

TLACUANI



César Alejandro Cisneros Rodriguez

A01152861

índice

Descripción de Proyecto	2
Reflexiones	3
César Cisneros	3
Lista Commits	4
Bitácora	5
Mensajes de sistema	6
Lexema	10
Palabras Reservadas	10
Tokens	10
Patrones para construcción	10
Sintaxis	11
DECLARACIÓN DE VARIABLES	11
ARREGLOS	11
DECLARACIÓN DE FUNCIONES	11
DECLARACIÓN DE PARAMETROS PARA FUNCIONES	11
FUNCIÓN MAIN	12
BLOQUES	12
ESTATUTOS	12
LLAMADAS DE FUNCIONES	12
CONDICIONALES	12
CICLO WHILE	12
ESCRITURA	13
LECTURA	13
EXPRESIONES: ASIGNACIÓN	13
EXPRESIONES	13
Generación de Código intermedio	14
Tipos de direcciones Virtuales	14
Diagramas de sintaxis	15
Construcción del Lenguaje:	27
Administración de Memoria	29
Máquina Virtual	30
Código ejemplo y ejecución:	31

Descripción de Proyecto

El Proyecto está principalmente enfocado a la parte educativa con el mínimo de conocimiento sobre la materia. Para este proyecto se utiliza lo que se sabe como funcionamientos básicos de programación.

Este lenguaje de programación está basado puramente en Python y maneja los tipos de argumento Integrales, Flotantes, Caracteres, Strings y Booleanos los cuales llamaremos en su interior como *INT*, *FLOAT*, *CHAR*, *String* y *Bool* respectivamente.

A pesar de que este sistema maneje elementos tipo *BOOL*, este no se permite ser utilizado como para almacenar en una variable o escrito sino más bien es utilizado como temporales a las respuestas de las operaciones como tal.

La forma de escritura está basado entre el Script de R, junto con el desarrollo de C++. Algo similar a lo que se nos enseñó en la clase de Compiladores para desarrollar este proyecto.

Este proyecto está hecho en base de Python con las librerías de *PLY*, *JSON*, *Math* y *Collections.abc* para hacer algunas funciones dentro del programa. Estas librerías vienen incluidas con Python por lo único que es necesario es instalar *PLY* para su ejecución.

Para su funcionamiento solo es de correr el archivo *Compile.py* en un IDE y poner un archivo en la carpeta de */testing/* del mismo directorio.

En este documento se empezará a describir las funcionalidades, los argumentos como el desarrollo del código y ejemplos de este explicando su funcionamiento para trabajar en este entorno.

Video de funcionamiento:

<https://youtu.be/YTDXA0YIAXw>

Liga a Github:

<https://github.com/Cealciro/Tlacuani>

Reflexiones

Este proyecto fue uno de los peores trabajos que pude haber tenido en lo que respecta mi periodo de vida. Me hace sentir estúpido, idiota, que no entiendo lo que está pasando y como es que llega a ser molesto el por más que intente estudiar la información no me embona por más que lo intente y llegar a tener dudas con respecto a mis decisiones de vida. Tuve que pedir ayuda a muchas personas durante todo este periodo para poder llevar a cabo este proyecto y jamás había sentido felicidad por que saliera un misero 3 por una suma que hice. Perdí muchas salidas, conferencias, trabajo, talleres, comunicación dentro del hogar todo por este trabajo. Al final de cuentas ver que funcione hace que uno salte de felicidad por ello por lo que siento algo pleno que pude hacer este trabajo. Pero preferiría no volver a pasar por este Espero que las desveladas hayan pagado sus frutos como el también el estrés que hice pasar como también la mala alimentación por estar demasiado tiempo detrás del monitor sin poder hacer otra cosa más que este proyecto.

Con más ansias espero ver la posibilidad de seguir estudiando o simplemente ir a trabajar en el área de videojuegos para una empresa o por mi cuenta. En si no veo que fue una pérdida de tiempo el desarrollar este proyecto ya que tengo un entendimiento completamente diferente para poder hacer otros tipos de procesos como también la facilidad para desarrollar en otros espacios de desarrollo con más facilidad para otras áreas.

A stylized, handwritten signature in black ink, consisting of a large loop and a series of intersecting lines.

César Cisneros

Lista Commits

Commits on Nov 22, 2022		
Update README.txt	Cealciro committed 3 minutes ago	Verified 88b6c9a <>
Merge branch 'main' of https://github.com/Cealciro/Tlacuani	Cealciro committed 11 minutes ago	6790c5f <>
Final Commit	Cealciro committed 12 minutes ago	b0feb15 <>
Update README.txt	Cealciro committed 22 hours ago	Verified d647242 <>
Please god help	Cealciro committed 23 hours ago	0058799 <>
Commits on Nov 21, 2022		
Borderline Suicidal	Cealciro committed 2 days ago	2168b66 <>
Commits on Nov 20, 2022		
Niggers	Cealciro committed 3 days ago	a0d1815 <>

Commits on Nov 11, 2022		
Corrección parser	Cealciro committed 11 days ago	57b1d36 <>
Cambio estructuras	Cealciro committed 12 days ago	bb57bbc <>
Commits on Nov 8, 2022		
Cambios Funciones parser y lexer	Cealciro committed 14 days ago	0cf6a68 <>
Commits on Nov 4, 2022		
cambios menores	Cealciro committed 19 days ago	9d7bdeb <>
Commits on Nov 2, 2022		
Cambios Nov 2	Cealciro committed 20 days ago	c43de1a <>
pasando el trabajo a git lmao	Cealciro committed 20 days ago	c534de8 <>
Initial commit	Cealciro committed 20 days ago	6aa4156 <>

Commits on Nov 19, 2022		
Cambios a funciones	Cealciro committed 4 days ago	9e5fd03 <>
Commits on Nov 18, 2022		
Ciclos	Cealciro committed 5 days ago	cdb3774 <>
Commits on Nov 16, 2022		
Cambios print	Cealciro committed 6 days ago	a7f7c98 <>
Check memory	Cealciro committed 7 days ago	901b17a <>
Commits on Nov 15, 2022		
cambios	Cealciro committed 8 days ago	3404bff <>
Commits on Nov 14, 2022		
cambios	Cealciro committed 9 days ago	654dc1e <>
Commits on Nov 13, 2022		
planificación	Cealciro committed 10 days ago	f999b03 <>

Bitácora

Titulo	Información
Avance 1	Se creo el archivo Lexer y Parser para el compilador El lexer incluye las palabras reservadas y expresiones regulares El Parser hace las reglas gramaticales para el lenguaje
Avance 2	Lexer: Se agrega Bool al Lexer Se cambian las variables True y False del Lexer a un tipo Parser: Se corrigen unos elementos menores de estatutos Se agrega Bool en Type y varcte Se genera la función de Error de Sintaxis SemanticCube: Se crea Cubo semántico para verificar compatibilidad con diferentes argumentos en el código VarTable: Se genera la Tabla de variables sigue en desarrollo Sema: Se genera la semántica, sigue en desarrollo
Avance 3	Parser: Se inicia el apartado de Puntos neurálgicos (Sigue en Desarrollo) Quad: Se crea la instancia para el cuádruplo del compilador Dir: Se genera para crear el directorio del sistema (Sigue en desarrollo) Sema: Se pospone desarrollo debido a estabilidad Se utiliza la misma base para esta entrega
Avance 4	Corrección de Parser y Lexer
Avance 5	Corrección del Parser y elementos del Lexer (Nov 1) -> se genera el Github Se agregaron funciones al Parser (Nov 2): ->Función While Se crea el archivo Compile.py para hacer las pruebas de una forma más eficiente sin usar el archivo del parser y no arruinar la funcionalidad del Parser siguiendo principios de programación Se crea el archivo de Structure.py el cual guarda información para la funcionalidad del programa, dentro del programa se especifica cada uno de las instancias. Se elimina Quad.py y es agregado al archivo Structure.py (Se modifica para funcionalidad básica) Se mueve Vartable.py a Structure.py (Se modifica para funcionalidad básica)
Avance 6	Se agrega las funciones en las gramáticas se cambian las estructuras como los tamaños de las direcciones correcciones funcionalidades del Parser Se agregaron puntos neurálgicos para el sistema Generación de cuádruplos
Avance 7	Puntos neurálgicos para las funciones Creación de la máquina Virtual Pruebas de funcionamiento básicas

Mensajes de sistema

No.	Mensaje	Información
1	"Carácter Ilegal '%s'" % t.value[0]	Error en la lectura de la gramática cuando se ingresa un elemento que no pertenece a un token
2	Exeso de memoria global variables INT	Se excede en la cantidad de memoria para declaraciones Globales
3	Exeso de memoria global variables FLOAT	Se excede en la cantidad de memoria para declaraciones Globales
4	Exeso de memoria global variables CHAR	Se excede en la cantidad de memoria para declaraciones Globales
5	"Variable \"%s\" ya declarada" %curr_var	La variable que se intenta ingresar ya está en declarada
6	Exeso de memoria Local variables INT	Se excede en la cantidad de memoria para declaraciones Locales
7	Exeso de memoria Local variables FLOAT	Se excede en la cantidad de memoria para declaraciones Locales
8	Exeso de memoria Local variables CHAR	Se excede en la cantidad de memoria para declaraciones Locales
9	Error no hay Variables	Esto pasa cuando se agrega un valor nulo como variable
10	Error, los arreglos deben de ser mayores a 1	Se pone cuando el usuario intenta poner un tamaño menor a uno
11	El arreglo pasa a ser variable.	Cuando el arreglo es de tamaño uno pasa a ser una variable
12	La matriz pasa a ser arreglo	Cuando la matriz tiene como tamaño [1][1] pasa a ser una variable
13	"El Nombre \"%s\" ya está reservado" %id	En el caso de crear parámetros para funciones las variables ya declaradas globalmente están protegidas
14	"El módulo \"%s\" ya existe" %id	Evita el nombre de módulos con el mismo nombre
15	"Error, el Modulo \"%s\" no está declarado" %p[-1]	Cuando se hace una llamada revisa si el nombre del módulo existe, de lo contrario manda este error
16	"Error, No hay argumentos, se necesitan: \"%s\" argumentos" %len(fn.val)	Este mensaje sale cuando el módulo necesita argumentos y el usuario no los ingresa
17	"Número de argumentos erróneo. Necesita \"%s\" se ingresaron \"%s\"" %(len(fn.val), len(li))	Cuando el número de argumentos al hacer la llamada y el módulo no concuerdan
18	"error, necesita ser un Booleano y se agregó: \"%s\"" %Stack.Sy[-1]	En los casos de un IF y WHILE como necesitan ser condicionales (True o False) revisa si la expresión tiene esta condición

19	"La variable \"%s\" es un arreglo favor de poner la dirección de memoria" %curr_var.ID	Cuando se hace llamar a un arreglo y no se pone la dirección para albergar la información
20	"Error, la variable \"%s\" no es un arreglo" %curr_var.ID	Cuando se pone una variable con coordenadas de un arreglo o matriz cuando no es ninguno de los anteriores mencionados
21	"Error, la variable \"%s\" se ingresó \"%s\" equivale a un \"%s\" necesita un INT" %(curr_var.ID, si[0], ty)	Cuando se intenta poner una variable o respuesta la cual no es un INT debido a la naturaleza de los arreglos
22	Error, las variables del arreglo solo pueden positivas	Cuando en la expresión sale de respuesta un número negativo
23	"Error, la variable \"%s\" excede las dimensiones, ingreso \"%s\" necesita \"%s\"" %(curr_var.ID, si[0], ro[0]-1)	La expresión sobrepasa el rango de dimensiones
24	"Error, la variable \"%s\" no es una matriz"	Se intenta poner una variable sin dimensiones como si fuera una matriz
25	"Error, la variable \"%s\" excede el límite ingreso \"%s\" tamaño máximo es \"%s\"" %(curr_var.ID, sim[0], ro[1]-1)	En la matriz, alguna de las direcciones sale del rango del sistema
26	"Error variable \"%s\" no declarada" %id	Cuando se ingresa una variable la cual no está declarada
27	Exceso de memoria Constantes INT	Se excede en la cantidad de memoria para las constantes
28	Exceso de memoria Constantes FLOAT	Se excede en la cantidad de memoria para las constantes
29	Exceso de memoria Constantes CHAR	Se excede en la cantidad de memoria para las constantes
30	Exceso de memoria Temporal INT	Se excede la cantidad de valores temporales que se crean en el sistema
31	Exceso de memoria Temporal Float	Se excede la cantidad de valores temporales que se crean en el sistema
32	Exceso de memoria Temporal Char	Se excede la cantidad de valores temporales que se crean en el sistema
33	Exceso de memoria Temporal Bool	Se excede la cantidad de valores temporales que se crean en el sistema
34	"No se puede dar un valor de retorno a esta función: \"%s\"" %ret	Si se pone Return a un módulo el cual no genera una variable de retorno
35	"Error en gramática, se agrego: %s ERROR {}".format(p) % (p.value)	Cuando la gramática se escribe erróneamente

36	"Caracteres inválidos: No corresponden al mismo tipo de elemento \"%s\" y \"%s\" \" % (righty, lefty)	Este caso es cuando se hacen dos tipos de variables con diferentes tipos
37	"Caracteres inválidos: Elementos no son compatibles \"%s\" y \"%s\" \" % (righty, lefty)	Los caracteres de la variable Return no tiene valores
38	"Error, la variable no tiene valor"	En compilación, la variable no tiene un valor específico
39	"Error, exceso de variables temporales"	Cuando se generan más variables temporales durante la compilación
40	"Error: Div 0 "	En compilación, se intenta dividir entre cero

Varios de estos mensajes no simplemente son de error sino también de avisos de que es lo que anda haciendo la maquina para el desarrollo de funciones como cuando a un usuario se le olvida utilizar ciertos tipos de argumentos para que este pueda corregirlos.

Dentro de los errores conocidos dentro del sistema se encuentran dos:

Al momento de poner una variable después de un Return este no lo puede leer y lo considera un carácter ilegal dentro de la gramática.

A la hora de poner una variable dentro de un arreglo este lo considera como un arreglo. Al momento de escribir este documento se conoce que es debido a como es que lee la variable como tal. En el Stack de Dimensiones solo lo ocupamos para buscar a que parte del directorio hay que ir y en la comparativa toma la variable que está dentro como si fuera el arreglo y no la que está fuera de las llaves ([,]).

Lexema

Palabras Reservadas

```
#Escritura general
'PROGRAM' : 'PROGRAM'
'MAIN'    : 'MAIN'
'FUNC'    : 'FUNC'

# Declaración Variables
'VAR'     : 'VAR'
'int'     : 'INT'
'float'   : 'FLOAT'
'char'    : 'CHAR'

# Declaración Funciones
'VOID'    : 'VOID'
'RETURN'  : 'RETURN'
'READ'    : 'READ'

    Lectura
'PRINT'   : 'PRINT'
    escritura

#Estatuto decisión
'IF'      : 'IF'
'THEN'    : 'THEN'
'ELSE'    : 'ELSE'

#Estatutos de repetición
    Condicional
'WHILE'   : 'WHILE'
'DO'      : 'DO'
    No Condicional
'FOR'     : 'FOR'
'TO'      : 'TO'

#Expresiones
'TRUE'    : 'TRUE'
'FALSE'   : 'FALSE'
'NOT'     : 'NOT'

#Funciones “Especiales”
'MEAN'    : 'MEAN'
'MODE'    : 'MODE'
'VARY'    : 'VARY'
'PLOTXY'  : 'PLOTXY'
```

Tokens

```
# Operadores matemáticos
PLUS = r'\+'
MINUS = r'\-'
MULT = r'\*'
DIV = r'\/'
ADD = r'\++'
DEC = r'\--'
ASSIGN = r'\='

# Operadores lógicos
EQUALS = r'\=='
NON_EQUAL = r'\!='
EQUAL_MAJOR = r'\>='
EQUAL_MINOR = r'\<='
LESS_THAN = r'\<'
MORE_THAN = r'\>'
AND = r'&'
OR = r'\|'

# llaves
L_PAR = r'\('
R_PAR = r'\)'
L_SQ = r'\['
R_SQ = r'\]'
L_COR = r'\{'
R_COR = r'\}'

# Puntuaciones
COLON = r'\:'
SEMICOLON = r'\;'
DOT = r'\.'
COMMA = r'\,'
```

Patrones para construcción

```
ID
    r'[a-zA-Z_]\w*'
CTEF
    r'\d+\.(\\d*)?(e(\\+|-)?\\d+)?'
CTEI
    r'\d+'
CTEC
    r'\\'[A-Za-z_]\\''
CTES
    r'"([^\\\\"\\n]+|\\.\\.)*"'
```

Sintaxis

program : PROGRAM ID ; pVars fun mein

DECLARACIÓN DE VARIABLES

pVars : vars
| empty

vars : VAR : vars2

vars2 : vare : type ; vars3

vars3 : vars2
| empty

type : INT
| FLOAT
| CHAR

ARREGLOS

ar : [siz]
| empty

siz : CTEI mat

mat : .. CTEI
| empty

DECLARACIÓN DE FUNCIONES

fun : Fun fuc
| empty

fuc : fun

Fun : FUNC Ftype ID (param) ; pVars bloq

Ftype : type
| VOID

DECLARACIÓN DE PARAMETROS PARA FUNCIONES

param : ID : type xparam
| empty

xparam : , ID : type xparam
| empty

FUNCIÓN MAIN

mein : MAIN () bloq

BLOQUES

bloq : { statbloq }

statbloq : estatuto statbloq
| empty

ESTATUTOS

estatuto : cond
| while
| prin
| assign
| mcall
| read
| ret

LLAMADAS DE FUNCIONES

mcall : Mcall ;

Mcall : ID (cparam)

cparam : expres cparam
| empty

cxparam : , expres cxparam
| empty

CONDICIONALES

cond : IF (expres) THEN bloq condex ;

condex : ELSE bloq
| empty

CICLO WHILE

whil : WHILE (expres) DO bloq

ESCRITURA

prin : PRINT (str ptr) ;

ptr : , str ptr
| empty

str : expres
| CTES

LECTURA

read : READ : variable ;

EXPRESIONES: ASIGNACIÓN

assign : variable = expres ;

variable : ID arr

ARREGLOS PARA LAS VARIABLES

arr : [dim]
| empty

dim : expres mar

mar : DOT expres
| empty

EXPRESIONES

Logicos

expres : contr c

c : con contr c
| empty

con : &
| |

Comparativos

contr : exp cn

cn : cont exp cn
| empty

cont : ==
| >=
| <=
| >
| <
| !=

Suma y Resta

exp : term t

t : opt term t
| empty

opt : PLUS
| MINUS

Multipliación y División

term : fact f

f : fopt fact f
| empty

fopt : *
| /

Operaciones Unarias y parentesis

fact : (expres)
| sign varcte

sign : opt
| Un
| empty

Un : ++
| --

Variables y Constantes

varcte : ID arr
| CTEC
| CTES
| CTEI
| CTEF

RETURN

ret : RETURN r SEMICOLON

r : Mcall
| expres

Generación de Código intermedio

Rangos	Tipos
00000 - 01900	INT Globales
02000 - 03999	FLOAT Globales
04000 - 05999	CHAR/STRING Globales
06000 - 08999	INT Locales
08000 - 09999	FLOAT Locales
10000 - 11999	CHAR/STRING Locales
12000 - 13999	INT Temporales
14000 - 15999	FLOAT Temporales
16000 - 17999	CHAR/STRING Temporales
18000 - 19999	BOOL Temporales
20000 - 21999	INT Constantes
22000 - 23999	FLOAT Constantes
24000 - 25999	CHAR/STRING Constantes

Tipos de direcciones Virtuales

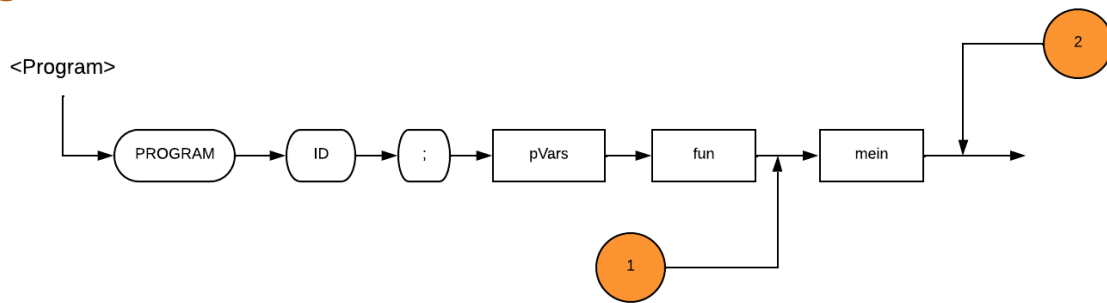
Globales: Son las variables que serán utilizadas en todos los módulos.

Locales: Estas variables solo están habilitadas en el módulo el cual está activo

Temporales: Variables atómicas que se generan mientras se hace la ejecución de operaciones lógicas como respuestas

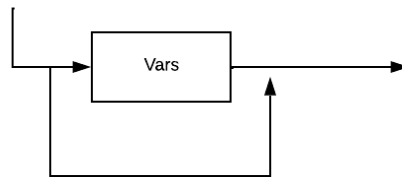
Constantes: son elementos los cuales tienen un valor establecido y no se modifican durante el contexto establecido

Diagramas de sintaxis

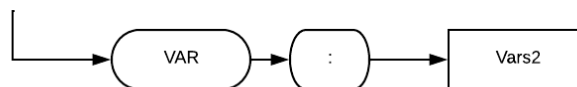


- 1) Se genera el Cuádruplo GoToMain
- 2) Se genera el cuádruplo END

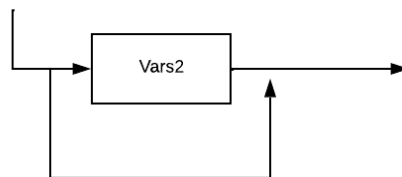
`<pVars>`



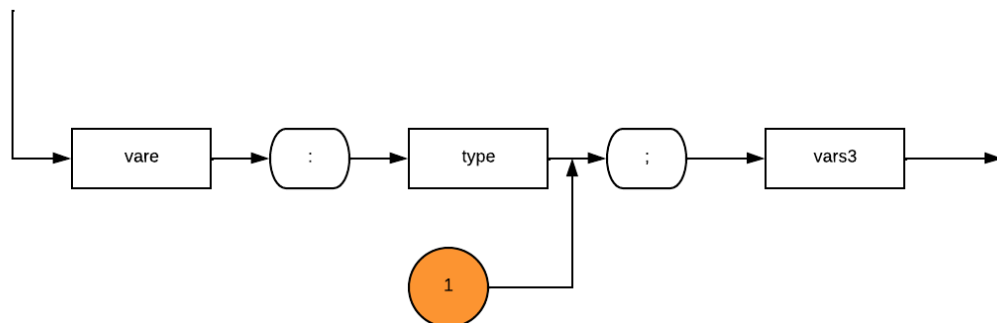
`<vars>`



`<vars3>`

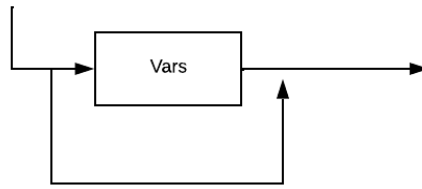


`<vars2>`

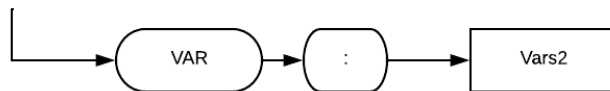


- 1) Se verifica que el ID y el tipo sean correctos y se les da un espacio en el directorio dependiendo de su tipo

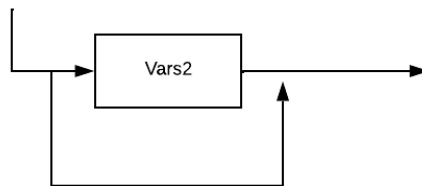
<pVars>



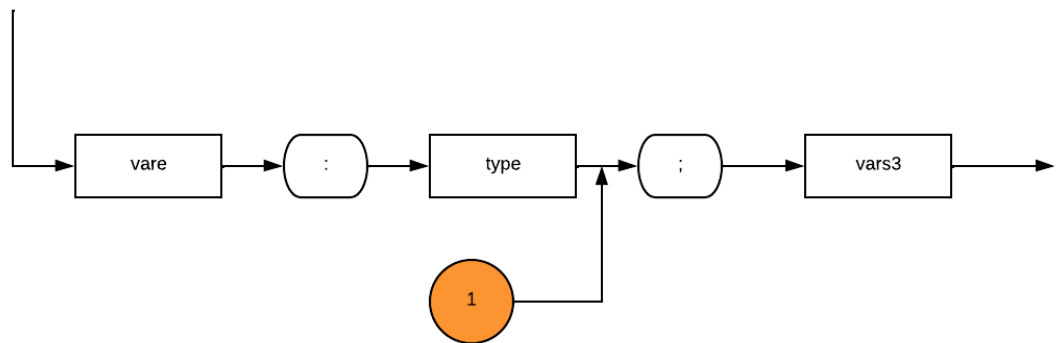
<vars>



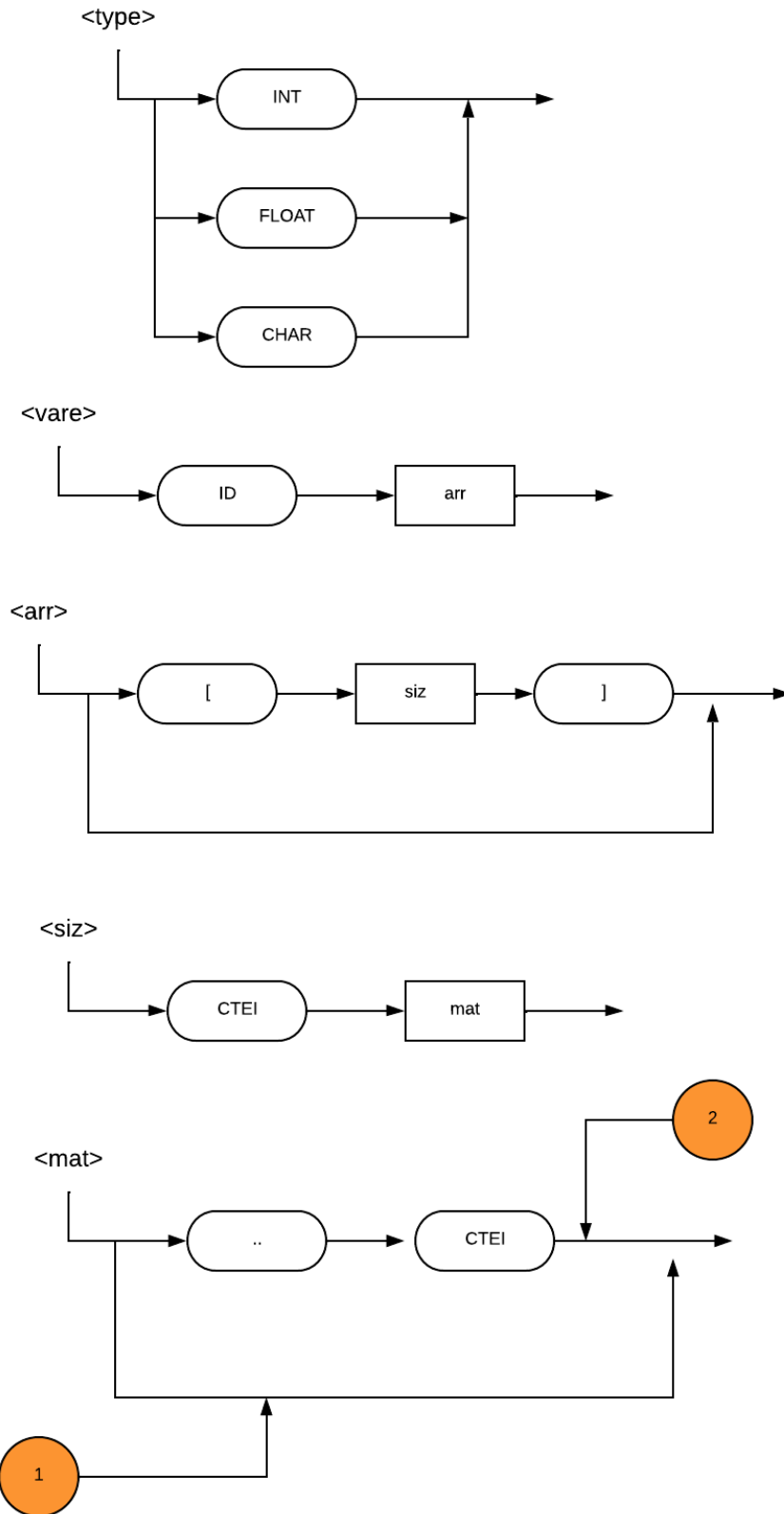
<vars3>



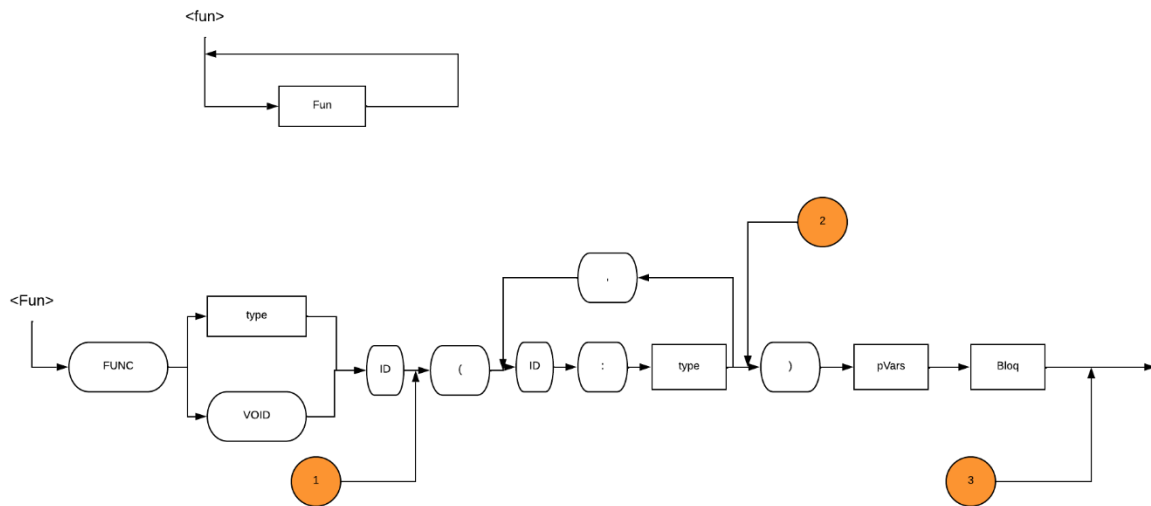
<vars2>



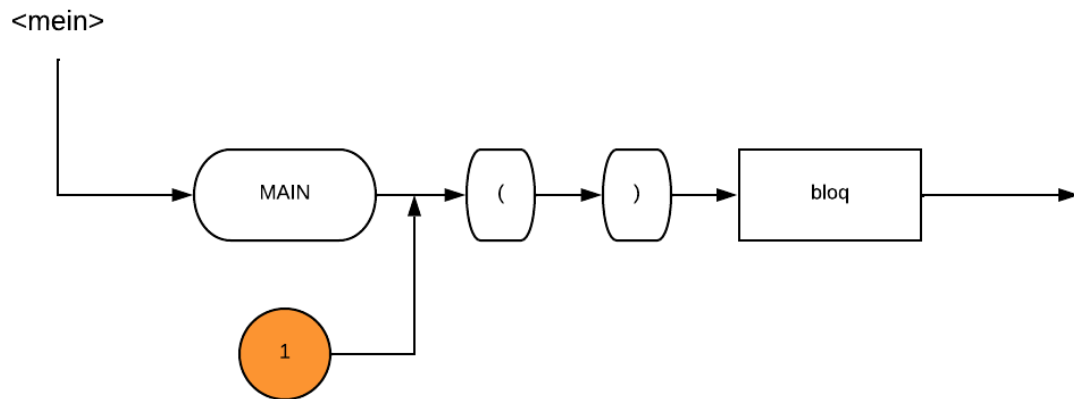
- 1) Revisa si la variable está declarada, en caso de estarlo lo guarda en el directorio, de lo contrario manda mensaje de error



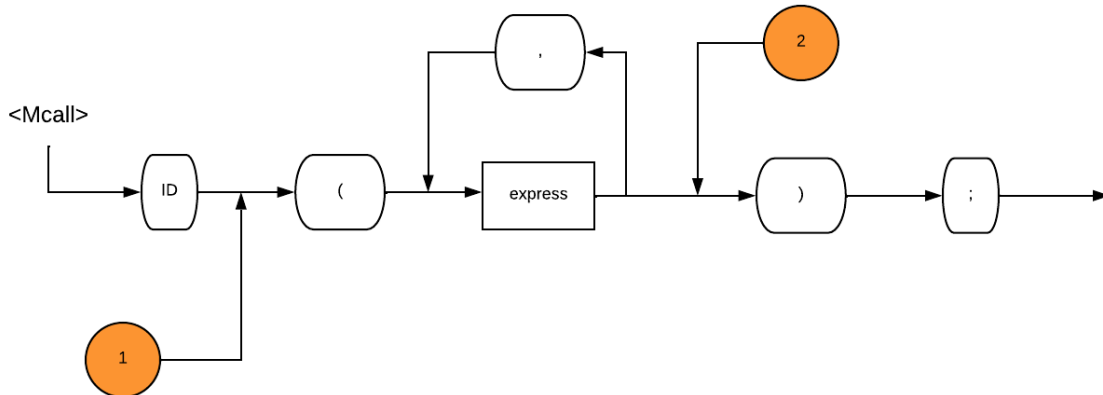
- 1) Revisa que los parámetros sean integrales, también si son arriba de 1, luego alberga todos sus valores en la lista de direcciones
- 2) Lo mismo que el punto dos pero ahora con los dos parámetros



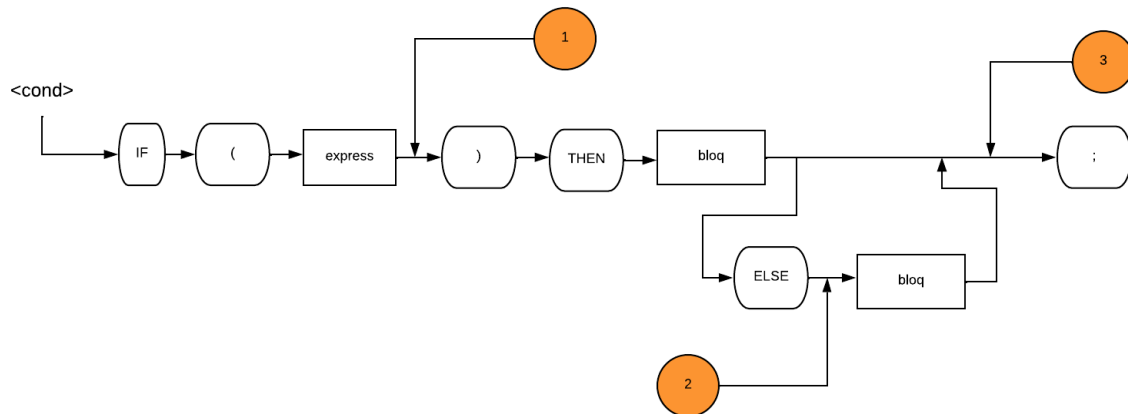
- 1) Revisa que la variable exista y alberga un espacio en la memoria global
- 2) Se verifican que los valores no existan y los alberga en la memoria local con su respectivo directorio
- 3) Se termina la función y se guardan los valores de memoria Se genera el cuádruplo EndFunc



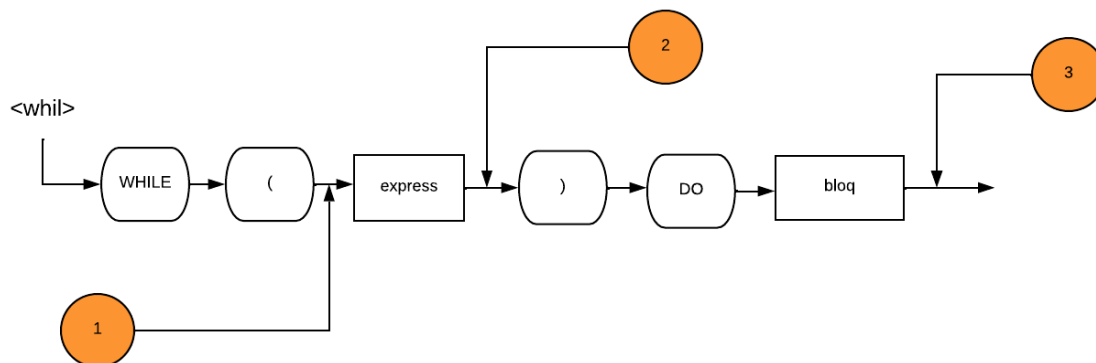
- 1) Se rellena el cuádruplo de GoToMain



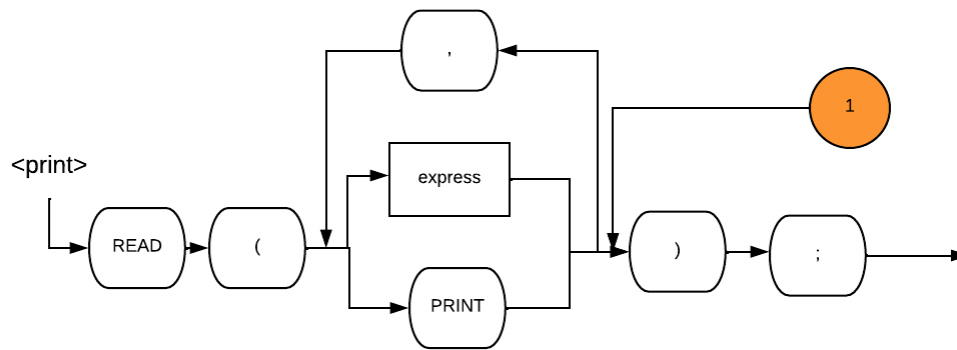
- 1) Se revisa que el ID corresponda a el de un módulo que exista en el programa
- 2) Se genera una lista y se comparan si los parámetros coinciden con el del módulo.



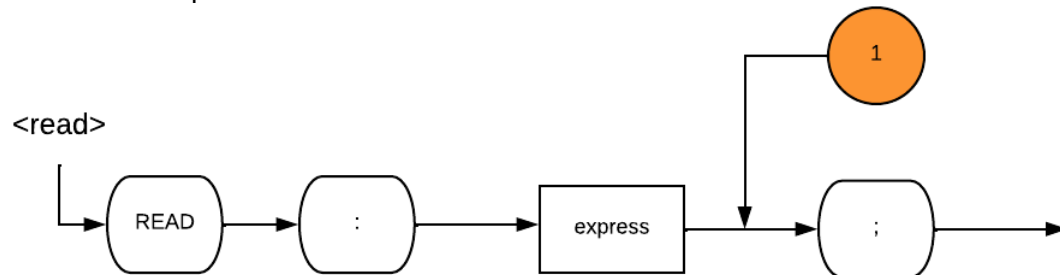
- 1) Se verifica que la expresión tenga un valor bool, luego se genera el cuádruplo con la variable temporal para hacer el salto de bloque
- 2) Está casilla es para rellenar el cuádruplo para cuando se hace el salto en caso de ser verdadero y saltar las instrucciones del Else
- 3) Rellena las casillas 1 y 2



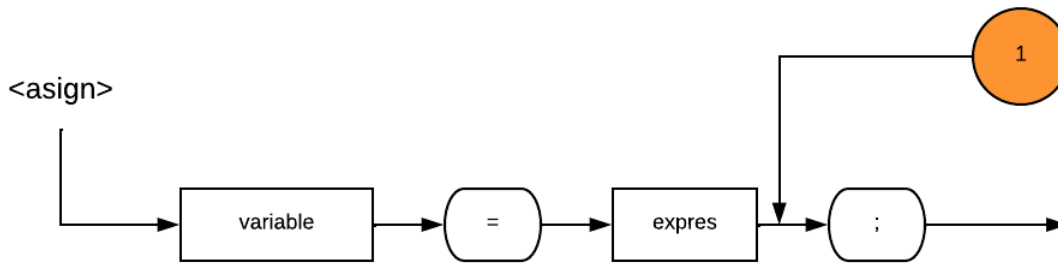
- 1) Se genera el salto para regresar
- 2) Se verifica que la expresión termine como Bool
- 3) Se regresa al punto con la información de la casilla 1



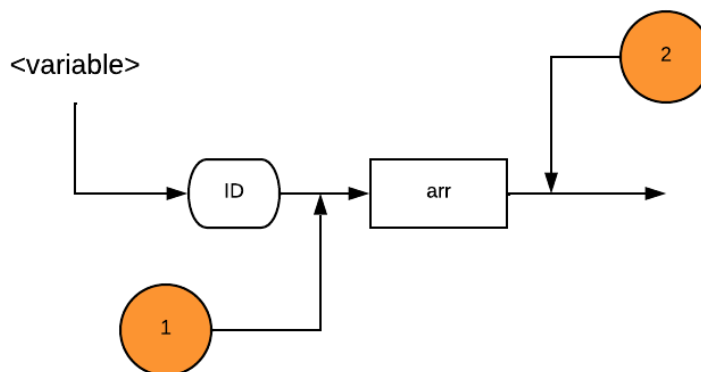
1) Genera cuádruplo con las variables establecidas



1) Genera cuádruplo con la variable para escritura

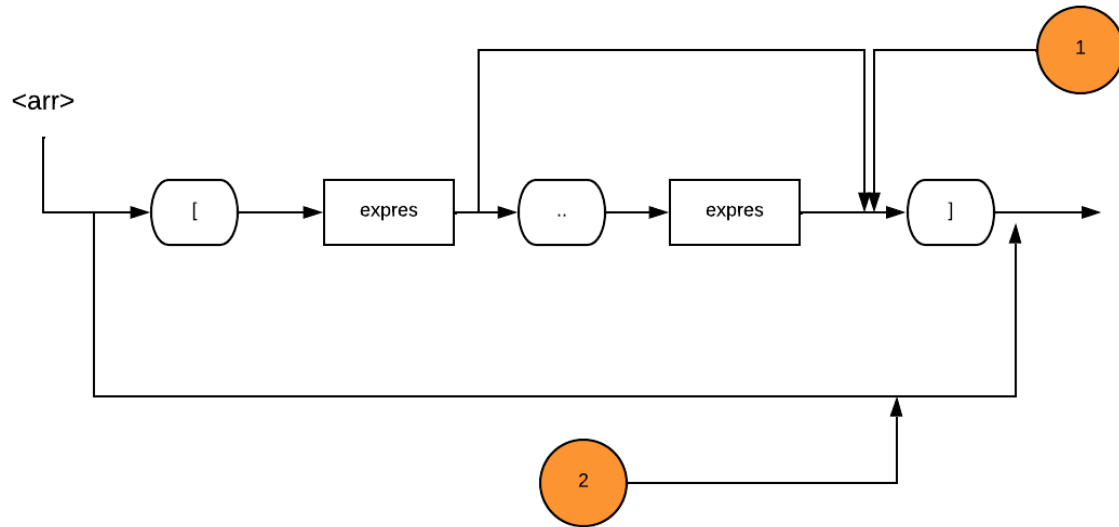


1) Genera cuádruplo para la asignación de valor con la variable

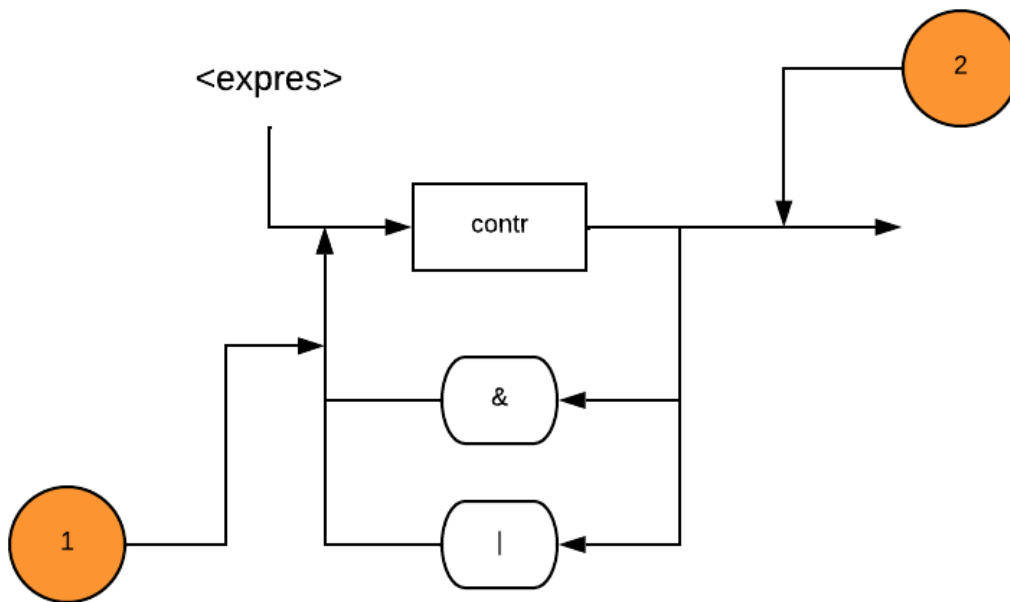


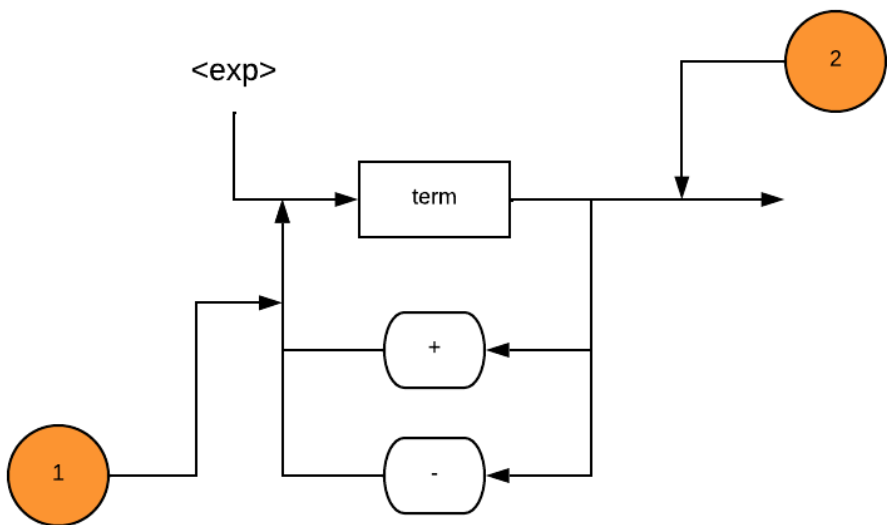
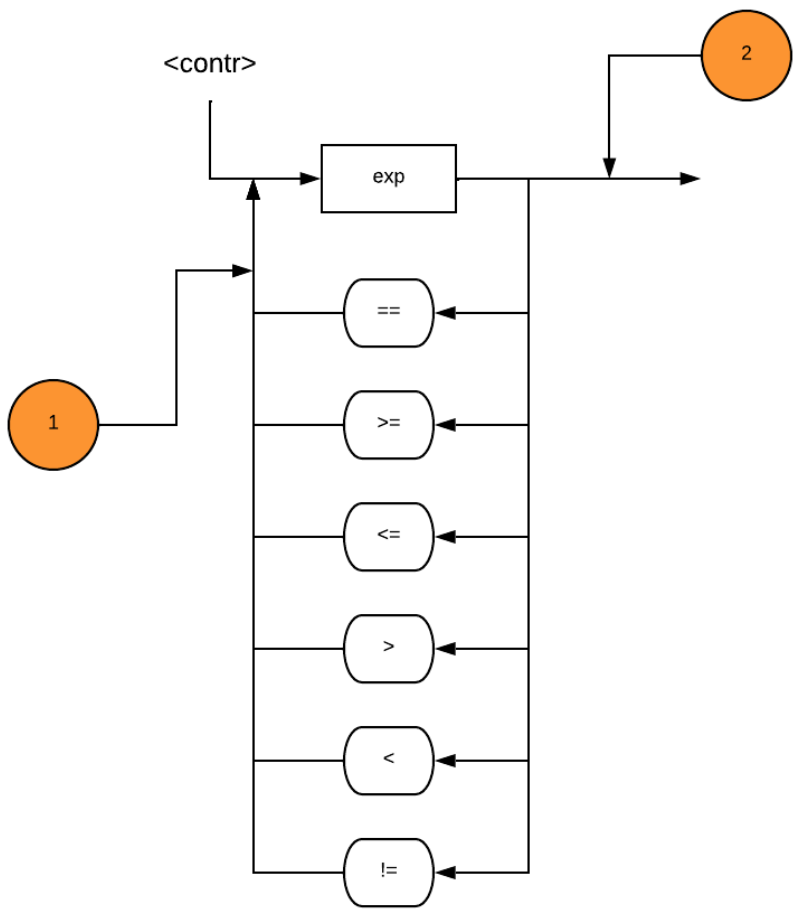
1) Verifica que el ID haya sido declarado

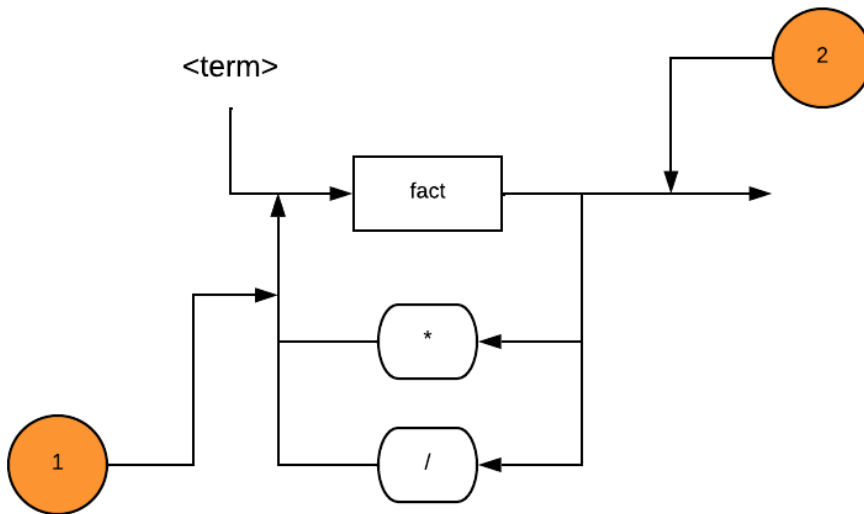
2) Obtiene la dirección de la variable y lo manda al stack



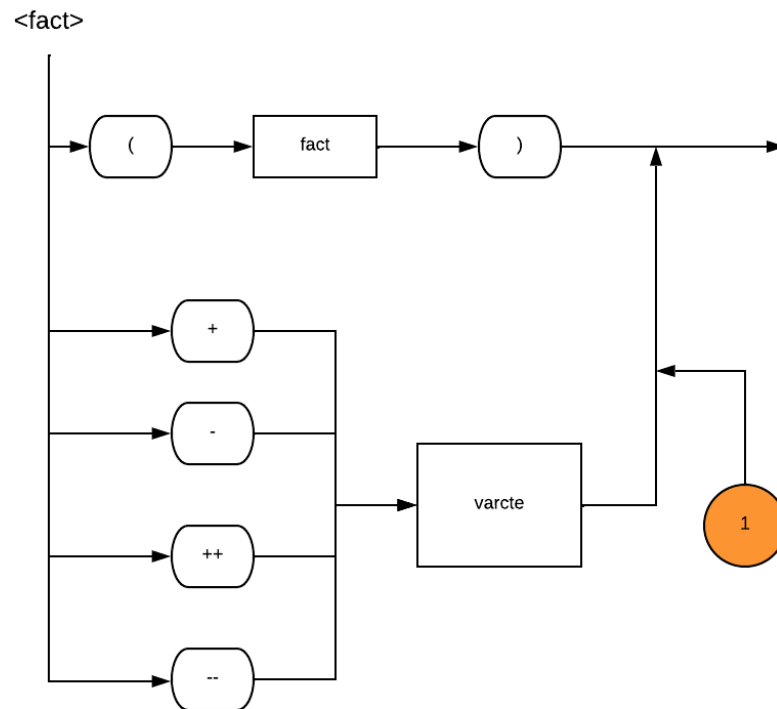
- 1) Verifica que los tamaños del arreglo coincidan con los de la variable
- 2) Verifica si la variable es un arreglo, manda con un rango 0 para los saltos dentro de la dirección



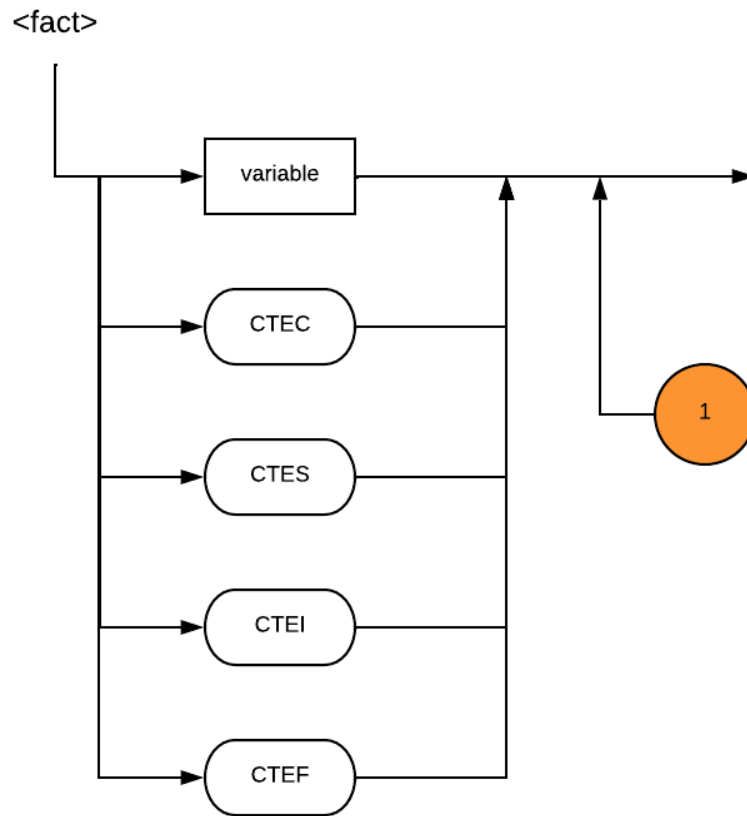




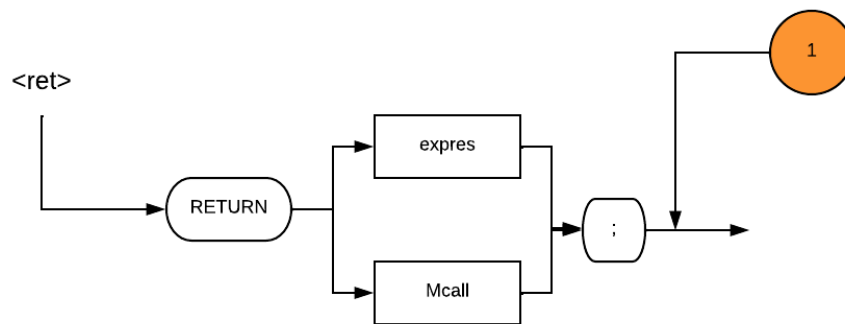
- 1) Agrega el operando en la pila
- 2) Si llega el mismo operando hace una comparación en el cubo semántico para ver si es posible hacer la operación



- 1) Genera la operación unaria, toma el operando y genera un cuádruplo con una operación con 1 de constante.
`++` => verifica si puede hacer una suma ($x + 1$) con la variable
`--` => verifica si puede hacer una resta ($x - 1$) con la variable
`+` => verifica si puede hacer una multiplicación ($x * 1$) con la variable
`-` => verifica si puede hacer una multiplicación ($x * 1$) con la variable



1) Manda el elemento a la pila de variables junto con su tipo



1) Genera el cuádruplo de retorno con la dirección

Construcción del Lenguaje:

En este apartado se va a describir el como se escribe el lenguaje para un mejor entendimiento de este.

PROGRAM id;

Este nos deja nombrar el programa a lo que le usuario desee como un cabezal para tener en cuenta en o el nombre del programa.

VAR :

Este estatuto es para abrir y hacer las declaraciones de las variables, de lo contrario no pueden usarse en el lenguaje de programación

Id : tipo ;

El ID es el nombre que le podemos poner a la variable, el tipo es cual es el que queremos utilizar para el trabajo, este puede ser INT, FLOAT o CHAR para trabajar en el lenguaje de programación

FUNC tipo id (parámetros) ;

Para las funciones se pueden poder cualquier tipo de retorno como los tipos para las variables (INT, FLOAT, CHAR) más el tipo de función Void el cual no tiene un valor de retorno. El ID es para nombrar el módulo y con este poder hacer la llamada dentro del bloque principal.

Parámetros : (id : tipo)

Para la declaración de parámetros se utiliza la misma forma para la declaración de variables.

MAIN ()

La función donde corre el código principal

**IF (expresión) THEN { bloque }
ELSE { };**

Un estatuto condicional, Este revisa si la expresión que termina es una condición BOOL (Si las condiciones son verdaderas o falsas). Si es verdadero hace las instrucciones del bloque que se encuentran después de THEN, en caso de ser falso pasa a las condiciones que se encuentran en el bloque ELSE. Nota, el ELSE puede ponerse opcional, si no está puesto simplemente termina la expresión.

WHILE (expresión) DO { bloque }

El estatuto de repetición WHILE hace que se repitan las instrucciones del bloque hasta que deje de cumplirse la condición que se encuentra en la expresión. La expresión funciona de la misma manera que el IF

`PRINT (String o Expresión) ;`

En este comando puede mandar a imprimir ejecuciones puestas por el usuario como también mensajes personalizados como lo que es el String.

`READ : id ;`

En este apartado lo que puede hacer es que pueda escribir una entrada personalizada por parte del usuario y está lo guarda en su dirección de memoria

Operaciones y Asignación:

Las operaciones que el lenguaje maneja son las matemáticas simples como suma, resta, multiplicación y división (+, -, *, /) como comparaciones (==, !=, >, <, >=, <=) y ejecución es lógicas como el AND y OR (&, |) y las asignaciones para cualquier tipo de variable (=).

Estas operaciones se manejan por medio de vector polaco y son ejecutadas por medio de FIFO (First In, First Out) con respecto a estas las operaciones se manejan de Izquierda a derecha a excepción de la asignación la cual se va de derecha a izquierda. Como se encuentra en el siguiente ejemplo,

`x: INT; 0`

`y: Float; 1`

`x = 7`

`y = 4 * 5 + (3 + 2 / 5) - (8 / 2)`

`OPER x = 4 * 5 + (3 + 2 / 5) - (8 / 2)`
`PVAR x 7 = y`

`if (y > 2 | x < 10 & y > x)`

`OPER x < 10 & y > x`
`PVAR y 2 > x 10 < y x > 8 |`

Administración de Memoria

El uso de memoria de Tlacuani se maneja por medio de Stacks a la hora de tener en cuenta los módulos de desarrollo. Esto es debido a que es necesario utilizar la memoria para cuando se hace llamadas a los módulos y seguir con el desarrollo de este una vez terminando su proceso. Dentro de la estructura está el de Vobj que es el valor único de cada una de las variables que se utilizan. Este tiene los valores de ID, Tipo, Dirección, Arreglo y valor.

El ID es el nombre que maneja la variable, el tipo como se establece es a que tipo de variable pertenece, la dirección es el espacio que va a utilizar en la maquina virtual, Arreglo es el tamaño de la variable si es un arreglo o una matriz y el valor es una variable temporal cuando se hacen ciertos tipos de cálculos para llevar a cabo durante el parsing y generar así los espacios temporales. Todos estos elementos son guardados dentro de una pila llamada VarTable teniendo estos valores a la mano.

De la misma manera se maneja con respecto a la creación de constantes solo que el cambio de las variables es simplemente con el ID y Dirección. ID es la constante y Dirección es en donde se encuentra ubicado en la memoria

Para la declaración de módulos es con las va con los elementos: ID, Tipo, variables, valores, comienzo y tamaño.

El ID es el nombre del módulo, el tipo es que tipo de módulo es con respecto a la descripción pasada. Variables son los elementos que se utilizan localmente junto con los globales, los valores son los elementos que necesitamos traer de otros módulos para su funcionamiento, Comienzo es en que lugar de la tabla de cuádruplos es que empieza el módulo y el tamaño son todos los elementos que se generaron durante su ejecución, variables, constantes, temporales etc. Este se maneja por medio de un Stack al igual que los anteriores.

El objeto de Pila nos guarda en los elementos que necesitamos para generar nuestras operaciones.

Operando guarda los símbolos de operación, símbolo y tipo guarda los valores que son cada uno de los elementos que se encuentra para la operación dentro del sistema, GoTo es nuestro control para hacer saltos dentro del cuádruplo para cuando necesitamos hacer una llamada o un salto para cuando hay alguna instrucción del sistema como lo que es un IF, While, Else como también ir a los módulos establecidos como el MAIN, o los que son creados por el usuario.

Los cuádruplos son como los enseñados en clase, maneja cuatro elementos. El Operador (Tiene también instrucciones como el GoTo, Return, Parámetro, EndModule, escritura y lectura), Operador izquierda, operador derecho que son las variables que utilizamos para nuestro sistema y el Temporal que es la dirección a donde es que se va a guardar la información que generamos por medio de las operaciones.

Máquina Virtual

Dentro de la maquina virtual no hay algo especial para hacer correr nuestro sistema más que la librería de JSON el cual necesitamos para generar nuestras tablas para utilizarlo en la máquina virtual.

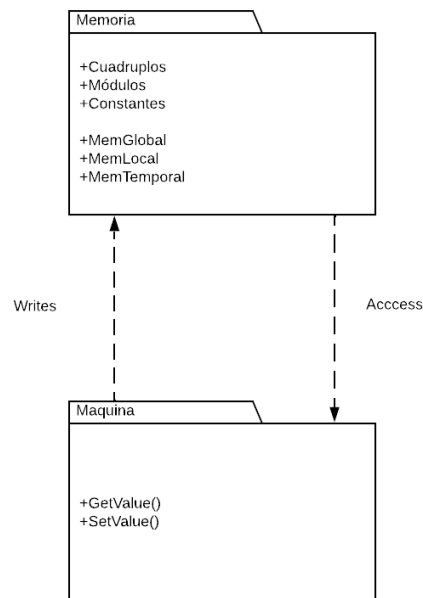
Una vez terminada la compilación pasamos a copiar los elementos de la tabla de Módulos, Constantes y Cuádruplos. Ya que se hizo los cálculos para generar las direcciones de memoria.

Dentro de estos se exporta en Módulos para hacer los cálculos. En este solo necesitamos exportar El Nombre, sus variables y el tamaño del modulo para poder hacer espacio dentro de la máquina virtual.

En Constantes necesitamos el ID y su dirección para poder asignar el valor a cualquiera de los elementos para hacer las operaciones matemáticas dentro de la maquina y por último necesitamos todos los elementos de la lista de cuádruplos. Dentro de la administración de memoria ya hicimos los cálculos y asignaciones para los elementos y las direcciones a donde es que va cada elemento que generamos dentro del sistema.

Una vez dentro de la maquina virtual utilizamos dos funciones, una para Obtener el valor que se encuentra guardado en la dirección y el otro para definir y poner un valor en la dirección asignada. Estas asignaciones de memoria se dan guardándolos en una lista. En cada una de las listas solamente guardamos el Valor y su dirección, algo similar a la tabla de constantes.

Para crear la memoria local simplemente se hace un append a la lista, creando una sublista con los elementos que vamos a estar utilizando durante el uso del módulo. Estas listas solo albergan información y para bajar los tiempos de búsqueda es de lo manejamos por tres listas diferentes. Globales, Locales y Temporales. En la lista Global vamos a tener todas nuestras variables globales, las que declaramos al principio del programa. Las locales con su dirección es donde se guardan las variables que vamos a estar modificando durante su desarrollo y las temporales son todas las que desarrollamos durante su ejecución.



Código ejemplo y ejecución:

```
PROGRAM TestCorrect;
VAR :
    y:int;
    x:int;
    m:int;
    z:float;
    w[2 .. 2]:int;
MAIN()
{
    x = 0;
    m = 0;

    w[0 .. 0]= 0;
    w[0 .. 1]= 1;
    w[1 .. 0]= 2;
    w[1 .. 1]= 3;

    PRINT(w[0 .. 0]);
    PRINT(w[0 .. 1]);
    PRINT(w[1 .. 0]);
    PRINT(w[1 .. 1]);

    y = 26;
    z = ( x + 3 * y)/4;
    IF (z > x) THEN {
        PRINT("hola mundo", z);
    };

    WHILE (x < 9) DO {
        IF (x < 2) THEN {
            WHILE (m < 2) DO {
                m = m + 1;
                PRINT("M =", m);
            }
            m=0;
        };
        PRINT ("X =", x);
        x = x + 1;
        PRINT("test");
    }
}
```

Compilación Completa!

```
0
1
2
3
"hola mundo"
19.5
"M ="
1
"M ="
2
"X ="
0
"test"
"M ="
1
"M ="
2
"X ="
1
"test"
"X ="
2
"test"
"X ="
3
"test"
"X ="
4
"test"
"X ="
5
"test"
"X ="
6
"test"
"X ="
7
"test"
"X ="
8
```

```

PROGRAM Fibbo;
VAR :
    y:int;
    v[5 .. 1]:int;
    x:int;
    z:float;
    w:char;

FUNC VOID fib (a:int) ;
VAR :
    c:int;
    t1:int;
    t2:int;
    t3:int;
{
    t1 = 0;
    t2 = 1;
    IF (a > 0) THEN {
        PRINT(t1);
    };
    IF (a > 1) THEN {
        PRINT(t2);
    };
    t3 = t1 + t2;
    c = 3;
    WHILE (a >= c) DO {
        PRINT(t3);
        t1 = t2;
        t2 = t3;
        t3 = t1 + t2;
        c = 1 + c;
    }
}

MAIN ()
{
    x = 1;
    y = 2;
    w = 'c';
    z = (x+y*3)/4;
    READ : x;
    fib (x);
}

```

```

Test número: 2
Corriendo "Accept Func.sn"

Compilación Completa!
5
0
1
1
2
3
Fin del programa

```