**ITESM Campus Monterrey**

16

**Din**

08

**Fall**

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

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

Proyecto Final de Compiladores

Iker Arbulu Lozano A01190690

Iker Arbulu Lozano A01190690

4 de Mayo 2016 Maestra: Elda Quiroga

Índice

Propósito, Objetivos y Alcance del Proyecto 3

Análisis de Requerimientos y Casos de Uso generales 3

Descripción de los principales test cases 4

Descripción del PROCESO general seguido 5

Nombre del lenguaje 6

Descripción genérica de las principales características del lenguaje 6

Errores que pueden ocurrir 7

Equipo de Cómputo, Lenguaje y Utilerías especiales 9

Descripción del Análisis del Léxico 9

Descripción del Análisis de Sintaxis 10

Descripción de Generación de Código Intermedio y Análisis Semántico 12

Descripción detallada del proceso de Administración de Memoria 23

Equipo de cómputo, lenguaje y utilerías 27

Descripción detallada del proceso de Administración de Memoria 27

Fibonaccis 29

Factoriales 33

Listas 36

Composición musical secuencial 44

Composición musical cíclica 45

Memoria 48

Máquina Virtual 49

Reconocimiento de apuntador en máquina virtual 49

Función search en ctes 50

Return 50

Funciones 51

For 53

If Else 54

Asignación Vars Dinámicas 55

Descripción del Proyecto

## Propósito, Objetivos y Alcance del Proyecto

El propósito es desarrollar un compilador para un lenguaje gráfico que permita a los usuarios programar música mediante la selección la utilización de estructuras de control y elementos musicales, todo con el fin de aprender a programar de una manera más interactiva y dinámica mediante la exploración de composiciones musicales.

El objetivo del lenguaje es enseñar programación a niños de secundaria mediante la utilización de bloques gráficos para que puedan entender estructuras de control, lógica de programación y que al mismo tiempo exploten su lado creativo que sería la producción de obras musicales o imitación de las mismas mediante estructuras de control.

El alcance está definido en que utilizará estructuras básicas de control (if’s, for’s, while’s) junto con operaciones básicas y el uso de módulos o funciones para poder programar en bloques gráficos sencillos para finalmente bajar el código y ejecutar para sacar un archivo .wav que representa los plays que ingresó el usuario en el orden en que los programó y se ejecutaron de acuerdo a la visión lógica del mismo. Es un lenguaje con operaciones básicas que permiten analizar un diseño lógico sencillo mediante la salida de notas para poder componer melodías de un solo instrumento.

## Análisis de Requerimientos y Casos de Uso generales

El requerimiento principal es que sea un lenguaje que pueda ser escrito mediante una interfaz gráfica y que a través de la compilación se puedan respetar los estatutos básicos de un lenguaje y al mismo tiempo produzca música como salida. Por lo que el requerimiento es que a través de una interfaz gráfica pasemos a una sintaxis particular en un archivo de texto que el usuario puede escribir directamente si quiere esa libertad, para finalmente compilar todo esto utilizando python como maquina virtual que lee y ejecuta los cuádruplos generados al parsear el lenguaje.

Los casos de uso son los definidos en el diagrama:



Aquí podemos ver que la compilación puede ser exitosa mediante el flujo de blockly -> archivo.txt -> terminal para finalmente ver lo que se despliega en consola y el archivo .wav que se genera si es que se incluyen plays. El flujo alterno en una compilación exitosa es brincarse blockly e ir directo a escribir en nuestro lenguaje al archivo.txt -> terminal para que genere los outputs en consola y el .wav según sea el caso. Todos los errores de compilación existirán hasta el momento en el que se utilice la consola, blockly solo está para que la programación sea más dinámica y sencilla. Los errores pueden ir desde léxico (palabras que no acepta), sintaxis (reglas específicas de escribir), semántica (acciones que corresponden a ciertos tipos de datos), hasta ejecución (divisiones entre 0, índices fuera de rango etc.).

## Descripción de los principales test cases

Los principales test cases son los siguientes:

* Fibonacci recursivo: Para demostrar que se pueden hacer funciones en las que el resultado depende de la llamada a dos veces la misma función
* Fibonacci iterativo: Prueba de mandar a llamar funciones y ciclos
* Factorial recursivo: Para demostrar recursividad sencilla y llamadas a funciones
* Factorial iterativo: Para demostrar llamada a funciones y ciclos
* Sort de una lista: Para demostrar la manipulación de listas, la utilización de variables globales a través de las funciones y los ciclos anidados. Al igual que la impresión de una lista mediante la utilización de ciclos.
* Find dentro de una lista: Para demostrar la búsqueda dentro de listas y la comparación de sus valores.
* Prueba de llamada a una función con mas de un parámetro: Para comprobar la funcionalidad de que se puede mandar una función como parámetro de otra función.
* Prueba de composición musical secuencial: Para probar como funcionan los plays si se ponen uno después del otro y como produce el .wav al final ya que se hizo secuencial
* Prueba de composición musical cíclica: Para probar como funcionan los plays si se utilizan con estructuras de control y produce un .wav que tiene los plays de acuerdo a los ciclos e if’s incluidos.
* Prueba de llamada a una función con una función como parámetro: Probar mandar funciones como parámetros de otra función la que primero se evalúa la función de parámetro y el resultado se manda como parámetro.

## Descripción del PROCESO general seguido

El Proceso que se siguió fue el de tener juntas semanales de 4 horas de trabajo siguiendo el siguiente flujo de trabajo:

1. Analizar lo que se tiene que hacer para la entrega semanal
2. Revisar de las notas y de patito que se puede rescatar
3. Empezar a diseñar las estructuras de datos a utilizar
4. Sesión de pair programming
5. Revisión de lo que se hizo en código
6. Diseñar las pruebas que se necesitan pasar
7. Realizar pruebas

Las bitácoras son los commits que hemos realizado ya que se describe que es lo que se trabajó semana a semana, a veces por compromisos se trabajó en áreas separadas. Esto sucedió más que nada al final del proyecto ya que teníamos que apurarle a la entrada gráfica y cerrar lo que faltaba de compilador.

Reflexión Iker:

En general fue muy enriquecedor programar semana a semana lo que íbamos aprendiendo en clase y sobre todo traducirlo a nuestro caso en particular, ya que en el salón a veces se veían ejemplos más fáciles o que seguían una sintaxis distinta. Aprendí a programar competitivamente en Python, acerca de las reglas de parseo, como va subiendo la regla y podemos meter acciones semánticas mediante puntos neurálgicos, como se conecta la máquina virtual con el compilador, la ventaja de que nos dan los cuádruplos y mediante las direcciones virtuales podemos sacar y meter valores a la memoria simulando que lo estuviéramos accediendo en una arquitectura computacional clásica. Al igual aprendí que el trabajo planeado y coordinado te hace batallar menos al final, siempre teniendo como objetivo entrega a entrega en lugar de querer aventarte el proyecto de un jalón ya que muchas veces te das cuenta de los bugs muy tarde.

Firma:

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

Reflexión Roberto:

En este proyecto aprendí a analizar mucho más a fondo lo que implican distintas declaraciones dentro del código que desarrollo en alguna aplicación cotidiana. Fue muy bueno haber visto desde el punto de vista del “behind the scenes” la compilación de un programa y como se administra la memoria para las distintas variables dimensionadas y no dimensionadas. Este proyecto me ayudó a apreciar de una mejor manera los lenguajes de programación que utilizo cotidianamente pues ahora se lo que el desarrollo de cada uno de ellos implicó y que las facilidades que brindan fueron gracias a técnicas eficientes en el desarrollo del compilador. Ahora pensaré un poco más a la hora de diseñar mis programas pues se lo que me puedo ahorrar al declarar en una forma más amigable para el compilador.

Firma:

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

Descripción del Lenguaje

## Nombre del lenguaje

Din

## Descripción genérica de las principales características del lenguaje

Es un lenguaje en el que se arrastran bloques en el browser para formar el lenguaje que se quiere construir y en general sigue la siguiente estructura:

* Variables: globales y locales por función. También se manejan listas de tamaño estático, la asignación por casilla o a la variable no dimensionada en general. Al igual se puede sacar la longitud de una variable dimensionada.
* Main del programa (función CANCION) con su tempo declarado de donde se tomará la ejecución del programa y las primeras instrucciones
* Funciones: Tipo void, int, float, char y bool. Recursión en las funciones teniendo mas de un return. Llamado de funciones mandando funciones como parámetro.
* Lógica: If/else
* Ciclos: for, en donde se asigna una variable y se reasigna hasta que cumpla con una condición. While en donde se sigue ejecutando hasta que se cumpla cierta condición.
* Print
* Instrucción de tocar nota
* Operaciones lógicas (AND,OR, >,>=,<,<=,NOT,==,!=)
* Operaciones aritméticas (+,-,\*,/)
* Paréntesis para prioridad de operadores
* Declaración de texto.

## Errores que pueden ocurrir

Compilación:

* Variable con ese ID ya existe en ese scope: Para cuando se quiere declarar una variable que ya existe en ese contexto.
* Dimensión no valida: Para cuando se ingresa una dimensión negativa en una variable dimensionada.
* Overflow de variables enteras globales: Para cuando se declaran más de 2000 variables enteras en el contexto global.
* Overflow de variables enteras locales: Para cuando se declaran más de 2000 variables enteras en el contexto de una función o el main (Canción)
* Overflow de variables flotantes globales: Para cuando se declaran más de 2000 variables flotantes en el contexto global.
* Overflow de variables flotantes locales: Para cuando se declaran más de 2000 variables flotantes en el contexto de una función o el main (Canción)
* Overflow de variables char globales: Para cuando se declaran más de 2000 variables char en el contexto global.
* Overflow de variables char locales: Para cuando se declaran más de 2000 variables char en el contexto de una función o el main (Canción)
* Overflow de variables booleanas globales: Para cuando se declaran más de 2000 variables booleanas en el contexto global.
* Overflow de variables booleanas locales: Para cuando se declaran más de 2000 variables booleanas en el contexto de una función o el main (Canción)
* Falta regresar parámetro de salida en función: Cuando no es una función void y no hubo un return en la función.
* Función con ese ID ya existe en el programa: Cuando se quiere declarar otra función con el mismo nombre de otra función anteriormente declarada.
* Parámetro con ese ID ya existe en ese scope: Para cuando ya existe un parámetro con ese nombre anteriormente en la declaración de la función.
* Type mismatch: Cuando el cubo semántico no permite realizar la operación con el(los) tipo(s) mandados.
* Asignación mal terminada: Cuando quedaron operaciones pendientes en una asignación.
* No existe tal variable a asignar: Cuando tratan de asignar una variable que no fue previamente declarada en ese contexto o en el contexto global.
* Tiene que tener un booleano como resultado de expresión: Para cuando se hace un estatuto lógico que requiero un GOTOF y la expresión interna no regresa un booleano.
* Overflow de temporales enteras: Para cuando se realizan más de 4000 operaciones de tipo enteras en un contexto causando overflow.
* Overflow de temporales flotantes: Para cuando se realizan más de 4000 operaciones de tipo flotantes en un contexto causando overflow.
* Overflow de temporales char: Para cuando se realizan más de 4000 operaciones de tipo char en un contexto causando overflow.
* Overflow de temporales bool: Para cuando se realizan más de 4000 operaciones de tipo booleanas en un contexto causando overflow.
* Missing opening parenthesis: Para cuando se cierra un paréntesis que no tiene su paréntesis que abre.
* No existe tal variable normal: Para cuando se trata de acceder al valor de una variable y no ha sido declarada anteriormente en el contexto.
* Overflow de constantes enteras: Para cuando se utilizan más de 4000 constantes enteras en todo el programa.
* Overflow de constantes flotantes: Para cuando se utilizan más de 400 constantes flotantes en todo el programa.
* Overflow de constantes booleanas: Para cuando se utilizan más de 400 constantes booleanas en todo el programa.
* Overflow de constantes char: Para cuando se utilizan más de 400 constantes char en todo el programa.
* El índice que tratas de accesar no es de tipo entero: Para cuando se trata de acceder a una casilla de una variable dimensionada y la expresión para acceder al índice no regresa un entero.
* No es una variable dimensionada: Para cuando se trata de acceder a la casilla de una variable que no es dimensionada.
* No existe tal variable dimensionada: Para cuando se trata de acceder a la casilla de una variable que no existe.
* Error en declaración de parámetros: Para cuando se manda un valor que no es del mismo tipo del parámetro al que busca acceder.
* Función con ese id no existe: Para cuando se llama a una función que no existe.
* Error en cantidad de parámetros: Cuando faltaron parámetros por enviar a la función del lado de la llamada.
* Llamada a función con operaciones pendientes: Cuando en una llamada a una función se queda con operaciones pendientes uno de los parámetros que se están enviando.
* Error en el tipo de retorno dado: Para cuando la expresión del return de una función no corresponde con el tipo con el cual se declaro.
* Syntax error at: Cuando hay una error en la sintaxis.
* Syntax error at EOF: Cuando se acaba el archivo y no se cumplió con la sintaxis.
* Caracter ilegal: Cuando hay un error de léxico

Ejecución:

* Índice fuera de rango: Para cuando se trata de acceder un índice de la lista que no está dentro de su tamaño.

Descripción del Compilador

## Equipo de Cómputo, Lenguaje y Utilerías especiales

Cualquier computadora que cuente con una línea de comandos y que pueda correr Python. Nosotros recomendamos el uso de un package manager como homebrew para la instalación correcta de Python, al igual la computadora tiene que tener un browser que pueda correr javascript (recomendamos Google Chrome). La otra utilería que tiene que tener instalada es la librería de numpy para Python, esto se puede instalar por medio de pip para que se integre con Python. El resto de las librerías que se necesitan ya vienen incluidas en el repositorio de github en el que se encuentra nuestro lenguaje.

## Descripción del Análisis del Léxico

Patrones de Construcción por token:

* PLAY : ‘PLAY’
* VOID: ‘VOID’
* WHILE: ‘WHILE’
* CANCION: ‘CANCION’
* VAR: ‘VAR’
* FUNC: ‘FUNC’
* LIST: ‘LIST’
* IF: ‘IF’
* ELSE: ‘ELSE’
* FOR: ‘FOR’
* NOT: ‘NOT’
* AND: ‘AND’
* OR: ‘OR’
* LENGTH: ‘LENGTH’
* NOTA: ‘([a-g][1-7])|([a,c,d,f,g][#][1-7])’
* PRINT: ‘PRINT’
* CALL: ‘CALL’
* RETURN: ‘RETURN’
* INT: ‘INT’
* CHAR: ‘CHAR’
* FLOAT: ‘FLOAT’
* BOOL: ‘BOOL’
* NOTEQ: ‘!=’
* LTHANEQ: ‘<=’
* MTHANEQ: ‘>=’
* EQ: ‘==’
* CTEF: ‘[0-9]+([eE]([+]|[-])?|[.][0-9]+[eE]([+]|[-])?|[.])[0-9]+’
* CTEE: ‘[0-9]+’
* CTECHAR: ‘\"[^\n"]+\"’
* CTEBOOL: ‘True| False’
* ID: ‘[A-Za-z]([\_]?([a-zA-Z]|[0-9]))\*’
* Literales: '(',')',',',':',';', '{','}','\*','/','',',',';','>','<','=','+','-','[',']','.'

## Descripción del Análisis de Sintaxis

Programa := a c cancion

a := empty

a := vars a

c := empty

c := función c

vars := VAR ID : tipo u;

función := FUNC z ID ( params ) f bloque

z := INT

z := CHAR

z := FLOAT

z := BOOL

z := VOID

f := empty

f := vars f

params := empty

params := tipo ID h

h := empty

h := , params

i := empty

i := estatuto i

bloque := { i }

cancion := CANCION ( CTEE ) f bloque

estatuto := asignacion

estatuto := if

estatuto := for

estatuto := while

estatuto := return

estatuto := play

estatuto := print

estatuto := callvoidfunc

asignacion := ID accesoVarDim = expresion ;

if := IF ( expression ) bloque l ;

l := empty

l := ELSE bloque

for := FOR ( asignacion expresion ; asignacion) bloque ;

expresion := m subexpresion

m := empty

m := NOT m

subexpresion := exp o

o := empty

o := AND subexpresion

o := OR subexpresion

exp := nexp p

p := empty

p := EQ nexp

p := NOTEQ nexp

p := > nexp

p := < nexp

p := MTHANEQ nexp

p := LTHANEQ nexp

nexp := termino q

q := + nexp

q := - nexp

termino := meteneg factor n

n := \* termino

n := / termino

factor := ( expresion )

factor := varcte

meteneg := empty

meteneg := -

varcte := ID r

varcte := CTEE

varcte := CTEF

varcte := CTEBOOL

varcte := callreturnfunc

varcte := CTECHAR

r := empty

r := accesoVarDim

r := length

accesoVarDim := [ nexp ]

length := . LENGTH ( )

while := WHILE ( expresion ) bloque ;

play := PLAY ( NOTA , CTEE ) ;

print := PRINT expresion ;

callreturnfunc := CALL ID ( s) ;

s := empty

s := expresion t

t := , s

callvoidfunc := CALL ID ( s ) ;

return := RETURN ( expresion ) ;

tipo := INT

tipo := CHAR

tipo := FLOAT

tipo := BOOL

u := empty

u := LIST ( CTEE )

empty := null

## Descripción de Generación de Código Intermedio y Análisis Semántico

Códigos de Operación:

# Multiplicación

MULT = 1

# División

DIV = 2

# And

AND = 3

# Or

OR = 4

# ==

EQEQ = 5

# !=

NOTEQ = 6

# >

GT = 7

# <

LT = 8

# >=

GTE = 9

# <=

LTE = 10

# Suma

PLUS = 11

# Resta

MINUS = 12

# Not (!)

NOT = 13

# Asignación

EQ = 14

# Play

PLAY = 15

# Imprime

PRINT = 16

# Brinco

GOTO = 17

# Brinco en falso

GOTOF = 18

# Brinco en verdadero

GOTOV = 19

# Cierra procedimiento

ENDPROC = 20

# Registro de activación

ERA = 21

# Entrada de parámetro

PARAMETRO = 22

# Arranca procedimiento

GOSUB = 23

# Verifica que este dentro de rango el arreglo

VERIFICA = 24

# Signo negativo

NEG = 25

# Error, usado para los errores en el cubo semántico

ERR = -1

Códigos de Tipos:

INT = 100

CHAR = 200

FLOAT = 300

BOOL = 400

Direcciones Virtuales

Variables globales enteras 1000 a 3999

Variables globales flotantes 3000 a 4999

Variables globales booleanas 5000 a 6999

Variables globales char 7000 a 8999

Variables locales enteras 9000 a 10999

Variables locales flotantes 11000 a 12999

Variables locales booleanas 13000 a 14999

Variables locales char 15000 a 16999

Variables locales temporales enteras 17000 a 20999

Variables locales temporales flotantes 21000 a 24999

Variables locales temporales booleanas 25000 a 28999

Variables locales temporales char 29000 a 32999

Constantes enteras 33000 a 36999

Constantes flotantes 37000 a 40999

Constantes booleanas 41000 a 44999

Constantes char 45000 a 48999

Constantes notas 49000 a infinito

Breve descripción de las acciones semánticas:

* Al terminar programa: Se vacía la tabla de variables globales y se imprime que se terminó con el archivo
* creadirprocgobal: Creamos el directorio de procedimientos global con las variables vacías, la dirección de inicio en None, el tamaño en 0's y con los parámetros vacíos
* punto neurálgico 22: Utilizado para generar el salto a cancion que representa el main del programa, lo hace al crear un GOTO y agrega a la pila de saltos lo que hay que rellenar al encontrarnos con Canción.
* Al terminar vars: Utilizada para declarar variables en cualquier función o globalmente utilizamos un diccionario auxiliar para saber en que scope estamos, revisamos si no existe una variable con es id en ese scope y se da de alta diferente si es global o local y por su tipo. Se genera al final el registro en el diccionario de variables en el que estamos.
* Al terminar función: Se valida si se hizo return en caso de que sea un función que no sea void, vaciamos la tabla de variables de la función, reiniciamos el conteo de variables locales y temporales locales, sacamos el scope actual y generamos la instrucción de ENDPROC para terminar la función.
* TIPOFUNCION (regla z): Le agrega al tipo a la función regresando el string convertido en código de operación.
* Meterfuncion: Se utiliza para meter una función al directorio de procedimiento, primero revisamos que no existe una función con ese nombre anteriormente declarada, iniciamos las variables locales y temporales, lo damos de alta en el directorio de procedimientos con su tipo, vars vacías, dirección virtual inicial, tamaño en ceros y params vacíos , finalmente agregamos el scope actual que va a comenzar.
* Punto neurálgico 23: Para sacar el cuádruplo donde inicia la función que se declaro, se rellena su registro de directorio de procedimientos en el campo de dirección virtual con el número del siguiente cuádruplo.
* Meterparams: utilizado para meter cada parámetro declarado en la función se toma en cuenta su tipo y que no se llame igual que una variable local, revisamos si no existe un parámetro con ese nombre en el directorio de variables de ese scope, lo agregamos como variable nueva con su tipo declarado y aumentamos la cantidad de variables locales del tipo del que se declaró.
* Al terminar canción: Vaciamos el diccionario de variables de canción y sacamos el scope de canción de la pila de scopes.
* Metercancion: Agregamos el scope de canción, reiniciamos las variables locales y temporales, creamos la entrada al directorio de procedimientos de canción donde tiene el tipo, las variables en vacío, none para dirección de inicio del procedimiento, en 0's la cantidad de variables y temporales, las variables en vacío y al final lleva el tempo que se manda a la función, se rellena el salto inicial ingresado en el primer cuádruplo al momento de empezar el programa ingresado en la pila de saltos y se rellena el GOTO con el siguiente cuádruplo que es donde comienza el main.
* Al terminar asignación: Revisamos que la operación pendiente en la pOper sea la asignación, sacamos el operando izquierdo de la pilaO y su tipo, luego sacamos lo que se quiere igualar de la pilaO y su tipo, hacemos la validación semántica con el cubo y creamos el cuádruplo de asignación.
* Punto neurálgico 8: Revisamos que la variable de la asignación exista y que no sea una variable dimensionada, si cumple entonces agregamos a la pilaO su dirección virtual y su tipo a la pila de tipos, finalmente agregamos el código de operación de la asignación a la pila de operaciones.
* Asiglista: Revisamos si se va a utilizar un acceso a una variable dimensionada o no y en base a eso lo regresamos al punto neurálgico 8 para que sepa si se trata de una variable dimensionada o no.
* Punto neurálgico 13: utilizado para meter las acciones correspondientes de los brincos por el estatuto lógico en caso de que sea falso todo dependiendo del resultado de la expresion del estatuto de control. Checamos si la expresion nos dejo un tipo bool al final, generamos el GOTOF todavía sin saber a donde vamos a brincar con el tope de la pilaO que tiene la respuesta de la expresión del if para saber si es verdadero o falso, finalmente agregamos a la pila de saltos para que sepa que es lo que se tiene que rellenar cuando se acabe el if o se encuentra con el else.
* Punto neurálgico 14: Utilizado para hacer un salto en caso de que haya else se utiliza porque si hay un else y entro al verdadero se tiene que saltar las instrucciones del else y al igual es el brinco que se hace para el GOTOF del if. Todo lo hacemos manipulando la pila de saltos, primero rellenado el salto pendiente del GOTOF y luego agregamos el GOTO que tiene pendiente su brinco y por lo tanto lo agregamos a la pila de saltos.
* Punto neurálgico 15: utilizado para cerrar el estatuto del if y generar las acciones correspondientes semánticas, aquí lo importante es que la pila de saltos ya se trabajo bien en los otros puntos del if ya que no importa si fue if else o if solo. Sacamos el pendiente del GOTO del else o del GOTOF, en caso de que no sea un if else, de la pila de saltos y rellenamos con el siguiente cuádruplo.
* Punto neurálgico 18: utilizado por el for para agregar a la pila de saltos el punto inicial antes de la expresion que se evalúa cada vez que se ejecuta de nuevo el ciclo. Este salto se agrega a la pila de saltos.
* Punto neurálgico 19: Utilizado por el for para revisar que la expresion sea de tipo booleana y agregamos los brincos correspondientes en falso y verdadero de acuerdo al resultado de la misma ya que siempre se debe dejar la segunda asignacion hasta el final del for. Generamos cuádruplos de GOTOV para que se vaya al bloque del for, todavía sin saber donde empieza agregando su posición a la pila de saltos, hacemos lo mismo para el GOTOF para que se brinque el bloque todavía sin saber a donde vamos agregando a la pila de saltos. Finalmente agregamos a la pila de saltos el siguiente cuádruplo ya que siempre se hace la asignación al final de bloque del for.
* Punto neurálgico 21: se utiliza para hacer el salto después de la segunda asignacion ya que después de ahí vuelve a evaluarse la expresion anteriormente en el for y se usa para saber que ya comienza el bloque entonces el salto del GOTOV se rellena. Se utiliza una pila auxiliar para los niveles anidados en la pila de saltos.
* Punto neurálgico 20: se utiliza para marcar el fin de la declaración del for y se utiliza para hacer el brinco a la segunda asignacion y rellenar el GOTOF de la expresión.
* Al terminar expresión: regla sintáctica que se utiliza para hacer el NOT a toda una expresion y marca la menor prioridad en las expresiones, se hace hasta el final generando el cuádruplo en caso de que se haya hecho el NOT y haciendo la validación en el cubo semántico.
* Agrega not: Se agrega el código de operación del NOT a la pOper en caso de que se haya incluido el NOT.
* Punto neurálgico 10: Utilizado para evaluar los AND's y OR's pendientes y agregar el resultado a la pilaO y a la pTipos el tipo. Sacando de la pilaO los dos operandos, generando una nueva temporal para el resultado y validando en el cubo semántico que se pueda hacer la operación y agregando el cuádruplo de operación con los operandos y el temporal resultante y agregando el resultado a la pilaO.
* Punto neurálgico 9\_1: Se utiliza para agregar el código de operación del AND a la pOper.
* Punto neurálgico 9\_2: Se utiliza para agregar el código de operación del OR a la pOper.
* Punto neurálgico 12: se utiliza para evaluar las comparaciones y poner el resultado en la pilaO y pTipos al igual hacer el cuádruplo, las operaciones son: ==, !=, >, >=, <=, <. Saca el operando izquierdo y el derecho de la pilaO, hace el temporal para el resultado, las validaciones semánticas del cubo y finalmente genera el cuádruplo guardando el resultado en la pilaO.
* Puntos neurálgicos 11: se utiliza para agregar la operación correspondiente de comparación (==,>,>=,<=,<,!=).
* Punto neurálgico 5: se utiliza para hacer la operación de suma o resta y agregar el resultado a la pilaO y a la pTipos, sacamos los operandos de la pilaO, hacemos el temporal resultante, hacemos validación semántica con el cubo y generamos el cuádruplo.
* Puntos neurálgicos 3: se utilizan para agregar la operación de suma o resta a la pOper.
* Punto neurálgico 4: utilizado para realizar la operación de multiplicación o división del termino y agregando el resultado a la pilaO y el tipo a la pTipos, sacamos los operandos de la pilaO hacemos la validación semántica con el cubo, generamos la temporal y generamos el cuádruplo de la operación.
* Puntos neurálgicos 2: se utiliza para agregar la operación a la pOper de multiplicación o división según sea el token leído.
* Al terminar factor: nivel mas alto en la jerarquía de expresiones en el que se pueden utilizar paréntesis y meter otra expresion para poner esa expresion en mayor prioridad o así bien meter constantes o variables. Aplicamos el negativo si es que existe como operación pendiente en la pOper, hacemos las validaciones pendientes con el operando de la pilaO en el cubo semántico, generamos la temporal correspondiente y finalmente generamos el cuádruplo de la operación.
* Meteneg: regla sintáctica para dar la opción de meter o no el signo negativo en el termino y agregarlo a la pOper.
* Punto neurálgico 6: utilizado para meter el fondo falso a la pila de operaciones para cuando se ingresa un paréntesis esto le da prioridad a las operaciones internas.
* Punto neurálgico 7: Utilizado para sacar el fondo falso de la pila de operaciones para cuando se termina la expresión y se cierra paréntesis y así ya pueden continuar las operaciones anteriores al paréntesis.
* Punto neurálgico var: Utilizado para agregar la variable a la pilaO y pTipos siempre y cuando exista en el diccionario de variables del scope actual y no sea una operación length. Validando que exista la variable en el diccionario de variables del scope actual o el scope global y agregamos su dirección virtual a la pilaO y su tipo a la pTipos.
* Punto neurálgico cte entera: utilizado para agregar constantes enteras y como las constantes están en un diccionario entonces revisa si ya existe esa constante y pone su dirección virtual o si no la agrega.
* Punto neurálgico cte flotante: utilizado para agregar constantes flotantes y como las constantes están en un diccionario entonces revisa si ya existe esa constante y pone su dirección virtual o si no la agrega.
* Punto neurálgico cte booleana: utilizado para agregar constantes booleanas y como las constantes están en un diccionario entonces revisa si ya existe esa constante y pone su dirección virtual o si no la agrega. Para esta en particular agregamos el string tru o fals debido a que sino Python lo parsea como 1 el True y 0 el False y por lo tanto no se pasa así tal cual.
* Punto neurálgico cte char: Utilizado para agregar constantes char y como las constantes están en un diccionario entonces revisa si ya existe esa constante y pone su dirección virtual o si no la agrega.
* accesoVarDim: utilizado para accesar el valor de una casilla de la variable dimensionada para esto se pueden utilizar expresiones numéricas entera para ingresar el índice de la casilla a la que se quiere acceder. Al final de sacar el resultado generamos el cuádruplo para sumar el resultado de la expresión numérica de adentro a la dirección base de la variable dimensionada y generamos un apuntado utilizando paréntesis en la pilaO
* punto neurálgico 27: Se utiliza para meter un fondo falso al momento de buscar el índice que se desea acceder mediante una expresion así dejamos en pausa todas las operaciones pendientes que teníamos en la expresion que tiene el acceso a la variable.
* Length: Se utiliza para saber el tamaño declarado de una variable dimensionada, regresa un entero metiéndolo a la pilaO y metiendo un entero a la pila de tipos. Validando que exista la variable y que sea dimensionada, sacando la dimensión de su diccionario de variables y poniendo en la pilaO.
* Punto neurálgico 16: Se utiliza en el while para definir el punto de brinco que va a tener el ciclo para siempre evaluar la condición cada vez que termina una vuelta.
* Punto neurálgico 17: se utiliza en while para cuando terminamos el ciclo que realice el GOTO al principio y vuelva a evaluar la condición para ver si vuelve a dar otra vuelta o no y después rellenamos el cuádruplo de GOTOF para saber a donde se va a ir el apuntador de instrucción si la condición es falsa.
* Al terminar el play: Estatuto que se utiliza para sacar un sonido o un silencio y que al final de programa se reúnan y se arme la melodía según fue encontrando estos estatutos. Valida que la nota exista o si no la agrega a las constantes y hace lo mismo con la constante entera.
* Al terminar el print: Generamos el cuádruplo del print con lo que hay en el tope de la piaO.
* Al terminar el callreturnfunc: en el scope global en el diccionario de procedimientos se declara una variable que representa el resultado de la función en donde se le asigna una dirección virtual, por medio de esta instrucción se recupera la dirección virtual y poder igualar una temporal y meterla a la pilaO para que siga con el resto de la expresión. revisamos que tipo es y en base a eso le asignamos el valor de la variable global ahora a una temporal para el scope que mando a llamar la función al igual la metemos a la pilaO y su tipo a la pila de tipos.
* Punto neurálgico 25: Se utiliza cada vez que se termina la expresion de un parámetro en la llamada a una función, sea void o con un tipo y se realiza una validación para ver si el parámetro es del mismo tipo y los esta ingresando en el orden correcto. La suma de los parámetros está en la pila de cuenta de parámetros sumando cada tipo, se valida contra el total declarado y se checa si esta declarando de más y luego checa si el resultado de la expresión es del mismo tipo, al final generando el cuádruplo de parámetro al número correspondiente en la nueva función.
* Punto neurálgico 24: Se utiliza en cualquier call a una función, genera un era para la nueva función a la que desea ir, agrega a la pila de funciones, a la pila de numero de función y finalmente deja todo listo para empezar a ingresar los parámetros.
* Punto neurálgico 26: se utiliza en cualquier llamada a función para revisar que se hayan dado el numero de parámetros correctos y genera el GOSUB que ya cambia el scope.
* Al terminar el return: se utiliza en las funciones que no son de tipo void esto marca que ya se asigna la variable global que corresponde a la función y se le asigna el resultado de la expresión. Se asigna mediante una operación de asignación en el cuádruplo, lo que se está generando de la expresión del return a la dirección virtual de su variable global y finalmente se hace un ENDPROC ya que ahí termina la función.

Tabla de consideraciones semánticas expresada en un diccionario de diccionarios:

#####################################

## multiplication operation cube ##

#####################################

cubo\_semantico[INT][INT][MULT]=INT

cubo\_semantico[INT][CHAR][MULT]=ERR

cubo\_semantico[INT][FLOAT][MULT]=FLOAT

cubo\_semantico[INT][BOOL][MULT]=ERR

cubo\_semantico[CHAR][CHAR][MULT]=ERR

cubo\_semantico[CHAR][FLOAT][MULT]=ERR

cubo\_semantico[CHAR][BOOL][MULT]=ERR

cubo\_semantico[FLOAT][FLOAT][MULT]=FLOAT

cubo\_semantico[FLOAT][BOOL][MULT]=ERR

cubo\_semantico[BOOL][BOOL][MULT]=ERR

#####################################

## división operation cube ##

#####################################

cubo\_semantico[INT][INT][DIV]=INT

cubo\_semantico[INT][CHAR][DIV]=ERR

cubo\_semantico[INT][FLOAT][DIV]=FLOAT

cubo\_semantico[INT][BOOL][DIV]=ERR

cubo\_semantico[CHAR][CHAR][DIV]=ERR

cubo\_semantico[CHAR][FLOAT][DIV]=ERR

cubo\_semantico[CHAR][BOOL][DIV]=ERR

cubo\_semantico[FLOAT][FLOAT][DIV]=FLOAT

cubo\_semantico[FLOAT][BOOL][DIV]=ERR

cubo\_semantico[BOOL][BOOL][DIV]=ERR

#####################################

## and operation cube ##

#####################################

cubo\_semantico[INT][INT][AND]=ERR

cubo\_semantico[INT][CHAR][AND]=ERR

cubo\_semantico[INT][FLOAT][AND]=ERR

cubo\_semantico[INT][BOOL][AND]=ERR

cubo\_semantico[CHAR][CHAR][AND]=ERR

cubo\_semantico[CHAR][FLOAT][AND]=ERR

cubo\_semantico[CHAR][BOOL][AND]=ERR

cubo\_semantico[FLOAT][FLOAT][AND]=ERR

cubo\_semantico[FLOAT][BOOL][AND]=ERR

cubo\_semantico[BOOL][BOOL][AND]=BOOL

#####################################

## or operation cube ##

#####################################

cubo\_semantico[INT][INT][OR]=ERR

cubo\_semantico[INT][CHAR][OR]=ERR

cubo\_semantico[INT][FLOAT][OR]=ERR

cubo\_semantico[INT][BOOL][OR]=ERR

cubo\_semantico[CHAR][CHAR][OR]=ERR

cubo\_semantico[CHAR][FLOAT][OR]=ERR

cubo\_semantico[CHAR][BOOL][OR]=ERR

cubo\_semantico[FLOAT][FLOAT][OR]=ERR

cubo\_semantico[FLOAT][BOOL][OR]=ERR

cubo\_semantico[BOOL][BOOL][OR]=BOOL

#####################################

## == operation cube ##

#####################################

cubo\_semantico[INT][INT][EQEQ]=BOOL

cubo\_semantico[INT][CHAR][EQEQ]=ERR

cubo\_semantico[INT][FLOAT][EQEQ]=BOOL

cubo\_semantico[INT][BOOL][EQEQ]=ERR

cubo\_semantico[CHAR][CHAR][EQEQ]=BOOL

cubo\_semantico[CHAR][FLOAT][EQEQ]=ERR

cubo\_semantico[CHAR][BOOL][EQEQ]=ERR

cubo\_semantico[FLOAT][FLOAT][EQEQ]=BOOL

cubo\_semantico[FLOAT][BOOL][EQEQ]=ERR

cubo\_semantico[BOOL][BOOL][EQEQ]=BOOL

#####################################

## notequals operation cube ##

#####################################

cubo\_semantico[INT][INT][NOTEQ]=BOOL

cubo\_semantico[INT][CHAR][NOTEQ]=ERR

cubo\_semantico[INT][FLOAT][NOTEQ]=BOOL

cubo\_semantico[INT][BOOL][NOTEQ]=ERR

cubo\_semantico[CHAR][CHAR][NOTEQ]=BOOL

cubo\_semantico[CHAR][FLOAT][NOTEQ]=ERR

cubo\_semantico[CHAR][BOOL][NOTEQ]=ERR

cubo\_semantico[FLOAT][FLOAT][NOTEQ]=BOOL

cubo\_semantico[FLOAT][BOOL][NOTEQ]=ERR

cubo\_semantico[BOOL][BOOL][NOTEQ]=BOOL

#####################################

## gt operation cube ##

#####################################

cubo\_semantico[INT][INT][GT]=BOOL

cubo\_semantico[INT][CHAR][GT]=ERR

cubo\_semantico[INT][FLOAT][GT]=BOOL

cubo\_semantico[INT][BOOL][GT]=ERR

cubo\_semantico[CHAR][CHAR][GT]=ERR

cubo\_semantico[CHAR][FLOAT][GT]=ERR

cubo\_semantico[CHAR][BOOL][GT]=ERR

cubo\_semantico[FLOAT][FLOAT][GT]=BOOL

cubo\_semantico[FLOAT][BOOL][GT]=ERR

cubo\_semantico[BOOL][BOOL][GT]=ERR

#####################################

## lt operation cube ##

#####################################

cubo\_semantico[INT][INT][LT]=BOOL

cubo\_semantico[INT][CHAR][LT]=ERR

cubo\_semantico[INT][FLOAT][LT]=BOOL

cubo\_semantico[INT][BOOL][LT]=ERR

cubo\_semantico[CHAR][CHAR][LT]=ERR

cubo\_semantico[CHAR][FLOAT][LT]=ERR

cubo\_semantico[CHAR][BOOL][LT]=ERR

cubo\_semantico[FLOAT][FLOAT][LT]=BOOL

cubo\_semantico[FLOAT][BOOL][LT]=ERR

cubo\_semantico[BOOL][BOOL][LT]=ERR

#####################################

## gte operation cube ##

#####################################

cubo\_semantico[INT][INT][GTE]=BOOL

cubo\_semantico[INT][CHAR][GTE]=ERR

cubo\_semantico[INT][FLOAT][GTE]=BOOL

cubo\_semantico[INT][BOOL][GTE]=ERR

cubo\_semantico[CHAR][CHAR][GTE]=ERR

cubo\_semantico[CHAR][FLOAT][GTE]=ERR

cubo\_semantico[CHAR][BOOL][GTE]=ERR

cubo\_semantico[FLOAT][FLOAT][GTE]=BOOL

cubo\_semantico[FLOAT][BOOL][GTE]=ERR

cubo\_semantico[BOOL][BOOL][GTE]=ERR

#####################################

## lte operation cube ##

#####################################

cubo\_semantico[INT][INT][LTE]=BOOL

cubo\_semantico[INT][CHAR][LTE]=ERR

cubo\_semantico[INT][FLOAT][LTE]=BOOL

cubo\_semantico[INT][BOOL][LTE]=ERR

cubo\_semantico[CHAR][CHAR][LTE]=ERR

cubo\_semantico[CHAR][FLOAT][LTE]=ERR

cubo\_semantico[CHAR][BOOL][LTE]=ERR

cubo\_semantico[FLOAT][FLOAT][LTE]=BOOL

cubo\_semantico[FLOAT][BOOL][LTE]=ERR

cubo\_semantico[BOOL][BOOL][LTE]=ERR

#####################################

## plus operation cube ##

#####################################

cubo\_semantico[INT][INT][PLUS]=INT

cubo\_semantico[INT][CHAR][PLUS]=ERR

cubo\_semantico[INT][FLOAT][PLUS]=FLOAT

cubo\_semantico[INT][BOOL][PLUS]=ERR

cubo\_semantico[CHAR][CHAR][PLUS]=ERR

cubo\_semantico[CHAR][FLOAT][PLUS]=ERR

cubo\_semantico[CHAR][BOOL][PLUS]=ERR

cubo\_semantico[FLOAT][FLOAT][PLUS]=FLOAT

cubo\_semantico[FLOAT][BOOL][PLUS]=ERR

cubo\_semantico[BOOL][BOOL][PLUS]=ERR

#####################################

## minus operation cube ##

#####################################

cubo\_semantico[INT][INT][MINUS]=INT

cubo\_semantico[INT][CHAR][MINUS]=ERR

cubo\_semantico[INT][FLOAT][MINUS]=FLOAT

cubo\_semantico[INT][BOOL][MINUS]=ERR

cubo\_semantico[CHAR][CHAR][MINUS]=ERR

cubo\_semantico[CHAR][FLOAT][MINUS]=ERR

cubo\_semantico[CHAR][BOOL][MINUS]=ERR

cubo\_semantico[FLOAT][FLOAT][MINUS]=FLOAT

cubo\_semantico[FLOAT][BOOL][MINUS]=ERR

cubo\_semantico[BOOL][BOOL][MINUS]=ERR

#####################################

## not operation cube ##

#####################################

cubo\_semantico[INT][NOT]=ERR

cubo\_semantico[CHAR][NOT]=ERR

cubo\_semantico[FLOAT][NOT]=ERR

cubo\_semantico[BOOL][NOT]=BOOL

#####################################

## equals operation cube ##

#####################################

cubo\_semantico[INT][INT][EQ]=INT

cubo\_semantico[INT][CHAR][EQ]=ERR

cubo\_semantico[INT][FLOAT][EQ]=ERR

cubo\_semantico[INT][BOOL][EQ]=ERR

cubo\_semantico[CHAR][CHAR][EQ]=CHAR

cubo\_semantico[CHAR][FLOAT][EQ]=ERR

cubo\_semantico[CHAR][BOOL][EQ]=ERR

cubo\_semantico[FLOAT][FLOAT][EQ]=FLOAT

cubo\_semantico[FLOAT][BOOL][EQ]=ERR

cubo\_semantico[BOOL][BOOL][EQ]=BOOL

### Caso especial para igualdad ya que solo

### se puede asignar un int a un float y no al

### revés

cubo\_semantico[FLOAT][INT][EQ] = FLOAT

#####################################

## neg operation cube ##

#####################################

cubo\_semantico[INT][NEG]=INT

cubo\_semantico[CHAR][NEG]=ERR

cubo\_semantico[FLOAT][NEG]=FLOAT

cubo\_semantico[BOOL][NEG]=ERR

## Descripción detallada del proceso de Administración de Memoria

# Pila de Operandos:

## Se utiliza para saber las direcciones virtuales

## de los operandos que están pendientes por transformar

## o utilizar

pilaO=[]



# Pila de Operadores:

## Se utiliza para saber los códigos de operación

## de las operaciones pendientes en el código actual

pOper=[]



# Pila de Tipos:

## Se utiliza para saber cuales son los códigos de los

## tipos que corresponden a la pila de operandos

pTipos=[]



# Diccionario de Cuádruplos:

## Este diccionario de listas se utiliza para ingresar los cuádruplos

## que el programa va ingresando (código intermedio) que tiene la

## siguiente organización:

## posicion0 = código de la operación a realizar

## posicion1 = dirección virtual del operando izquierdo

## posicion2 = dirección virtual del operando derecho

## posicion3 = dirección virtual del resultado

cuádruplos={}



# Diccionario de Procedimientos:

## Este diccionario de los procedimientos del programa

## es un diccionario de listas que tiene la siguiente

## organización:

## posicion0 = tipo del procedimiento

## posicion1 = diccionario de variables del procedimiento

## posicion2 = dirección cuádruplo inicio del procedimiento

## posicion3 = diccionario del tamaño del procedimiento, representado en cada tipo y por variables y constantes

## posicion4 = lista de parámetros que tiene el procedimiento

# Diccionario de Variables:

## Este diccionario es un diccionario de listas que tiene

## la siguiente organización:

## posicion0 = tipo de la variable

## posicion1 = dirección virtual de la variable

## posicion2 = lista de dimensiones de la variable (para variables de tipo lista)

### No esta declarada aquí porque se declara cada vez que se

### manda a crear un nuevo procedimiento y se crea adentro

### de la posición de variables en el diccionario de procedimientos



# Pila de Numero de Funciones:

## Esta pila se utiliza para llevar un registro de a que

## funciones se están llamando para que cuando se mande

## a llamar una función adentro de otra entonces sepa a

## cual registrarle parámetros, se utilizan números

## ya que puede mandarse a llamar la misma función adentro

## de la otra

pilaNumFuncs = []



# Pila auxParamCount:

## Esta pila se utiliza para llevar un registro de en que

## parámetro se encuentran las funciones en pilaNumFuncs

## esto se utiliza para cuando se mande a llamar una función

## adentro de otra

pilaAuxParamCount = []



# Pila de Funciones:

## Esta pila se utiliza para llevar un registro de a que

## funciones se están llamando para que cuando se mande

## a llamar una función adentro de otra entonces sepa a

## cual registrarle parámetros a diferencia de la pilaNumFuncs

## esta es para saber el nombre de la función a la que se le

## están enviando las cosas

pilaFuncs = []



# Pila de Saltos:

## Esta pila/queue se utiliza para llevar un registro de

## los saltos que tiene pendiente por rellenar el programa

## o por asignar mas adelante, se utilizo pila/queue para

## que se puedan sacar por ambos lados

pSaltos = deque([])



# Diccionario de constantes:

## Aquí se guardan las constantes que se van declarando

## la organización es que la llave es el valor de la constante

## y el valor es la dirección virtual para los cuádruplos

ctes = {}



Descripción de la Máquina Virtual

## Equipo de cómputo, lenguaje y utilerías

Son las mismas que las del compilador.

## Descripción detallada del proceso de Administración de Memoria

Especificación gráfica

## clase de memoria la que se utiliza para pedir memoria

## cuando llamas a una nueva funcion, la global o la de

## cancion, nos ayuda a definir el tamano en variables

## del elemento y las casillas y los valores de ejecucion

## que agarra cada casilla, inicialmente arrancan en 0





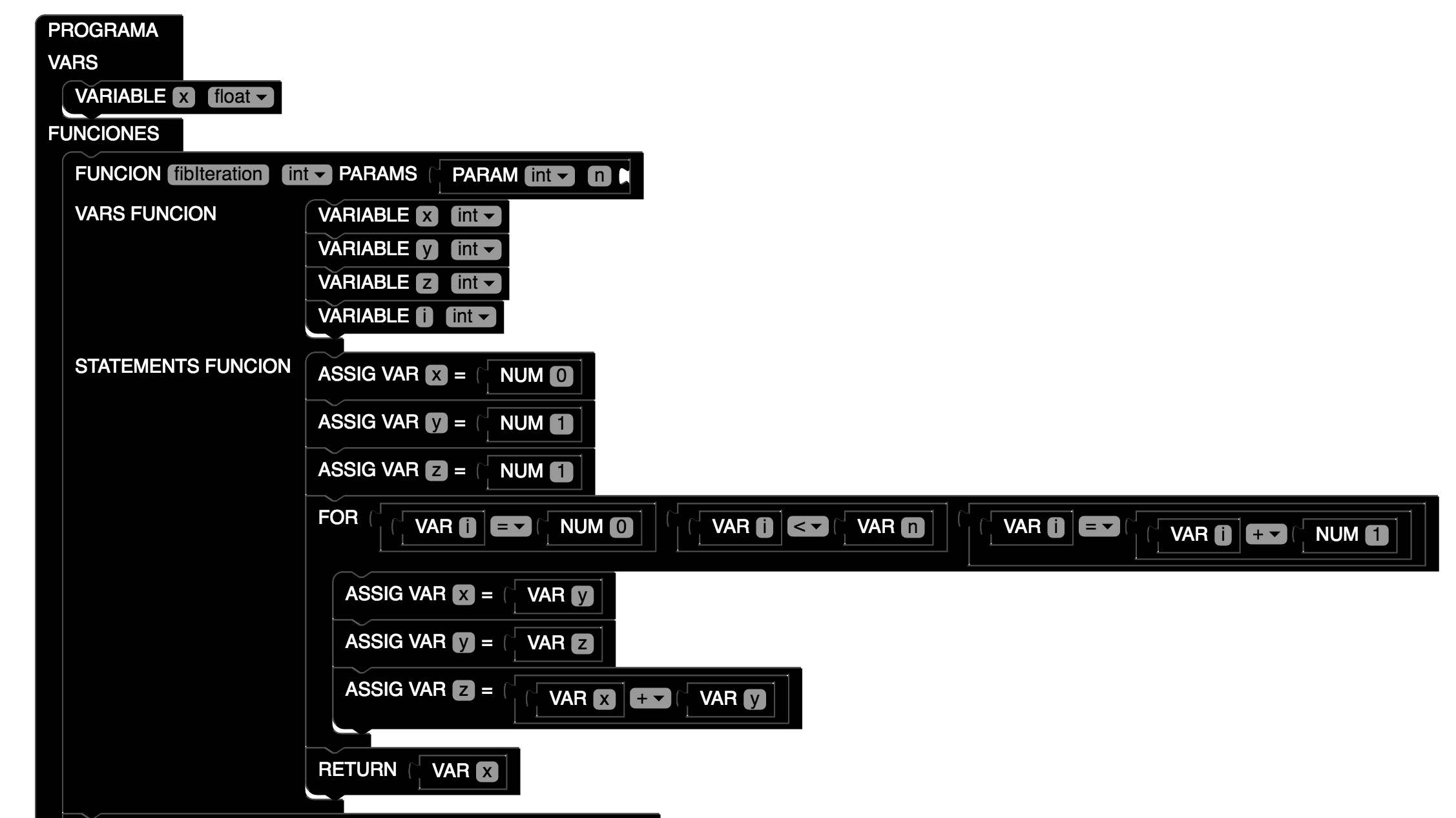
La asociación con la memoria se ve así:

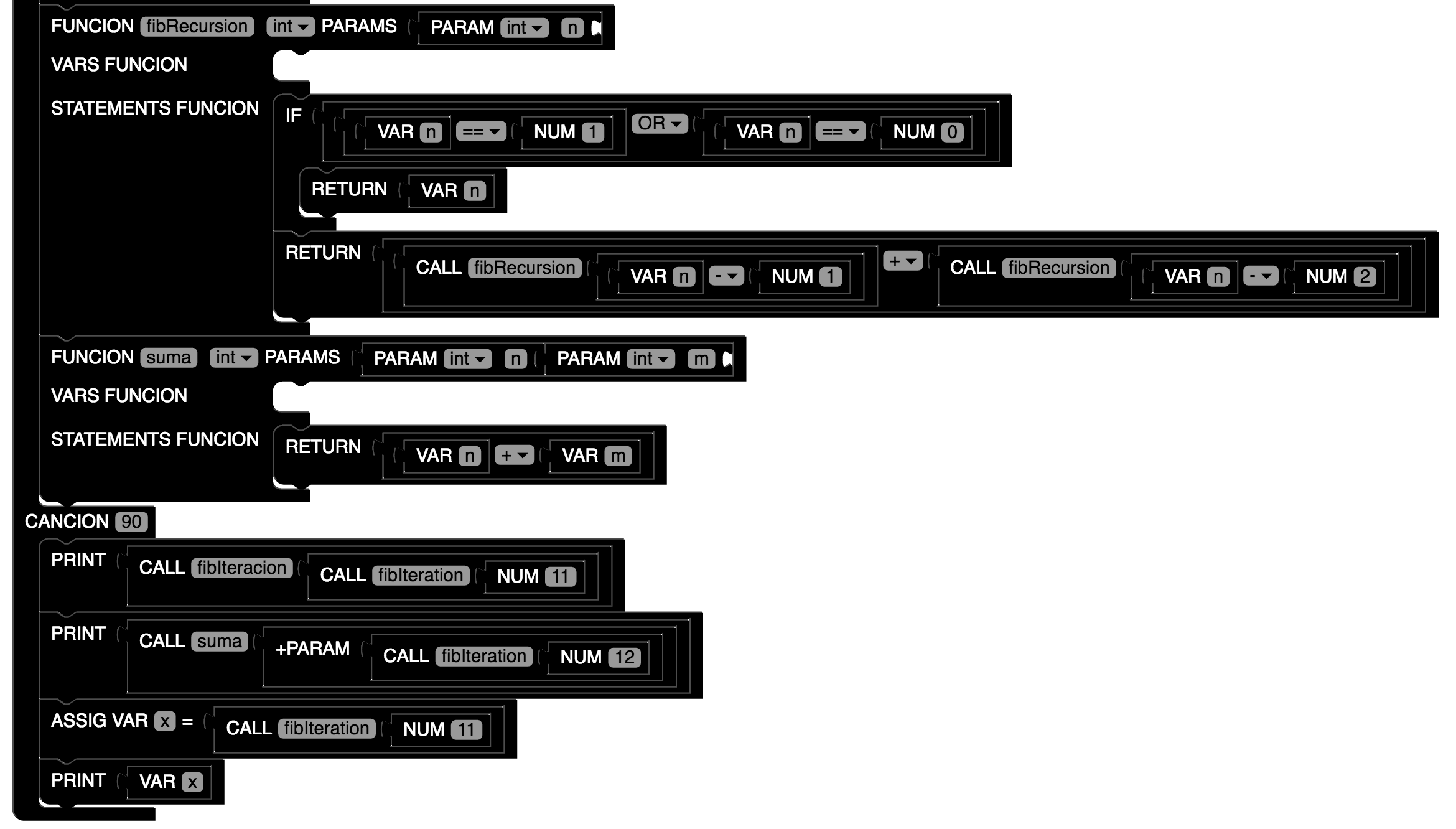
Si la flag de global se prende en la clase entonces las variables arrancan en distinto número. Según las variables que se declararon entonces se crean las llaves de los diccionarios de ints, chars, bools y floats ahí se mezclan las temporales con las variables y todas se inicializan en 0. Así se tradujo a la memoria y entonces así se respeta la dirección virtual que viene en los cuádruplos para buscar en la memoria de acuerdo al tipo.

Pruebas del Funcionamiento del Lenguaje

## Fibonaccis

Blockly





Archivo generado

VAR x : FLOAT;

FUNC INT fibIteration(INT n)

VAR x : INT;

VAR y : INT;

VAR z : INT;

VAR i : INT;

{

x = 0;

y = 1;

z = 1;

FOR (i = 0; i < n; i=i+1;) {

x = y;

y = z;

z = x + y;

};

RETURN (x);

}

FUNC INT fibRecursion(INT n) {

IF ((n == 1) OR (n == 0)) {

RETURN (n);

};

RETURN (CALL fibRecursion(n - 1); + CALL fibRecursion(n - 2););

}

FUNC INT suma(INT n,INT m) {

RETURN (n + m);

}

CANCION (90)

{

PRINT CALL fibIteration(CALL fibIteration(11););;

PRINT CALL suma(CALL fibIteration(12);,10);;

PRINT CALL fibRecursion(11);;

x = CALL fibIteration(11);;

PRINT x;

}

Impresion en terminal

{ 1: [17, None, None, 44],

2: [14, 33000, None, 9001],

3: [14, 33001, None, 9002],

4: [14, 33001, None, 9003],

5: [14, 33000, None, 9004],

6: [8, 9004, 9000, 25000],

7: [19, 25000, None, 12],

8: [18, 25000, None, 17],

9: [11, 9004, 33001, 17000],

10: [14, 17000, None, 9004],

11: [17, None, None, 6],

12: [14, 9002, None, 9001],

13: [14, 9003, None, 9002],

14: [11, 9001, 9002, 17001],

15: [14, 17001, None, 9003],

16: [17, None, None, 9],

17: [14, 9001, None, 1000],

18: [20, None, None, None],

19: [20, None, None, None],

20: [5, 9000, 33001, 25000],

21: [5, 9000, 33000, 25001],

22: [4, 25000, 25001, 25002],

23: [18, 25002, None, 26],

24: [14, 9000, None, 1001],

25: [20, None, None, None],

26: [21, 'fibRecursion', None, None],

27: [12, 9000, 33001, 17000],

28: [22, 17000, None, 9000],

29: [23, 'fibRecursion', None, None],

30: [14, 1001, None, 17001],

31: [21, 'fibRecursion', None, None],

32: [12, 9000, 33002, 17002],

33: [22, 17002, None, 9000],

34: [23, 'fibRecursion', None, None],

35: [14, 1001, None, 17003],

36: [11, 17001, 17003, 17004],

37: [14, 17004, None, 1001],

38: [20, None, None, None],

39: [20, None, None, None],

40: [11, 9000, 9001, 17000],

41: [14, 17000, None, 1002],

42: [20, None, None, None],

43: [20, None, None, None],

44: [21, 'suma', None, None],

45: [21, 'fibIteration', None, None],

46: [22, 33003, None, 9000],

47: [23, 'fibIteration', None, None],

48: [14, 1000, None, 17000],

49: [22, 17000, None, 9000],

50: [22, 33004, None, 9001],

51: [23, 'suma', None, None],

52: [14, 1002, None, 17001],

53: [16, 17001, None, None],

54: [21, 'fibRecursion', None, None],

55: [22, 33003, None, 9000],

56: [23, 'fibRecursion', None, None],

57: [14, 1001, None, 17002],

58: [16, 17002, None, None],

59: [21, 'fibRecursion', None, None],

60: [22, 33003, None, 9000],

61: [23, 'fibRecursion', None, None],

62: [14, 1001, None, 17003],

63: [14, 17003, None, 3000],

64: [16, 3000, None, None]}

[]

[]

[]

deque([])

comienza maquina virtual

{0: 33000, 1: 33001, 2: 33002, 10: 33004, 5: 33003}

[17, None, None, 44]

[21, 'suma', None, None]

[21, 'fibIteration', None, None]

[22, 33003, None, 9000]

[23, 'fibIteration', None, None]

[14, 33000, None, 9001]

[14, 33001, None, 9002]

[14, 33001, None, 9003]

[14, 33000, None, 9004]

[8, 9004, 9000, 25000]

[19, 25000, None, 12]

[14, 9002, None, 9001]

[14, 9003, None, 9002]

[11, 9001, 9002, 17001]

[14, 17001, None, 9003]

[17, None, None, 9]

[11, 9004, 33001, 17000]

[14, 17000, None, 9004]

[17, None, None, 6]

[8, 9004, 9000, 25000]

[19, 25000, None, 12]

[14, 9002, None, 9001]

[14, 9003, None, 9002]

[11, 9001, 9002, 17001]

[14, 17001, None, 9003]

[17, None, None, 9]

[11, 9004, 33001, 17000]

[14, 17000, None, 9004]

[17, None, None, 6]

[8, 9004, 9000, 25000]

[19, 25000, None, 12]

[14, 9002, None, 9001]

[14, 9003, None, 9002]

[11, 9001, 9002, 17001]

[14, 17001, None, 9003]

[17, None, None, 9]

[11, 9004, 33001, 17000]

[14, 17000, None, 9004]

[17, None, None, 6]

[8, 9004, 9000, 25000]

[19, 25000, None, 12]

[14, 9002, None, 9001]

[14, 9003, None, 9002]

[11, 9001, 9002, 17001]

[14, 17001, None, 9003]

[17, None, None, 9]

[11, 9004, 33001, 17000]

[14, 17000, None, 9004]

[17, None, None, 6]

[8, 9004, 9000, 25000]

[19, 25000, None, 12]

[14, 9002, None, 9001]

[14, 9003, None, 9002]

[11, 9001, 9002, 17001]

[14, 17001, None, 9003]

[17, None, None, 9]

[11, 9004, 33001, 17000]

[14, 17000, None, 9004]

[17, None, None, 6]

[8, 9004, 9000, 25000]

[19, 25000, None, 12]

[18, 25000, None, 17]

[14, 9001, None, 1000]

[20, None, None, None]

[14, 1000, None, 17000]

[22, 17000, None, 9000]

[22, 33004, None, 9001]

[23, 'suma', None, None]

[11, 9000, 9001, 17000]

[14, 17000, None, 1002]

[20, None, None, None]

[14, 1002, None, 17001]

[16, 17001, None, None]

15

[21, 'fibRecursion', None, None]

[22, 33003, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17002]

[16, 17002, None, None]

5

[21, 'fibRecursion', None, None]

[22, 33003, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[21, 'fibRecursion', None, None]

[12, 9000, 33001, 17000]

[22, 17000, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17001]

[21, 'fibRecursion', None, None]

[12, 9000, 33002, 17002]

[22, 17002, None, 9000]

[23, 'fibRecursion', None, None]

[5, 9000, 33001, 25000]

[5, 9000, 33000, 25001]

[4, 25000, 25001, 25002]

[18, 25002, None, 26]

[14, 9000, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[11, 17001, 17003, 17004]

[14, 17004, None, 1001]

[20, None, None, None]

[14, 1001, None, 17003]

[14, 17003, None, 3000]

[16, 3000, None, None]

5

## Factoriales

Blockly





Archivo de texto

FUNC INT factorialRecur(INT number)

VAR temp : INT;

{

IF(number <= 1){

RETURN (1);

};

temp = number \* CALL factorialRecur(number - 1);;

RETURN (temp);

}

FUNC INT factorialIter(INT number)

VAR temp : INT;

VAR i : INT;

{

FOR (i = 0; i <= number; i= i + 1;){

IF (i == 0){

temp = 1;

}

ELSE{

temp = temp \* i;

};

};

RETURN (temp);

}

CANCION(90)

VAR number : INT;

{

number = 7;

PRINT CALL factorialRecur(number);;

PRINT CALL factorialIter(number);;

}

Impresion en Terminal

{ 1: [17, None, None, 33],

2: [10, 9000, 33000, 25000],

3: [18, 25000, None, 6],

4: [14, 33000, None, 1000],

5: [20, None, None, None],

6: [21, 'factorialRecur', None, None],

7: [12, 9000, 33000, 17000],

8: [22, 17000, None, 9000],

9: [23, 'factorialRecur', None, None],

10: [14, 1000, None, 17001],

11: [1, 9000, 17001, 17002],

12: [14, 17002, None, 9001],

13: [14, 9001, None, 1000],

14: [20, None, None, None],

15: [20, None, None, None],

16: [14, 33001, None, 9002],

17: [10, 9002, 9000, 25000],

18: [19, 25000, None, 23],

19: [18, 25000, None, 30],

20: [11, 9002, 33000, 17000],

21: [14, 17000, None, 9002],

22: [17, None, None, 17],

23: [5, 9002, 33001, 25001],

24: [18, 25001, None, 27],

25: [14, 33000, None, 9001],

26: [17, None, None, 29],

27: [1, 9001, 9002, 17001],

28: [14, 17001, None, 9001],

29: [17, None, None, 20],

30: [14, 9001, None, 1001],

31: [20, None, None, None],

32: [20, None, None, None],

33: [14, 33002, None, 9000],

34: [21, 'factorialRecur', None, None],

35: [22, 9000, None, 9000],

36: [23, 'factorialRecur', None, None],

37: [14, 1000, None, 17000],

38: [16, 17000, None, None],

39: [21, 'factorialIter', None, None],

40: [22, 9000, None, 9000],

41: [23, 'factorialIter', None, None],

42: [14, 1001, None, 17001],

43: [16, 17001, None, None]}

[]

[]

[]

deque([])

comienza maquina virtual

{0: 33001, 1: 33000, 7: 33002}

[17, None, None, 33]

[14, 33002, None, 9000]

[21, 'factorialRecur', None, None]

[22, 9000, None, 9000]

[23, 'factorialRecur', None, None]

[10, 9000, 33000, 25000]

[18, 25000, None, 6]

[21, 'factorialRecur', None, None]

[12, 9000, 33000, 17000]

[22, 17000, None, 9000]

[23, 'factorialRecur', None, None]

[10, 9000, 33000, 25000]

[18, 25000, None, 6]

[21, 'factorialRecur', None, None]

[12, 9000, 33000, 17000]

[22, 17000, None, 9000]

[23, 'factorialRecur', None, None]

[10, 9000, 33000, 25000]

[18, 25000, None, 6]

[21, 'factorialRecur', None, None]

[12, 9000, 33000, 17000]

[22, 17000, None, 9000]

[23, 'factorialRecur', None, None]

[10, 9000, 33000, 25000]

[18, 25000, None, 6]

[21, 'factorialRecur', None, None]

[12, 9000, 33000, 17000]

[22, 17000, None, 9000]

[23, 'factorialRecur', None, None]

[10, 9000, 33000, 25000]

[18, 25000, None, 6]

[21, 'factorialRecur', None, None]

[12, 9000, 33000, 17000]

[22, 17000, None, 9000]

[23, 'factorialRecur', None, None]

[10, 9000, 33000, 25000]

[18, 25000, None, 6]

[21, 'factorialRecur', None, None]

[12, 9000, 33000, 17000]

[22, 17000, None, 9000]

[23, 'factorialRecur', None, None]

[10, 9000, 33000, 25000]

[18, 25000, None, 6]

[14, 33000, None, 1000]

[20, None, None, None]

[14, 1000, None, 17001]

[1, 9000, 17001, 17002]

[14, 17002, None, 9001]

[14, 9001, None, 1000]

[20, None, None, None]

[14, 1000, None, 17001]

[1, 9000, 17001, 17002]

[14, 17002, None, 9001]

[14, 9001, None, 1000]

[20, None, None, None]

[14, 1000, None, 17001]

[1, 9000, 17001, 17002]

[14, 17002, None, 9001]

[14, 9001, None, 1000]

[20, None, None, None]

[14, 1000, None, 17001]

[1, 9000, 17001, 17002]

[14, 17002, None, 9001]

[14, 9001, None, 1000]

[20, None, None, None]

[14, 1000, None, 17001]

[1, 9000, 17001, 17002]

[14, 17002, None, 9001]

[14, 9001, None, 1000]

[20, None, None, None]

[14, 1000, None, 17001]

[1, 9000, 17001, 17002]

[14, 17002, None, 9001]

[14, 9001, None, 1000]

[20, None, None, None]

[14, 1000, None, 17000]

[16, 17000, None, None]

5040

[21, 'factorialIter', None, None]

[22, 9000, None, 9000]

[23, 'factorialIter', None, None]

[14, 33001, None, 9002]

[10, 9002, 9000, 25000]

[19, 25000, None, 23]

[5, 9002, 33001, 25001]

[18, 25001, None, 27]

[14, 33000, None, 9001]

[17, None, None, 29]

[17, None, None, 20]

[11, 9002, 33000, 17000]

[14, 17000, None, 9002]

[17, None, None, 17]

[10, 9002, 9000, 25000]

[19, 25000, None, 23]

[5, 9002, 33001, 25001]

[18, 25001, None, 27]

[1, 9001, 9002, 17001]

[14, 17001, None, 9001]

[17, None, None, 20]

[11, 9002, 33000, 17000]

[14, 17000, None, 9002]

[17, None, None, 17]

[10, 9002, 9000, 25000]

[19, 25000, None, 23]

[5, 9002, 33001, 25001]

[18, 25001, None, 27]

[1, 9001, 9002, 17001]

[14, 17001, None, 9001]

[17, None, None, 20]

[11, 9002, 33000, 17000]

[14, 17000, None, 9002]

[17, None, None, 17]

[10, 9002, 9000, 25000]

[19, 25000, None, 23]

[5, 9002, 33001, 25001]

[18, 25001, None, 27]

[1, 9001, 9002, 17001]

[14, 17001, None, 9001]

[17, None, None, 20]

[11, 9002, 33000, 17000]

[14, 17000, None, 9002]

[17, None, None, 17]

[10, 9002, 9000, 25000]

[19, 25000, None, 23]

[5, 9002, 33001, 25001]

[18, 25001, None, 27]

[1, 9001, 9002, 17001]

[14, 17001, None, 9001]

[17, None, None, 20]

[11, 9002, 33000, 17000]

[14, 17000, None, 9002]

[17, None, None, 17]

[10, 9002, 9000, 25000]

[19, 25000, None, 23]

[5, 9002, 33001, 25001]

[18, 25001, None, 27]

[1, 9001, 9002, 17001]

[14, 17001, None, 9001]

[17, None, None, 20]

[11, 9002, 33000, 17000]

[14, 17000, None, 9002]

[17, None, None, 17]

[10, 9002, 9000, 25000]

[19, 25000, None, 23]

[5, 9002, 33001, 25001]

[18, 25001, None, 27]

[1, 9001, 9002, 17001]

[14, 17001, None, 9001]

[17, None, None, 20]

[11, 9002, 33000, 17000]

[14, 17000, None, 9002]

[17, None, None, 17]

[10, 9002, 9000, 25000]

[19, 25000, None, 23]

[5, 9002, 33001, 25001]

[18, 25001, None, 27]

[1, 9001, 9002, 17001]

[14, 17001, None, 9001]

[17, None, None, 20]

[11, 9002, 33000, 17000]

[14, 17000, None, 9002]

[17, None, None, 17]

[10, 9002, 9000, 25000]

[19, 25000, None, 23]

[18, 25000, None, 30]

[14, 9001, None, 1001]

[20, None, None, None]

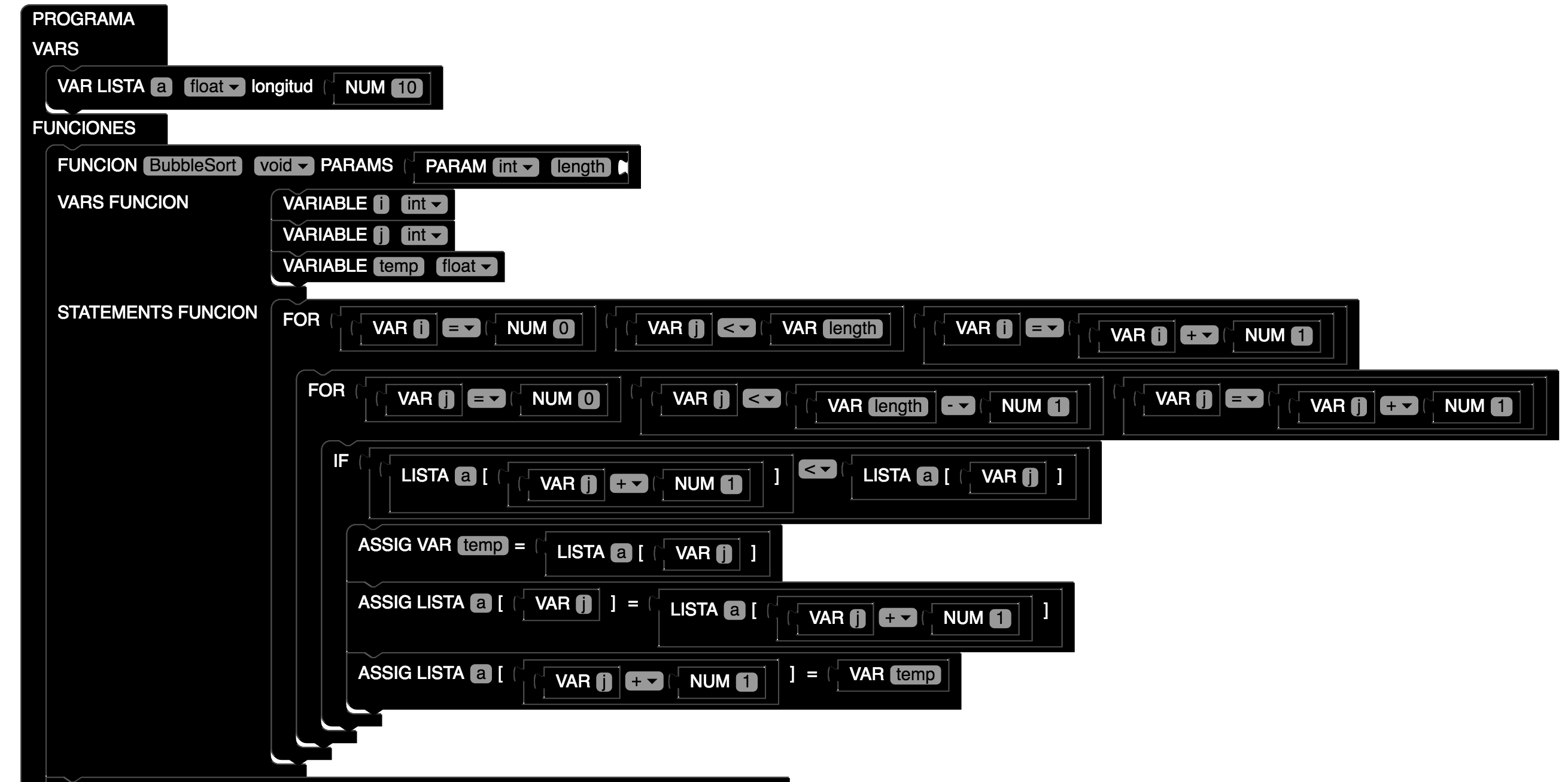
[14, 1001, None, 17001]

[16, 17001, None, None]

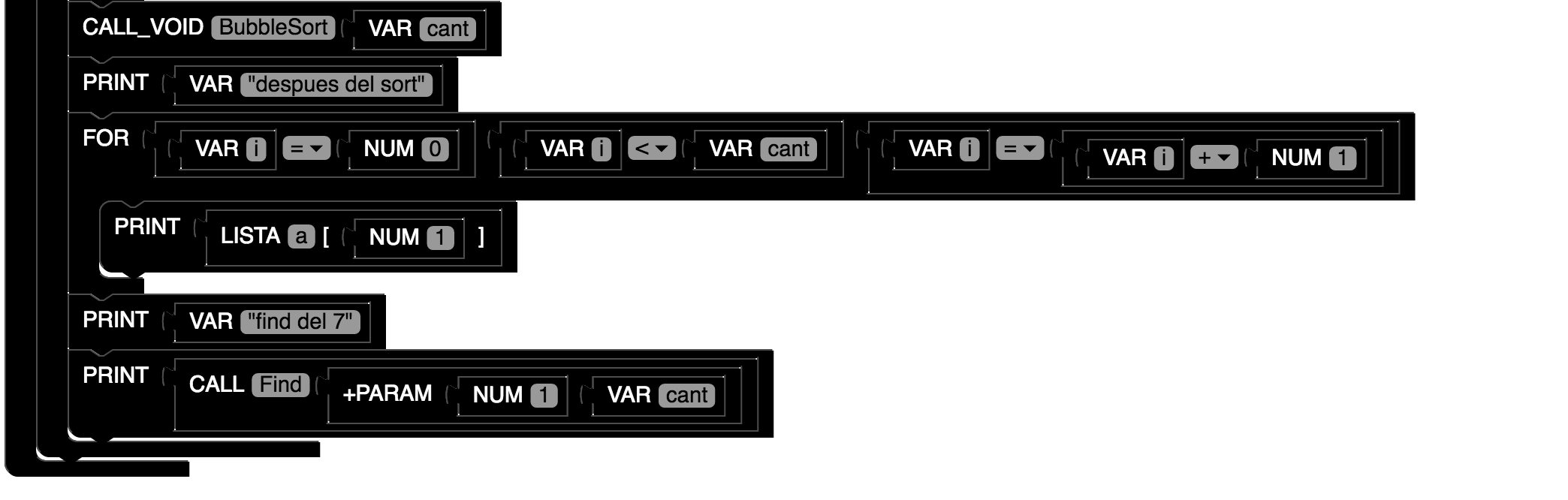
5040

## Listas

Blockly







Archivo de texto

VAR a : FLOAT LIST(10);

FUNC VOID BubbleSort (INT length)

VAR i : INT;

VAR j : INT;

VAR temp : FLOAT;

{

FOR (i = 0; i < length; i=i+1;)

{

FOR (j = 0; j < length - 1; j = j+1;)

{

IF (a[j + 1] < a[j])

{

temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

};

};

};

}

FUNC INT Find (FLOAT dato, INT length)

VAR i : INT;

{

FOR (i = 0; i < length; i=i+1;)

{

IF (a[i] == dato)

{

RETURN (i);

};

};

RETURN (-1);

}

CANCION(200)

VAR cant : INT;

VAR i : INT;

{

cant = 5;

FOR(i = 0; i < cant;i = i+1;){

a[i] = 10 - i;

};

PRINT "antes del sort";

FOR(i = 0; i < cant;i = i+1;){

PRINT a[i];

};

CALL BubbleSort(cant);

PRINT "despues del sort";

FOR(i = 0; i < cant;i = i+1;){

PRINT a[i];

};

PRINT "Find del 7:";

PRINT CALL Find(7,cant);;

}

Impresión en terminal:

{ 1: [17, None, None, 58],

2: [14, 33000, None, 9001],

3: [8, 9001, 9000, 25000],

4: [19, 25000, None, 9],

5: [18, 25000, None, 39],

6: [11, 9001, 33001, 17000],

7: [14, 17000, None, 9001],

8: [17, None, None, 3],

9: [14, 33000, None, 9002],

10: [12, 9000, 33001, 17001],

11: [8, 9002, 17001, 25001],

12: [19, 25001, None, 17],

13: [18, 25001, None, 38],

14: [11, 9002, 33001, 17002],

15: [14, 17002, None, 9002],

16: [17, None, None, 10],

17: [11, 9002, 33001, 17003],

18: [24, 17003, 0, 9],

19: [11, 17003, 33002, 17004],

20: [24, 9002, 0, 9],

21: [11, 9002, 33002, 17005],

22: [8, '(17004)', '(17005)', 25002],

23: [18, 25002, None, 37],

24: [24, 9002, 0, 9],

25: [11, 9002, 33002, 17006],

26: [14, '(17006)', None, 11000],

27: [24, 9002, 0, 9],

28: [11, 9002, 33002, 17007],

29: [11, 9002, 33001, 17008],

30: [24, 17008, 0, 9],

31: [11, 17008, 33002, 17009],

32: [14, '(17009)', None, '(17007)'],

33: [11, 9002, 33001, 17010],

34: [24, 17010, 0, 9],

35: [11, 17010, 33002, 17011],

36: [14, 11000, None, '(17011)'],

37: [17, None, None, 14],

38: [17, None, None, 6],

39: [20, None, None, None],

40: [14, 33000, None, 9001],

41: [8, 9001, 9000, 25000],

42: [19, 25000, None, 47],

43: [18, 25000, None, 54],

44: [11, 9001, 33001, 17000],

45: [14, 17000, None, 9001],

46: [17, None, None, 41],

47: [24, 9001, 0, 9],

48: [11, 9001, 33002, 17001],

49: [5, '(17001)', 11000, 25001],

50: [18, 25001, None, 53],

51: [14, 9001, None, 1000],

52: [20, None, None, None],

53: [17, None, None, 44],

54: [25, 33001, None, 17002],

55: [14, 17002, None, 1000],

56: [20, None, None, None],

57: [20, None, None, None],

58: [14, 33003, None, 9000],

59: [14, 33000, None, 9001],

60: [8, 9001, 9000, 25000],

61: [19, 25000, None, 66],

62: [18, 25000, None, 71],

63: [11, 9001, 33001, 17000],

64: [14, 17000, None, 9001],

65: [17, None, None, 60],

66: [24, 9001, 0, 9],

67: [11, 9001, 33002, 17001],

68: [12, 33004, 9001, 17002],

69: [14, 17002, None, '(17001)'],

70: [17, None, None, 63],

71: [16, 45000, None, None],

72: [14, 33000, None, 9001],

73: [8, 9001, 9000, 25001],

74: [19, 25001, None, 79],

75: [18, 25001, None, 83],

76: [11, 9001, 33001, 17003],

77: [14, 17003, None, 9001],

78: [17, None, None, 73],

79: [24, 9001, 0, 9],

80: [11, 9001, 33002, 17004],

81: [16, '(17004)', None, None],

82: [17, None, None, 76],

83: [21, 'BubbleSort', None, None],

84: [22, 9000, None, 9000],

85: [23, 'BubbleSort', None, None],

86: [16, 45001, None, None],

87: [14, 33000, None, 9001],

88: [8, 9001, 9000, 25002],

89: [19, 25002, None, 94],

90: [18, 25002, None, 98],

91: [11, 9001, 33001, 17005],

92: [14, 17005, None, 9001],

93: [17, None, None, 88],

94: [24, 9001, 0, 9],

95: [11, 9001, 33002, 17006],

96: [16, '(17006)', None, None],

97: [17, None, None, 91],

98: [16, 45002, None, None],

99: [21, 'Find', None, None],

100: [22, 33005, None, 11000],

101: [22, 9000, None, 9000],

102: [23, 'Find', None, None],

103: [14, 1000, None, 17007],

104: [16, 17007, None, None]}

[]

[]

[]

deque([])

comienza maquina virtual

{0: 33000, 1: 33001, 5: 33003, 7: 33005, '"despues del sort"': 45001, 10: 33004, '"antes del sort"': 45000, '"Find del 7:"': 45002, 3000: 33002}

[17, None, None, 58]

[14, 33003, None, 9000]

[14, 33000, None, 9001]

[8, 9001, 9000, 25000]

[19, 25000, None, 66]

[24, 9001, 0, 9]

[11, 9001, 33002, 17001]

[12, 33004, 9001, 17002]

[14, 17002, None, '(17001)']

[17, None, None, 63]

[11, 9001, 33001, 17000]

[14, 17000, None, 9001]

[17, None, None, 60]

[8, 9001, 9000, 25000]

[19, 25000, None, 66]

[24, 9001, 0, 9]

[11, 9001, 33002, 17001]

[12, 33004, 9001, 17002]

[14, 17002, None, '(17001)']

[17, None, None, 63]

[11, 9001, 33001, 17000]

[14, 17000, None, 9001]

[17, None, None, 60]

[8, 9001, 9000, 25000]

[19, 25000, None, 66]

[24, 9001, 0, 9]

[11, 9001, 33002, 17001]

[12, 33004, 9001, 17002]

[14, 17002, None, '(17001)']

[17, None, None, 63]

[11, 9001, 33001, 17000]

[14, 17000, None, 9001]

[17, None, None, 60]

[8, 9001, 9000, 25000]

[19, 25000, None, 66]

[24, 9001, 0, 9]

[11, 9001, 33002, 17001]

[12, 33004, 9001, 17002]

[14, 17002, None, '(17001)']

[17, None, None, 63]

[11, 9001, 33001, 17000]

[14, 17000, None, 9001]

[17, None, None, 60]

[8, 9001, 9000, 25000]

[19, 25000, None, 66]

[24, 9001, 0, 9]

[11, 9001, 33002, 17001]

[12, 33004, 9001, 17002]

[14, 17002, None, '(17001)']

[17, None, None, 63]

[11, 9001, 33001, 17000]

[14, 17000, None, 9001]

[17, None, None, 60]

[8, 9001, 9000, 25000]

[19, 25000, None, 66]

[18, 25000, None, 71]

[16, 45000, None, None]

"antes del sort"

[14, 33000, None, 9001]

[8, 9001, 9000, 25001]

[19, 25001, None, 79]

[24, 9001, 0, 9]

[11, 9001, 33002, 17004]

[16, '(17004)', None, None]

10

[17, None, None, 76]

[11, 9001, 33001, 17003]

[14, 17003, None, 9001]

[17, None, None, 73]

[8, 9001, 9000, 25001]

[19, 25001, None, 79]

[24, 9001, 0, 9]

[11, 9001, 33002, 17004]

[16, '(17004)', None, None]

9

[17, None, None, 76]

[11, 9001, 33001, 17003]

[14, 17003, None, 9001]

[17, None, None, 73]

[8, 9001, 9000, 25001]

[19, 25001, None, 79]

[24, 9001, 0, 9]

[11, 9001, 33002, 17004]

[16, '(17004)', None, None]

8

[17, None, None, 76]

[11, 9001, 33001, 17003]

[14, 17003, None, 9001]

[17, None, None, 73]

[8, 9001, 9000, 25001]

[19, 25001, None, 79]

[24, 9001, 0, 9]

[11, 9001, 33002, 17004]

[16, '(17004)', None, None]

7

[17, None, None, 76]

[11, 9001, 33001, 17003]

[14, 17003, None, 9001]

[17, None, None, 73]

[8, 9001, 9000, 25001]

[19, 25001, None, 79]

[24, 9001, 0, 9]

[11, 9001, 33002, 17004]

[16, '(17004)', None, None]

6

[17, None, None, 76]

[11, 9001, 33001, 17003]

[14, 17003, None, 9001]

[17, None, None, 73]

[8, 9001, 9000, 25001]

[19, 25001, None, 79]

[18, 25001, None, 83]

[21, 'BubbleSort', None, None]

[22, 9000, None, 9000]

[23, 'BubbleSort', None, None]

[14, 33000, None, 9001]

[8, 9001, 9000, 25000]

[19, 25000, None, 9]

[14, 33000, None, 9002]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[24, 9002, 0, 9]

[11, 9002, 33002, 17006]

[14, '(17006)', None, 11000]

[24, 9002, 0, 9]

[11, 9002, 33002, 17007]

[11, 9002, 33001, 17008]

[24, 17008, 0, 9]

[11, 17008, 33002, 17009]

[14, '(17009)', None, '(17007)']

[11, 9002, 33001, 17010]

[24, 17010, 0, 9]

[11, 17010, 33002, 17011]

[14, 11000, None, '(17011)']

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[24, 9002, 0, 9]

[11, 9002, 33002, 17006]

[14, '(17006)', None, 11000]

[24, 9002, 0, 9]

[11, 9002, 33002, 17007]

[11, 9002, 33001, 17008]

[24, 17008, 0, 9]

[11, 17008, 33002, 17009]

[14, '(17009)', None, '(17007)']

[11, 9002, 33001, 17010]

[24, 17010, 0, 9]

[11, 17010, 33002, 17011]

[14, 11000, None, '(17011)']

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[24, 9002, 0, 9]

[11, 9002, 33002, 17006]

[14, '(17006)', None, 11000]

[24, 9002, 0, 9]

[11, 9002, 33002, 17007]

[11, 9002, 33001, 17008]

[24, 17008, 0, 9]

[11, 17008, 33002, 17009]

[14, '(17009)', None, '(17007)']

[11, 9002, 33001, 17010]

[24, 17010, 0, 9]

[11, 17010, 33002, 17011]

[14, 11000, None, '(17011)']

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[24, 9002, 0, 9]

[11, 9002, 33002, 17006]

[14, '(17006)', None, 11000]

[24, 9002, 0, 9]

[11, 9002, 33002, 17007]

[11, 9002, 33001, 17008]

[24, 17008, 0, 9]

[11, 17008, 33002, 17009]

[14, '(17009)', None, '(17007)']

[11, 9002, 33001, 17010]

[24, 17010, 0, 9]

[11, 17010, 33002, 17011]

[14, 11000, None, '(17011)']

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[18, 25001, None, 38]

[17, None, None, 6]

[11, 9001, 33001, 17000]

[14, 17000, None, 9001]

[17, None, None, 3]

[8, 9001, 9000, 25000]

[19, 25000, None, 9]

[14, 33000, None, 9002]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[24, 9002, 0, 9]

[11, 9002, 33002, 17006]

[14, '(17006)', None, 11000]

[24, 9002, 0, 9]

[11, 9002, 33002, 17007]

[11, 9002, 33001, 17008]

[24, 17008, 0, 9]

[11, 17008, 33002, 17009]

[14, '(17009)', None, '(17007)']

[11, 9002, 33001, 17010]

[24, 17010, 0, 9]

[11, 17010, 33002, 17011]

[14, 11000, None, '(17011)']

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[24, 9002, 0, 9]

[11, 9002, 33002, 17006]

[14, '(17006)', None, 11000]

[24, 9002, 0, 9]

[11, 9002, 33002, 17007]

[11, 9002, 33001, 17008]

[24, 17008, 0, 9]

[11, 17008, 33002, 17009]

[14, '(17009)', None, '(17007)']

[11, 9002, 33001, 17010]

[24, 17010, 0, 9]

[11, 17010, 33002, 17011]

[14, 11000, None, '(17011)']

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[24, 9002, 0, 9]

[11, 9002, 33002, 17006]

[14, '(17006)', None, 11000]

[24, 9002, 0, 9]

[11, 9002, 33002, 17007]

[11, 9002, 33001, 17008]

[24, 17008, 0, 9]

[11, 17008, 33002, 17009]

[14, '(17009)', None, '(17007)']

[11, 9002, 33001, 17010]

[24, 17010, 0, 9]

[11, 17010, 33002, 17011]

[14, 11000, None, '(17011)']

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[18, 25001, None, 38]

[17, None, None, 6]

[11, 9001, 33001, 17000]

[14, 17000, None, 9001]

[17, None, None, 3]

[8, 9001, 9000, 25000]

[19, 25000, None, 9]

[14, 33000, None, 9002]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[24, 9002, 0, 9]

[11, 9002, 33002, 17006]

[14, '(17006)', None, 11000]

[24, 9002, 0, 9]

[11, 9002, 33002, 17007]

[11, 9002, 33001, 17008]

[24, 17008, 0, 9]

[11, 17008, 33002, 17009]

[14, '(17009)', None, '(17007)']

[11, 9002, 33001, 17010]

[24, 17010, 0, 9]

[11, 17010, 33002, 17011]

[14, 11000, None, '(17011)']

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[24, 9002, 0, 9]

[11, 9002, 33002, 17006]

[14, '(17006)', None, 11000]

[24, 9002, 0, 9]

[11, 9002, 33002, 17007]

[11, 9002, 33001, 17008]

[24, 17008, 0, 9]

[11, 17008, 33002, 17009]

[14, '(17009)', None, '(17007)']

[11, 9002, 33001, 17010]

[24, 17010, 0, 9]

[11, 17010, 33002, 17011]

[14, 11000, None, '(17011)']

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[18, 25001, None, 38]

[17, None, None, 6]

[11, 9001, 33001, 17000]

[14, 17000, None, 9001]

[17, None, None, 3]

[8, 9001, 9000, 25000]

[19, 25000, None, 9]

[14, 33000, None, 9002]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[24, 9002, 0, 9]

[11, 9002, 33002, 17006]

[14, '(17006)', None, 11000]

[24, 9002, 0, 9]

[11, 9002, 33002, 17007]

[11, 9002, 33001, 17008]

[24, 17008, 0, 9]

[11, 17008, 33002, 17009]

[14, '(17009)', None, '(17007)']

[11, 9002, 33001, 17010]

[24, 17010, 0, 9]

[11, 17010, 33002, 17011]

[14, 11000, None, '(17011)']

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[18, 25001, None, 38]

[17, None, None, 6]

[11, 9001, 33001, 17000]

[14, 17000, None, 9001]

[17, None, None, 3]

[8, 9001, 9000, 25000]

[19, 25000, None, 9]

[14, 33000, None, 9002]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[11, 9002, 33001, 17003]

[24, 17003, 0, 9]

[11, 17003, 33002, 17004]

[24, 9002, 0, 9]

[11, 9002, 33002, 17005]

[8, '(17004)', '(17005)', 25002]

[18, 25002, None, 37]

[17, None, None, 14]

[11, 9002, 33001, 17002]

[14, 17002, None, 9002]

[17, None, None, 10]

[12, 9000, 33001, 17001]

[8, 9002, 17001, 25001]

[19, 25001, None, 17]

[18, 25001, None, 38]

[17, None, None, 6]

[11, 9001, 33001, 17000]

[14, 17000, None, 9001]

[17, None, None, 3]

[8, 9001, 9000, 25000]

[19, 25000, None, 9]

[18, 25000, None, 39]

[20, None, None, None]

[16, 45001, None, None]

"despues del sort"

[14, 33000, None, 9001]

[8, 9001, 9000, 25002]

[19, 25002, None, 94]

[24, 9001, 0, 9]

[11, 9001, 33002, 17006]

[16, '(17006)', None, None]

6

[17, None, None, 91]

[11, 9001, 33001, 17005]

[14, 17005, None, 9001]

[17, None, None, 88]

[8, 9001, 9000, 25002]

[19, 25002, None, 94]

[24, 9001, 0, 9]

[11, 9001, 33002, 17006]

[16, '(17006)', None, None]

7

[17, None, None, 91]

[11, 9001, 33001, 17005]

[14, 17005, None, 9001]

[17, None, None, 88]

[8, 9001, 9000, 25002]

[19, 25002, None, 94]

[24, 9001, 0, 9]

[11, 9001, 33002, 17006]

[16, '(17006)', None, None]

8

[17, None, None, 91]

[11, 9001, 33001, 17005]

[14, 17005, None, 9001]

[17, None, None, 88]

[8, 9001, 9000, 25002]

[19, 25002, None, 94]

[24, 9001, 0, 9]

[11, 9001, 33002, 17006]

[16, '(17006)', None, None]

9

[17, None, None, 91]

[11, 9001, 33001, 17005]

[14, 17005, None, 9001]

[17, None, None, 88]

[8, 9001, 9000, 25002]

[19, 25002, None, 94]

[24, 9001, 0, 9]

[11, 9001, 33002, 17006]

[16, '(17006)', None, None]

10

[17, None, None, 91]

[11, 9001, 33001, 17005]

[14, 17005, None, 9001]

[17, None, None, 88]

[8, 9001, 9000, 25002]

[19, 25002, None, 94]

[18, 25002, None, 98]

[16, 45002, None, None]

"Find del 7:"

[21, 'Find', None, None]

[22, 33005, None, 11000]

[22, 9000, None, 9000]

[23, 'Find', None, None]

[14, 33000, None, 9001]

[8, 9001, 9000, 25000]

[19, 25000, None, 47]

[24, 9001, 0, 9]

[11, 9001, 33002, 17001]

[5, '(17001)', 11000, 25001]

[18, 25001, None, 53]

[17, None, None, 44]

[11, 9001, 33001, 17000]

[14, 17000, None, 9001]

[17, None, None, 41]

[8, 9001, 9000, 25000]

[19, 25000, None, 47]

[24, 9001, 0, 9]

[11, 9001, 33002, 17001]

[5, '(17001)', 11000, 25001]

[18, 25001, None, 53]

[14, 9001, None, 1000]

[20, None, None, None]

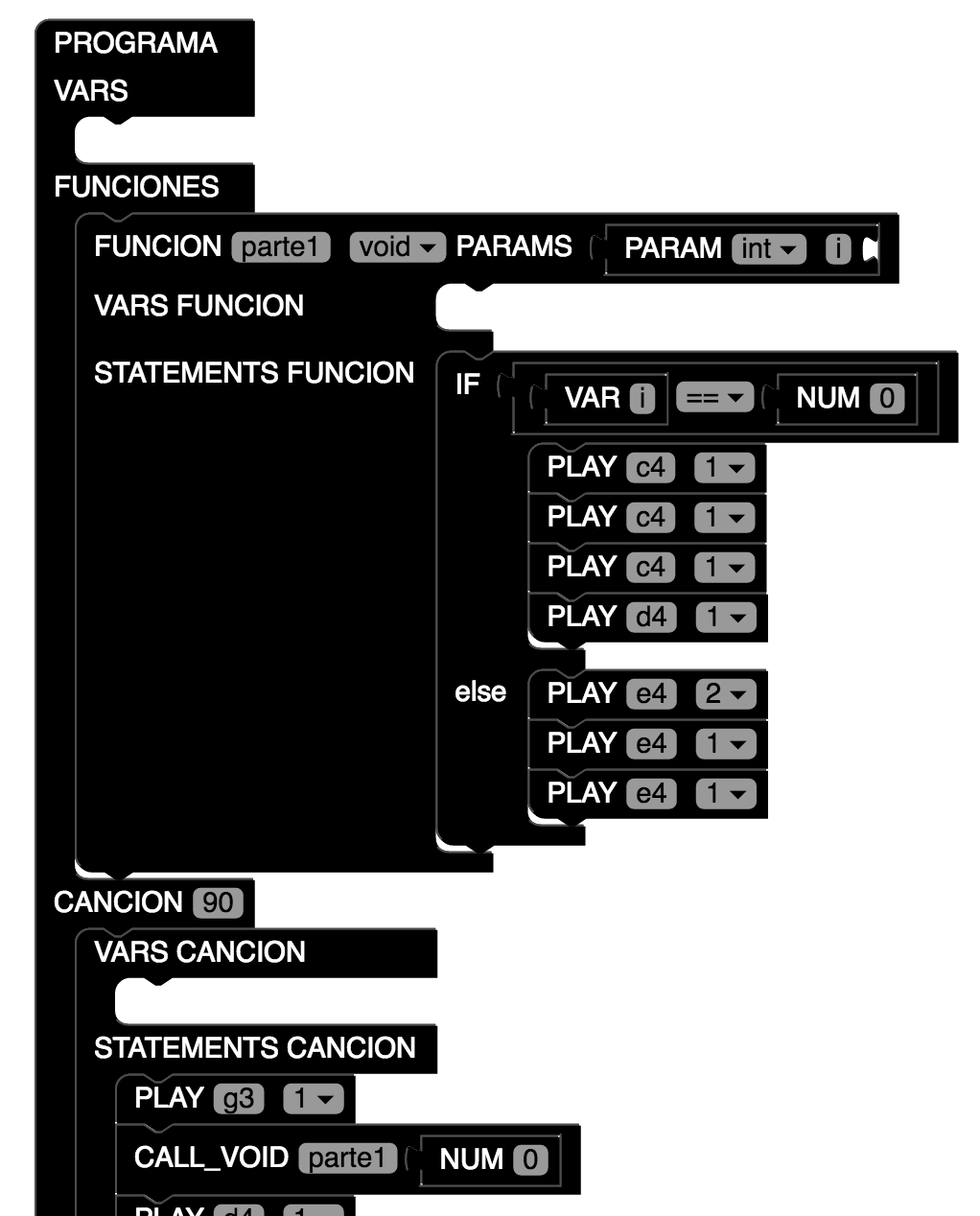
[14, 1000, None, 17007]

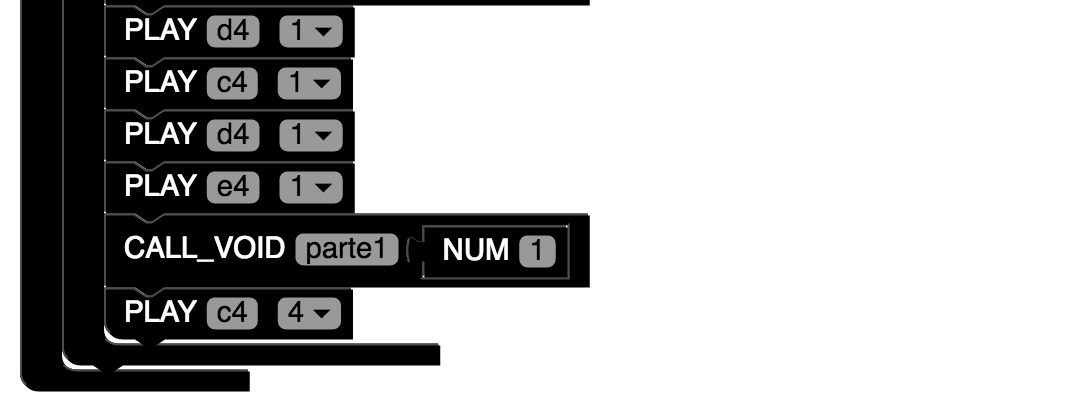
[16, 17007, None, None]

1

## Composición musical secuencial

Blockly





Archivo de Texto

FUNC VOID parte1(INT i)

{

IF (i==0){

PLAY(c4,4);PLAY(c4,4);PLAY(c4,4);PLAY(d4,4);

}

ELSE {

PLAY(e4,3);PLAY(e4,4);PLAY(e4,4);

};}

CANCION (90)

{

PLAY(g3,4);CALL parte1(0);PLAY(d4,4);PLAY(c4,4);PLAY(d4,4);PLAY(e4,4);CALL parte1(1);PLAY(c4,1);}

Impresión en Terminal

{ 1: [17, None, None, 13],

2: [5, 9000, 33000, 25000],

3: [18, 25000, None, 9],

4: [15, 49000, 33001, None],

5: [15, 49000, 33001, None],

6: [15, 49000, 33001, None],

7: [15, 49001, 33001, None],

8: [17, None, None, 12],

9: [15, 49002, 33002, None],

10: [15, 49002, 33001, None],

11: [15, 49002, 33001, None],

12: [20, None, None, None],

13: [15, 49003, 33001, None],

14: [21, 'parte1', None, None],

15: [22, 33000, None, 9000],

16: [23, 'parte1', None, None],

17: [15, 49001, 33001, None],

18: [15, 49000, 33001, None],

19: [15, 49001, 33001, None],

20: [15, 49002, 33001, None],

21: [21, 'parte1', None, None],

22: [22, 33003, None, 9000],

23: [23, 'parte1', None, None],

24: [15, 49000, 33003, None]}

[]

[]

[]

deque([])

comienza maquina virtual

{0: 33000, 1: 33003, 3: 33002, 4: 33001, 'g3': 49003, 'e4': 49002, 'd4': 49001, 'c4': 49000}

[17, None, None, 13]

[15, 49003, 33001, None]

[21, 'parte1', None, None]

[22, 33000, None, 9000]

[23, 'parte1', None, None]

[5, 9000, 33000, 25000]

[18, 25000, None, 9]

[15, 49000, 33001, None]

[15, 49000, 33001, None]

[15, 49000, 33001, None]

[15, 49001, 33001, None]

[17, None, None, 12]

[20, None, None, None]

[15, 49001, 33001, None]

[15, 49000, 33001, None]

[15, 49001, 33001, None]

[15, 49002, 33001, None]

[21, 'parte1', None, None]

[22, 33003, None, 9000]

[23, 'parte1', None, None]

[5, 9000, 33000, 25000]

[18, 25000, None, 9]

[15, 49002, 33002, None]

[15, 49002, 33001, None]

[15, 49002, 33001, None]

[20, None, None, None]

[15, 49000, 33003, None]

/Users/ikerarbululozano/Google Drive/Noveno Semestre/Compiladores/proyecto/codigo/parser/pysynth\_b.py:277: VisibleDeprecationWarning: using a non-integer number instead of an integer will result in an error in the future

data = np.zeros((repeat+1)\*t\_len + 441000.)

[1/13]

[5/13]

[9/13]

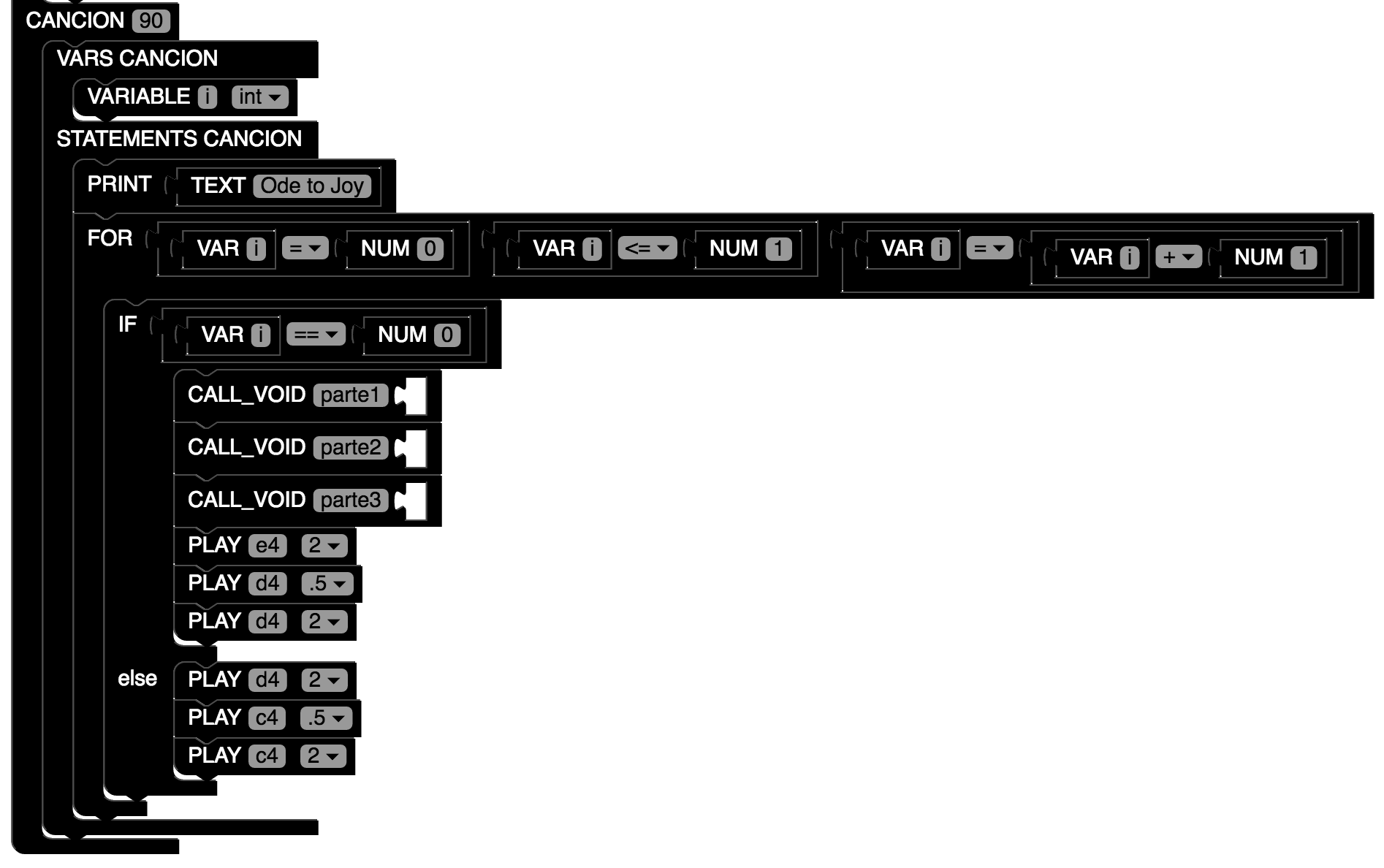
[13/13]

Writing to file song.wav

## Composición musical cíclica

Blockly





Archivo de Texto

FUNC VOID parte1()

{

PLAY(e4,4);PLAY(e4,4);PLAY(f4,4);PLAY(g4,4);}

FUNC VOID parte2()

{

PLAY(g4,4);PLAY(f4,4);PLAY(e4,4);PLAY(d4,4);}

FUNC VOID parte3()

{

PLAY(c4,4);PLAY(c4,4);PLAY(d4,4);PLAY(e4,4);}

CANCION (90) VAR i : INT;

{

PRINT "Ode to Joy";FOR (i=0;i<=1;i=i+1;){

IF (i==0){

CALL parte1();CALL parte2();CALL parte3();PLAY(e4,3);PLAY(d4,8);PLAY(d4,3);

}

ELSE {

PLAY(d4,3);PLAY(c4,8);PLAY(c4,3);

};

};}

Impresión en Terminal

{ 1: [17, None, None, 17],

2: [15, 49000, 33000, None],

3: [15, 49000, 33000, None],

4: [15, 49001, 33000, None],

5: [15, 49002, 33000, None],

6: [20, None, None, None],

7: [15, 49002, 33000, None],

8: [15, 49001, 33000, None],

9: [15, 49000, 33000, None],

10: [15, 49003, 33000, None],

11: [20, None, None, None],

12: [15, 49004, 33000, None],

13: [15, 49004, 33000, None],

14: [15, 49003, 33000, None],

15: [15, 49000, 33000, None],

16: [20, None, None, None],

17: [16, 45000, None, None],

18: [14, 33001, None, 9000],

19: [10, 9000, 33002, 25000],

20: [19, 25000, None, 25],

21: [18, 25000, None, 41],

22: [11, 9000, 33002, 17000],

23: [14, 17000, None, 9000],

24: [17, None, None, 19],

25: [5, 9000, 33001, 25001],

26: [18, 25001, None, 37],

27: [21, 'parte1', None, None],

28: [23, 'parte1', None, None],

29: [21, 'parte2', None, None],

30: [23, 'parte2', None, None],

31: [21, 'parte3', None, None],

32: [23, 'parte3', None, None],

33: [15, 49000, 33003, None],

34: [15, 49003, 33004, None],

35: [15, 49003, 33003, None],

36: [17, None, None, 40],

37: [15, 49003, 33003, None],

38: [15, 49004, 33004, None],

39: [15, 49004, 33003, None],

40: [17, None, None, 22]}

[]

[]

[]

deque([])

comienza maquina virtual

{0: 33001, '"Ode to Joy"': 45000, 'g4': 49002, 'f4': 49001, 1: 33002, 8: 33004, 3: 33003, 'e4': 49000, 'd4': 49003, 4: 33000, 'c4': 49004}

[17, None, None, 17]

[16, 45000, None, None]

"Ode to Joy"

[14, 33001, None, 9000]

[10, 9000, 33002, 25000]

[19, 25000, None, 25]

[5, 9000, 33001, 25001]

[18, 25001, None, 37]

[21, 'parte1', None, None]

[23, 'parte1', None, None]

[15, 49000, 33000, None]

[15, 49000, 33000, None]

[15, 49001, 33000, None]

[15, 49002, 33000, None]

[20, None, None, None]

[21, 'parte2', None, None]

[23, 'parte2', None, None]

[15, 49002, 33000, None]

[15, 49001, 33000, None]

[15, 49000, 33000, None]

[15, 49003, 33000, None]

[20, None, None, None]

[21, 'parte3', None, None]

[23, 'parte3', None, None]

[15, 49004, 33000, None]

[15, 49004, 33000, None]

[15, 49003, 33000, None]

[15, 49000, 33000, None]

[20, None, None, None]

[15, 49000, 33003, None]

[15, 49003, 33004, None]

[15, 49003, 33003, None]

[17, None, None, 40]

[17, None, None, 22]

[11, 9000, 33002, 17000]

[14, 17000, None, 9000]

[17, None, None, 19]

[10, 9000, 33002, 25000]

[19, 25000, None, 25]

[5, 9000, 33001, 25001]

[18, 25001, None, 37]

[15, 49003, 33003, None]

[15, 49004, 33004, None]

[15, 49004, 33003, None]

[17, None, None, 22]

[11, 9000, 33002, 17000]

[14, 17000, None, 9000]

[17, None, None, 19]

[10, 9000, 33002, 25000]

[19, 25000, None, 25]

[18, 25000, None, 41]

/Users/ikerarbululozano/Google Drive/Noveno Semestre/Compiladores/proyecto/codigo/parser/pysynth\_b.py:277: VisibleDeprecationWarning: using a non-integer number instead of an integer will result in an error in the future

data = np.zeros((repeat+1)\*t\_len + 441000.)

[1/18]

[5/18]

[9/18]

[13/18]

[17/18]

Writing to file song.wav

Song.wav es un pasaje del himno a la alegría

Listados Perfectamente Documentados

## Memoria

## clase de memoria la que se utiliza para pedir memoria

## cuando llamas a una nueva funcion, la global o la de

## cancion, nos ayuda a definir el tamano en variables

## del elemento y las casillas y los valores de ejecucion

## que agarra cada casilla, inicialmente arrancan en 0

class Memoria:

## unica funcion de la clase memoria en la que recibe la cantidad

## de variables por tipo y temporales por tipo y un booleano

## que dice si es global o no

def \_\_init\_\_(self, vi, vf, vb, vc, ti, tf, tb, tc, globalmem):

## contador que empieza en 0 para cada ciclo

aux = 0

## creamos los diccionarios para cada tipo de dato

self.ints = {}

self.floats = {}

self.bools = {}

self.chars = {}

## si es global los iniciales son distintos de los

## locales

if globalmem:

aux\_vi = 1000

aux\_vf = 3000

aux\_vb = 5000

aux\_vc = 7000

else:

aux\_vi = 9000

aux\_vf = 11000

aux\_vb = 13000

aux\_vc = 15000

## solo hay temporales locales por lo que no importa

## si es global o local

aux\_ti = 17000

aux\_tf = 21000

aux\_tb = 25000

aux\_tc = 29000

## hacemos las casillas para todas las variables

## enteras

while aux < vi:

## inicialmente las declaramos en 0

self.ints[aux\_vi]=0

## sumamos el numero de casilla

aux\_vi += 1

## sumamos el contador para el ciclo

aux += 1

## reiniciamos el contador de ciclo

aux = 0

## hacemos las casillas para todas las variables

## flotantes

while aux < vf:

## inicialmente las declaramos en 0.0

self.floats[aux\_vf]=0.0

## sumamos el numero de casilla

aux\_vf += 1

## sumamos el contador para el ciclo

aux += 1

## reiniciamos el contador del ciclo

aux = 0

## hacemos las casillas para todas las variables

## booleanas

while aux < vb:

## inicialmente las declaramos en True

self.bools[aux\_vb] = True

## sumamos el numero de casilla

aux\_vb += 1

## sumamos el contador para el ciclo

aux += 1

## reiniciamos el contador de ciclo

aux = 0

## hacemos las casillas para todas las variables

## char

while aux < vc:

## inicialmente las declaramos en ''

self.chars[aux\_vc] = ''

## sumamos el numero de casilla

aux\_vc += 1

## sumamos el contador para el ciclo

aux += 1

## reiniciamos el contador de ciclo

aux = 0

## hacemos las casillas para todas las temporales

## enteras

while aux < ti:

## inicialmente las declaramos en 0

self.ints[aux\_ti] = 0

## sumamos el numero de casilla

aux\_ti += 1

## sumamos el contador de ciclo

aux += 1

## reiniciamos el contador de ciclo

aux = 0

## hacemos las casillas para todas las temporales

## flotantes

while aux < tf:

## inicialmente las declaramos en 0.0

self.floats[aux\_tf] = 0.0

## sumamos el numero de casilla

aux\_tf += 1

## sumamos el contador de ciclo

aux += 1

## reiniciamos el contador de ciclo

aux = 0

## hacemos las casillas para todas las temporales

## booleanas

while aux < tb:

## inicialmente las declaramos en True

self.bools[aux\_tb] = True

## sumamos el numero de casilla

aux\_tb += 1

## sumamos el contador de ciclo

aux += 1

## reiniciamos el contador de ciclo

aux = 0

## hacemos las casillas para todas las temporales

## char

while aux < tc:

## inicialmente las declaramos en ''

self.chars[aux\_tc] = ''

## sumamos el numero de casilla

aux\_tc += 1

## sumamos el contador de ciclo

aux += 1

## Máquina Virtual

######################################

## maquina virtual ##

######################################

## aqui comienza la maquina virtual

print "comienza maquina virtual"

## imprimimos constantes

print ctes

## nos traemos las variables necesarias

global current\_cuad

global song

## indicamos que empezamos en el primer cuadruplo

current\_cuad = 1

## declaramos la memoria global con la clase memoria y su

## tamano de acuerdo a las variables que necesita

memoriaGlobal = memoria.Memoria(dir\_procs['global'][pos\_dics\_tam]['vi'], dir\_procs['global'][pos\_dics\_tam]['vf'], dir\_procs['global'][pos\_dics\_tam]['vb'], dir\_procs['global'][pos\_dics\_tam]['vc'], dir\_procs['global'][pos\_dics\_tam]['ti'], dir\_procs['global'][pos\_dics\_tam]['tf'], dir\_procs['global'][pos\_dics\_tam]['tb'], dir\_procs['global'][pos\_dics\_tam]['tc'], True)

## hacemos lo mismo pero con nuestra clase main que nunca se

## genera un era para ella porque siempre se ejecuta en automatico

memoriaActiva = memoria.Memoria(dir\_procs['CANCION'][pos\_dics\_tam]['vi'], dir\_procs['CANCION'][pos\_dics\_tam]['vf'], dir\_procs['CANCION'][pos\_dics\_tam]['vb'], dir\_procs['CANCION'][pos\_dics\_tam]['vc'], dir\_procs['CANCION'][pos\_dics\_tam]['ti'], dir\_procs['CANCION'][pos\_dics\_tam]['tf'], dir\_procs['CANCION'][pos\_dics\_tam]['tb'], dir\_procs['CANCION'][pos\_dics\_tam]['tc'], False)

## declaramos la memoria Dormida vacia

memoriaDormida = []

## mientras que el cuadruplo en el que vamos sea

## menor que la cantidad de parametros seguimos mandando a

## llamar la funcion que lee cada cuadruplo

while current\_cuad < contCuad:

func\_maq\_virtual()

## nos traemos la libreria para hacer musica

import pysynth

## declaramos que existe antes

existe = True

## nos traemos la libreria que nos deja cambier el beat

from pysynth\_b import \*

## revisamos si utilizo plays el usuario

try:

song

except NameError:

## si arroja la excepcion significa que no se utilizo

## ningun play entonces no hay musica

existe = False

## si ya existe simplemente hacemos el wav correspondiente y

## lo guardamos en el archivo song.wav en el directorio de

## del ejecutable, se hace play con el beat que se declaro

## en la funcion Cancion

if existe:

make\_wav(song, fn = "song.wav", leg\_stac = .7, bpm = dir\_procs['CANCION'][5])

## Reconocimiento de apuntador en máquina virtual

## sacamos el operando izquierdo

opdoIzq = cuadruplos[current\_cuad][1]

## revisamos si es un apuntador porque tiene los parentesis

if isinstance(opdoIzq,basestring) and opdoIzq[0] == '(' and opdoIzq[-1]==')':

## en caso de que lo sea le hacemos un trim al string y lo

## convertimos en int

opdoIzq = int(opdoIzq[1:-1])

## y del apuntador sacamos el valor correspondiente dentro de la

## memoria, recorriendo cada uno de los rangos para saber de que

## parte sacarlo y luego ya sacamos el operando con su valor

if opdoIzq >= var\_glob\_int\_inicio and opdoIzq < var\_glob\_float\_inicio:

opdoIzq = memoriaGlobal.ints[opdoIzq]

elif opdoIzq >= var\_glob\_float\_inicio and opdoIzq < var\_glob\_bool\_inicio:

opdoIzq = memoriaGlobal.floats[opdoIzq]

elif opdoIzq >= var\_glob\_bool\_inicio and opdoIzq < var\_glob\_char\_inicio:

opdoIzq = memoriaGlobal.bools[opdoIzq]

elif opdoIzq >= var\_glob\_char\_inicio and opdoIzq < var\_loc\_int\_inicio:

opdoIzq = memoriaGlobal.chars[opdoIzq]

elif (opdoIzq >= var\_loc\_int\_inicio and opdoIzq < var\_loc\_float\_inicio) or (opdoIzq >= var\_loc\_temp\_int\_inicio and opdoIzq < var\_loc\_temp\_float\_inicio):

opdoIzq = memoriaActiva.ints[opdoIzq]

elif (opdoIzq >= var\_loc\_float\_inicio and opdoIzq < var\_loc\_bool\_inicio) or (opdoIzq >= var\_loc\_temp\_float\_inicio and opdoIzq < var\_loc\_temp\_bool\_inicio):

opdoIzq = memoriaActiva.floats[opdoIzq]

elif (opdoIzq >= var\_loc\_bool\_inicio and opdoIzq < var\_loc\_char\_inicio) or (opdoIzq >= var\_loc\_temp\_bool\_inicio and opdoIzq < var\_loc\_temp\_char\_inicio):

opdoIzq = memoriaActiva.bools[opdoIzq]

elif (opdoIzq >= var\_loc\_char\_inicio and opdoIzq < var\_loc\_temp\_int\_inicio) or (opdoIzq >= var\_loc\_temp\_char\_inicio and opdoIzq < cte\_int\_inicio):

opdoIzq = memoriaActiva.chars[opdoIzq]

elif opdoIzq >= cte\_int\_inicio and opdoIzq < cte\_float\_inicio:

opdoIzq = int(search(ctes,opdoIzq))

elif opdoIzq >= cte\_float\_inicio and opdoIzq < cte\_bool\_inicio:

opdoIzq = float(search(ctes,opdoIzq))

elif opdoIzq >= cte\_bool\_inicio and opdoIzq < cte\_char\_inicio:

## aqui lo convertimos a booleano de acuerdo a nuestra

## sintaxis

if search(ctes,opdoIzq) == 'tru':

opdoIzq = True

else:

opdoIzq = False

elif opdoIzq >= cte\_char\_inicio:

opdoIzq = search(ctes,opdoIzq)

## Función search en ctes

##############################################################

## funcion search ##

### funcion utilizada para sacar la constante dada una ##

### direccion virtual, se utiliza en la maquina virtual ##

### cuando le llega una direccion virtual y tiene que ##

### saber cual es su constante buscando en el diccionario ##

### dada el valor y se tiene que encontrar la llave que ##

### corresponde ##

##############################################################

def search(values, searchFor):

## values es el diccionario y searchFor es la direccion

## virtual a encontrar, k empieza con la primera llave

## del diccionario y recorre todas

for k in values:

## en caso de que se encuentre entonces se regresa

## k que representa la constante y llave y se

## termina el ciclo

if searchFor == values[k]:

return k

## en caso de que termine el ciclo y no haya salido

## entonces no existe y regresamos None

return None

## Return

##############################################################

## return ##

### se utiliza en las funciones que no son de tipo void ##

### esto marca que ya se asigna la variable global que ##

### corresponde a la funcion y se le asigna el resultado ##

### de la expresion ##

##############################################################

def p\_return(p):

'return : RETURN "(" expresion ")" ";"'

## nos traemos las variables necesarias

global pos\_dics\_tipo

global pos\_dics\_var

global contCuad

global var\_glob\_int

global var\_glob\_float

global var\_glob\_float\_inicio

global var\_glob\_bool

global var\_glob\_bool\_inicio

global var\_glob\_char

global var\_glob\_char\_inicio

global var\_loc\_int\_inicio

## sacamo el resultado de la expresion y sera el retorno

## tambien sacamos el tipo resultante

retorno = pilaO.pop()

tipoRetorno = pTipos.pop()

## preparamos una asignacion ya que el return eso es lo que hace

op = EQ

## checamos si el retorno corresponde con el que se declaron en la variable

## vuelve a pasar el caso especial con regresar un entero cuando es flotante

if tipoRetorno == dir\_procs[scope[-1]][pos\_dics\_tipo] or (tipoRetorno == INT and dir\_procs[scope[-1]][pos\_dics\_tipo] == FLOAT):

## revisamos si ya existe la variable global para el scope

if scope[-1] in dir\_procs["global"][pos\_dics\_var]:

## si ya esta sacamos la direccion virtual y hacemos el cuadruplo de asignacion

var\_aux = dir\_procs["global"][pos\_dics\_var][scope[-1]][pos\_vars\_dir\_virtual]

cuadruplos[contCuad] = [op,retorno,None,var\_aux]

## si aun no existe

else:

## agregamos de acuerdo al tipo y luego armamos el cuadruplo

## de asignacion con la direccion virtual nueva

if dir\_procs[scope[-1]][pos\_dics\_tipo] == INT:

dir\_procs["global"][pos\_dics\_tam]['vi']+=1

if var\_glob\_int + 1 < var\_glob\_float\_inicio:

dir\_procs["global"][pos\_dics\_var][scope[-1]] = [dir\_procs[scope[-1]][pos\_dics\_tipo],var\_glob\_int,None]

cuadruplos[contCuad] = [op,retorno,None,var\_glob\_int]

var\_glob\_int += 1

else:

print "Overflow de variables enteras globales"

exit()

elif dir\_procs[scope[-1]][pos\_dics\_tipo] == FLOAT:

dir\_procs["global"][pos\_dics\_tam]['vf']+=1

if var\_glob\_float + 1 < var\_glob\_bool\_inicio:

dir\_procs["global"][pos\_dics\_var][scope[-1]] = [dir\_procs[scope[-1]][pos\_dics\_tipo],var\_glob\_float,None]

cuadruplos[contCuad] = [op,retorno,None,var\_glob\_float]

var\_glob\_float += 1

else:

print "Overflow de variables enteras globales"

exit()

elif dir\_procs[scope[-1]][pos\_dics\_tipo] == CHAR:

dir\_procs["global"][pos\_dics\_tam]['vc']+=1

if var\_glob\_char + 1 < var\_loc\_int\_inicio:

dir\_procs["global"][pos\_dics\_var][scope[-1]] = [dir\_procs[scope[-1]][pos\_dics\_tipo],var\_glob\_char,None]

cuadruplos[contCuad] = [op,retorno,None,var\_glob\_char]

var\_glob\_char += 1

else:

print "Overflow de variables enteras globales"

exit()

else:

dir\_procs["global"][pos\_dics\_tam]['vb']+=1

if var\_glob\_bool + 1 < var\_glob\_char\_inicio:

dir\_procs["global"][pos\_dics\_var][scope[-1]] = [dir\_procs[scope[-1]][pos\_dics\_tipo],var\_glob\_bool,None]

cuadruplos[contCuad] = [op,retorno,None,var\_glob\_bool]

var\_glob\_bool += 1

else:

print "Overflow de variables enteras globales"

exit()

contCuad += 1

else:

print "Error en el tipo de retorno dado"

exit()

## generamos el fin del procedimiento que al final cambia

## el socpe actual y vuelve a pasar a control de la funcion

## que lo llamo

op = ENDPROC

cuadruplos[contCuad]=[op,None,None,None]

contCuad += 1

pass

## Funciones

##########################################################

## callvoidfunc ##

### estatuto que permite al usuario cambiar al scope ##

### de una funcion de tipo void enviandole parametros ##

### y ejecutando instrucciones dentreo del scope de la ##

### funcion ##

##########################################################

def p\_callvoidfunc(p):

'callvoidfunc : CALL ID neur24 "(" s ")" neur26 ";"'

pass

##############################################################

## punto neuralgico 24 ##

### se utiliza en cualquier call a una funcion, genera ##

### un era para la nueva funcion a la que desea ir, ##

### agrega a la pila de funciones, a la pila de numero ##

### de funcion y finalmente deja todo listo para empezar ##

### a ingresar los parametros ##

##############################################################

def p\_neur24(p):

'neur24 : '

## nos traemos las variables necesarias

global contCuad

global auxFuncDestinoDir

global tempFunc

global currentFunc

global pOper

global pilaFuncs

## revisamos que exista la funcion a la que se quiere

## llamar

if p[-1] in dir\_procs:

## lo agregamos a la pila de funciones y agregamos

## el numero de parametros dados de alta en 0's

pilaFuncs.append(p[-1])

auxFuncDestinoDir = p[-1]

pilaAuxParamCount.append([0,0,0,0])

## generamos la operacion de era

op = ERA

cuadruplos[contCuad] = [op,p[-1],None,None]

contCuad += 1

## metemos el fondo falso representado por el parametro

pOper.append(PARAMETRO)

## apuntamos al numero de funcion corrrespondiente

currentFunc = tempFunc

## agregamos ese numero de funcion a la pila de numero de

## funciones

pilaNumFuncs.append(tempFunc)

## aumentamos el numero temporal

tempFunc += 1

## marcamos error si no esta en el directorio de procedimiento

else:

print "Funcion con ese id no existe"

exit()

pass

##########################################################

## punto neuralgico 26 ##

### se utiliza en cualquier llamada a funcion para ##

### revisar que se hayan dado el numero de parametros ##

### correctos y genera el GOSUB que ya cambia el scope ##

##########################################################

def p\_neur26(p):

'neur26 : '

## nos traemos las variables necesarias

global auxFuncDestinoDir

global contCuad

global pilaFuncs

global currentFunc

## definimos la funcion destino

auxFuncDestinoDir = pilaFuncs[-1]

## llevamos la cuenta de la cantidad e parametros al sumar todos los

## declarados por tipo

paramCount = pilaAuxParamCount[-1][0]+ pilaAuxParamCount[-1][1]+ pilaAuxParamCount[-1][2]+pilaAuxParamCount[-1][3]

## revisamos que el numero de parametros dados de alta sean iguales a la

## longitud de el arreglo de parametros en el diccionario de procedimientos

## no vaya a ser que dio parametros de menos

if paramCount != len(dir\_procs[auxFuncDestinoDir][pos\_dics\_params]):

print "Error en cantidad de parametros"

exit()

else:

## si no fallo el pasado entonces ya damos de alta el GOSUB

op = GOSUB

## checamos si esta el fondo falso

if(pOper[-1] == PARAMETRO):

## sacamos fondo falso, la funcion y el numero de parametros

## y al final cambiamos la funcion actual

pOper.pop()

pilaNumFuncs.pop()

pilaAuxParamCount.pop()

if pilaNumFuncs != []:

currentFunc = pilaNumFuncs[-1]

## se quedo con operaciones pendientes en los parametros

else:

print "Llamada a funcion con operaciones pendientes"

exit()

## generamos el GOSUB

cuadruplos[contCuad] = [op,auxFuncDestinoDir,None,None]

contCuad += 1

## sacamos la funcion

pilaFuncs.pop()

pass

##################################################################

## Regla sintactica s para poder meter los valores de los ##

## parametros por medio de expresiones ya que la funcion puede ##

## o no tener parametros y puede tener uno o muchos por eso se ##

## hace tambien un ciclo con la regla t ##

##################################################################

def p\_s(p):

'''s : empty

| expresion neur25 t'''

pass

##########################################################

## punto neuralgico 25 ##

### se utiliza cada vez que se termina la expresion de ##

### un parametro en la llamada a una funcion, sea void ##

### o con un tipo y se realiza una validacion para ver ##

### si el parametro es del mismo tipo y los esta ##

### ingresando en el orden correcto ##

##########################################################

def p\_neur25(p):

'neur25 : '

## nos traemos las variables necesarias

global dir\_procs

global contCuad

global auxFuncDestinoDir

global currentFunc

global pilaAuxParamCount

## llevamos la cuenta de la cantidad e parametros al sumar todos los

## declarados por tipo

paramCount = pilaAuxParamCount[-1][0]+ pilaAuxParamCount[-1][1]+ pilaAuxParamCount[-1][2]+pilaAuxParamCount[-1][3]

## como permitimos la llamada de una funcion como parametro

## de otra funcion tenemos que manejar una pila en la que

## el tope es a la funcion a la que se desea llamar y para

## eso utilizamos un auxiliar

auxFuncDestinoDir = pilaFuncs[-1]

## checamos que el pOper no este vacio y que la pila

## que marca que numero de funcion a la que estamos llamando

## sea la misma a la que tenemos en el tope, esto se hace

## para cuando el parametro de una funcion es si misma

if pOper != [] and currentFunc == pilaNumFuncs[-1]:

## checamos que la pilaO no este vacia y que la operacion

## pendiente o la pila del pOper sea el fondo falso de

## parametro

if len(pilaO)>0 and pOper[-1] == PARAMETRO:

## sacamos el argumento que es el resultado de la

## expresion

argumento = pilaO.pop()

## sacamos el tipo de la pila de tipos

tipoarg = pTipos.pop()

## checamos si es el mismo tipo el que mandamos que el numero de parametro

## al que se lo estamos mandando, esto se debe al tope de la pilaAuxParamCount que nos dice

## que numero de parametro estamos llamando y hacemos el caso especial que si

## nos manda un int en la llamada a un float en el destino se la aceptamos

if tipoarg == dir\_procs[auxFuncDestinoDir][pos\_dics\_params][paramCount] or (tipoarg == INT and dir\_procs[auxFuncDestinoDir][pos\_dics\_params][paramCount]==FLOAT):

## hacemos la operacion parametro y dependendo del tipo entonces

## utilizamos la variable local correspondiente

op = PARAMETRO

if tipoarg == INT:

## validacion especial cuando mandan un int a un float

if tipoarg == INT and dir\_procs[auxFuncDestinoDir][pos\_dics\_params][paramCount]==FLOAT:

cuadruplos[contCuad] = [op,argumento,None,var\_loc\_float\_inicio + pilaAuxParamCount[-1][1]]

## sumamos la cantidad de parametros flotantes

pilaAuxParamCount[-1][1] += 1

else:

cuadruplos[contCuad] = [op,argumento,None,var\_loc\_int\_inicio + pilaAuxParamCount[-1][0]]

## sumamos la cantidad de parametros enteros

pilaAuxParamCount[-1][0] += 1

elif tipoarg == FLOAT:

cuadruplos[contCuad] = [op,argumento,None,var\_loc\_float\_inicio + pilaAuxParamCount[-1][1]]

## sumamos la cantidad de parametros flotantes

pilaAuxParamCount[-1][1] += 1

elif tipoRes == CHAR:

cuadruplos[contCuad] = [op,argumento,None,var\_loc\_char\_inicio + pilaAuxParamCount[-1][2]]

## sumamos la cantidad de parametros char

pilaAuxParamCount[-1][2] += 1

else:

cuadruplos[contCuad] = [op,argumento,None,var\_loc\_bool\_inicio + pilaAuxParamCount[-1][3]]

## sumamos la cantidad de parametros booleanos

pilaAuxParamCount[-1][3] += 1

contCuad+=1

else:

print "Error en declaracion de parametros"

exit()

else:

argumento = None

tipoarg = None

else:

argumento = None

tipoarg = None

pass

## For

######################################################################

## for ##

### estatuto ciclico en el que se hace una asignacion una sola ##

### vez seguido de una expresion que siempre se evalua en cada ##

### pasada del ciclo y finalmente se hace una asignacion cada ##

### vez que se vuelve termina de ejecutar el ciclo ##

######################################################################

def p\_for(p):

'for : FOR "(" asignacion neur18 expresion ";" neur19 asignacion ")" neur21 bloque ";" neur20'

pass

######################################################################

## punto neuralgico 18 ##

### utilizado por el for para agregar a la pila de saltos ##

### el punto inicial antes de la expresion que se evalua cada ##

### vez que se ejecuta de nuevo el ciclo ##

######################################################################

def p\_neur18(p):

'neur18 : '

## nos traemos las variables necesarias

global contCuad

## agregamos a la pila de saltos el cuadruplo siguiente

## para que sepamos a donde brincar al comienzo del ciclo

pSaltos.append(contCuad)

pass

######################################################################

## punto neuralgico 19 ##

### utilizado por el for para revisar que la expresion sea de ##

### tipo booleana y agregamos los brincos correspondientes en falso ##

### y verdadero de acuerdo al resultado de la misma ya que siempre ##

### se debe dejar la segunda asignacion hasta el final del for ##

######################################################################

def p\_neur19(p):

'neur19 : '

## nos traemos las variable necesarias

global contCuad

## checamos que el tipo de la expresion sea BOOL

if pTipos[-1] == BOOL:

## lo sacamos de la pila de tipos

pTipos.pop()

## sacamos el opdoIzq que vendria a ser el resultado

## de la expresion para hacer los brincos en GOTOV y GOTOF

opdoIzq = pilaO.pop()

## armamos el GOTOV todavia sin destino porque no sabemos

## aun en que punto arranca el bloque del for

op = GOTOV

cuadruplos[contCuad] = [op,opdoIzq,None,None]

contCuad+=1

## agregamos a la pila de saltos el numero de cuadruplo

## del GOTOV que va a rellenar mas adelante cuando se encuentre

## el bloque del for

pSaltos.append(contCuad-1)

## armamos el GOTOF todavia sin destino porque no sabemos

## aun en que punto termina el for

op = GOTOF

cuadruplos[contCuad] = [op,opdoIzq,None,None]

contCuad+=1

## agregamos a la pila de saltos el numero de cuadruplo

## del GOTOF que va a rellenar mas adelante cuando se acabe

## el bloque del for

pSaltos.append(contCuad-1)

## agregamos a la pila de saltos el siguiente cuadruplo que

## realiza la segunda asignacion del for y como se hace hasta el final

## este se va con un GOTO al final del bloque del for

pSaltos.append(contCuad)

## si el resultado de la expresion no es bool hay un error

else:

print ("Tiene que tener un booleano como resultado de expresion")

exit()

pass

##################################################################

## punto neuralgico 21 ##

### se utiliza para hacer el salto despues de la segunda ##

### asignacion ya que despues de ahi vuelve a evaluarse ##

### la expresion anteriormente en el for y se usa para saber ##

### que ya comienza el bloque entonces el salto del GOTOV ##

### se rellena ##

##################################################################

def p\_neur21(p):

'neur21 : '

## nos traemos las variables necesarias

global contCuad

## marcamos la operacion del GOTO

op = GOTO

## utilizamos una pila auxiliar ya que el salto antes de la

## expresion no esta en el tope y se tiene que buscar

auxPila = []

## se encuentra dos niveles anidado el salto en la pila

i = 0

while i < 3:

## se transfieren de los saltos a la auxiliar

## se utiliza una pila para respetar el orden que se tenia

## en la pila

auxPila.append(pSaltos.pop())

i += 1

pass

## ya cuando llegamos al salto que queremos lo sacamos

## y ya hacemos la operacion del GOTO que nos permitira

## volver a comenzar el ciclo

ciclo = pSaltos.pop()

cuadruplos[contCuad] = [op,None,None,ciclo]

contCuad+=1

## ahora se encuentra en la pila auxiliar el GOTOV de

## la expresion y como ya viene el bloque lo sacamos y

## rellenamos el GOTOV con el cuadruplo que viene

verdadero = auxPila.pop()

cuadruplos[verdadero][3] = contCuad

## restablecemos la pila de saltos para que sigan el

## GOTOF y el punto para el brinco a la segunda asignacion

pSaltos.append(auxPila.pop())

pSaltos.append(auxPila.pop())

pass

##############################################################

## punto neuralgico 20 ##

### se utiliza para marcar el fin de la declaracion del ##

### for y se utiliza para hacer el brinco a la segunda ##

### asignacion y rellenar el GOTOF de la expresion ##

##############################################################

def p\_neur20(p):

'neur20 : '

## nos traemos las variables necesarias

global contCuad

## generamos la accion del GOTO que sera el

## salto a la segunda asignacion al final del for

op = GOTO

asigna = pSaltos.pop()

cuadruplos[contCuad] = [op,None,None,asigna]

contCuad+=1

## sacamos el numero de cuadruplo para el GOTOF

## de la expresion y lo rellenamos con el siguiente

## cuadruplo porque ahi ya termino el for

falso = pSaltos.pop()

cuadruplos[falso][3] = contCuad

pass

## If Else

##################################################################

## if ##

### regla responsable de encargarse del estatuto if en el que ##

### se especifica la sintaxis del mismo estatuto y manda a ##

### llamar las reglas correspondientes para hacer un estatuto ##

### de control logico ##

##################################################################

def p\_if(p):

'if : IF "(" expresion ")" neur13 bloque l ";" neur15'

pass

##################################################################

## punto neuralgico 13 ##

### utilizado para meter las acciones correspondientes de los ##

### brincos por el estatuto logico en caso de que sea falso ##

### todo dependiendo del resultado de la expresion del estatuto ##

### de control ##

##################################################################

def p\_neur13(p):

'neur13 : '

## nos traemos las variables necesarias

global contCuad

## checamos si la expresion nos dejo un tipo bool al final

if pTipos[-1] == BOOL:

## sacamos de la pila de tipos, agarramos el opdoIzq

## que es el resultdado de la expresion anterior que esta

## en la pilaO

pTipos.pop()

opdoIzq = pilaO.pop()

## hacemos un goto en falso pero en el que ponemos el

## resultado de la expresion como lo que se va a evaluar

## todavia sin saber a donde brincaremos

op = GOTOF

cuadruplos[contCuad] = [op,opdoIzq,None,None]

contCuad+=1

## agregamos a la pila de saltos el cuadruplo del GOTOF

## para saber a donde brincar en caso de ser falso

pSaltos.append(contCuad-1)

## si el resultado de la expresion no es un booleano hay error

else:

print ("Tiene que tener un booleano como resultado de expresion")

exit()

pass

###################################################################

## Regla sintactica l para poder meter la opcion del else en un ##

## if o que se quede el if sin else ##

###################################################################

def p\_l(p):

'''l : empty

| ELSE neur14 bloque'''

pass

##################################################################

## punto neuralgico 14 ##

### utilizado para hacer un salto en caso de que haya else ##

### se utiliza porque si hay un else y entro al verdadero se ##

### tiene que saltar las instrucciones del else y al igual es ##

### el brinco que se hace para el GOTOF del if ##

##################################################################

def p\_neur14(p):

'neur14 : '

## nos traemos las variables necesarias

global contCuad

## hacemos la operacion del goto y sacamos

## el brinco en falso que traiamos del if en la pila

## de saltos

op = GOTO

falso = pSaltos.pop()

## generamos el cuadruplo de goto pero sin saber a donde

## brincaremos porque no sabemos cuando se acaba el else

cuadruplos[contCuad] = [op,None,None,None]

contCuad += 1

## agregamos a la pila de saltos el brinco del else que

## esta pendiente por resolverse

pSaltos.append(contCuad-1)

## rellenamos el cuadruplo del GOTOF que traia el if

## ya que ya sabemos a donde va a brincar, justo despues

## del GOTO generado por el else para que sepa

## que si es falso va directo al bloque del else

cuadruplos[falso][3] = contCuad

pass

########################################################################

## punto neuralgico 15 ##

### utilizado para cerrar el estatuto del if y generar ##

### las acciones correspondientes semanticas, aqui lo importante ##

### es que la pila de saltos ya se trabajo bien en los otros ##

### puntos del if ya que no importa si fue if else o if solo ##

########################################################################

def p\_neur15(p):

'neur15 : '

## nos traemos las variables que necesitamos

global contCuad

## sacamos de la pila de salto el pendiente del GOTO del

## else o el pendiente del GOTOF si es un if sin else

falso = pSaltos.pop()

## rellenamos el cuadruplo correspondiente con el numero

## de cuadruplo actual

cuadruplos[falso][3] = contCuad

pass

## Asignación Vars Dinámicas

######################################################################

## asignacion ##

### estatuto de menor jerarquia en las operaciones aritmeticas, ##

### se utiliza para asignar un valor a una variable de lo que sale ##

### de una expresion ##

######################################################################

def p\_asignacion(p):

'asignacion : ID asiglista "=" neur8 expresion ";"'

## nos traemos las variables necesarias

global contCuad

global dir\_procs

global scope

## revisamos si la operacion pendiente es el de asignacion

if pOper[-1] == EQ:

## sacamos lo necesario de las pilas para el cuadruplo

op = pOper.pop()

opdoIzq = pilaO.pop()

tipoIzq = pTipos.pop()

igualdad = pilaO.pop()

tipoIgualdad = pTipos.pop()

## revisamos los tipos en el cubo semantico

if tipoIzq in cubo\_semantico[tipoIgualdad] and op in cubo\_semantico[tipoIgualdad][tipoIzq]:

tipoRes = cubo\_semantico[tipoIgualdad][tipoIzq][op]

## revisamos los tipos en el cubo semantico inversamente tambien

else:

tipoRes = cubo\_semantico[tipoIzq][tipoIgualdad][op]

## si el tipo resultante en el cubo semantico no es error

if tipoRes != ERR :

## genera el cuadruplo

cuadruplos[contCuad] = [op,opdoIzq,None,igualdad]

contCuad+=1

else:

print("Type mismatch")

exit()

## si nos quedamos con operaciones pendientes es una asignacion mal terminada

else:

print("Asignacion mal terminada")

exit()

pass

#######################################################################

## punto neuralgico 8 ##

### usado para validar que la variable de asignacion existe ##

### y la agregamos a la pilaO y la pila de tipos y agregamos ##

### el operando tambien ##

#######################################################################

def p\_neur8(p):

'neur8 : '

## utilizamos el diccionario adicional para apuntar a las variables

auxDic = dir\_procs[scope[-1]][pos\_dics\_var]

## si no es una lista lo que estamos agregando

if p[-2] == -1:

## revisamos si existe una variable con ese id

if p[-3] in auxDic:

## lo agregamos a la pilaO y a la pila de tipos

pilaO.append(auxDic[p[-3]][pos\_vars\_dir\_virtual])

pTipos.append(auxDic[p[-3]][0])

elif p[-3] in dir\_procs['global'][pos\_dics\_var]:

pilaO.append(dir\_procs['global'][pos\_dics\_var][p[-3]][pos\_vars\_dir\_virtual])

pTipos.append(dir\_procs['global'][pos\_dics\_var][p[-3]][0])

## si no existe es un error

else:

print "No existe tal variable a asignar"

exit()

## agregamos la operacion a la pOper tambien

pOper.append(EQ)

pass

#############################################################################

## asiglista ##

### utilizado para revisar si se quiere asignar un elemento ##

### de tipo lista o no, si es empty entonces se regresa -1, ##

### sino pues se regresa 1 y se realizan distintas acciones semanticas ##

### en la asignacion ##

#############################################################################

def p\_asiglista(p):

'''asiglista : empty

| accesoVarDim'''

## en caso de que no haya asignacion de lista

## regresamos -1

if p[1] == None:

p[0] = -1

## en caso de que si haya regresamos 1 y nos

## vamos a la regla accesoVarDim

else:

p[0] = 1

pass

##############################################################

## accesoVarDim ##

### utilizado para accesar el valor de una cassilla de ##

### la variable dimensionada para esto se puede utilizar ##

### expresion numerica entera para ingresar el indice ##

### de la casilla a la que se quiere acceder ##

##############################################################

def p\_accesoVarDim(p):

'accesoVarDim : "[" neur27 nexp "]"'

## regresamos que hicimos una operacion de lista

p[0] = 1

## nos traemos las variables necesarias

global contCuad

global var\_loc\_temp\_int

global var\_loc\_temp\_int

global var\_loc\_temp\_float

global var\_loc\_temp\_float\_inicio

global var\_loc\_temp\_bool

global var\_loc\_temp\_bool\_inicio

global var\_loc\_temp\_char

global var\_loc\_temp\_char\_inicio

global cte\_int\_inicio

global cte\_int

global cte\_float\_inicio

## revisamos si la variable existe en el diccionario de variables

if p[-1] in dir\_procs[scope[-1]][pos\_dics\_var]:

## revisamos que sea una variable dimensionada por su campo de dimension

## en el diccionario

if dir\_procs[scope[-1]][pos\_dics\_var][p[-1]][pos\_vars\_dim] != None:

## revisamos que sea un entero el resultado de la expresion

## de acceso al indice

if pTipos[-1] == INT:

## hacemos la operacion para verificar que este

## dentro del rango de la dimension y que lo revise en

## ejecucion y para eso hacemos este cuadruplo

op = VERIFICA

opdoIzq = pilaO[-1]

opdoDer = dir\_procs[scope[-1]][pos\_dics\_var][p[-1]][pos\_vars\_dim][1]

res = dir\_procs[scope[-1]][pos\_dics\_var][p[-1]][pos\_vars\_dim][0]

cuadruplos[contCuad] = [op,opdoIzq,opdoDer,res]

contCuad += 1

## vamos a hacer la operacion de suma del resultado de la operacion

## interna con la direccion virtual base de la variable dimensionada

## opdoIzq es el resultado de la operacion y el opdoDer es la direccion

## base de la variable convertida en direccion virtual de constante

op = PLUS

opdoIzq = pilaO.pop()

pTipos.pop()

## aqui saco la direccion base con la direccion virtual

aux = dir\_procs[scope[-1]][pos\_dics\_var][p[-1]][pos\_vars\_dir\_virtual]

## aqui lo agregue a las constantes si no estaba

if (not aux in ctes):

if cte\_int + 1 < cte\_float\_inicio:

ctes[aux] = cte\_int

cte\_int += 1

else:

print "Overflow de constantes enteras"

exit()

## aqui ya definimos el opdoDer para el cuadruplo

opdoDer = ctes[aux]

dir\_procs[scope[-1]][pos\_dics\_tam]['ti']+=1

if var\_loc\_temp\_int + 1 < var\_loc\_temp\_float\_inicio:

## se genera el cuadruplo

cuadruplos[contCuad] = [op,opdoIzq,opdoDer,var\_loc\_temp\_int]

contCuad += 1

## aqui agrego la direccion representada como meter el numero

## de la direccion virtual entre parentesis para que de esta manera

## lo trate como apuntador la maquina virtual y de esta manera

## encuentra el valor de la casilla

pilaO.append("("+str(var\_loc\_temp\_int)+")")

var\_loc\_temp\_int += 1

else:

print "Overflow de temporales enteras"

exit()

## aqui agrego el tipo de la variable dimensionada

pTipos.append(dir\_procs[scope[-1]][pos\_dics\_var][p[-1]][pos\_vars\_tipo])

## aqui saco el fondo falso que se mete cuando se manda a llamar la

## expresion interna para saber el desplazamiento que se quiere hacer

if pOper[-1] == '[':

pOper.pop()

## si no es entera la expresion interna entonces es un error

else:

print "El indice que tratas de acccesar no es de tipo entero"

exit()

## si la variable no es dimensionada entonces es un error

else:

print "No es una variable dimensionada"

exit()

## hacemos lo mismo para ver si existe como global

elif p[-1] in dir\_procs['global'][pos\_dics\_var]:

## revisamos que sea una variable dimensionada por su campo de dimension

## en el diccionario

if dir\_procs['global'][pos\_dics\_var][p[-1]][pos\_vars\_dim] != None:

## revisamos que sea un entero el resultado de la expresion

## de acceso al indice

if pTipos[-1] == INT:

## hacemos la operacion para verificar que este

## dentro del rango de la dimension y que lo revise en

## ejecucion y para eso hacemos este cuadruplo

op = VERIFICA

opdoIzq = pilaO[-1]

opdoDer = dir\_procs['global'][pos\_dics\_var][p[-1]][pos\_vars\_dim][1]

res = dir\_procs['global'][pos\_dics\_var][p[-1]][pos\_vars\_dim][0]

cuadruplos[contCuad] = [op,opdoIzq,opdoDer,res]

contCuad += 1

## vamos a hacer la operacion de suma del resultado de la operacion

## interna con la direccion virtual base de la variable dimensionada

## opdoIzq es el resultado de la operacion y el opdoDer es la direccion

## base de la variable convertida en direccion virtual de constante

op = PLUS

opdoIzq = pilaO.pop()

pTipos.pop()

## aqui saco la direccion base con la direccion virtual

aux = dir\_procs['global'][pos\_dics\_var][p[-1]][pos\_vars\_dir\_virtual]

## aqui lo agregue a las constantes si no estaba

if (not aux in ctes):

if cte\_int + 1 < cte\_float\_inicio:

ctes[aux] = cte\_int

cte\_int += 1

else:

print "Overflow de constantes enteras"

exit()

## aqui ya definimos el opdoDer para el cuadruplo

opdoDer = ctes[aux]

dir\_procs[scope[-1]][pos\_dics\_tam]['ti']+=1

if var\_loc\_temp\_int + 1 < var\_loc\_temp\_float\_inicio:

## se genera el cuadruplo

cuadruplos[contCuad] = [op,opdoIzq,opdoDer,var\_loc\_temp\_int]

contCuad += 1

## aqui agrego la direccion representada como meter el numero

## de la direccion virtual entre parentesis para que de esta manera

## lo trate como apuntador la maquina virtual y de esta manera

## encuentra el valor de la casilla

pilaO.append("("+str(var\_loc\_temp\_int)+")")

var\_loc\_temp\_int += 1

else:

print "Overflow de temporales enteras"

exit()

## aqui agrego el tipo de la variable dimensionada

pTipos.append(dir\_procs['global'][pos\_dics\_var][p[-1]][pos\_vars\_tipo])

## aqui saco el fondo falso que se mete cuando se manda a llamar la

## expresion interna para saber el desplazamiento que se quiere hacer

if pOper[-1] == '[':

pOper.pop()

## si no es entera la expresion interna entonces es un error

else:

print "El indice que tratas de acccesar no es de tipo entero"

exit()

## si la variable no es dimensionada entonces es un error

else:

print "No es una variable dimensionada"

exit()

## si no existe la variable es un error

else:

print "No existe tal variable dimensionada "+repr(p[-1])

exit()

pass

##################################################################

## punto neuralgico 27 ##

### se utiliza para meter un fondo falso al momento de buscar ##

### el indice que se desea acceder mediante una expresion ##

### asi dejamos en pausa todas las operaciones pendientes que ##

### teniamos en la expresion que tiene el acceso a la variable ##

##################################################################

def p\_neur27(p):

'neur27 : '

pOper.append('[')

pass