Code to Paint

Proyecto final Diseño de compiladores.

3 de mayo del 2017

David Eusebio Cazares Pérez

DESCRIPCION DEL PROYECTO, 3

DESCRIPCIÓN DEL LENGUAJE, 6

DESCRIPCION DEL COMPILADOR, 7

DESCRIPCION DE LA MÁQUINA VIRTUAL, 18

PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE, 20

LISTADOS PERFECTAMENTE DOCUMENTADOS DEL PROYECTO, 28

DESCRIPCION DEL PROYECTO

El objetivo principal del proyecto es desarrollar, por medio de herramientas y recursos existentes, un lenguaje de programación que permita de manera sencilla dibujar figuras y trazos en la pantalla de la computadora.

La categoría del proyecto entra en la sección de un programa para ayudar a jóvenes a desarrollar un entendimiento básico de programación de manera divertida por medio de comandos sencillos. Dichos comandos se utilizarán para la creación de un dibujo que se presentará en la pantalla de la computadora. Mediante comandos sencillos y básicos, como mover a la derecha cierta cantidad de distancia en la pantalla, formarán diferentes trazos que conformarán al final un dibujo.

El proyecto utilizará la librería de “Turtle graphics” para dibujar sobre la pantalla. Se usará ply para la creación de reglas sintácticas y gramaticales.

Se espera que el proyecto sea capaz de generar dibujos básicos por medio de las diferentes funciones y así mismo que pueda lograr las funciones básicas de un lenguaje de programación. Entre dichas funciones se encuentran la generación de funciones y variables, tanto como la recursividad, ciclos y condicionales. También se espera que se logre el uso de variables dimensionadas.

Análisis de requerimientos y casos de uso:

1. El lenguaje maneja valores booleanos.
2. El lenguaje maneja valores dobles(flotantes).
3. El lenguaje maneja valores enteros.
4. La suma entre un valor entero y doble regresará un entero.
5. La resta entre un valor entero y doble resultará en un entero.
6. La asignación de un valor doble a una variable entera regresará un entero.
7. La comparación menor que entre dos dígitos, enteros o dobles, regresará un booleano.
8. La comparación mayor que entre dos dígitos, enteros o dobles, regresará un booleano.
9. La comparación igual que entre dos dígitos, enteros o dobles, regresará un booleano.
10. Las operaciones entre enteros y dobles no establecidas en los puntos anteriores regresarán un doble.
11. Los valores booleanos no pueden ser asignados a variables enteras o dobles.
12. Los valores numéricos al ser usados en comparaciones se tomarán como verdad si el valor es mayor o igual a cero o falso de ser negativo.
13. Todas las comparaciones regresan un valor booleano.
14. Se permite el uso de variables globales.
15. Se permite el uso de variables locales.
16. Se permite el uso de variables constantes.
17. Se permite el uso de variables dimensionadas de N dimensiones.
18. Las variables dimensionadas deben ser declaradas con un valor entero constante.
19. Las variables dimensionadas pueden ser accedidas con cualquier tipo de valor entero.
20. Se permite la declaración de funciones.
21. Todas las funciones deben contener un y solo un retorno.
22. La declaración de funciones debe poseer los parámetros y el tipo de retorno.
23. Es posible realizar funciones aritméticas.
24. Es posible realizar funciones de comparación.
25. Es posible la asignación de variables a la hora de la declaración.
26. Se permite la llamada de funciones dentro de funciones, permitiendo el uso de recursividad.
27. Se permite el uso de ciclos(while, do while).
28. Se permite el uso de decisiones (if) y la opción de tener o no un else que le siga.
29. Todas las expresiones a ser evaluadas dentro de la decisión de los estatutos if, while y dowhile deben ser de tipo booleano.
30. El lenguaje cuenta con una salida gráfica que se verá en pantalla a la hora de ejecución pero cerrará a la hora de terminar.
31. La salida del lenguaje(dibujo) se guardará al final de la ejecución en un archivo con nombre “drawing” y con extensión “ews”.
32. Se cuenta con funciones para la manipulación del ambiente de dibujo, las cuales son “PenUp”, “PenDown”, “PenColor”, “PenSize”, “SetX”, “SetY”.
33. Se cuenta con la función de “MoveForward” para dibujar cierta cantidad de pixeles hacia la dirección que se esté apuntando.
34. Las funciones “MoveOnX” y “MoveOnY” siempre dibujaran en línea recta sobre el eje de las X y el eje de las Y respectivamente. Y no moverán la dirección del cursor.
35. Las funciones “RotateLeft” y “RotateRight” permiten girar el cursor hacia cierta dirección en grados.
36. Las funciones de movimiento no dibujarán a menos que se haya utilizado la función “PenDown” con anterioridad.
37. Al inicio del programa la función “PenDown” ha sido llamada así que dibujará a menos que se ejecute la función “PenUp”.
38. Las funciones de “Rectangle”, “Circle” y “Triangle” dibujarán según las medidas pasadas pero siempre hacia la dirección que apunte el cursor.
39. El lenguaje debe ser capaz de detectar errores de sintaxis, es decir un estatuto mal formado.
40. El lenguaje es capaz de detectar errores de tipos no compatibles, es decir tratar de realizar acciones con tipos no compatibles entre sí.
41. El lenguaje debe ser capaz de detectar si una variable no ha sido declarada o asignada.
42. El lenguaje debe ser capaz de detectar si una función ya ha sido declarada o asignada.

Casos de pruebas generales:

* Realización de operación aritmética con dos valores.
* Realización de operación lógica con dos valores.
* Realización de operación relacional con dos valores.
* Pruebas para ingresar un valor desde el teclado.
* Pruebas para imprimir un valor en pantalla.
* Realización de función Fibonacci usando recursividad.
* Realización de función Fibonacci usando ciclos.
* Realización de una función de ordenamiento sobre una variable dimensionada desordenada.
* Realización de una función de búsqueda para una variable dimensionada.
* Realización de un dibujo en pantalla usando los diferentes elementos y funciones de dibujo.

El proceso para el desarrollo del proyecto fue llevado acabo por medio de un manejador de versiones, github, y se separó en diferentes etapas. Dichas etapas fueron el léxico y gramática. El cubo semántico. Directorios de variables y funciones. La lógica básica de expresiones aritméticas, lógicas y relacionales. La generación de cuádruplos. La elaboración de lógica para procedimientos. Finalmente la máquina virtual.

En lo personal las cosas que más se aprendieron fueron la lógica de expresiones y términos para llegar a generar los cuádruplos. El manejo de memoria fue algo complicado por la razón que costaba entender la separación de la memoria a la hora de compilar y ejecutar. Colocar las diferentes acciones en su lugar correspondiente, ya sea en la máquina virtual o en la generación de código intermedio o semántica/gramática.

DESCRIPCION DEL LENGUAJE:

Se decidió que el nombre del lenguaje sería “Code To Paint”.

El lenguaje utiliza diferentes herramientas básicas que componen un lenguaje de programación, como lo son ciclos y condiciones, para generar en la pantalla diferentes trazos que conforman un dibujo. El lenguaje posee tres diferentes tipos de datos; enteros, doble o flotantes y booleanos. El lenguaje está destinado a que gente sin muchos conocimientos se adentren al mundo de la programación al poder ver como mediante la lógica se generan diferentes trazos para lograr un dibujo. El lenguaje es similar a C en muchos aspectos a la hora de desarrollar en él.

Entre sus principales características se encuentra la posibilidad de mover el cursos para generar trazos así como mediante funciones especiales para la generación de figuras básicas como rectángulos, triángulos y círculos.

A pesar de tener salida gráfica también se puede utilizar para comprender conceptos básicos de programación como lo son la declaración de variables y funciones. La resolución de expresiones aritméticas. La generación de ciclos y condicionales y la lógica detrás de variables no lineales.

Los errores que pueden ocurrir son los siguientes:

“variable already defined on global scope” .- Sucede cuando una variable ha sido declarada anteriormente.

“var A has 2 dimensions, 3 given”.- Sucede cuando la cantidad de dimensiones dadas para acceder a una variable no concuerda con la cantidad a la hora de ser definida.

“var A not defined on line” .- La variable que se intenta utilizar no ha sido definida con anterioridad.

“var A is an array, on line”.- La variable intenta ser utilizada sin dimensiones cuando fue declarada como un arreglo.

"var A is not an array, on line ".- La variable intenta ser accedida con dimensiones cuando no es un arreglo.

"arrays must be accessed by an int, boolean given in var A, on line.- Se intenta acceder a la variable usando dimensiones diferentes a números enteros.

“function myFunction needs 4 elements, 2 given, on line”.- Se intenta realizar la llamada a una función con una cantidad diferente de elementos a la definida.

“function B not declared on line”.- La función llamada no ha sido declarada con anterioridad.

“Type mismatch on line”.- Se intenta realizar una operación con tipos incompatibles. Diferente tipo de argumentos.

"Function B already declared on line ".- La función ya ha sido declarada previamente.

“If must have a boolean as expression on line “.- Se intenta realizar un If con algo diferente a un booleano.

“While must have a boolean as expression on line”.- Igual al error pasado pero con un estatuto while.

"return type must be int, boolean given on function B, on line ".- Se utilizó un valor de retorno diferente al declarado en el encabezado de la función.

"Input needs a regular variable, not an array".- El estatuto input solo se puede utilizar con variable lineales.

“type mismatch, PenColor uses 3 integers between 0 and 255, on line “.- Existen diferentes tipos de errores para las funciones especiales cada uno indicando el error al ser utilizadas. El más común es intentar utilizar la función con parámetros incompatibles.

DESCRIPCION DEL COMPILADOR:

El equipo que se utilizó para el desarrollo fue una computadora personal de la marca Apple con sistema operativo Sierra versión 10.12.14. Se utiliza también el lenguaje Python versión 2.7 y las librerías de PLY para el desarrollo del lenguaje. Para el output gráfico se ocupó la librería de Turtle Graphics.

Las palabras reservadas que se utilizarán en el proyecto son las siguientes:

PROGRAM ::= 'program'

MAIN ::= 'main'

END ::= 'end'

FUNCTION ::= 'function'

RETURN ::= 'return'

IF ::= 'if'

ELSE ::= 'else'

WHILE ::= 'while'

DO ::= 'do'

INT ::= 'int'

DOUBLE ::= 'double'

BOOLEAN ::= 'boolean'

PENUP ::= 'penUp'

PENDOWN ::= 'penDown'

PENCOLOR ::= 'penColor'

PENSIZE ::= 'penSize'

SETX ::= 'setX'

SETY ::= 'setY'

CLEAR ::= 'clear'

MOVEONX ::= 'moveOnX'

MOVEONY ::= 'moveOnY'

ROTATETORIGHT ::= 'rotateToRight'

ROTATETOLEFT ::= 'rotateToLeft'

RECTANGLE ::= 'rectangle'

TRIANGLE ::= 'triangle'

CIRCLE ::= 'circle'

MOVEFORWARD ::= ‘moveForward’

PRINT ::=’print’

INPUT ::=’input’

Las tokens que se utilizaron, con sus respectivas expresiones regulares son los siguientes:

PLUS = '\+'

MINUS = '-'

MULT = '\\*'

DIVIDE = '/'

LPAREN = '\('

RPAREN = '\)'

LBRACE = '\{'

RBRACE = '\}'

LBRACKET = '\['

RBRACKET = '\]'

EQUALS = '=='

ASSIGN = '='

LESS = '<'

GREATER = '>'

SEMICOLON = ';'

COMMA = ','

AND = '&'

OR = '\|'

COMMENT = '\#.\*(\n | $) '

CONST\_INT = '-?\d+'

CONST\_DOUBLE = '-?\d+\.\d+'

CONST\_BOOLEAN = '(True|False)'

ID = '[a-zA-Z\_][a-zA-Z\_0-9]\*'

Descripción del Análisis de Sintaxis.

Las siguientes fueron las reglas que se utilizaron para formar la grámatica formal. De las reglas a continuación el código que se utilizó en cada una fue dejado fuera y algunas reglas que se crearon con el único propósito de insertar código fueron omitidas al igual. Las palabras en mayúsculas representan Tokens mientras que las palabras en minúsculas representan el paso a alguna otra regla. Lo anterior al igual que los nombre escogidos para las reglas fueron determinados bajo una convención.

* program: PROGRAM ID SEMICOLON global\_declaration function\_declaration main END SEMICOLON
* global\_declaration : declaration\_statute global\_declaration | empty
* function\_declaration: function\_header function\_main function\_declaration | empty
* function\_header: FUNCTION return\_type\_declared LPAREN params\_declaration RPAREN
  + return\_type\_declared: DOUBLE ID | INT ID | BOOLEAN ID
  + param\_declaration: INT ID mult\_params\_declaration | DOUBLE ID mult\_params\_declaration | BOOLEAN ID mult\_params\_declaration
  + mult\_params\_declaration: COMMA param\_declaration | empty
* function\_main: LBRACE statute return\_statute RBRACE
  + return\_statute: RETURN expression SEMICOLON
* main: MAIN LBRACE statute RBRACE
* declaration\_statute: INT array int\_assignation SEMICOLON | DOUBLE array double\_assignation SEMICOLON| BOOLEAN array boolean\_assignation SEMICOLON
  + array: LBRACKET CONST\_INT RBRACKET array | empty
* int\_assignation: ASSIGN expression | empy
* double\_assignation: ASSIGN expression | empy
* boolean\_assignation: ASSIGN expression | empy
* assignation\_statute: ID array\_u ASSIGN expression SEMICOLON
* array\_u: LBRACKET expression RBRACKET array\_u | empty
* function: LPAREN params RPAREN | array\_u
  + params: expression mult\_params | empty
  + mult\_params: COMMA params | empty
* expression: logical
* logical: relational logical\_main
  + logical\_main: AND logical | OR logical | empty
* relational: sum relational\_main
  + relational\_main: EQUALS relational | LESS relational | GREATER relational | empty
* sum: mult sum\_main
  + sum\_main: PLUS sum | MINUS sum | empty
* mult: term mult\_main
  + mult\_main: MULT mult | DIVIDE mult | empty
* term: CONST\_INT | CONST\_DOUBLE | CONST\_BOOLEAN| ID function | LPAREN logical RPAREN
* statute: if\_statute statute | while\_statute statute| do\_while\_statute statute| assignation\_statute statute| declaration\_statute statute| function\_statute statute| comment\_statute statute| print\_statute statute| graphic\_statute statute| input\_statute statute| empty
* if\_statute: IF LPAREN expression RPAREN LBRACE statute RBRACE else
  + else ELSE LBRACE statute RBRACE| empty
* while\_statute: WHILE LPAREN expression RPAREN LBRACE statute RBRACE
* do\_while\_statute: DO LBRACE statute RBRACE WHILE LPAREN expression RPAREN
* function\_statute: ID LPAREN params RPAREN SEMICOLON
* comment\_statute: COMMENT
* print\_statute: PRINT LPAREN expression RPAREN SEMICOLON
* input\_statute: INPUT LPAREN ID RPAREN SEMICOLON
* graphic\_statute: pen\_up\_statute| pen\_down\_statute| pen\_size\_statute| pen\_color\_statute| set\_x\_statute| set\_y\_statute| clear\_statute| move\_on\_x\_statute| move\_on\_y\_statute| move\_forward\_statute| rotate\_to\_right\_statute| rotate\_to\_left\_statute| rectangle\_statute| triangle\_statute| circle\_statute
* pen\_up\_statute: PENUP LPAREN RPAREN SEMICOLON
* pen\_down\_statute: PENDOWN LPAREN RPAREN SEMICOLON
* pen\_size\_statute: PENSIZE LPAREN expression RPAREN SEMICOLON
* pen\_color\_statute: PENCOLOR LPAREN expression COMMA expression COMMA expression RPAREN SEMICOLON
* set\_x\_statute: SETX LPAREN expression RPAREN SEMICOLON
* set\_y\_statute: SETY LPAREN expression RPAREN SEMICOLON
* clear\_statute: CLEAR LPAREN RPAREN SEMICOLON
* move\_on\_x\_statute: MOVEONX LPAREN expression RPAREN SEMICOLON
* move\_on\_y\_statute: MOVEONY LPAREN expression RPAREN SEMICOLON
* move\_forward\_statute: MOVEFORWARD LPAREN expression RPAREN SEMICOLON
* rotate\_to\_right\_statute: ROTATETORIGHT LPAREN expression RPAREN SEMICOLON
* rotate\_to\_left\_statute: ROTATETOLEFT LPAREN expression RPAREN SEMICOLON
* rectangle\_statute: RECTANGLE LPAREN expression COMMA expression RPAREN SEMICOLON
* triangle\_statute: TRIANGLE LPAREN expression RPAREN SEMICOLON
* circle\_statute: CIRCLE LPAREN expression RPAREN SEMICOLON

Código de operaciones:

'PLUS' : 0,

'MINUS' : 1,

'MULT' : 2,

'DIVIDE' : 3,

'EQUALS' : 4,

'ASSIGN' : 5,

'LESS' : 6,

'GREATER' : 7,

'AND' : 8,

'OR' : 9,

'goto' : 10,

'gotoF' : 11,

'gotoT' : 12,

'return' : 13,

'era' : 14,

'goSub' : 15,

'param' : 16,

'end' : 17,

'verify' : 18,

'penDown' : 19,

'penUp' : 20,

'penSize' : 21,

'penColor' : 22,

'setX' : 23,

'setY' : 24,

'rotateRight' : 25,

'rotateLeft' : 26,

'moveX' : 27,

'moveY' : 28,

'clear' : 29,

'moveForward' : 30,

'circle' : 31,

'print' : 32,

'loadMemory' : 33,

'restoreMemory' : 34,

'input' : 35,

'beginFill' : 36,

'endFill' : 37

Memoria virtual:

Globales enteros: 0

Globales dobles: 1000

Globales booleanos: 2000

Constantes enteras: 3000

Constantes dobles: 4000

Constantes booleanas: 5000

Temporales enteros: 6000

Temporales dobles : 7000

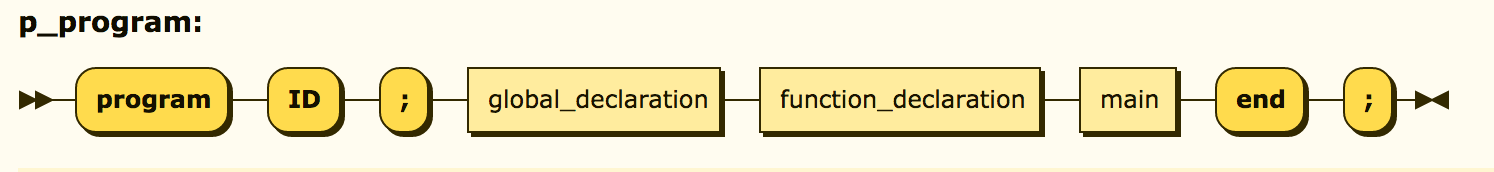
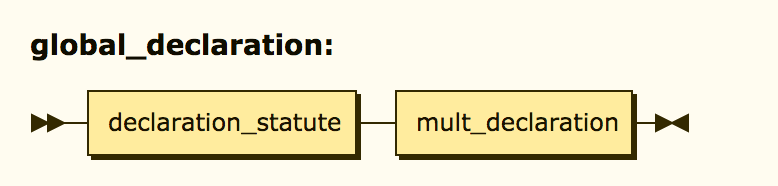
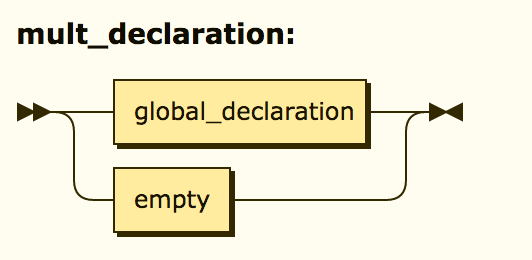
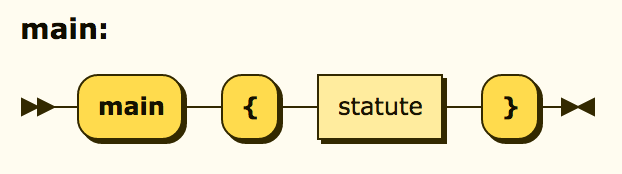
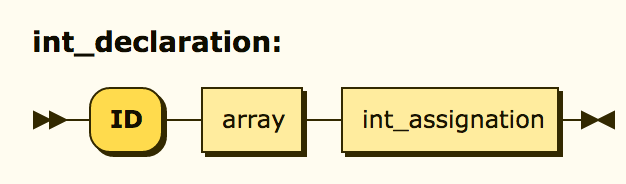
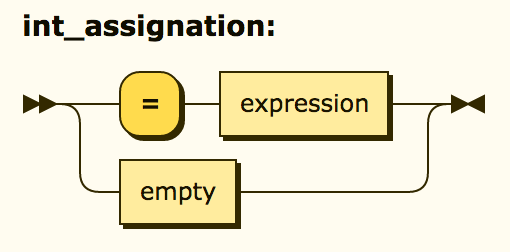
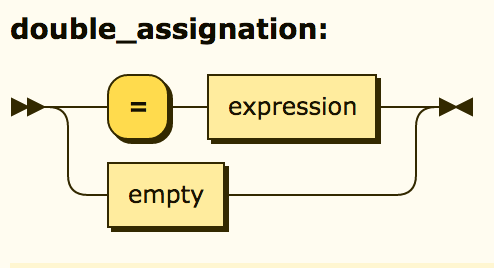
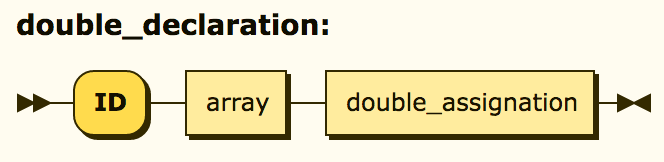
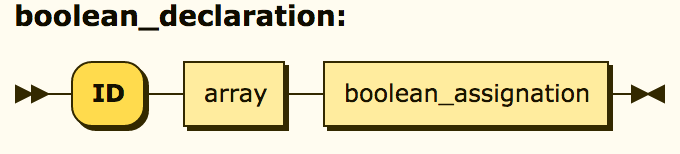
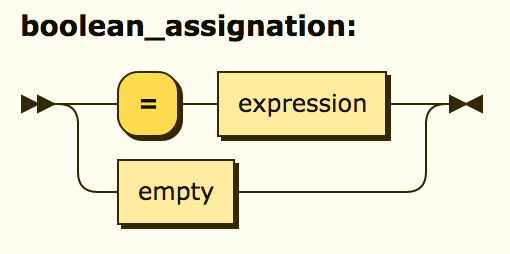
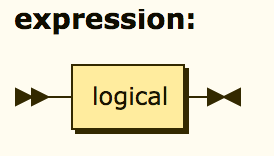
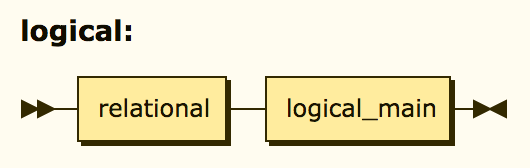
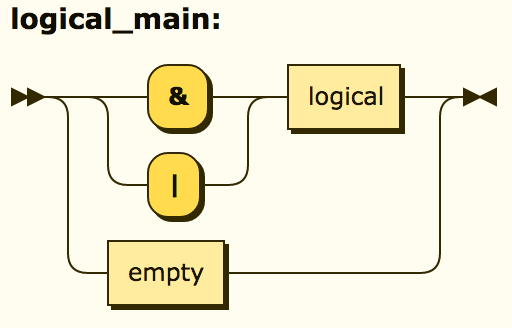
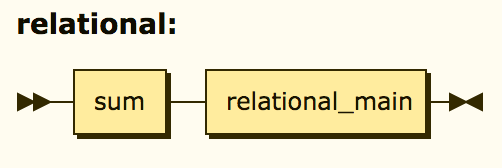
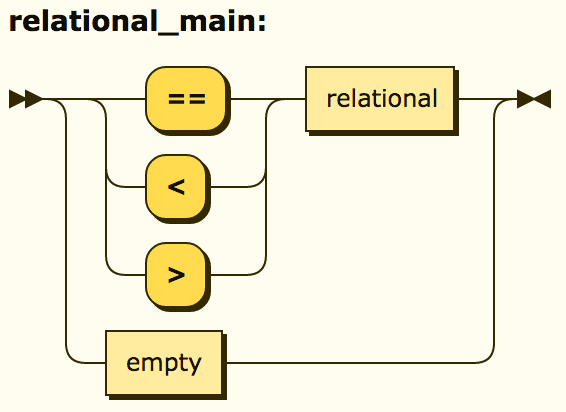
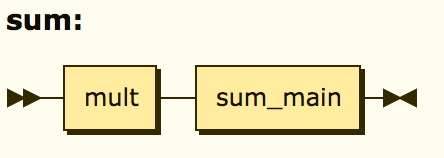
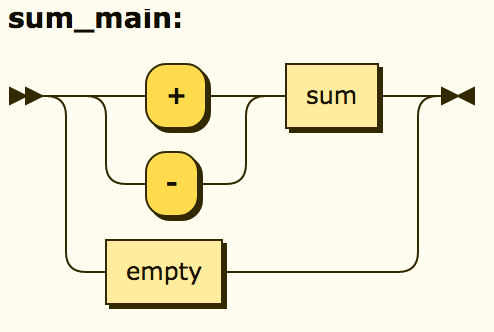
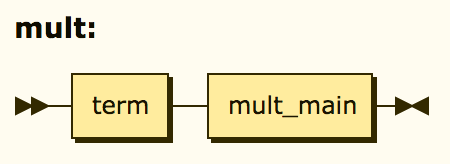
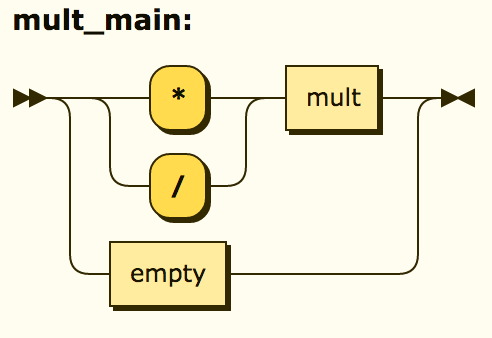
Temporales booleanos: 8000

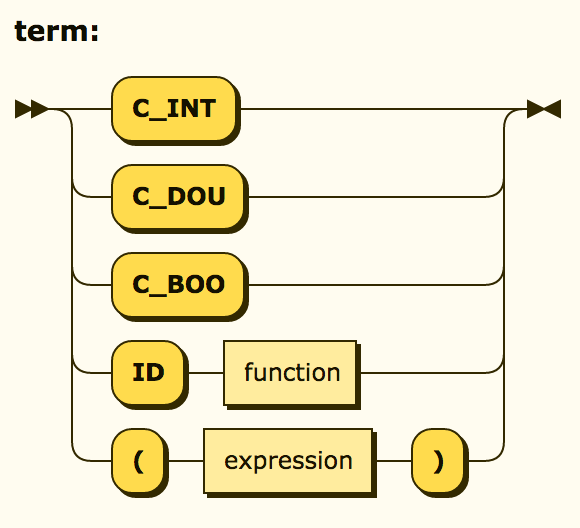
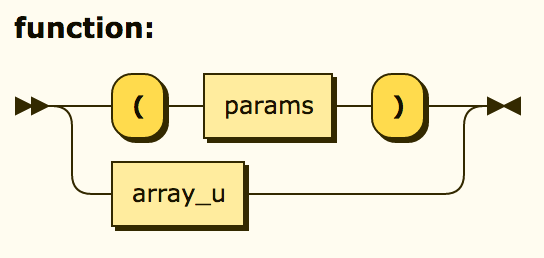
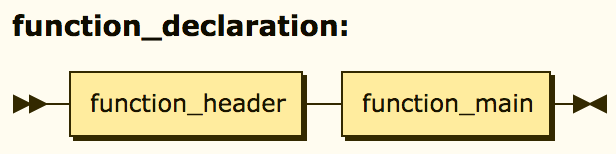
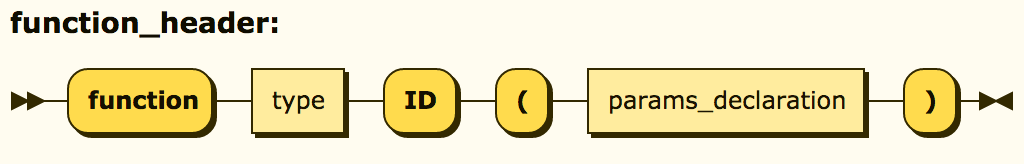
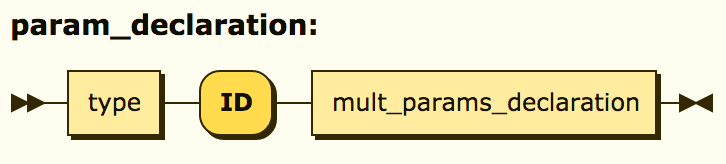
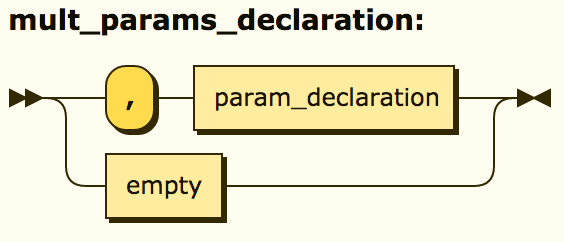
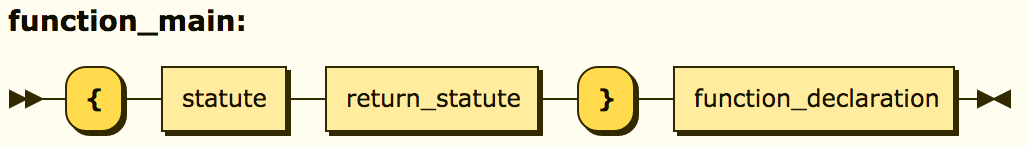
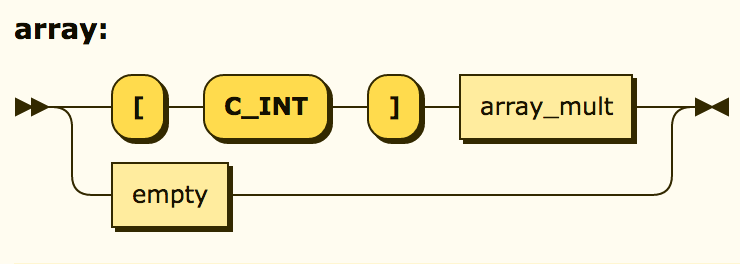
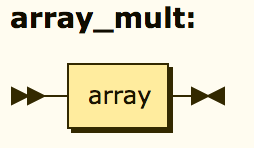
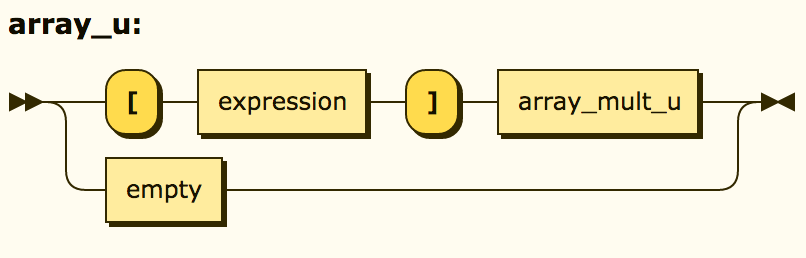
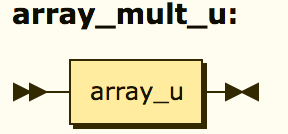
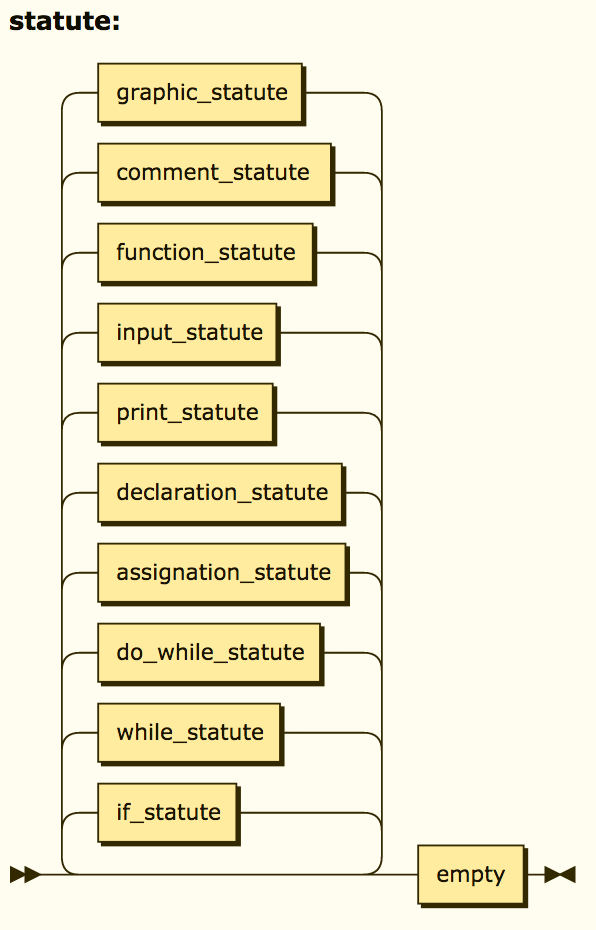
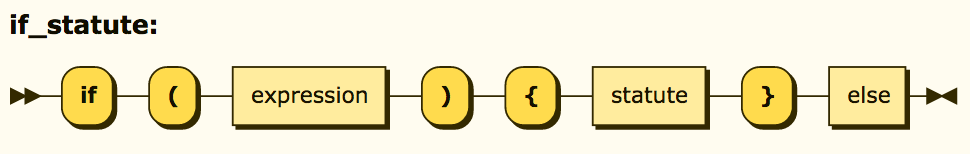
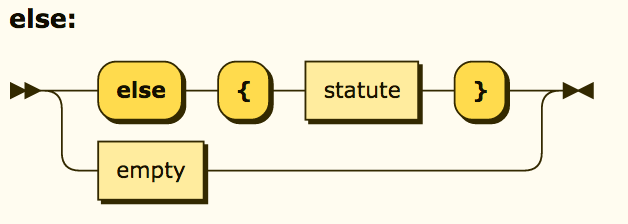
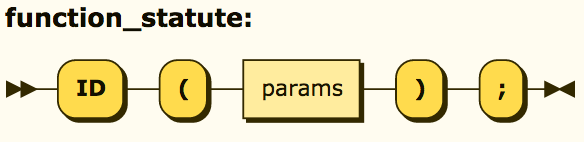
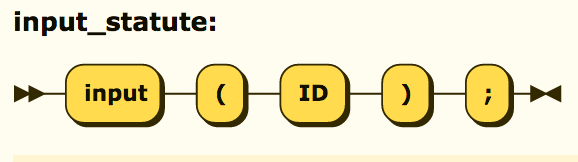
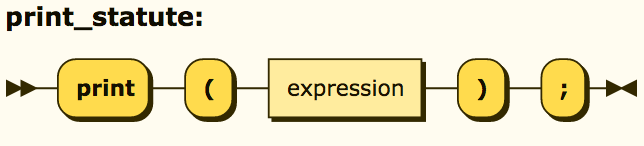
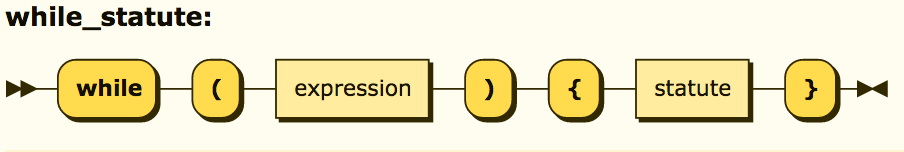
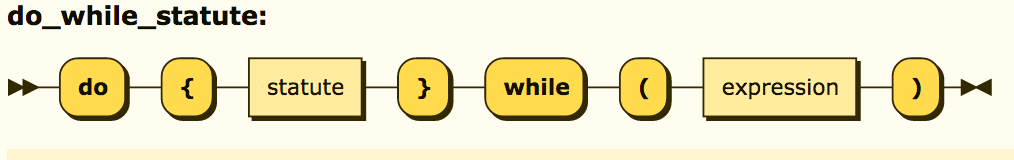
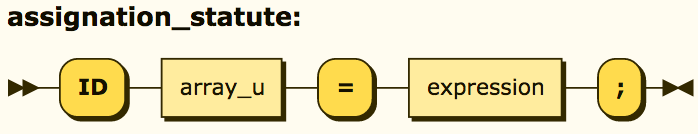
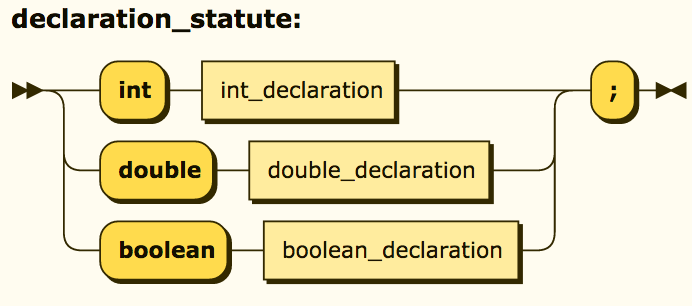
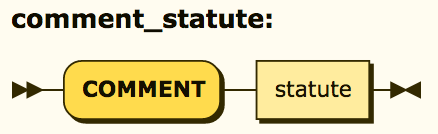
Locales enteros: 9000

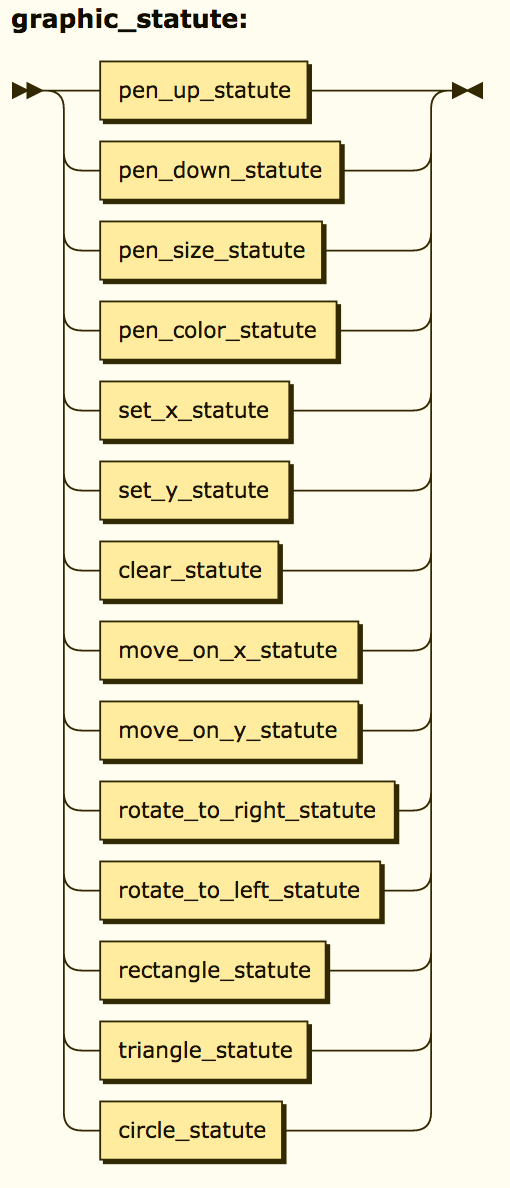
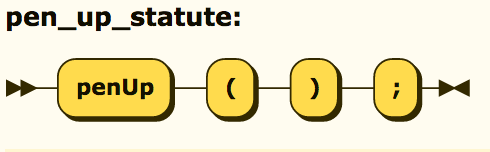
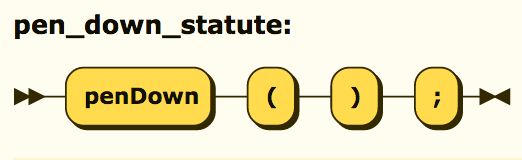
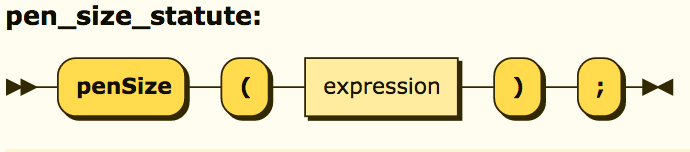
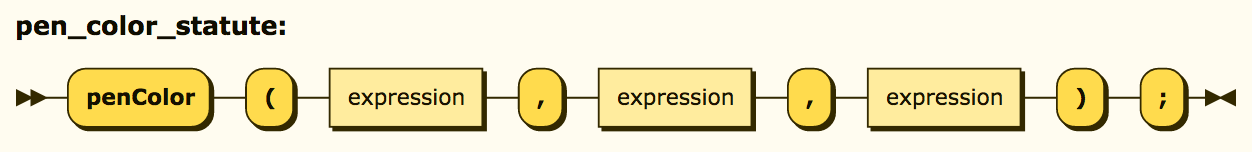
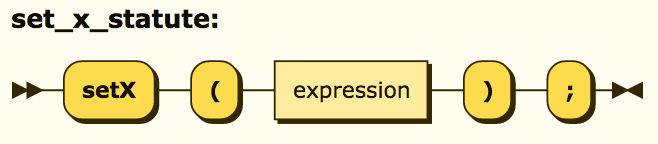
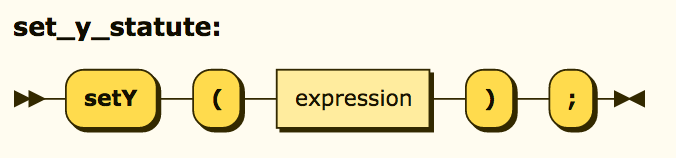
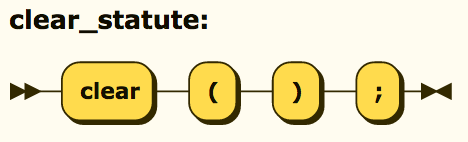
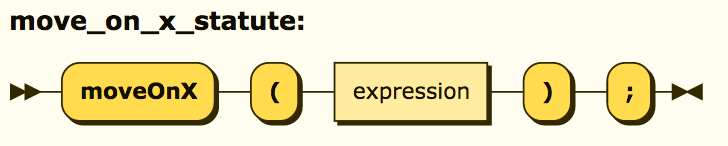
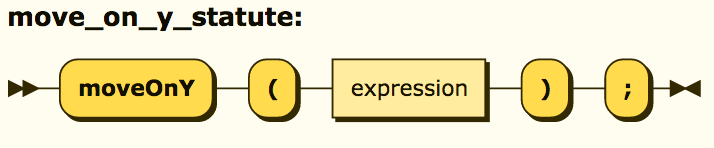
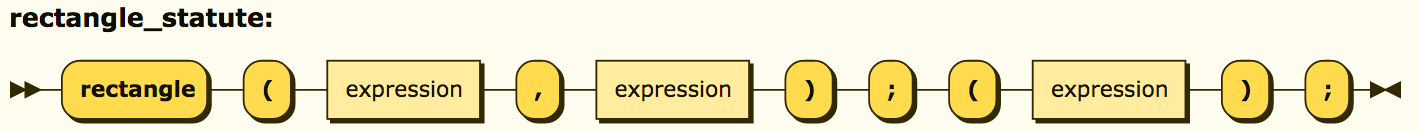
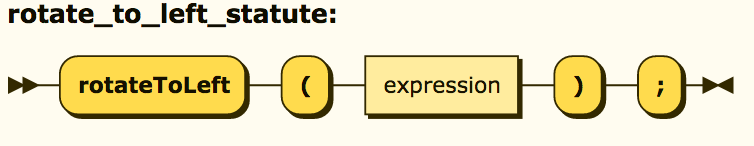
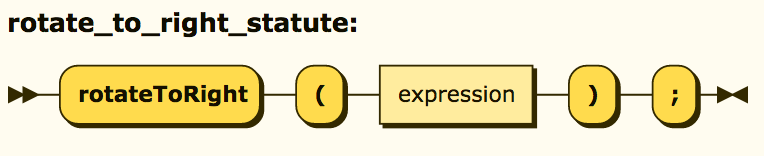
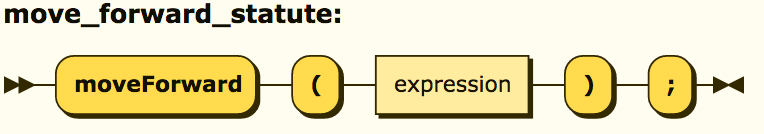
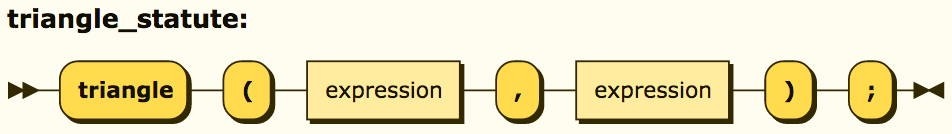
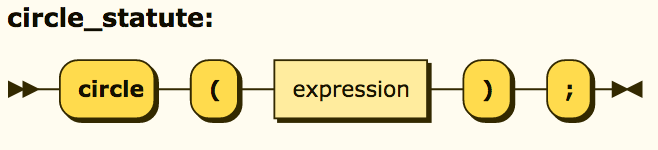
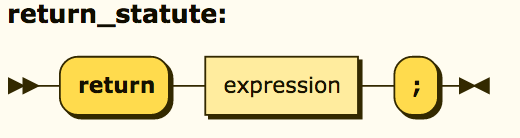
Locales dobles: 10000

Locales booleanos: 11000

Diagramas de sintaxis.







Consideraciones semánticas:

#0 +

int + int = int

int + double = int

int + boolean = -1

double + int = int

double + double = double

double + boolean = -1

boolean + int = -1

boolean + double = -1

boolean + boolean = -1

#1 -

int - int = int

int - double = int

int - boolean = -1

double - int = int

double - double = double

double - boolean = -1

boolean - int = -1

boolean - double = -1

boolean - boolean = -1

#2 \*

int \* int = int

int \* double = double

int \* boolean = -1

double \* int = double

double \* double = double

double \* boolean = -1

boolean \* int = -1

boolean \* double = -1

boolean \* boolean = -1

#3 /

int / int = int

int / double = double

int / boolean = -1

double / int = double

double / double = double

double / boolean = -1

boolean / int = -1

boolean / double = -1

boolean / boolean = -1

#4 ==

int == int = boolean

int == double = boolean

int == boolean = -1

double == int = boolean

double == double = boolean

double == boolean = -1

boolean == int = -1

boolean == double = -1

boolean == boolean = -1

#5 =

int = int = int

int = double = int

int = boolean = -1

double = int = double

double = double = double

double = boolean = -1

boolean = int = -1

boolean = double = -1

boolean = boolean = boolean

#6 <

int < int = boolean

int < double = boolean

int < boolean = -1

double < int = boolean

double < double = boolean

double < boolean = -1

boolean < int = -1

boolean < double = -1

boolean < boolean = -1

#7 >

int > int = boolean

int > double = boolean

int > boolean = -1

double > int = boolean

double > double = boolean

double > boolean = -1

boolean > int = -1

boolean > double = -1

boolean > boolean = -1

#8 &

int & int = boolean

int & double = boolean

int & boolean = boolean

double & int = boolean

double & double = boolean

double & boolean = boolean

boolean & int = boolean

boolean & double = boolean

boolean & boolean = boolean

#9 |

int | int = boolean

int | double = boolean

int | boolean = boolean

double | int = boolean

double | double = boolean

double | boolean = boolean

boolean | int = boolean

boolean | double = boolean

boolean | boolean = boolean

#18 verify

int verify int = int

int verify double = -1

int verify boolean = -1

double verify int = -1

double verify double = -1

double verify boolean = -1

boolean verify int = -1

boolean verify double = -1

boolean verify boolean = -1

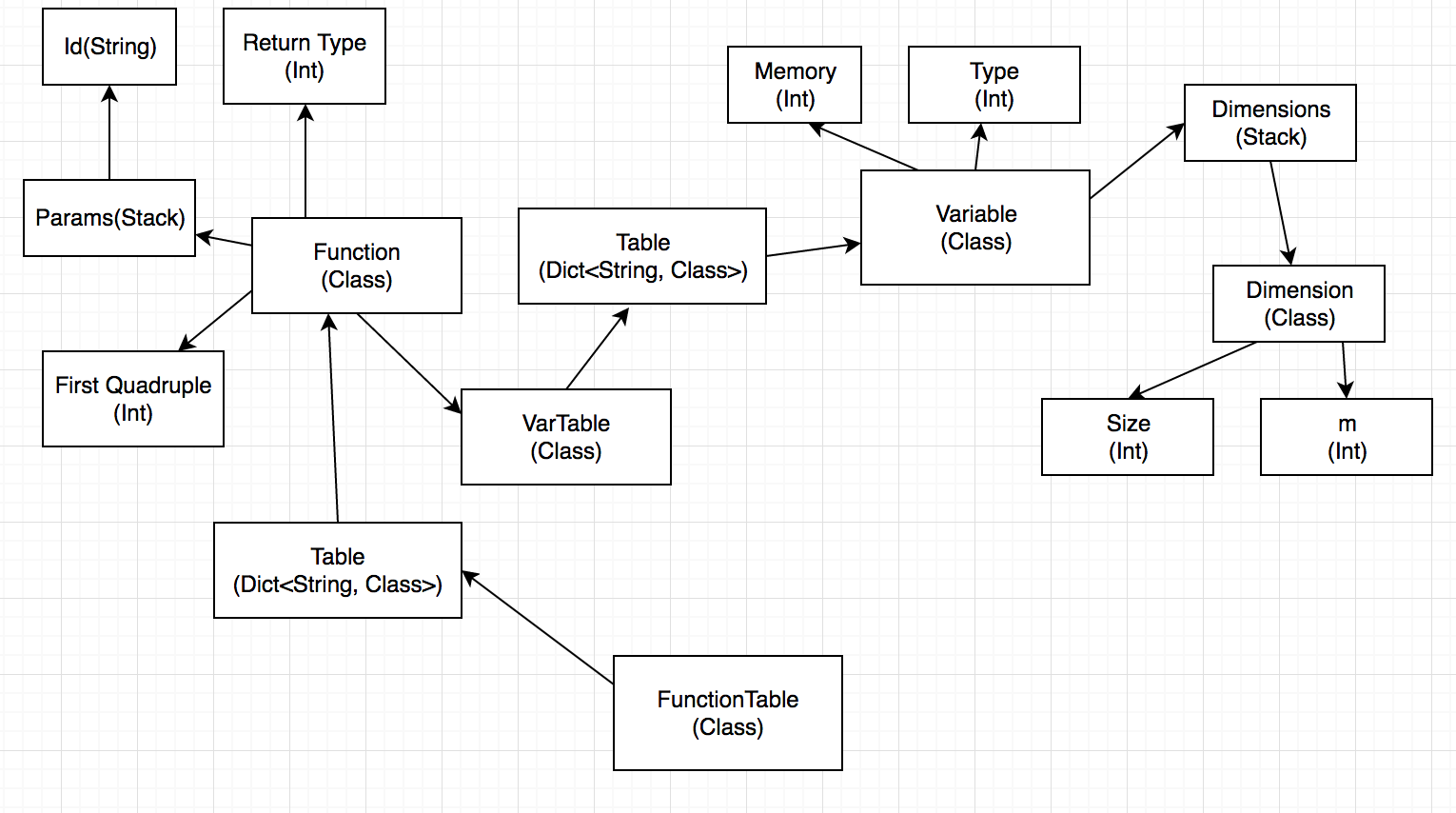
Administración de memoria.

Directorio de funciones y variables.

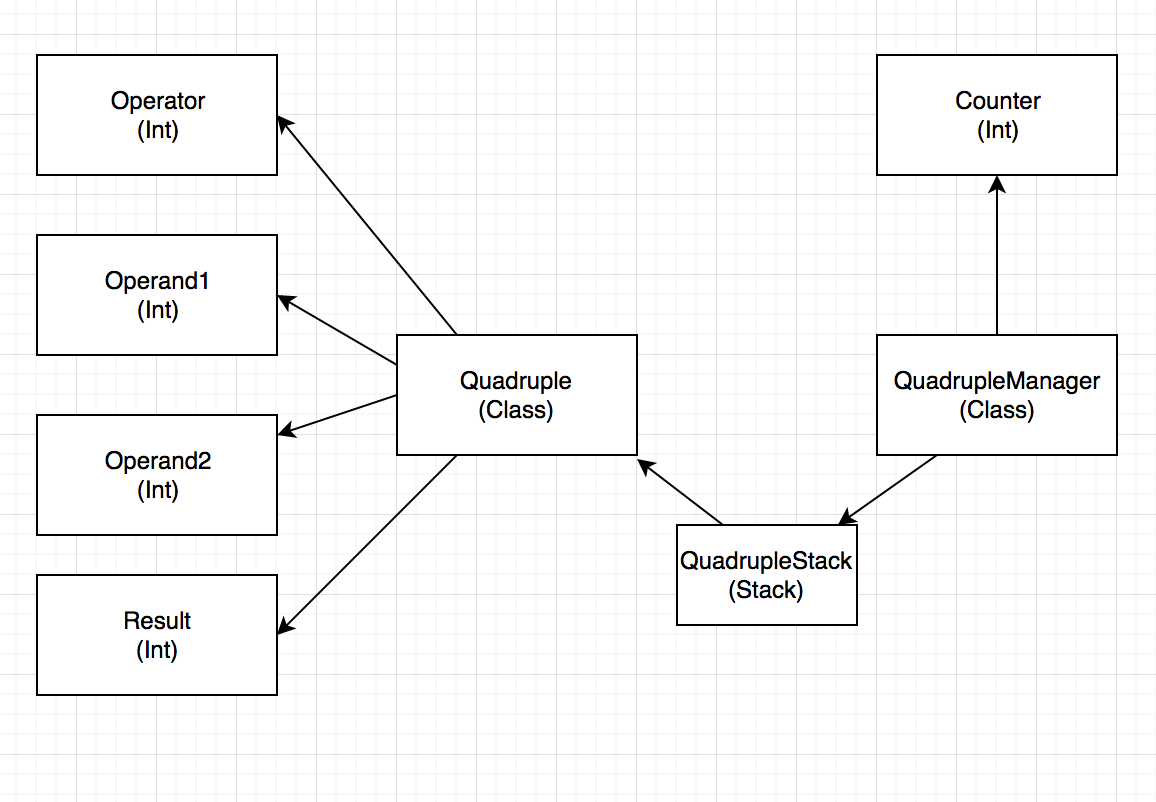
La memoria a la hora de compilación fue manejada por una clase MemoryManager la cuál poseía diferentes métodos para incrementar la memoria. En cuanto a variables se utilizó las funciones dentro de la tabla de funciones para llevar el conteo de la memoria utilizada y de la siguiente en ser asignada. Para ello se llevaba el conteo de la memoria y así mismo se guardó la cantidad de variables que requeriría una función a la hora de ser llamada.

La estructura utilizada fue la siguiente en cuanto el almacenamiento de variables y funciones.

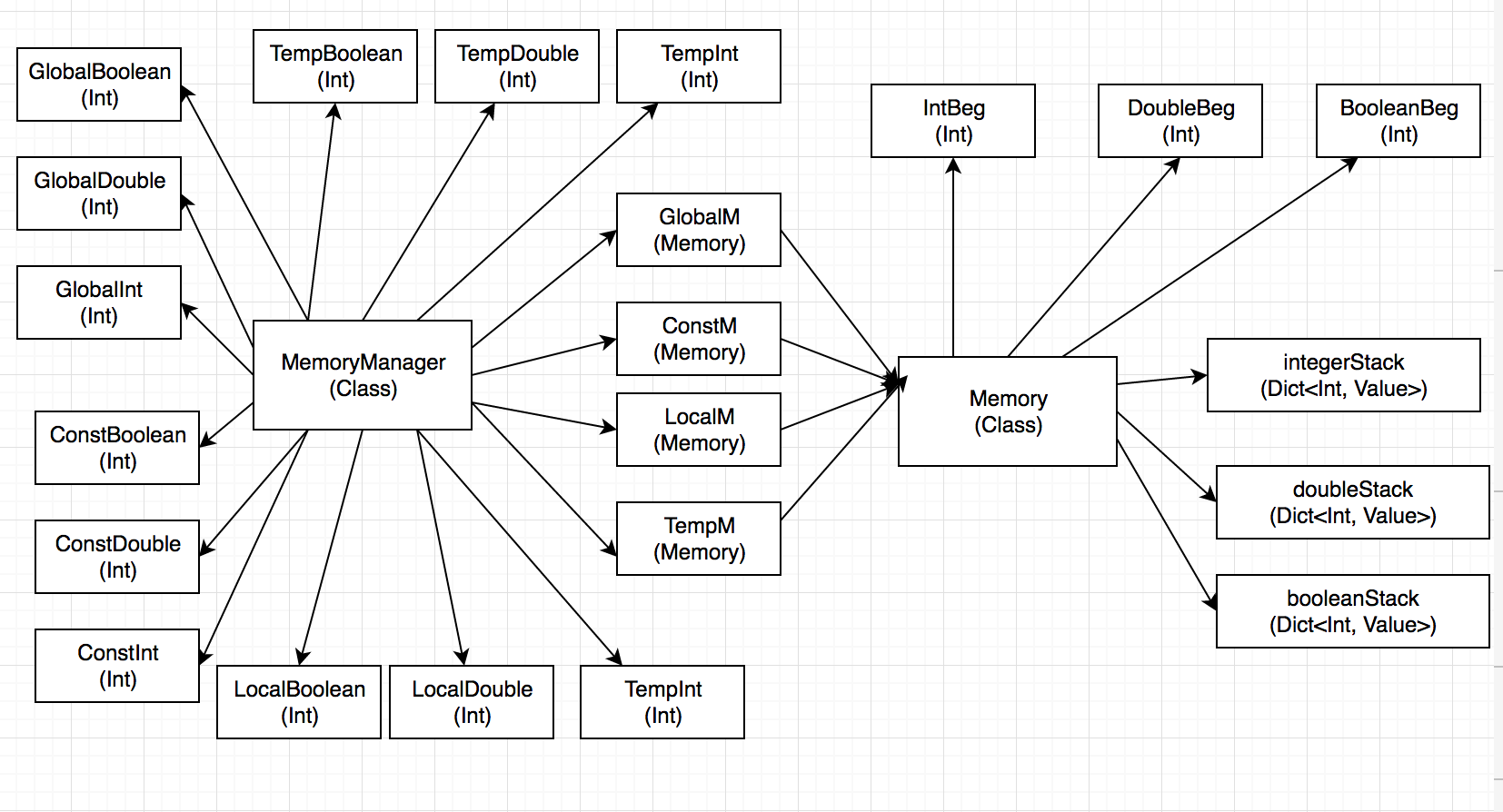
Sus elementos son los siguientes:



* La clase FunctionTable: Clase creada que cuenta con un solo atributo llamado Table. La razón de la generación de una clases especial es para poder generar métodos que permitan facilitar la manipulación de funciones. Métodos para agregar y regresar cierta función o para saber si cierta función existe dentro de la tabla.
  + Table: Es un directorio que utiliza como String la llave de acceso y guarda un valor de tipo Function, una clase especial. La llave es un String para facilitar el acceso a la función usando el id.
* La clase Function: es una clase creada que representa la instancia de una función dentro de la tabla. Cuenta con contadores para sabes la cantidad de memoria que será utilizada a la hora de la ejecución de la misma. Los contadores son : IntLocalMemoryRquired, DoubleLocalMemoryRquired, BooleanLocalMemoryRquired e igualmente para la memoria temporal.
  + Return type: guarda el tipo de retorno de la función usando un número entero que se define en una lista de tipos.
  + Params: Stack de Strings que representa los parámetros de la función. Ya que la variable es guardada como tal en una tabla éste stack solo guarda el id de los parámetros.
    - Id: String del parámetro alojado.
  + First Quadruple: guarda en un entero el valor donde se encuentra el primer cuádruplo en los cuádruplos una vez generados. Se utiliza para comenzar la función una vez que fue llamada.
  + Var table: variable que hace referencia a la tabla de variable de la función.
* La clase Var Table: Clase generada con un solo atributo y que se generó para facilitar las operaciones posibles en cuanto a la inserción y obtención de variables.
  + Table: atributo de la clase que es un directorio con String como llave y Clase variable como valor. El string representa el id de la variable y el valor la instancia de la variable como tal.
* La clase Variable: Clase que representa la instancia de una variable.
  + Memory: el valor de la memoria virtual guardada que es donde se localiza el valor real de la variable.
  + Type: tipo de la variable.
  + Dimensions: Stack que contiene un tipo de dato especial llamado Dimension. Es usada para variables dimensionadas.
    - Dimension: clase generada que instancia una dimensión de una variable.
      * Size: tamaño de dicha dimensión.
      * M: valor usado para calcular el desplazamiento al momento de acceder a una variable dimensionada. Se comprende como la multiplicación de las dimensiones faltantes.

Cuádruplo:

* QuadrupleManager: clase especial generada para llevar un control de los cuádruplos generados. Nuevamente la clase fue generada teniendo a consideración funciones para agregar, rellenar y acceder a cuádruplos, así como para llevar un conteo.
  + Counter: contador de la cantidad de cuádruplos generados.
  + QuadrupleStack: se utilizó una estructura de tipo Stack y que cuenta con todos los cuádruplos generados.
* Quadruple: clase especial que representa la instancia de un cuádruplo.
  + Operator: el operador dentro del cuádruplo, es decir la operación a ser realizada.
  + Operand1: primer operando del cuádruplo. Éste valor guarda la memoria del operando y no el valor en sí.
  + Operand2: segundo operando del cuádruplo. Éste valor guarda la memoria del operando y no el valor en sí.
  + Result: podría ser considerado como el tercer operando. Es la memoria donde se guardará el valor una vez realizada la operación.

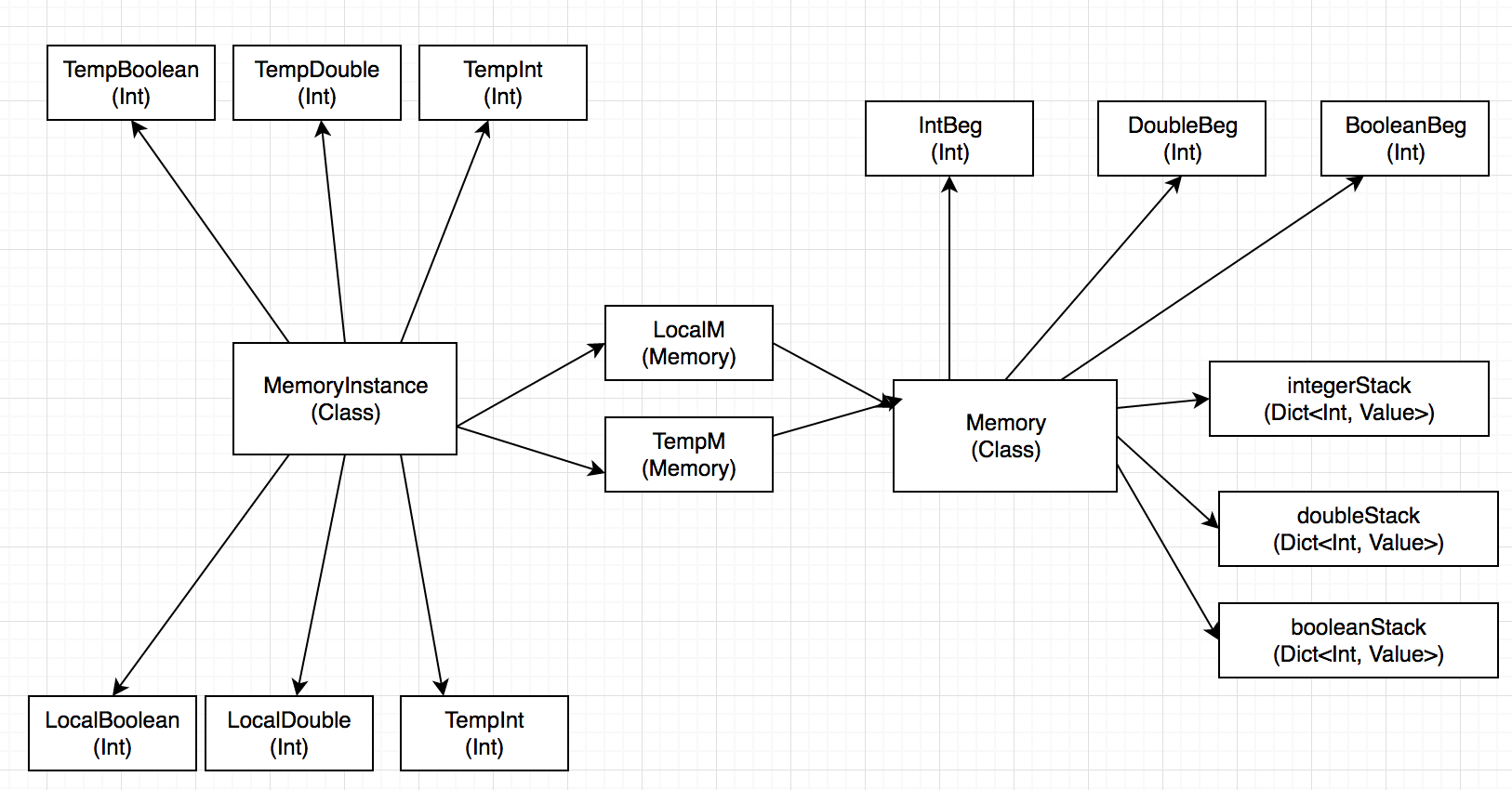
Memoria:

* MemoryManager: Clase generada para llevar un control de la memoria. Posee funciones como requestMemory e increaseMemory.
  + TempInt: Valor donde comienza el stack de memoria temporal entera.
  + TempDouble: Valor donde comienza el stack de memoria temporal doble.
  + TempBoolean: Valor donde comienza el stack de memoria temporal booleana.
  + LocalInt: Valor donde comienza el stack de memoria local entera.
  + LocalDouble: Valor donde comienza el stack de memoria local doble.
  + LocalBoolean: Valor donde comienza el stack de memoria local booleana.
  + GlobalInt: Valor donde comienza el stack de memoria global entera.
  + GlobalDouble: Valor donde comienza el stack de memoria global doble.
  + GlobalBoolean: Valor donde comienza el stack de memoria global booleana.
  + ConstantInt: Valor donde comienza el stack de memoria constante entera.
  + ConstantDouble: Valor donde comienza el stack de memoria constante doble.
  + ConstantBoolean: Valor donde comienza el stack de memoria constante booleana.
  + GlobalM: instancia de clase memoria que representa la memoria global.
  + ConstM: instancia de clase memoria que representa la memoria de constantes.
  + LocalM: instancia de clase memoria que representa la memoria local.
  + TempM: instancia de clase memoria que representa la memoria temporal.
* Memory: clase que representa una instancia de memoria. Cada instancia tiene diferentes valores para controlar stacks de memoria entera, doble y local.
  + IntBeg: entero que representa el comienzo del stack de memoria entera dentro.
  + DoubleBeg: entero que representa el comienzo del stack de memoria doble dentro.
  + BooleanBeg: entero que representa el comienzo del stack de memoria booleana dentro.
  + integerStack: stack de memoria en sí representa un stack de memoria entera. En vez de usar un stack se decidió usar una estructura de Stack se optó por usar un diccionario con las llaves como dirección de memoria.
  + doubleStack: stack de memoria en sí representa un stack de memoria doble. En vez de usar un stack se decidió usar una estructura de Stack se optó por usar un diccionario con las llaves como dirección de memoria.
  + booleanStack: stack de memoria en sí representa un stack de memoria booleana. En vez de usar un stack se decidió usar una estructura de Stack se optó por usar un diccionario con las llaves como dirección de memoria.

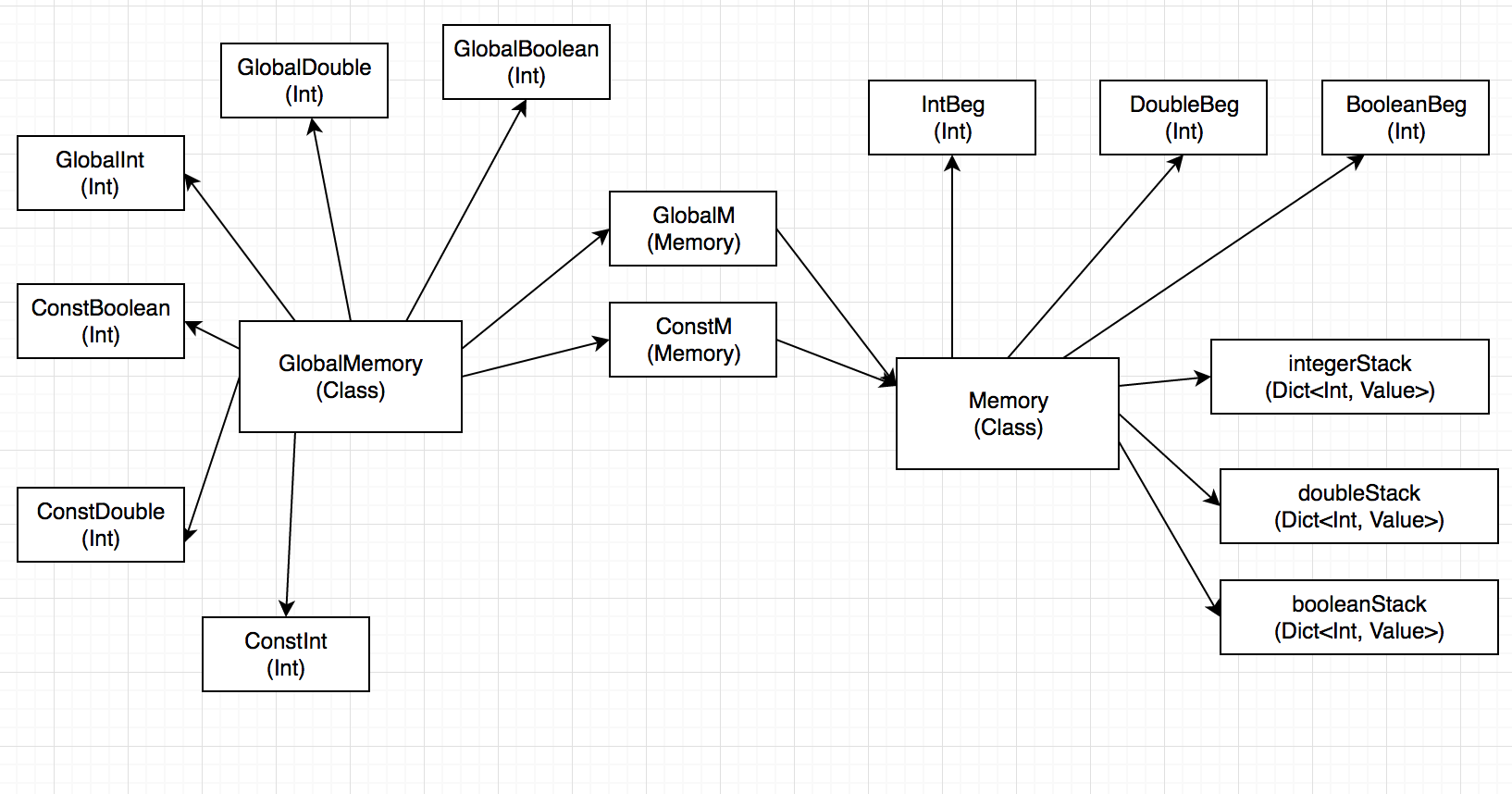
DESCRIPCION DE LA MÁQUINA VIRTUAL

Memoria en la máquina virtual:

La memoria a la hora de ejecución es representada por una estructura muy similar a la usada en la generación de cuádruplos pero con modificaciones para permitir la instancia de memoria local y temporal sin tener que poseer memoria global o constante en las instancias.

A la hora de acceder a las variables se llevaba un conteo de la memoria ya asignada y al llamar una función se le indicaba el offset de la memoria, es decir que memoria estaba ocupada, para que la instancia de memoria pudiera calcular el valor real de la memoria.

* MemoryInstance: instancia de memoria. Posee a su vez clases de memoria local y temporal.
  + TempInt: Valor donde comienza el stack de memoria temporal entera.
  + TempDouble: Valor donde comienza el stack de memoria temporal doble.
  + TempBoolean: Valor donde comienza el stack de memoria temporal booleana.
  + LocalInt: Valor donde comienza el stack de memoria local entera.
  + LocalDouble: Valor donde comienza el stack de memoria local doble.
  + LocalBoolean: Valor donde comienza el stack de memoria local booleana.
  + LocalM: instancia de clase memoria que representa la memoria local.
  + TempM: instancia de clase memoria que representa la memoria temporal.
* Memory: clase que representa una instancia de memoria. Cada instancia tiene diferentes valores para controlar stacks de memoria entera, doble y local.
  + IntBeg: entero que representa el comienzo del stack de memoria entera dentro.
  + DoubleBeg: entero que representa el comienzo del stack de memoria doble dentro.
  + BooleanBeg: entero que representa el comienzo del stack de memoria booleana dentro.
  + integerStack: stack de memoria en sí representa un stack de memoria entera. En vez de usar un stack se decidió usar una estructura de Stack se optó por usar un diccionario con las llaves como dirección de memoria.
  + doubleStack: stack de memoria en sí representa un stack de memoria doble. En vez de usar un stack se decidió usar una estructura de Stack se optó por usar un diccionario con las llaves como dirección de memoria.
  + booleanStack: stack de memoria en sí representa un stack de memoria booleana. En vez de usar un stack se decidió usar una estructura de Stack se optó por usar un diccionario con las llaves como dirección de memoria.

La representación de la memoria global utiliza la misma estructura que la memoria local y temporal con la diferencia que siempre se encuentra cargada en memoria y no posee ni local ni temporal.

* GlobalMemory: única instancia de la memoria global. Siempre se encuentra cargada en el stack de memoria.
  + GlobalInt: Valor donde comienza el stack de memoria global entera.
  + GlobalDouble: Valor donde comienza el stack de memoria global doble.
  + GlobalBoolean: Valor donde comienza el stack de memoria global booleana.
  + ConstantInt: Valor donde comienza el stack de memoria constante entera.
  + ConstantDouble: Valor donde comienza el stack de memoria constante doble.
  + ConstantBoolean: Valor donde comienza el stack de memoria constante booleana.
  + GlobalM: instancia de clase memoria que representa la memoria global.
  + ConstM: instancia de clase memoria que representa la memoria de constantes.
* Memory: clase que representa una instancia de memoria. Cada instancia tiene diferentes valores para controlar stacks de memoria entera, doble y local.
  + IntBeg: entero que representa el comienzo del stack de memoria entera dentro.
  + DoubleBeg: entero que representa el comienzo del stack de memoria doble dentro.
  + BooleanBeg: entero que representa el comienzo del stack de memoria booleana dentro.
  + integerStack: stack de memoria en sí representa un stack de memoria entera. En vez de usar un stack se decidió usar una estructura de Stack se optó por usar un diccionario con las llaves como dirección de memoria.
  + doubleStack: stack de memoria en sí representa un stack de memoria doble. En vez de usar un stack se decidió usar una estructura de Stack se optó por usar un diccionario con las llaves como dirección de memoria.
  + booleanStack: stack de memoria en sí representa un stack de memoria booleana. En vez de usar un stack se decidió usar una estructura de Stack se optó por usar un diccionario con las llaves como dirección de memoria.

PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE

Pruebas de operaciones relacionales:

program relational;

main{

boolean a = True;

boolean b = False;

int i = 1;

if (a & i | b){

print (i);

}

}

end;

10, , ,1

5,5000, ,11000

5,5001, ,11001

5,3000, ,9000

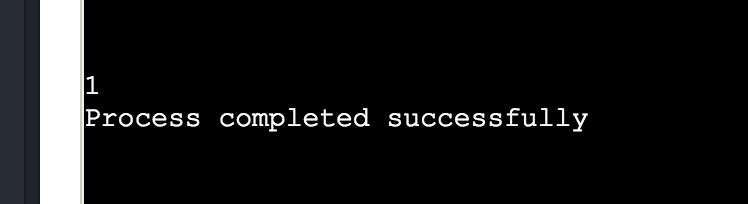
8,11000,9000,8000

9,8000,11001,8001

11,8001, ,8

32,9000, ,

17, , ,



Prueba de factorial recursivo:

program factorial\_r;

function int factorial(int n){

int f;

if (n == 1){

f = 1;

} else {

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

}

return f;

}

main{

print (factorial(5));

}

end;

10, , ,14

4,9000,3000,8000

11,8000, ,5

5,3001, ,9001

10, , ,13

14,factorial, ,

1,9000,3002,6000

16,6000, ,9000

33, , ,

15,factorial, ,1

34,0, ,6001

2,9000,6001,6002

5,6002, ,9001

13,9001, ,0

14,factorial, ,

16,3003, ,9000

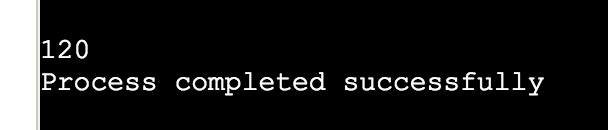
33, , ,

15,factorial, ,1

34,0, ,6000

32,6000, ,

17, , ,



Prueba de factorial cíclico:

program factorial\_c;

function int factorial(int n){

int i = n;

int r = 1;

while ( i > 1 ){

r = r \* i;

i = i - 1;

}

return r;

}

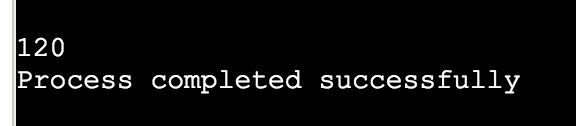
main{

int i = 5;

print (factorial(i));

}

end;



10, , ,11

5,9000, ,9001

5,3000, ,9002

7,9001,3001,8000

11,8000, ,10

2,9002,9001,6000

5,6000, ,9002

1,9001,3002,6001

5,6001, ,9001

10, , ,3

13,9002, ,0

5,3003, ,9000

14,factorial, ,

16,9000, ,9000

33, , ,

15,factorial, ,1

34,0, ,6000

32,6000, ,

17, , ,

Prueba de Fibonacci recursivo

program fibonacci\_r;

function int fibonacci(int n){

int f;

if (n == 1 | n == 2){

f = 1;

} else {

f = fibonacci(n - 1) + fibonacci(n - 2);

}

return f;

}

main{

print (fibonacci(7));

}

end;

10, , ,22

4,9000,3000,8000

4,9000,3001,8001

9,8000,8001,8002

11,8002, ,7

5,3002, ,9001

10, , ,21

14,fibonacci, ,

1,9000,3003,6000

16,6000, ,9000

33, , ,

15,fibonacci, ,1

34,0, ,6001

14,fibonacci, ,

1,9000,3004,6002

16,6002, ,9000

33, , ,

15,fibonacci, ,1

34,0, ,6003

0,6001,6003,6004

5,6004, ,9001

13,9001, ,0

14,fibonacci, ,

16,3005, ,9000

33, , ,

15,fibonacci, ,1

34,0, ,6000

32,6000, ,

17, , ,



Prueba de Fibonacci cíclico.

program fibonacci\_c;

function int fibonacci(int n){

int i = 3;

int sum = 0;

int fLast = 1;

int sLast = 1;

if (n == 1 | n == 2){

# values is 1 for fi with values 1 and 2

sum = 1;

} else {

while (i < n | i == n){

sum = fLast + sLast;

sLast = fLast;

fLast = sum;

i = i + 1;

}

}

return sum;

}

main{

print (fibonacci(7));

}

end;

10, , ,23

5,3000, ,9001

5,3001, ,9002

5,3002, ,9003

5,3003, ,9004

4,9000,3004,8000

4,9000,3005,8001

9,8000,8001,8002

11,8002, ,11

5,3006, ,9002

10, , ,22

6,9001,9000,8003

4,9001,9000,8004

9,8003,8004,8005

11,8005, ,22

0,9003,9004,6000

5,6000, ,9002

5,9003, ,9004

5,9002, ,9003

0,9001,3007,6001

5,6001, ,9001

10, , ,11

13,9002, ,0

14,fibonacci, ,

16,3008, ,9000

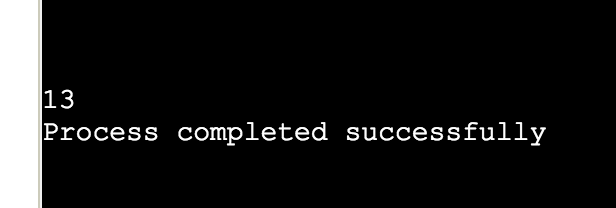
33, , ,

15,fibonacci, ,1

34,0, ,6000

32,6000, ,

17, , ,



program bubbleSort;

main{

int n = 10;

int m[10] = 1;

int i = 0;

#generates array values

while (i < 10){

m[i] = 10 - i;

i = i + 1;

}

#prints array before sort

i = 0;

while (i < 10){

print(m[i]);

i = i + 1;

}

boolean swapped = True;

int c = 1;

int temp = 0;

#sort

while (swapped){

swapped = False;

while (c < n){

if (m[c - 1] > m[c]){

#swap

temp = m[c - 1];

m[c - 1] = m[c];

m[c] = temp;

swapped = True;

}

c = c + 1;

}

c = 1;

}

#prints array

i = 0;

while (i < 10){

print(m[i]);

i = i + 1;

}

}

end;

10, , ,1

5,3000, ,9000

5,3001, ,9001

5,3001, ,9002

5,3001, ,9003

5,3001, ,9004

5,3001, ,9005

5,3001, ,9006

5,3001, ,9007

5,3001, ,9008

5,3001, ,9009

5,3001, ,9010

5,3002, ,9011

6,9011,3003,8000

11,8000, ,22

18,9011,10,

0,9011,\*9001,6000

1,3004,9011,6001

5,6001, ,&6000

0,9011,3005,6002

5,6002, ,9011

10, , ,13

5,3006, ,9011

6,9011,3007,8001

11,8001, ,31

18,9011,10,

0,9011,\*9001,6003

32,&6003, ,

0,9011,3008,6004

5,6004, ,9011

10, , ,23

5,5000, ,11000

5,3009, ,9012

5,3010, ,9013

11,11000, ,64

5,5001, ,11000

6,9012,9000,8002

11,8002, ,62

1,9012,3011,6005

18,6005,10,

0,6005,\*9001,6006

18,9012,10,

0,9012,\*9001,6007

7,&6006,&6007,8003

11,8003, ,59

1,9012,3012,6008

18,6008,10,

0,6008,\*9001,6009

5,&6009, ,9013

1,9012,3013,6010

18,6010,10,

0,6010,\*9001,6011

18,9012,10,

0,9012,\*9001,6012

5,&6012, ,&6011

18,9012,10,

0,9012,\*9001,6013

5,9013, ,&6013

5,5002, ,11000

0,9012,3014,6014

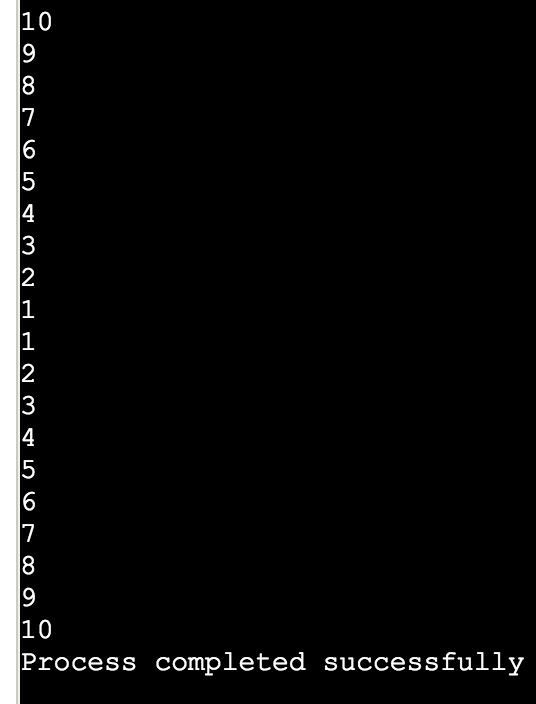
5,6014, ,9012

10, , ,36

5,3015, ,9012

10, , ,34

5,3016, ,9011

6,9011,3017,8004

11,8004, ,73

18,9011,10,

0,9011,\*9001,6015

32,&6015, ,

0,9011,3018,6016

5,6016, ,9011

10, , ,65

17, , ,

program find;

main{

int m[20];

int i = 0;

int index = -1;

int valueToFind = 11;

while (i < 20){

#inverted array

m[i] = 20 - i;

i = i + 1;

}

i = 0;

while (i < 20){

if (m[i] == valueToFind){

index = i;

}

i = i + 1;

}

print (index);

}

end;

10, , ,1

5,3000, ,9020

5,3001, ,9021

5,3002, ,9022

6,9020,3003,8000

11,8000, ,13

18,9020,20,

0,9020,\*9000,6000

1,3004,9020,6001

5,6001, ,&6000

0,9020,3005,6002

5,6002, ,9020

10, , ,4

5,3006, ,9020

6,9020,3007,8001

11,8001, ,24

18,9020,20,

0,9020,\*9000,6003

4,&6003,9022,8002

11,8002, ,21

5,9020, ,9021

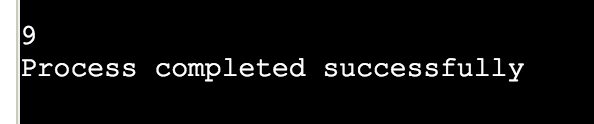
0,9020,3008,6004

5,6004, ,9020

10, , ,14

32,9021, ,

17, , ,



Prueba de output gráfico usando diferentes funciones creadas.

program draw;

function int sphere(int d){

circle(d);

return 0;

}

main{

int radius = 5;

int first = 80;

int second = 60;

int third = 40;

int fourth = 20;

int trunkH = 30;

int trunkW = 20;

int c;

penUp();

moveOnY(-50);

#trunk

penDown();

rectangle(trunkW, trunkH);

penUp();

moveOnY(trunkH);

moveOnX( 0 - (first - trunkW) / 2);

penDown();

c = sphere(radius);

triangle(first);

penUp();

moveOnX(first);

penDown();

c = sphere(radius);

penUp();

moveOnX( 0 - first/2);

moveOnY(first/2);

moveOnX( 0 - second / 2);

penDown();

c = sphere(radius);

triangle(second);

penUp();

moveOnX(second);

penDown();

c = sphere(radius);

penUp();

moveOnX( 0 - second/2);

moveOnY(second/2);

moveOnX( 0 - third / 2);

penDown();

c = sphere(radius);

triangle(third);

penUp();

moveOnX(third);

penDown();

c = sphere(radius);

penUp();

moveOnX( 0 - third/2);

moveOnY(fourth/2);

moveOnX( 0 - fourth / 2);

penDown();

c = sphere(radius);

triangle(fourth);

penUp();

moveOnX(fourth);

penDown();

c = sphere(radius);

penUp();

}

end;

10, , ,3

31,9000, ,

13,3000, ,0

5,3001, ,9000

5,3002, ,9001

5,3003, ,9002

5,3004, ,9003

5,3005, ,9004

5,3006, ,9005

5,3007, ,9006

20, , ,

28,3008, ,

19, , ,

30,9006, ,

26,3009, ,

30,9005, ,

26,3009, ,

30,9006, ,

26,3009, ,

30,9005, ,

26,3009, ,

20, , ,

28,9005, ,

1,9001,9006,6000

3,6000,3011,6001

1,3010,6001,6002

27,6002, ,

19, , ,

14,sphere, ,

16,9000, ,9000

33, , ,

15,sphere, ,1

34,0, ,6003

5,6003, ,9007

30,9001, ,

26,3012, ,

30,9001, ,

26,3012, ,

30,9001, ,

26,3012, ,

20, , ,

27,9001, ,

19, , ,

14,sphere, ,

16,9000, ,9000

33, , ,

15,sphere, ,1

34,0, ,6004

5,6004, ,9007

20, , ,

3,9001,3014,6005

1,3013,6005,6006

27,6006, ,

3,9001,3015,6007

28,6007, ,

3,9002,3017,6008

1,3016,6008,6009

27,6009, ,

19, , ,

14,sphere, ,

16,9000, ,9000

33, , ,

15,sphere, ,1

34,0, ,6010

5,6010, ,9007

30,9002, ,

26,3018, ,

30,9002, ,

26,3018, ,

30,9002, ,

26,3018, ,

20, , ,

27,9002, ,

19, , ,

14,sphere, ,

16,9000, ,9000

33, , ,

15,sphere, ,1

34,0, ,6011

5,6011, ,9007

20, , ,

3,9002,3020,6012

1,3019,6012,6013

27,6013, ,

3,9002,3021,6014

28,6014, ,

3,9003,3023,6015

1,3022,6015,6016

27,6016, ,

19, , ,

14,sphere, ,

16,9000, ,9000

33, , ,

15,sphere, ,1

34,0, ,6017

5,6017, ,9007

30,9003, ,

26,3024, ,

30,9003, ,

26,3024, ,

30,9003, ,

26,3024, ,

20, , ,

27,9003, ,

19, , ,

14,sphere, ,

16,9000, ,9000

33, , ,

15,sphere, ,1

34,0, ,6018

5,6018, ,9007

20, , ,

3,9003,3026,6019

1,3025,6019,6020

27,6020, ,

3,9004,3027,6021

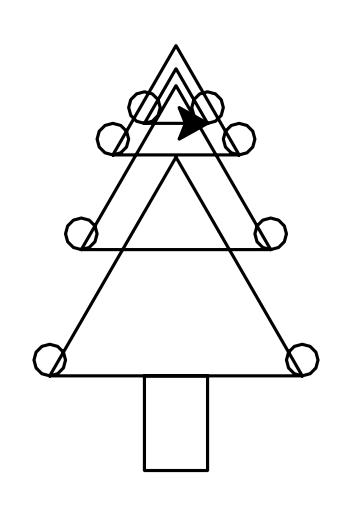
28,6021, ,

3,9004,3029,6022

1,3028,6022,6023

27,6023, ,

19, , ,

14,sphere, ,

16,9000, ,9000

33, , ,

15,sphere, ,1

34,0, ,6024

5,6024, ,9007

30,9004, ,

26,3030, ,

30,9004, ,

26,3030, ,

30,9004, ,

26,3030, ,

20, , ,

27,9004, ,

19, , ,

14,sphere, ,

16,9000, ,9000

33, , ,

15,sphere, ,1

34,0, ,6025

5,6025, ,9007

20, , ,

17, , ,

LISTADOS PERFECTAMENTE DOCUMENTADOS DEL PROYECTO

Algunas partes importantes del código son las siguientes:

Diferentes funciones usadas para el acceso a tablas de variables y funciones.

#checks if function exists

def functionExists(key):

return functionTable.exists(key)

# checks if var exists on function

def varExistsOnFunction(var, function):

return functionTable.getFunction(function).hasVariable(var)

#get last variable on stack declared from a function table

def getLastVariableDeclaredFromFunction(function):

return functionTable.getFunction(function).getVariable(getLastVariable())

#get last variable declared on stack from last function declared on stack

def getLastVariableDeclaredFromLastFunction():

return functionTable.getFunction(getLastFunction()).getVariable(getLastVariable())

#add variabe to var table on function

def addVariableToFunction(variable, function):

functionTable.getFunction(function).addVariable(variable)

#add variabe to var table on function

def addVariableToLastFunction(variable):

functionTable.getFunction(getLastFunction()).addVariable(variable)

#get a variable from a var table of a function

def getVariableFromFunction(variable, function):

return functionTable.getFunction(function).getVariable(variable)

El cálculo de las variables dimensionadas.

#calculates m's and return the total size of array

def getTotalMemoryDimension(self):

if len (self.dimension) == 0:

return 1

if len (self.dimension) == 1:

return self.dimension[0].size

totalMemory = 1

for d in self.dimension:

totalMemory \*= d.size

#at this point we have the total memory

for d in self.dimension:

d.m = totalMemory/d.size#calculates all m's

self.dimension[len(self.dimension) - 1].m = 0 #last dimension m's 0

return self.dimension[0].m \* self.dimension[0].size #Total size

La generación de cuádruplos básicos de operaciones:

#used to generates quadruple. A common function used by the different operators

def generateOperatorNextQuadruple(operator):

#global temporalCounter

if isSameOrder(operator):#last operator in stack is a mult

if not getPenultimateType() is None:

#remove last operator

operator = operatorsStack.pop()

#last operand

lastOpe = operandsStack.pop()

#penultimate operand

penultimateOpe = operandsStack.pop()

#last type

lastType = typesStack.pop()

#penultimate type

penultimateType = typesStack.pop()

resultType = semantic\_cube[lastType][operator][penultimateType]

if resultType != -1 : #can do operation

#add result type to stack

typesStack.append(resultType)

if not onGlobalScope():

lastFunction = functionTable.getFunction(getLastFunction())

if getLastType() == TYPES['int']:

#temp increase memory by one

temp = memoryManager.tempM.requestIntMemory(1)

result = quadrupleManager.addQuadruple(operator, penultimateOpe, lastOpe, temp)

#add result to stack

operandsStack.append(result)

lastFunction.increaseIntTempMemoryRequired(1)#result always increments one by one

elif getLastType() == TYPES['double']:

#temp increase memory by one

temp = memoryManager.tempM.requestDoubleMemory(1)

result = quadrupleManager.addQuadruple(operator, penultimateOpe, lastOpe, temp)

#add result to stack

operandsStack.append(result)

lastFunction.increaseDoubleTempMemoryRequired(1)#result always increments one by one

elif getLastType() == TYPES['boolean']:

#temp increase memory by one

temp = memoryManager.tempM.requestBooleanMemory(1)

result = quadrupleManager.addQuadruple(operator, penultimateOpe, lastOpe, temp)

#add result to stack

operandsStack.append(result)

lastFunction.increaseBooleanTempMemoryRequired(1)#result always increments one by one

return 1

else: # on global scope

g = functionTable.getFunction('global')

if getLastType() == TYPES['int']:

#temp increase memory by one

temp = memoryManager.tempM.requestIntMemory(1)

result = quadrupleManager.addQuadruple(operator, penultimateOpe, temp)

#add result to stack

operandsStack.append(result)

g.increaseIntTempMemoryRequired(1)#result always increments one by one

elif getLastType() == TYPES['double']:

#temp increase memory by one

temp = memoryManager.tempM.requestDoubleMemory(1)

result = quadrupleManager.addQuadruple(operator, penultimateOpe, lastOpe, temp)

#add result to stack

operandsStack.append(result)

g.increaseDoubleTempMemoryRequired(1)#result always increments one by one

elif getLastType() == TYPES['boolean']:

#temp increase memory by one

temp = memoryManager.tempM.requestBooleanMemory(1)

result = quadrupleManager.addQuadruple(operator, penultimateOpe, lastOpe, temp)

#add result to stack

operandsStack.append(result)

g.increaseBooleanTempMemoryRequired(1)#result always increments one by one

return 1

else :

return 0

return -1