||5027

175|GOTO|||159

Ingredients retrieved succesfully.

Cooking menu...

30

36

42

66

81

96

102

126

150

Menu cooked successfully.

**Listados Perfectamente Documentados del Proyecto**

Nosotros utilizamos una gema de ruby desarrollada por nosotros en un hackathon hace unos años que se llama Martha y te documenta el código solita y tu le pasas la descripción de lo que hace el método.

En lo que más nos tardamos hacer fueron los cuádruplos pero sin una buena máquina virtual los cuádruplos de nada sirven y no gusto la forma eficiente en la que implementamos nuestra máquina virtual. Solo incluimos los métodos más complejos.

class VirtualMachine  
#Title: gosub  
#Input: func\_name, jump  
#Input Description: function name and jump for execution  
#Output Description: Nothing  
#Description: Creates new context for function and adds function return to stack for recursive case and quad number set to the start of function  
#Author: Victor Elizalde  
 def gosub(func\_name, jump) #method  
 @current\_context = @new\_context  
 @return\_stack.push(@quad\_number)  
 @quad\_number = jump - 1  
 end  
  
#Input Description: Nothing  
#Output Description: Nothing  
#Description: Starts execution, calls main function and interprets each quadruple and when done it calls the finish function to stop everything  
#Author: Victor Elizalde  
 def start() #method  
 puts "Cooking menu..."  
 while @quad\_number < @quads.length  
 interpret\_quad(@quads[@quad\_number])  
 @quad\_number += 1  
 end  
 finish()  
 end

#Title: endfunc  
#Input:  
#Input Description: Nothing  
#Output Description: Nothing  
#Description: kills function context and return to pending one and also kills return  
#Author: Victor Elizalde  
 def endfunc() #method  
 @current\_context = @context\_stack.pop()  
 @quad\_number = @return\_stack.pop()  
 end  
  
#Title: verify\_limits  
#Input: value, limit1, limit2  
#Input Description: dim variable and limits  
#Output Description: Nothing  
#Description: verifies limit of dimensional object and raises error if not correct  
#Author: Victor Elizalde  
 def verify\_limits(value, limit1, limit2) #method  
 value = get\_value(value)  
 if !value.between?(limit1, limit2)  
 puts "Fatal Error: #{value} exceeds limit."  
 exit  
 end  
 end  
  
#Title: era  
#Input: object\_name, func\_name  
#Input Description: object and function  
#Output Description: Nothing  
#Description: Initializes context either for function or method  
#Author: Victor Elizalde  
 def era(object\_name, func\_name) #method  
 @functions\_stack.push(func\_name)  
 @context\_stack.push(@current\_context)  
  
 if object\_name == nil  
 @new\_context = {}  
 else  
 if @object\_contexts[object\_name] == nil  
 @object\_contexts[object\_name] = {}  
 @new\_context = @object\_contexts[object\_name]  
 else  
 @new\_context = @object\_contexts[object\_name]  
 end  
 end  
 end  
  
#Title: interpret\_quad  
#Input: quad  
#Input Description: quadruple  
#Output Description: Nothing  
#Description: calls function for quadruple type  
#Author: Victor Elizalde  
 def interpret\_quad(quad) #method  
 process = quad.operator  
 case process  
 when '+'  
 add(quad.first\_col, quad.second\_col, quad.result)  
 when '-'  
 subtract(quad.first\_col, quad.second\_col, quad.result)  
 when '\*'  
 multiply(quad.first\_col, quad.second\_col, quad.result)  
 when '/'  
 divide(quad.first\_col, quad.second\_col, quad.result)  
 when '<'  
 less\_than(quad.first\_col, quad.second\_col, quad.result)  
 when '>'  
 greater\_than(quad.first\_col, quad.second\_col, quad.result)  
 when '<='  
 less\_or\_equal\_than(quad.first\_col, quad.second\_col, quad.result)  
 when '>='  
 greater\_or\_equal\_than(quad.first\_col, quad.second\_col, quad.result)  
 when '=='  
 equal(quad.first\_col, quad.second\_col, quad.result)  
 when '<>'  
 different\_from(quad.first\_col, quad.second\_col, quad.result)  
 when '&&'  
 and\_op(quad.first\_col, quad.second\_col, quad.result)  
 when '||'  
 or\_op(quad.first\_col, quad.second\_col, quad.result)  
 when '='  
 assignation(quad.first\_col, quad.result)  
 when 'GOTO'  
 @quad\_number = quad.result - 1  
 when 'GOTOF'  
 gotof(quad.first\_col, quad.result)  
 when 'ERA'  
 era(quad.first\_col, quad.result)  
 when 'PARAM'  
 param(quad.first\_col, quad.result)  
 when 'GOSUB'  
 gosub(quad.first\_col, quad.result)  
 when 'ENDFUNC'  
 endfunc()  
 when 'VER'  
 verify\_limits(quad.first\_col, quad.second\_col, quad.result)  
 when 'read'  
 reading(quad.second\_col, quad.result)  
 when 'print'  
 writing(quad.first\_col)  
 when 'return'  
 return\_func(quad.first\_col)  
 end  
 end  
  
#Title: reading  
#Input: var, var\_type  
#Input Description: variable to put value, variable type  
#Output Description: Nothing  
#Description: Assigns user input value to variable  
#Author: Victor Elizalde  
 def reading(var, var\_type) #method  
 user\_input = $stdin.gets.chomp  
 user\_input\_type = get\_type(user\_input)  
 user\_input\_class = user\_input\_type.class  
 type = digest\_type(var\_type, user\_input)  
 if user\_input\_class.to\_s == type  
 @current\_context[var] = user\_input\_type  
 else  
 puts "Fatal Error: semantic cube error, expected #{var\_type} and got #{user\_input\_class}."  
 exit  
 end  
 end  
end

**Manual de Usuario foodLOOP**

Es un lenguaje orientado a objetos que utiliza términos culinarios para programar, haciendo más fácil para la gente empezar a aprender fundamentos de programación.

**Tipos de Datos**

| **Tipo** | **foodLOOP** |
| --- | --- |
| Integer | nocortable |
| Float | cortable |
| String | nombre |
| Booleano | estado |

**Operadores**

| **Operador** | **Tipo** |
| --- | --- |
| **+** | **Aritmético** |
| **-** | **Aritmético** |
| **\*** | **Aritmético** |
| **/** | **Aritmético** |
| **<** | **Lógico** |
| **>** | **Lógico** |
| **<=** | **Lógico** |
| **>=** | **Lógico** |
| **<>** | **Lógico** |
| **==** | **Lógico** |
| **&&** | **Lógico** |
| **||** | **Lógico** |

**Caracteres Especiales**

| **Carácter** | **Uso** |
| --- | --- |
| = | Asignación |
| ; | Final de línea |
| . | llamada de método de objeto |

**Declaración de Variables**

Todas las variables son ingredientes.

Ejemplo:

Ingrediente a : nocortable; <------ Varible vacia

Ingrediente a : nocortable = 5; <------ Varible con valor

Ingrediente a : nocortable[5]; <------ Arreglo

Ingrediente a : nocortable[5,5]; <------ Matriz

**Asignar valores a arreglos o matrices**

Ejemplo:

A[0] = 2;

A[0,0] = 4;

**Declaración de funciones**

Se define el tipo de retorno luego el nombre de variable, los parametros y servir es el return.

Ejemplo:

receta nocortable suma(nocortable a, nocortable b)

servir a+b;

terminar receta

**Llamada de funciones**

Ejemplo:

a = suma(5,5);

**if and else**

Ejemplo:

si(1 < 0) :

a = 1;

sino :

A = 0;

terminar si

**While**

Ejemplo:

mientras (1<0):

A = a + 1;

terminar mientras

**Lectura**

Ejemplo:

a = traer();

**Escritura**

Ejemplo:

mostrar(a); ←-- valor de variable

mostrar(“a”); ←-- String

**Clases**

Ejemplo:

platillo Pechuga\_Pollo:

ingrediente sal : nombre;

receta nombre getSal()

servir sal;

terminar receta

terminar platillo

**Herencia**

Ejemplo:

platillo Pechuga\_Pollo:

ingrediente sal : nombre = “sal de mar”;

receta nombre getSal()

servir sal;

terminar receta

terminar platillo

platillo Cordon\_Bleu << Pechuga\_Pollo:

ingrediente sal : nombre;

receta nombre getSal()

servir sal;

terminar receta

terminar platillo

**Declarar Objetos**

Ejemplo:

ingrediente restaurante1 : Pechuga\_Pollo;

**Llamada a métodos de objetos**

Ejemplo:

restaurante1.getSal();

**Extensión de programas**

La extensión de los programas es .fl para que los reconozca el compilador

**Estructura del programa**

Primero se ponen las clases, luego las funciones y después el programa, todos son opcionales de poner menos el programa.

**Instalación**

Bajar el ambiente y correr todos los programas con **ruby test.rb nombre\_de\_mi\_programa.fl**