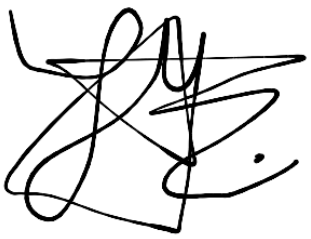
TC3048 Diseño de Compiladores

Lenguaje Luyer

22 de noviembre de 2022

Luis Yerik Arámbula Barrera

A00825080



Descripción del Proyecto

Propósito y Alcance

El propósito de este proyecto es crear un lenguaje de programación simple con enfoque estadístico. Dicho lenguaje contará con las características de los típicos lenguajes de programación, como módulos, arreglos, condicionales, ciclos, etc. Este lenguaje es de tipo estático con un paradigma declarativo, es decir, las variables deberán llevar su tipo correspondiente. Al crear el lenguaje se aprenderán y aplicarán conceptos importantes de las ciencias computacionales.

Análisis de Requerimientos

- Estatutos.
- Expresiones matemáticas, lógicas y relacionales.
- Módulos
- Variables locales y globales
- Elementos estructurados
- Funciones estadísticas

Descripción de los Casos de Prueba

Los casos de prueba se encuentran en archivos "`.ly`" en la carpeta "`pruebas`".

Nombre	*Descripción*
`array_test.ly`	Probar el funcionamiento de arreglos.
`esp_test.ly`	Probar las funciones predefinidas del lenguaje.
`fact_iter.ly`	Probar las operaciones aritméticas y ciclos con el algoritmo factorial iterativo.
`fibo_recursivo.ly`	Probar las funciones y recursividad con el algoritmo para la serie fibonacci.
`mat_mul.ly`	Probar elementos estructurados mediante la multiplicación de matrices.

`sort.ly`	Probar arreglos mediante un algoritmo de bubble sort.
`fibo_iter.ly`	Probar ciclos y funciones mediante el algoritmo fibonacci de forma iterativa.
`find.ly`	Probar arreglos mediante la búsqueda de un elemento en uno.
`fact_recursivo.ly`	Probar las funciones y recursividad con el algoritmo factorial iterativo.

Bitácora semanal

Semana 1

- Se implementó análisis léxico y sintáctico con LARK.

Semana 2

- Se agregó el cubo semántico

Semana 3

- Se agregó semántica inicial para funciones y main, junto con el directorio de funciones y tablas de variables correspondientes
- Se implementaron puntos neurálgicos para operaciones aritméticas de suma, resta, multiplicación y división
- Se creó un objeto para cuádruplos y se empezaron a imprimir los primeros con los puntos anteriores

Semana 4

- Se agregó la semántica para asignaciones, condiciones y ciclo while.
- Los valores de las variables ahora son manejados correctamente por el directorio
- Se agregaron errores comunes como variables duplicadas o valores inválidos
- Se agregaron variables globales

Semana 5

- Se agregó la semántica para funciones
- Se agregó la tabla de párametros
- Se creó la clase Memoria mediante varias estructuras
- Se manejan direcciones de memoria para todas las variables correctamente
- Se guardan los cuádruplos de inicio de cada función y del main
- Se inició la implementación de cuádruplos de funciones, return aun sin funcionar

Semana 6

- Se modificó la sintáxis para el uso de return, funciones void y con tipo
- Se creó la tabla de constantes
- Se agregó la tabla de parámetros
- Los parámetros se verifican, con su tipo y orden
- Las funciones void y return se manejan correctamente
- Se manejan llamadas dentro de llamadas con una pila de llamadas

Semana 7

- Se maneja la declaración de arreglos de n dimensiones utilizando indexado de tipo C
- Se maneja el acceso de arreglos y su uso en operaciones
- La memoria se aparta correctamente para los arreglos
- Se maneja la asignación en arreglos
- Se creó la máquina virtual con una clase
- La máquina virtual lee cuádruplos en el orden correcto
- La máquina virtual realiza operaciones aritméticas y de asignación
- La clase memoria maneja asignaciones de valores correctamente
- La máquina virtual ejecuta estatutos condicionales y ciclos
- La máquina virtual ejecuta funciones correctamente
- La máquina virtual maneja la recursividad correctamente con una pila de IP
- Se agregaron funciones predefinidas
- Se probaron los casos de prueba exitosamente

Lista de Commits

Esta sección fue obtenida utilizando el output del comando `git log` el cual lista todos los commits del más reciente al primero.

```
commit 10e52b0c6ad33d94dd7b8105d3f74daab6bb5c9b
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Tue Nov 22 22:33:52 2022 -0600

    actualizar readme

commit 5035ec7ef01a6e57cea7d964386cc805bc27ecea
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Tue Nov 22 15:34:46 2022 -0600

    pruebas

commit 65dd86299956aca4417e3c0e25baed4b04bf3055
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Nov 21 22:18:50 2022 -0600

    actualizar README

commit d52dce4e802e1a91514bea9c7f4815cebb4c652a
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Nov 21 22:18:11 2022 -0600

    funciones estadisticas

commit 04be918cce66fe49a96f3ee9c97c7cd5796c00d2
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Sun Nov 20 22:25:23 2022 -0600

    ejecución de arreglos
```

```
commit c0726cf57b24845cb67f6f3e4cd4d83cddd4d98e
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Sat Nov 19 20:02:59 2022 -0600

    ejecución de funciones

commit 4700336b51fefa9d32768100b7da45fb58e61c95
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Fri Nov 18 16:10:20 2022 -0600

    maquina virtual

commit d24c10592d6810945a7016e2bc1ff2e8d5b7f65f
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Thu Nov 17 17:25:13 2022 -0600

    actualizar readme

commit b9dcdf409e7ef2d4b2c41c503796901a540c959e
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Thu Nov 17 17:24:30 2022 -0600

    add array assignment

commit ae498b63798c9dd059546b882be5e4ac46c9391b
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Wed Nov 16 20:32:22 2022 -0600

    agregar arreglos de n dimensiones

commit dc2e2e6f3cf577fb21b960ab4f663e8a135f6694
Author: Luis Yerik Arambula <A00825080@itesm.mx>
```

```
Date:   Wed Nov 16 01:33:10 2022 -0600

    sintaxis de arreglos, puntos para declaraciones de arreglos de n
dim

commit 31b5924841be71ef7f52bc5f36551da8dae95b73
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Tue Nov 15 09:44:57 2022 -0600

    manejar llamadas anidadas

commit d01035942c173cffaf5ece6836fdbfb756e97cdb
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Sun Nov 13 18:49:02 2022 -0600

    actualizar readme

commit a94960d507db166aeb4dd32b238c5be82f4f033c
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Sun Nov 13 18:48:15 2022 -0600

    manejo de funciones tipadas y void

commit ef0c16f9b4479102e470e4413f2a2fbf18f22c19
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Nov 7 23:20:50 2022 -0600

    actualizar README

commit 1b86d50fea2a46ec6cd1ade11c229f7054ea9688
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Nov 7 23:17:12 2022 -0600
```

```
    generacion de cuadruplos de funciones

commit 79c31037f9d23d1a2b9e521bcebbe5210417d6b5
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Sun Nov 6 20:50:26 2022 -0600

    agregar clase para memoria y manejar direcciones

commit 3183b49c0f480d323a1d5a48123f4ff238fc3e96
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Fri Nov 4 16:23:57 2022 -0600

    actualizar readme

commit 91492d9c64c598463176953dec572cf18969e5d5
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Fri Nov 4 16:21:34 2022 -0600

    add globals

commit c2214e07f9d969ac6bf381f316dbbae3c7888912
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Oct 31 17:28:26 2022 -0600

    agregar parentesis en operaciones, arreglar ciclos

commit d94898da32eca4c2aadc8b77dbb02a4fec2d6c73
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Sat Oct 29 19:08:59 2022 -0500

    actualizar readme

commit e282ff6d44f58455c2327350900122329ba04c9f
```

```
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Sat Oct 29 18:59:19 2022 -0500

    generacion de cuadruplos con objeto y agregar while

commit 5b0fcd8cd2da74df79eb9c34ad6266bef7c13f4c
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Wed Oct 26 19:47:10 2022 -0500

    agregar asignación

commit 6ef506b431cb18dbebc879837ce6a779489a39b3
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Oct 17 23:32:54 2022 -0500

    actualizar README

commit 7b2b78c89456e42c17c899b433bbd37190e4b0fc
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Oct 17 23:30:13 2022 -0500

    agregar semántica de variables y operaciones básicas

commit e6555ac2508cde3f9d76b5330e1bf1660465a3e6
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Wed Oct 12 19:36:23 2022 -0500

    agregar directorio de funciones y variables

commit 10e616847724acad465bee37e4826beea6a66e86
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Oct 10 22:31:28 2022 -0500
```

```
    actualizar readme

commit f95fc049421bccff5aebbfae215d4ad781f3eadc
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Oct 10 21:17:20 2022 -0500

    agregar cubo

commit 078f2a60851d31e83d6838846dfc2ad70455a032
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Oct 3 16:54:09 2022 -0500

    Actualizar readme

commit 2cf2f36656c5ead47c0b530e716d691fd89b6c8c
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Oct 3 16:51:30 2022 -0500

    lexico y sintaxis

commit c5ae9039976c8a395bdaa8c44ec556a0746eea57
Author: Luis Yerik Arambula <A00825080@itesm.mx>
Date:   Mon Oct 3 15:39:26 2022 -0500

    Initial commit
```

Reflexión

Este proyecto sin lugar a duda ha sido uno de los más pesados que he tenido en la carrera pero también ha sido el más gratificante de realizar. Siempre me ha fascinado la complejidad de la computación hoy en día. Aunque este proyecto no sea un lenguaje sofisticado me ayudó a darme cuenta de la increíble complejidad que hay detrás de la computación moderna. Incluso si no llegamos a ver conceptos más difíciles o funciones de lenguajes más avanzadas creo que este curso es un

buen cierre a todas las materias más “teóricas” de la carrera, como lo son matemáticas computacionales, organización computacional, matemáticas discretas, lenguajes de programación, etc.

Además de comprender cómo es que puede funcionar un lenguaje de programación, las diferentes fases y procesos que permiten que esto pueda ser traducido a algo entendible por una computadora, creo que algo muy valioso que me llevo de este proyecto es la habilidad de debuggear.

A lo largo de todo el proyecto me topé con gran cantidad de errores. Aunque algunos los ayuda resuelto utilizando sólo `print` a medida que el proyecto llegaba a más líneas de código, variables y complejidad en general, resultaba más difícil averiguar qué era lo que estaba mal. De ahí que el uso del debugger fue vital para resolver errores críticos para el funcionamiento del lenguaje.

Descripción del Lenguaje

Luyer es un lenguaje de programación estático y declarativo con funciones estadísticas. En él se pueden desarrollar algoritmos simples con operaciones básicas como suma, resta, división y multiplicación. Además tiene la capacidad de crear diferentes módulos para mantener el código limpio y estructurado. Sus funciones estadísticas se dan sólo en elementos estructurados, los cuales pueden ser del número de dimensiones que se desee, mientras no se exceda de la memoria límite del lenguaje. Estas funciones son características estadísticas como promedio, desviación estándar, máximos y mínimos.

El lenguaje maneja errores típicos de un lenguaje de programación, como:

- Nombres repetidos, por ejemplo, declaración múltiple de variables o funciones
- Una función no puede llamarse global pues habría un conflicto en el directorio de funciones
- Parámetros repetidos
- Returns de tipo incorrecto
- Returns en funciones void

- Llamadas a funciones que no existen
- Usar una llamada a una función void en operaciones
- Cantidad de parámetros de una llamada incorrectos
- Tipos de parámetros incorrectos
- Variables indefinidas o inexistentes
- Operaciones entre tipos inválidos
- Índices inválidos en arreglos
- Inicialización inválida en arreglos
- Acceso incorrecto a arreglos
- Índices fuera de rango

Luyer tiene varias limitaciones, por ejemplo, por el momento no son soportadas las funciones de tipos estructurados y estos tampoco pueden ser utilizados como parámetros de funciones. Además, las llamadas a funciones anidadas no han sido manejadas adecuadamente en ejecución. Como solución al primer problema, recomiendo utilizar elementos estructurados globales, si es que se planean manipular dentro de una función. También funcionan localmente en cualquier función, mientras no sean parámetros de ella.

Descripción del Compilador

Equipo, lenguaje y utilerías

El compilador de Luyer fue programado en Python, utilizando la librería LARK como analizador sintáctico y léxico. Además, la librería NumPy fue utilizada para las funciones estadísticas de este lenguaje. Luyer fue desarrollado en una MacBook Air 2020 con procesador M1 y 8GB de RAM, además, VS Code fue utilizado en todo el desarrollo como editor de texto y debugger.

Descripción del Análisis Léxico

La librería LARK maneja el análisis léxico en el mismo lugar que el sintáctico, simplificando mucho este proceso. Los tokens están definidos encerrando a símbolos con comillas, por ejemplo, para utilizar el token del punto y coma basta con

escribir “;” para que LARK lo identifique. Por lo tanto, se definieron una mínima cantidad de tokens. Estos fueron definidos de la siguiente manera:

```
ID: /[a-zA-Z_][a-zA-Z0-9_]*/
INT: "int"
FLOAT: "float"
STRING: "string"
BOOL : "bool"
TRUE: "true"
FALSE: "false"
RET: "ret"
VOID: "void"
PRINT: "out"
MEAN: "mean"
FILL: "fill"
STD: "std"
MIN: "min"
MAX: "max"
LEN: "len"

%import common.SIGNED_FLOAT
%import common.SIGNED_INT
%import common.NUMBER
%import common.ESCAPED_STRING
%import common.WS
%ignore WS
```

Nótese que LARK también cuenta con tokens predeterminados, por ejemplo `%import common.SIGNED_FLOAT`, importa una expresión regular para floats, desde la librería de LARK.

Descripción del Análisis Sintáctico

La siguiente gramática fue definida para el lenguaje luyer. LARK maneja un formato ligeramente diferente a las gramáticas convencionales, por ejemplo el vacío puede ser interpretado como `regla: otra_regla |` en donde el símbolo | significa que hay otra opción de regla. Debido a que no hay nada después, es interpretado como el epsilon.

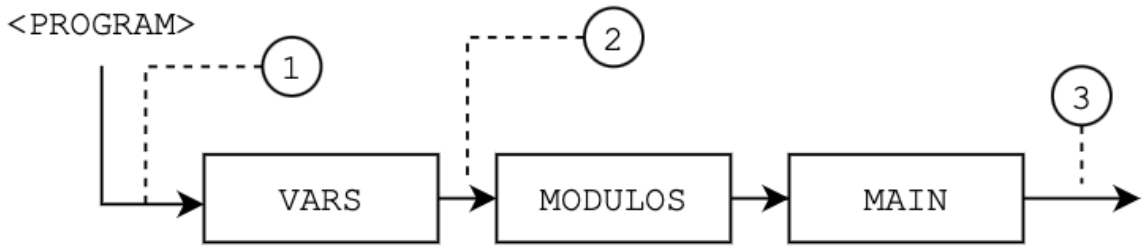
```
start: program
program: globals vars modulos main
modulos: modulos1
modulos1: funcion modulos |

funcion: "func" tipo_func ":" ID "(" func_vars? ")" bloque_ret|
tipo_func: tipo1 | VOID

func_vars: tipo ":" ID func_vars1
func_vars1: "," func_vars |

main: "main" bloque_ret

bloque_ret: "{" vars estatuto1 "}"
bloque: "{" estatuto1 "}"
estatuto1: estatuto2 |
estatuto2: estatuto estatuto1

estatuto: while | condicion | predef | asignacion | asignacion_arr |
return | llamada_void

llamada_void: llamada ";"
predef: out | fill

out: PRINT "(" expresion out1* ")" ";"
out1: "," expresion escritura
```

```
fill: FILL "(" ESCAPED_STRING "," ID ")" ";"

return: RET expresion end_return ";"

llamada: ID "(" parametros? ")"

parametros: expresion parametros2
parametros2: "," parametros |

asignacion: ID "=" expresion ";"
asignacion_arr : array_access "=" expresion ";"

expresion: expresion1
expresion1: expresion2 expresion4
expresion4: simbolo3 expresion1 |
simbolo3: "&&" | "||"

expresion2: exp expresion3
expresion3: simbolo expresion2 |
simbolo: "<" | ">" | "!=" | "<=" | ">=" | "=="

exp: termino exp1
exp1: simbolo1 exp |
simbolo1: "+" | "-"

termino: factor termino1
termino1: simbolo2 termino |
simbolo2: "*" | "/"
factor: "(" expresion ")" | var_cte

condicion: "if" "(" expresion ")" bloque condicion1
condicion1: "else" bloque |
while: "while" "(" expresion ")" bloque
```

```
vars: vars1*
vars1: tipo ":" ID vars2? ";"
     | tipo ":" ID "[" dims "]" ";"
dims: var_cte dims1
dims1: "," dims |
vars2: asg_sign expresion
asg_sign: "="

array_access: ID "[" dims2 "]"
dims2: expresion dims3
dims3: "," dims2 |

var_cte: SIGNED_FLOAT
| SIGNED_INT
| llamada
| TRUE
| FALSE
| ID
| ESCAPED_STRING
| array_access
| MEAN "(" ID ")"
| STD "(" ID ")"
| MAX "(" ID ")"
| MIN "(" ID ")"
| LEN "(" ID ")"

tipo: INT | FLOAT | STRING | BOOL
```

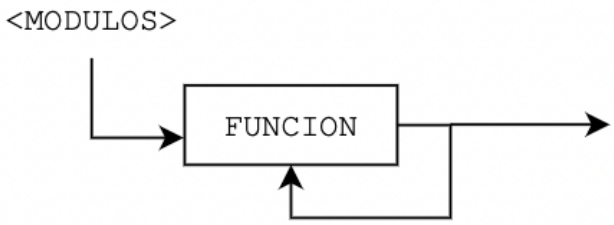
Descripción de Generación de Código Intermedio y Análisis Semántico

Programa

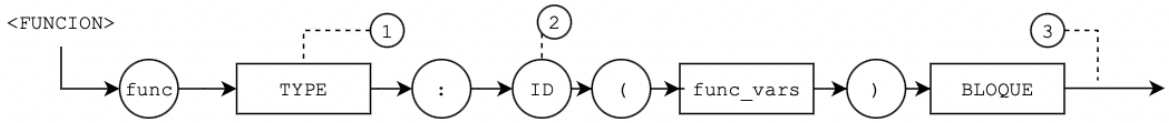


1. Crear directorio de funciones, crear una sección global
2. Insertar goto a main
3. Terminar programa

Módulos

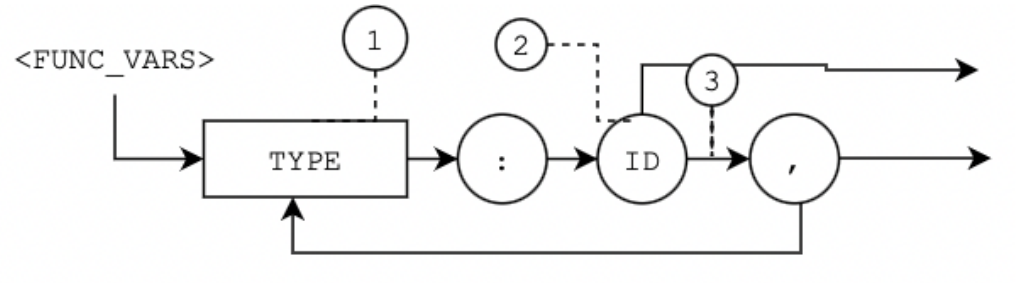


Función



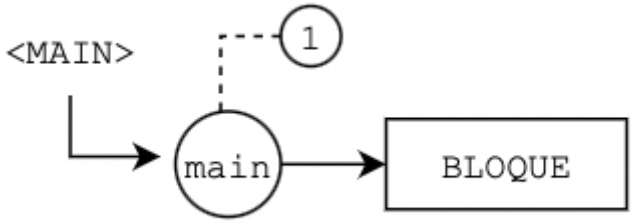
1. Obtener tipo
2. Obtener ID
3. Generar entrada en directorio de funciones con todos los datos necesarios (tipo, id, memoria, número de cuádruplo)

Parámetros



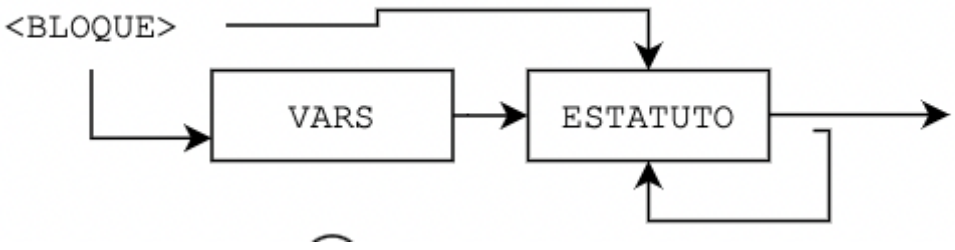
1. Obtener tipo
2. Obtener ID
3. Verificar ID, asignar dirección local e Insertar datos en tabla de variables y parámetros correspondientes

Main

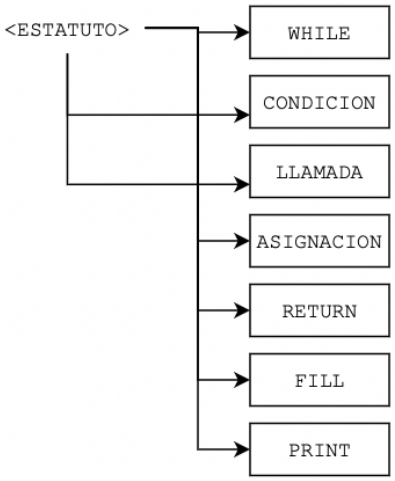


1. Cambiar scope a main

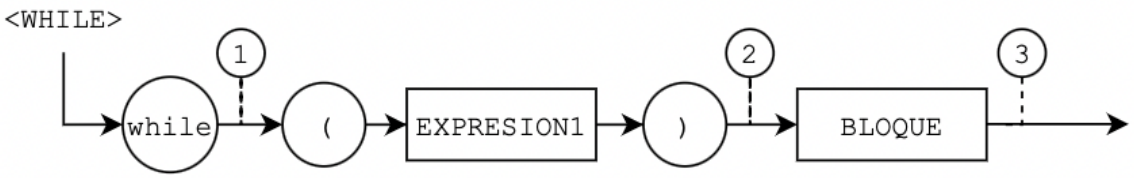
Bloque



Estatuto

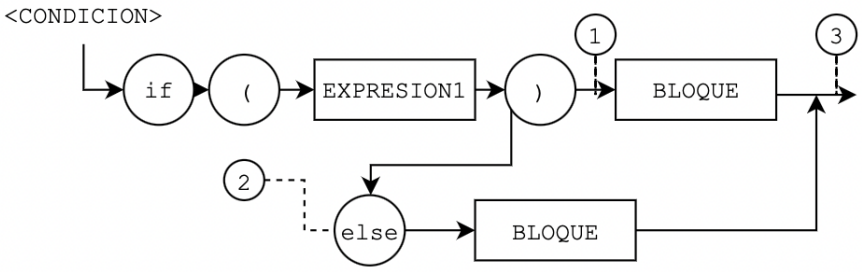


While



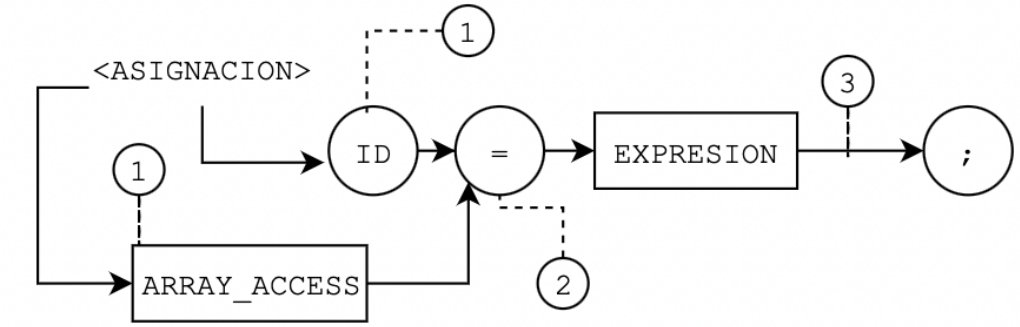
1. Insertar # cuádruplo en pila de saltos (migaja de pan)
2. Verificar expresión e insertar gotoF
3. Goto para evaluar expresión de nuevo

Condición



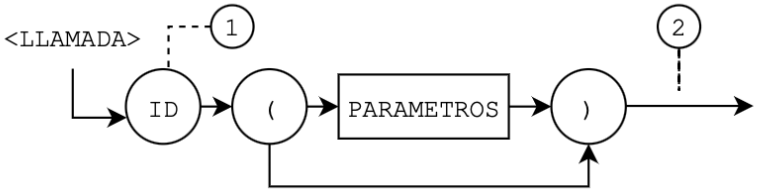
1. Verificar expresión y generar gotoF hacia el fin del if o el inicio del else
2. Goto al fin del if
3. Rellenar

Asignación



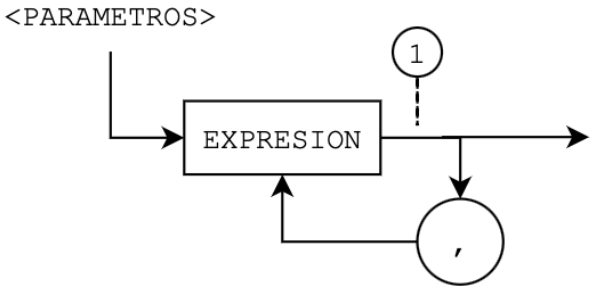
1. Verificar ID
2. Meter = a pOper
3. Verificar expresión y generar cuádruplo

Llamada



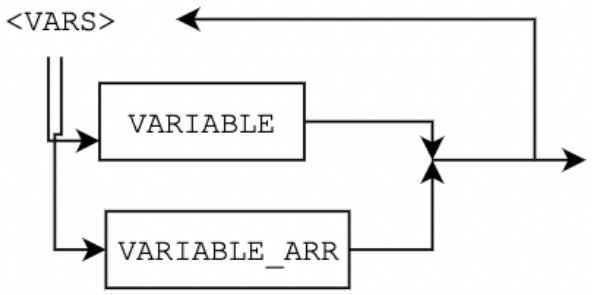
1. Obtener ID y verificar en directorio de funciones. Generar era. Iniciar contador de parámetros.
2. Generar gosub.

Parámetros

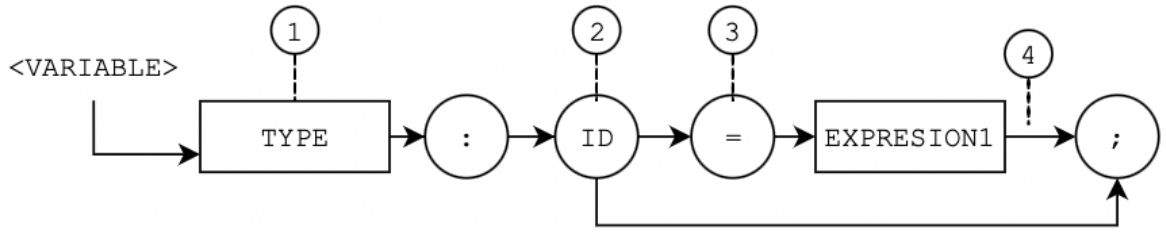


1. Verificar parámetro, tipo, número de parámetro y generar cuádruplo param

Variables

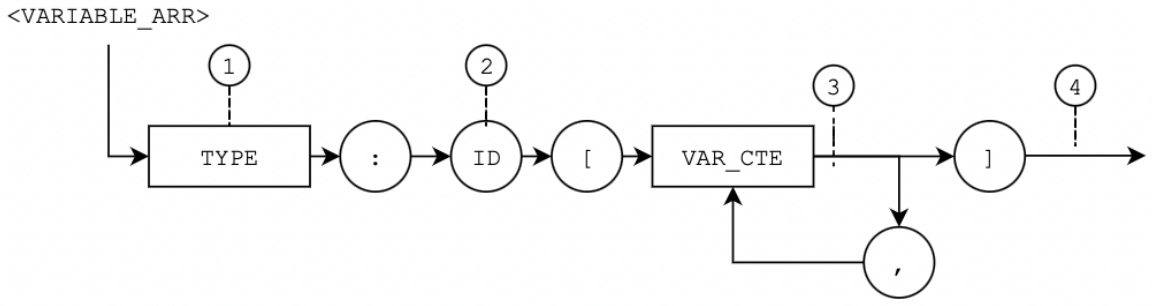


Variable normal



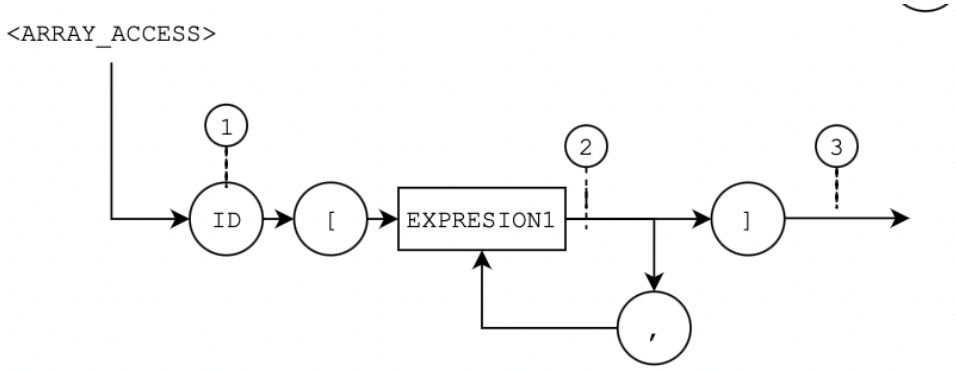
1. Obtener tipo
2. Obtener ID, verificar existencia, insertar en tabla, generar dirección.
3. Insertar = en pOper
4. Generar quad para asignar resultado de expresión

Variable de arreglo



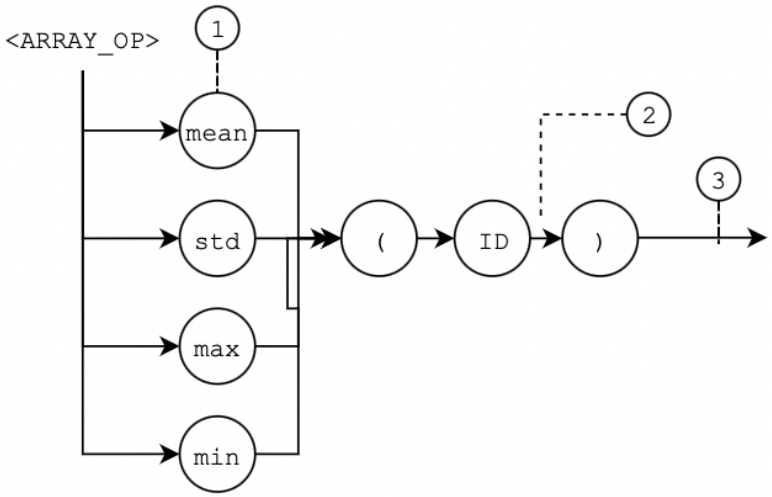
1. Obtener tipo
2. Obtener ID, verificar ID, insertar en tabla, señalar tipo dimensionado, inicializar estructura para dimensiones,
3. Verificar tipo de inicialización, crear nodo
4. Guardar tamaño, calcular Mn para cada dimensión, asignar direcciones de memoria

Acceso a arreglos



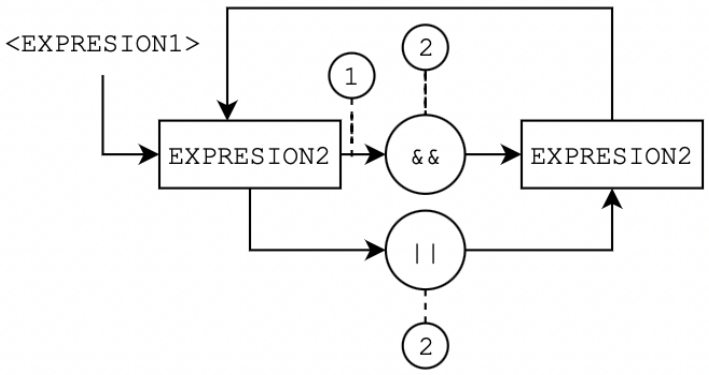
1. Obtener ID y verificar, pushear a pilaDim
2. Obtener resultado de expresión, generar cuádruplo para verificar límites, generar cuádruplos para obtener dirección del índice
3. Obtener dirección y crear temporal pointer

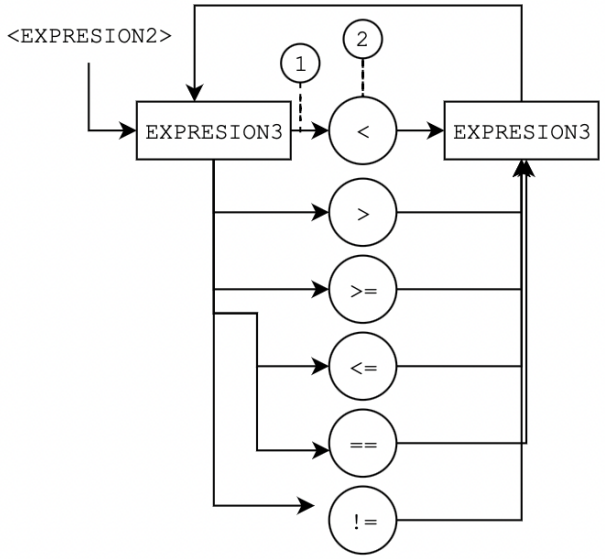
Operaciones de Arreglos

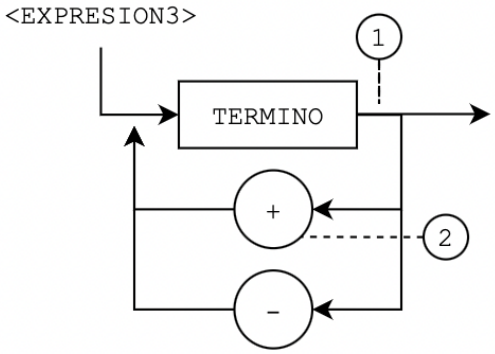


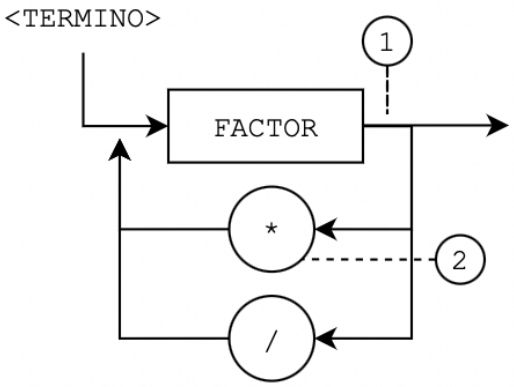
1. Obtener operación
2. Verificar que ID sea un arreglo
3. Generar cuádruplo correspondiente

Expresiones



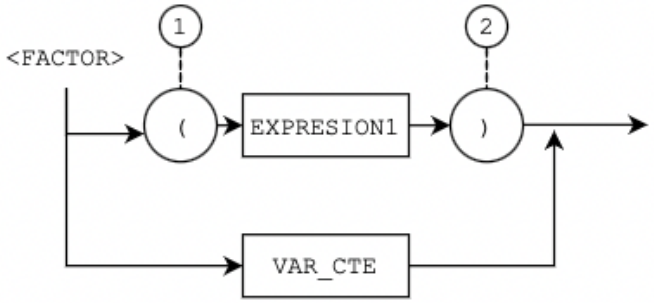






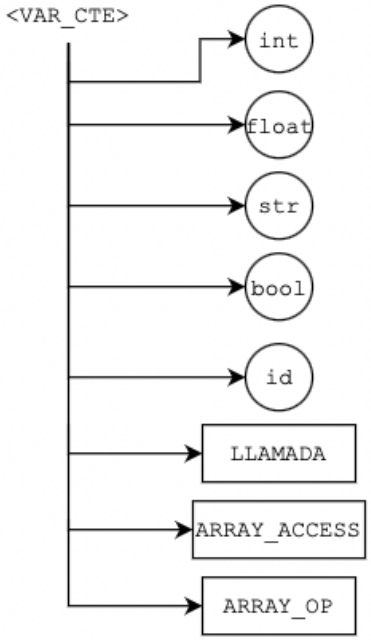
1. Verificar símbolos previos y generar cuádruplo de operación si hay
2. Meter símbolo a pOper

Factor

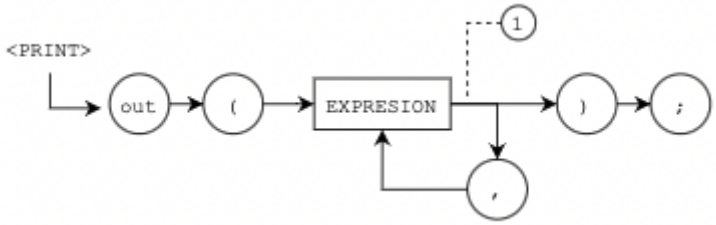


1. Meter fondo falso
2. Sacar fondo falso

Constantes

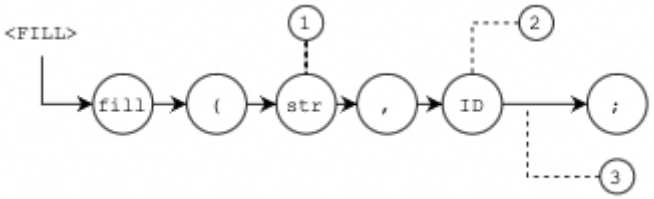


Escritura



1. Generar cuádruplo con resultado de la expresión

Fill



1. Verificar que el string sea un valor correcto
2. Verificar ID
3. Generar cuádruplo fil

Adicionalmente los diagramas pueden ser vistos aquí.

Descripción de la Administración de Memoria en Compilación

La memoria en compilación está manejada por un sólo objeto de la clase Memoria. Este objeto está definido de la siguiente manera.

```
class Memoria():
 def __init__(self):
   self.memoria_global = {'int' : [], 'float': [], 'string': [], 'bool' : []}
   self.memoria_local = {'int' : [], 'float': [], 'string': [], 'bool' : []}
   self.memoria_temporal = {'int' : [], 'float': [], 'string': [], 'bool' : [], 'pointer' : []}
   self.memoria_constante = {'int' : [], 'float': [], 'string': [], 'bool' : []}
```

Como se puede apreciar, hay 4 diccionarios representando los tipos de memoria, y a su vez estos tienen otros diccionarios representando el tipo de dato guardado en una respectiva lista. En compilación, estas listas son llenadas por valores vacíos los cuales son strings `'na'`, a excepción de la memoria constante la cual sí guarda valores.

Para apartar memoria en compilación, existen métodos push para cada tipo. Los cuales generan direcciones virtuales con base en el tipo de memoria y el tipo de valor. Además se definen offsets para cada tipo de memoria y tipo de valor.

```
   if tipo == 'float':
     current_size = len(self.memoria_global['float'])
     if current_size >= offset:
         raise stackOverflow("Stack memory exceeded!")
     self.memoria_global['float'].append('na')
     return current_size + offset * 2
```

El ejemplo anterior describe la sección para apartar memoria global del tipo float. El offset inicial es de 1000, este offset también es el máximo número de valores para cada tipo por lo que las direcciones globales del tipo float abarcan las direcciones 2000 a 2999. Adicionalmente a este offset existen otros para los tipos de memoria. De esta manera la fórmula para asignar direcciones es `current_size + offset * tipo + memoria_offset`. La siguiente tabla muestra todos los tipos de memoria y su traducción a direcciones virtuales.

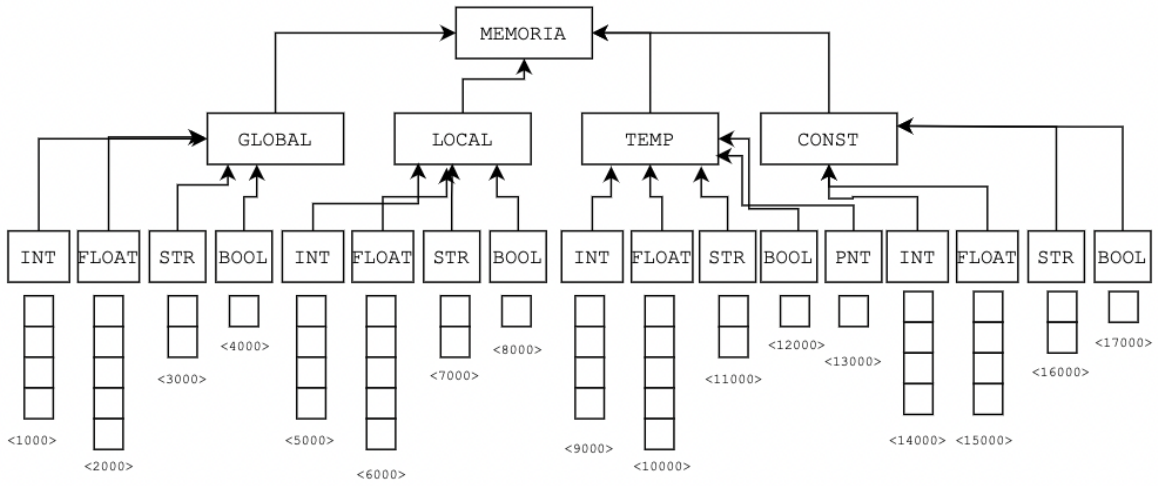
Tabla de traducción de direcciones virtuales

Memoria	Tipo	Offset Tipo	Offset Memoria	Dirección Virtual Min.	Dirección Virtual Max.
global	int	1000	0	1000	1999
	float	1000 * 2	0	2000	2999
	string	1000 * 3	0	3000	3999
	bool	1000 * 4	0	4000	4999
local	int	1000	4000	5000	5999
	float	1000 * 2	4000	6000	6999
	string	1000 * 3	4000	7000	7999
	bool	1000 * 4	4000	8000	8999
temporal	int	1000	8000	9000	9999
	float	1000 * 2	8000	10000	10999
	string	1000 * 3	8000	11000	11999
	bool	1000 * 4	8000	12000	12999
	pointer	1000 * 5	8000	13000	13999
constante	int	1000	12000	14000	14999
	float	1000 * 2	12000	15000	15999
	string	1000 * 3	12000	16000	16999
	bool	1000 * 4	12000	17000	17999

Para manejar valores constantes, también se cuenta con un directorio de constantes el cual es simplemente un diccionario cuyas llaves son las constantes y valores direcciones.

Este diagrama es una representación gráfica de la estructura del objeto memoria. Este objeto mantiene la misma estructura ya sea en compilación o ejecución.

Diagrama del Objeto Memoria



Directorio de Funciones

El directorio de funciones se realizó utilizando una serie de diccionarios. De esta manera se aseguró el acceso rápido a sus componentes y tablas de variables. Las llaves del directorio se acceden y crean con base en la variable `scope` definida.

Este scope está definido por la sección de código en la que se está compilando, por ejemplo, hay un scope para globales, otro para main y otro para cada función que exista en un programa.

De esta manera, la manera para acceder a todos los datos de un scope es simplemente `directorio_funciones[self.scope]`. Dentro de esto se encontrarán más llaves para la tabla de variables, cuádruplo de inicio, tabla de parámetros, etc.

Ejemplo de directorio de funciones

```
{
'global': {
    'tipo': 'global',
    'nombre': 'global',
    'tabla_vars': {
        'sum': {
            'nombre': 'sum',
            'tipo': 'int',
            'direccion': 1000
        }
    }
},
'sum': {
    'tipo': 'int',
    'nombre': 'sum',
    'tabla_vars': {
        'x': {
            'nombre': 'x',
            'tipo': 'int',
            'direccion': 5000
        },
        'y': {
            'nombre': 'y',
            'tipo': 'int',
            'direccion': 5001
        }
    },
    'inicio': 2,
    'params': [{
        'tipo': 'int',
        'direccion': 5000
    }, {
        'tipo': 'int',
```

```
            'direccion': 5001
        }],
        'memoria': {
            'local': [2, 0, 0, 0],
            'temporal': [1, 0, 0, 0, 0]
        }
    },
    'main': {
        'tipo': 'main',
        'nombre': 'main',
        'tabla_vars': {
            'mat': {
                'nombre': 'mat',
                'tipo': 'int',
                'direccion': 5000,
                'arr': True,
                'dims': [
                    [2, 4],
                    [4, 1]
                ],
                'size': 8
            },
            'st': {
                'nombre': 'st',
                'tipo': 'string',
                'direccion': 7000
            }
        },
        'inicio': 5
    }
}
```

Pilas y Cubo Semántico

Para las pilas se utilizaron deques, aunque las listas hubieran funcionado de la misma manera.

El cubo semántico consiste en un gran diccionario donde las llaves son una combinación de operador, operando, operando y resultado.

```
cubo = {
 '+' : {
   'int' : {
     'int' : 'int',
     'float' : 'float',
     'string' : 'ERROR',
     'bool' : 'ERROR'
   },
   'float' : {
     'int' : 'float',
     'float' : 'float',
     'string' : 'ERROR',
     'bool' : 'ERROR'
```

El cubo inicia de la forma vista arriba, extendiéndose bastante debido a la gran combinación de operadores y operandos que pueden darse.

Descripción de la Máquina Virtual

Objeto VM

La máquina virtual está dada por una clase llamada VM. La cual está inicializada con una lista conteniendo todos los cuádruplos generados, una lista para manejar todas las memorias que se generen, una lista de Instruction Pointers para manejar los saltos entre funciones y una variable para generar una nueva memoria cuando se ejecuta el cuádruplo ERA.

```
class VM():
 def __init__(self, quads):
   self.quads = quads
   self.call_stack = []
   self.ip_stack = [1]
   self.next_mem = None
```

Los métodos principales de la máquina virtual son `execute` y `execute_quad`. El primero de estos toma la lista de cuádruplos generados, los cuales son un objeto simple de la siguiente estructura:

```
class Quad():
 def __init__(self, op, opr1, opr2, res):
   self.op = op
   self.opr1 = opr1
   self.opr2 = opr2
   self.res = res
```

Y manda el cuádruplo correspondiente al Instruction Pointer a la segunda función. Esta función `execute_quad` se encarga de realizar la operación correspondiente. Hay que tener en cuenta que el Instruction Pointer Stack comienza con 1, para que se ejecute el primer cuádruplo.

Con esto en mente podemos definir estas instrucciones para comenzar a ejecutar la máquina virtual:

1. Obtener lista de cuádruplos
2. Meter 1 al IP Stack
3. Mientras que Quads[IP].opr no sea “endprogram” ejecuta
4. Ejecuta la instrucción correspondiente

Manejo de Memoria de Ejecución

Para el manejo de memoria en ejecución, se definieron en la clase Memoria métodos de asignación y para obtener valores, con base en los offsets y direcciones virtuales generadas en compilación.

```
def get_value(self, address):
   #Obtener valores globales
   if address >= 1000 and address <= 4999:
     if address <= 1999:
       return self.memoria_global['int'][address - 1000]
     elif address <= 2999:
       return self.memoria_global['float'][address - 2000]
     elif address <= 3999:
       return self.memoria_global['string'][address - 3000]
     elif address <= 4999:
       return self.memoria_global['bool'][address - 4000]
 def assign(self, address, value):
   #Asignar valores globales
   if address >= 1000 and address <= 4999:
     if address <= 1999:
       self.memoria_global['int'][address - 1000] = value
     elif address <= 2999:
       self.memoria_global['float'][address - 2000] = value
     elif address <= 3999:
       self.memoria_global['string'][address - 3000] = value
     elif address <= 4999:
       self.memoria_global['bool'][address - 4000] = value
```

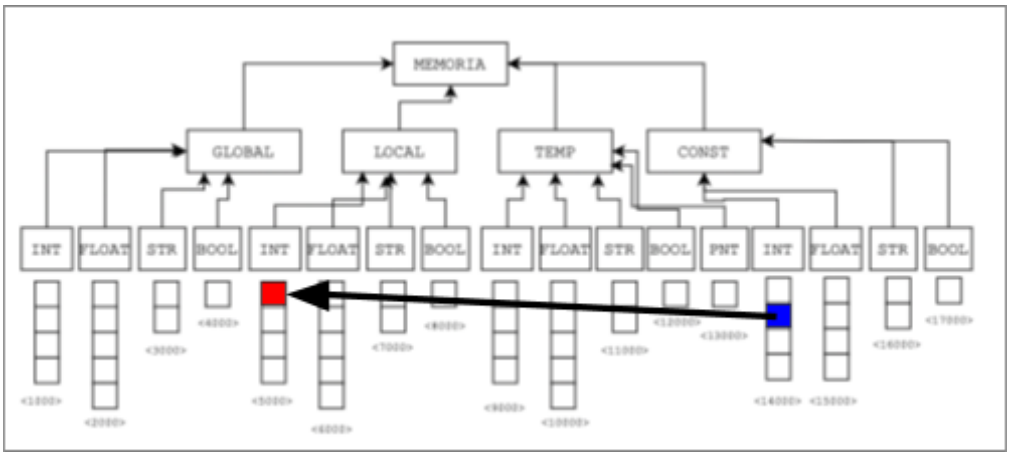
De esta manera se obtiene y asigna de manera eficiente el valor de cada dirección virtual.

Por ejemplo, si tenemos un cuádruplo `“=” 14001 None 5000` debemos asignar el valor que se encuentra en la dirección 14001 a la variable que se encuentra en la

dirección 5001. Podemos identificar que estas direcciones corresponden a variables locales enteras y constantes enteras (si recordamos la forma de asignar direcciones virtuales).

Con esto en mente, podemos obtener las direcciones del objeto cuádruplo y asignar los valores correspondientes.

```
def assign(self, quad):
  #Obtener direcciones
  add1 = self.get_address(quad.opr1)
  add2 = self.get_address(quad.res)
  #Obtener el valor a asignar
  value = self.call_stack[-1].get_value(add1)
  #Realizar asignación de valor
  self.call_stack[-1].assign(add2, value)
  #Move to next quad
  self.ip_stack[-1] += 1
```



Para el manejo de scopes se realiza lo siguiente:

1. Era: Creamos un nuevo objeto memoria
2. Sacamos el nombre de la función del cuádruplo, utilizamos el directorio de funciones para ver la memoria que hay que apartar
3. Param: Obtenemos el valor del parámetro y lo asignamos a la memoria nueva
4. Gosub: Insertamos la nueva memoria en el call_stack, haciendo esta memoria la actual, copiamos globales y constantes a la nueva memoria
5. Cambiamos el IP al inicio de la función, sacado del directorio de funciones
6. Return: Se asigna el valor de retorno a la memoria anterior, se hace pop() del call_stack para regresar a la memoria inicial y pop() al ip_stack para continuar en la instrucción anterior.

Pruebas del Funcionamiento del Lenguaje

Array_test.ly

Código

```
main {
  int : mat[2,4];
  int : arr[3];
  int : i = 0;
  int : j = 0;
  int : cont = 1;
  arr[2] = 1;
  while(i < 2){
    j = 0;
    while(j < 4){
      mat[i, j] = cont;

      j = j + 1;
      cont = cont + 1;
    }
    i = i + 1;
  }
  i = 0;
  j = 0;
  while(i < 2){
    j = 0;
    while(j < 4){
      out(mat[i, j]);
      j = j + 1;
    }
    out("/n");
    i = i + 1;
  }
  out(mat[arr[2], 2]);
```

```
}
```

Cuádruplos

```
1 : goto None None 2
2 : = 14003 None 5011
3 : = 14003 None 5012
4 : = 14004 None 5013
5 : ver 14000 0 3
6 : + 14000 5008 13000
7 : = 14004 None 13000
8 : < 5011 14000 12000
9 : gotoF 12000 None 27
10 : = 14003 None 5012
11 : < 5012 14001 12001
12 : gotoF 12001 None 24
13 : ver 5011 0 2
14 : * 5011 4 9000
15 : ver 5012 0 4
16 : + 9000 5012 9001
17 : + 9001 5000 13001
18 : = 5013 None 13001
19 : + 5012 14004 9002
20 : = 9002 None 5012
21 : + 5013 14004 9003
22 : = 9003 None 5013
23 : goto None None 11
24 : + 5011 14004 9004
25 : = 9004 None 5011
26 : goto None None 8
27 : = 14003 None 5011
28 : = 14003 None 5012
29 : < 5011 14000 12002
30 : gotoF 12002 None 47
31 : = 14003 None 5012
```

```
32 : < 5012 14001 12003
33 : gotoF 12003 None 43
34 : ver 5011 0 2
35 : * 5011 4 9005
36 : ver 5012 0 4
37 : + 9005 5012 9006
38 : + 9006 5000 13002
39 : print None None 13002
40 : + 5012 14004 9007
41 : = 9007 None 5012
42 : goto None None 32
43 : print None None 16000
44 : + 5011 14004 9008
45 : = 9008 None 5011
46 : goto None None 29
47 : ver 14000 0 3
48 : + 14000 5008 13003
49 : ver 13003 0 2
50 : * 13003 4 9009
51 : ver 14000 0 4
52 : + 9009 14000 9010
53 : + 9010 5000 13004
54 : print None None 13004
55 : endprogram None None None
```

Orden de Ejecución de Cuádruplos

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 11 12 13
14 15 16 17 18 19 20 21 22 23 11 12 13 14 15 16 17 18 19 20 21 22 23
11 12 13 14 15 16 17 18 19 20 21 22 23 11 12 24 25 26 8 9 10 11 12
13 14 15 16 17 18 19 20 21 22 23 11 12 13 14 15 16 17 18 19 20 21 22
23 11 12 13 14 15 16 17 18 19 20 21 22 23 11 12 13 14 15 16 17 18 19
20 21 22 23 11 12 24 25 26 8 9 27 28 29 30 31 32 33 34 35 36 37 38
39 40 41 42 32 33 34 35 36 37 38 39 40 41 42 32 33 34 35 36 37 38 39
40 41 42 32 33 34 35 36 37 38 39 40 41 42 32 33 43 44 45 46 29 30 31
```

```
32 33 34 35 36 37 38 39 40 41 42 32 33 34 35 36 37 38 39 40 41 42 32
33 34 35 36 37 38 39 40 41 42 32 33 34 35 36 37 38 39 40 41 42 32 33
43 44 45 46 29 30 47 48 49 50 51 52 53 54
```

Resultado

```
1 2 3 4
5 6 7 8
7
```

esp_test.ly

Código

```
main {
  int : arr[100];
  float : ans;
  fill("random", arr);

  ans = mean(arr);
  out(ans);
  ans = std(arr);
  out(ans);
  ans = min(arr);
  out(ans);
  ans = max(arr);
  out(ans);
}
```

Cuádruplos generados

```
1 : goto None None 2
2 : fill random 100 5000
3 : mean 5000 100 10000
4 : = 10000 None 6000
5 : print None None 6000
6 : std 5000 100 10001
7 : = 10001 None 6000
8 : print None None 6000
```

```
9 : min 5000 100 10002
10 : = 10002 None 6000
11 : print None None 6000
12 : max 5000 100 10003
13 : = 10003 None 6000
14 : print None None 6000
15 : endprogram None None None
```

Orden de Ejecución de Cuádruplos

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

Resultado

```
47.82 27.45956299725107 1 98
```

fact_iter.ly

Código

```
main {
  int : fact = 1;
  int : n = 5;
  while(n > 0){
    fact = fact * n;
    n = n - 1;
  }
  out(fact);
}
```

Cuádruplos Generados

```
1 : goto None None 2
2 : = 14000 None 5000
3 : = 14001 None 5001
4 : > 5001 14002 12000
5 : gotoF 12000 None 11
6 : * 5000 5001 9000
7 : = 9000 None 5000
8 : - 5001 14000 9001
9 : = 9001 None 5001
```

```
10 : goto None None 4
11 : print None None 5000
12 : endprogram None None None
```

Orden de Ejecución de Cuádruplos

```
1 2 3 4 5 6 7 8 9 10 4 5 6 7 8 9 10 4 5 6 7 8 9 10 4 5 6 7 8 9 10 4
5 6 7 8 9 10 4 5 11
```

Resultado

```
120
```

fact_recursivo.ly

Código

```
func int : fact(int : n){
  if(n == 0){
    ret 1;
  }
  ret n * fact(n - 1);
}

main{
  out(fact(5));
}
```

Cuádruplos Generados

```
1 : goto None None 13
2 : == 5000 14000 12000
3 : gotoF 12000 None 5
4 : return fact None 14001
5 : era None None fact
6 : - 5000 14001 9000
7 : parameter 9000 1 fact
8 : gosub None None fact
9 : = 1000 None 9001
10 : * 5000 9001 9002
11 : return fact None 9002
```

```
12 : endfunc None None None
13 : era None None fact
14 : parameter 14002 1 fact
15 : gosub None None fact
16 : = 1000 None 9000
17 : print None None 9000
18 : endprogram None None None
```

Orden de Ejecución de Cuádruplos

```
1 13 14 15 2 3 5 6 7 8 2 3 5 6 7 8 2 3 5 6 7 8 2 3 5 6 7 8 2 3 5 6 7
8 2 3 4 9 10 11 9 10 11 9 10 11 9 10 11 9 10 11 16 17
```

Resultado

```
120
```

fibo_iter.ly

Código

```
func int : fibo(int : n){
  int : a = 0;
  int : b = 1;
  int : c;
  int : i = 2;
  if(n <= 1){
    ret n;
  }
  while(i <= n){
    c = a + b;
    a = b;
    b = c;
    i = i + 1;
  }
  ret b;
}

main {
```

```
  out(fibo(9));
}
```

Cuádruplos Generados

```
1 : goto None None 19
2 : = 14000 None 5001
3 : = 14001 None 5002
4 : = 14002 None 5004
5 : <= 5000 14001 12000
6 : gotoF 12000 None 8
7 : return fibo None 5000
8 : <= 5004 5000 12001
9 : gotoF 12001 None 17
10 : + 5001 5002 9000
11 : = 9000 None 5003
12 : = 5002 None 5001
13 : = 5003 None 5002
14 : + 5004 14001 9001
15 : = 9001 None 5004
16 : goto None None 8
17 : return fibo None 5002
18 : endfunc None None None
19 : era None None fibo
20 : parameter 14003 1 fibo
21 : gosub None None fibo
22 : = 1000 None 9000
23 : print None None 9000
24 : endprogram None None None
```

Orden de Ejecución de Cuádruplos

```
1 19 20 21 2 3 4 5 6 8 9 10 11 12 13 14 15 16 8 9 10 11 12 13 14 15
16 8 9 10 11 12 13 14 15 16 8 9 10 11 12 13 14 15 16 8 9 10 11 12 13
14 15 16 8 9 10 11 12 13 14 15 16 8 9 10 11 12 13 14 15 16 8 9 10 11
12 13 14 15 16 8 9 17 22 23
```

Resultado

```
34
```

fibo_recursivo.ly

Código

```
func int : fibo (int : x){
  if(x <= 1){
    ret x;
  }
  ret fibo(x - 1) + fibo(x - 2);
}

main{
  int : ans;
  ans = fibo(6);
  out(ans);
}
```

Cuádruplos Generados

```
1 : goto None None 18
2 : <= 5000 14000 12000
3 : gotoF 12000 None 5
4 : return fibo None 5000
5 : era None None fibo
6 : - 5000 14000 9000
7 : parameter 9000 1 fibo
8 : gosub None None fibo
9 : = 1000 None 9001
10 : era None None fibo
11 : - 5000 14001 9002
12 : parameter 9002 1 fibo
13 : gosub None None fibo
14 : = 1000 None 9003
15 : + 9001 9003 9004
16 : return fibo None 9004
```

```
17 : endfunc None None None
18 : era None None fibo
19 : parameter 14002 1 fibo
20 : gosub None None fibo
21 : = 1000 None 9000
22 : = 9000 None 5000
23 : print None None 5000
24 : endprogram None None None
```

Orden de Ejecución de Cuádruplos

```
1 18 19 20 2 3 5 6 7 8 2 3 5 6 7 8 2 3 5 6 7 8 2 3 5 6 7 8 2 3 5 6 7
8 2 3 4 9 10 11 12 13 2 3 4 14 15 16 9 10 11 12 13 2 3 4 14 15 16 9
10 11 12 13 2 3 5 6 7 8 2 3 4 9 10 11 12 13 2 3 4 14 15 16 14 15 16
9 10 11 12 13 2 3 5 6 7 8 2 3 5 6 7 8 2 3 4 9 10 11 12 13 2 3 4 14
15 16 9 10 11 12 13 2 3 4 14 15 16 14 15 16 9 10 11 12 13 2 3 5 6 7
8 2 3 5 6 7 8 2 3 5 6 7 8 2 3 4 9 10 11 12 13 2 3 4 14 15 16 9 10 11
12 13 2 3 4 14 15 16 9 10 11 12 13 2 3 5 6 7 8 2 3 4 9 10 11 12 13 2
3 4 14 15 16 14 15 16 14 15 16 21 22 23
```

Resultado

```
8
```

find.ly

Código

```
main {
  int : i = 0;
  int : size;
  int : arr[100];
  int : ans = 0;

  fill("random", arr);
  size = len(arr);
  out("Looking for 101/n");
  arr[72] = 101;
  while(i < size){
```

```
    if(arr[i] == 101){
      ans = i;
    }
    i = i + 1;
  }
  out("found in index: ", ans);
}
```

Cuádruplos Generados

```
1 : goto None None 2
2 : = 14000 None 5000
3 : = 14000 None 5102
4 : fill random 100 5002
5 : len 5002 100 9000
6 : = 9000 None 5001
7 : print None None 16000
8 : ver 14002 0 100
9 : + 14002 5002 13000
10 : = 14003 None 13000
11 : < 5000 5001 12000
12 : gotoF 12000 None 21
13 : ver 5000 0 100
14 : + 5000 5002 13001
15 : == 13001 14003 12001
16 : gotoF 12001 None 18
17 : = 5000 None 5102
18 : + 5000 14004 9001
19 : = 9001 None 5000
20 : goto None None 11
21 : print None None 16001
22 : print None None 5102
23 : endprogram None None None
```

Orden de Ejecución de Cuádruplos

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18
19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13
14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19
20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14
15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20
11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15
16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11
12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16
18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12
13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18
19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13
14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19
20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14
15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20
11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15
16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11
12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16
18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12
13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18
19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13
14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19
20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14
15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20
11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15
16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11
12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16
18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12
13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18
19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 17 18 19 20 11 12
13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18
19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13
14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19

```
20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14
15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20
11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15
16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11
12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16
18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12
13 14 15 16 18 19 20 11 12 13 14 15 16 18 19 20 11 12 13 14 15 16 18
19 20 11 12 13 14 15 16 18 19 20 11 12 21 22
```

Resultado

```
Looking for 101
found in index:  72
```

mat_mul.ly

Código

```
main {
  int : mat1[2,3];
  int : mat2[3,2];
  int : res[2,2];
  int : i = 0;
  int : j = 0;
  int : k = 0;

  mat1[0,0] = 1;
  mat1[0,1] = 2;
  mat1[0,2] = 3;
  mat1[1,0] = 4;
  mat1[1,1] = 5;
  mat1[1,2] = 6;
  mat2[0,0] = 10;
  mat2[0,1] = 11;
  mat2[1,0] = 20;
  mat2[1,1] = 21;
  mat2[2,0] = 30;
```

```
  mat2[2,1] = 31;

  while(i < 2){
    j = 0;
    while(j < 2){
      res[i, j] = 0;
      k = 0;
      while(k < 3){
        res[i, j] = res[i, j] + mat1[i, k] * mat2[k, j];
        k = k + 1;
      }
      j = j + 1;
    }
    i = i + 1;
  }

  i = 0;
  j = 0;

  while(i < 2){
    j = 0;
    while(j < 2){
      out(res[i, j]);
      j = j + 1;
    }
    out("/n");
    i = i + 1;
  }
}
```

Cuádruplos Generados

```
1 : goto None None 2
2 : = 14002 None 5016
3 : = 14002 None 5017
```

```
4 : = 14002 None 5018
5 : ver 14002 0 2
6 : * 14002 3 9000
7 : ver 14002 0 3
8 : + 9000 14002 9001
9 : + 9001 5000 13000
10 : = 14003 None 13000
11 : ver 14002 0 2
12 : * 14002 3 9002
13 : ver 14003 0 3
14 : + 9002 14003 9003
15 : + 9003 5000 13001
16 : = 14000 None 13001
17 : ver 14002 0 2
18 : * 14002 3 9004
19 : ver 14000 0 3
20 : + 9004 14000 9005
21 : + 9005 5000 13002
22 : = 14001 None 13002
23 : ver 14003 0 2
24 : * 14003 3 9006
25 : ver 14002 0 3
26 : + 9006 14002 9007
27 : + 9007 5000 13003
28 : = 14004 None 13003
29 : ver 14003 0 2
30 : * 14003 3 9008
31 : ver 14003 0 3
32 : + 9008 14003 9009
33 : + 9009 5000 13004
34 : = 14005 None 13004
35 : ver 14003 0 2
36 : * 14003 3 9010
```

```
37 : ver 14000 0 3
38 : + 9010 14000 9011
39 : + 9011 5000 13005
40 : = 14006 None 13005
41 : ver 14002 0 3
42 : * 14002 2 9012
43 : ver 14002 0 2
44 : + 9012 14002 9013
45 : + 9013 5006 13006
46 : = 14007 None 13006
47 : ver 14002 0 3
48 : * 14002 2 9014
49 : ver 14003 0 2
50 : + 9014 14003 9015
51 : + 9015 5006 13007
52 : = 14008 None 13007
53 : ver 14003 0 3
54 : * 14003 2 9016
55 : ver 14002 0 2
56 : + 9016 14002 9017
57 : + 9017 5006 13008
58 : = 14009 None 13008
59 : ver 14003 0 3
60 : * 14003 2 9018
61 : ver 14003 0 2
62 : + 9018 14003 9019
63 : + 9019 5006 13009
64 : = 14010 None 13009
65 : ver 14000 0 3
66 : * 14000 2 9020
67 : ver 14002 0 2
68 : + 9020 14002 9021
69 : + 9021 5006 13010
```

```
70 : = 14011 None 13010
71 : ver 14000 0 3
72 : * 14000 2 9022
73 : ver 14003 0 2
74 : + 9022 14003 9023
75 : + 9023 5006 13011
76 : = 14012 None 13011
77 : < 5016 14000 12000
78 : gotoF 12000 None 123
79 : = 14002 None 5017
80 : < 5017 14000 12001
81 : gotoF 12001 None 120
82 : ver 5016 0 2
83 : * 5016 2 9024
84 : ver 5017 0 2
85 : + 9024 5017 9025
86 : + 9025 5012 13012
87 : = 14002 None 13012
88 : = 14002 None 5018
89 : < 5018 14001 12002
90 : gotoF 12002 None 117
91 : ver 5016 0 2
92 : * 5016 2 9026
93 : ver 5017 0 2
94 : + 9026 5017 9027
95 : + 9027 5012 13013
96 : ver 5016 0 2
97 : * 5016 2 9028
98 : ver 5017 0 2
99 : + 9028 5017 9029
100 : + 9029 5012 13014
101 : ver 5016 0 2
102 : * 5016 3 9030
```

```
103 : ver 5018 0 3
104 : + 9030 5018 9031
105 : + 9031 5000 13015
106 : ver 5018 0 3
107 : * 5018 2 9032
108 : ver 5017 0 2
109 : + 9032 5017 9033
110 : + 9033 5006 13016
111 : * 13015 13016 9034
112 : + 13014 9034 9035
113 : = 9035 None 13013
114 : + 5018 14003 9036
115 : = 9036 None 5018
116 : goto None None 89
117 : + 5017 14003 9037
118 : = 9037 None 5017
119 : goto None None 80
120 : + 5016 14003 9038
121 : = 9038 None 5016
122 : goto None None 77
123 : = 14002 None 5016
124 : = 14002 None 5017
125 : < 5016 14000 12003
126 : gotoF 12003 None 143
127 : = 14002 None 5017
128 : < 5017 14000 12004
129 : gotoF 12004 None 139
130 : ver 5016 0 2
131 : * 5016 2 9039
132 : ver 5017 0 2
133 : + 9039 5017 9040
134 : + 9040 5012 13017
135 : print None None 13017
```

```
136 : + 5017 14003 9041
137 : = 9041 None 5017
138 : goto None None 128
139 : print None None 16000
140 : + 5016 14003 9042
141 : = 9042 None 5016
142 : goto None None 125
143 : endprogram None None None
```

Orden de Ejecución de Cuádruplos

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113
114 115 116 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105
106 107 108 109 110 111 112 113 114 115 116 89 90 91 92 93 94 95 96
97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114
115 116 89 90 117 118 119 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
112 113 114 115 116 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103
104 105 106 107 108 109 110 111 112 113 114 115 116 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
112 113 114 115 116 89 90 117 118 119 80 81 120 121 122 77 78 79 80
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
103 104 105 106 107 108 109 110 111 112 113 114 115 116 89 90 91 92
93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
112 113 114 115 116 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103
104 105 106 107 108 109 110 111 112 113 114 115 116 89 90 117 118
119 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 89
90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 89 90 91 92 93 94 95 96 97 98 99 100
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 89
```

```
90 117 118 119 80 81 120 121 122 77 78 123 124 125 126 127 128 129
130 131 132 133 134 135 136 137 138 128 129 130 131 132 133 134 135
136 137 138 128 129 139 140 141 142 125 126 127 128 129 130 131 132
133 134 135 136 137 138 128 129 130 131 132 133 134 135 136 137 138
128 129 139 140 141 142 125 126
```

Resultado

```
140 146
320 335
```

sort.ly

Código

```
main {
  int : arr[100];
  int : i = 0;
  int : j = 0;
  int : size;
  int : temp;

  fill("random", arr);
  size = len(arr);

  out("unsorted/n");
  while(i < size){
    out(arr[i]);
    i = i + 1;
  }

  i = 0;
  while(i < size - 1){
    j = 0;
    while(j < size - i - 1){
      if(arr[j] > arr[j + 1]){
        temp = arr[j];
```

```
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
      j = j + 1;
    }
    i = i + 1;
  }
  out("/n");
  out("sorted/n");
  i = 0;
  while(i < size){
    out(arr[i]);
    i = i + 1;
  }
}
```

Cuádruplos Generados

```
1 : goto None None 2
2 : = 14001 None 5100
3 : = 14001 None 5101
4 : fill random 100 5000
5 : len 5000 100 9000
6 : = 9000 None 5102
7 : print None None 16000
8 : < 5100 5102 12000
9 : gotoF 12000 None 16
10 : ver 5100 0 100
11 : + 5100 5000 13000
12 : print None None 13000
13 : + 5100 14002 9001
14 : = 9001 None 5100
15 : goto None None 8
16 : = 14001 None 5100
17 : - 5102 14002 9002
```

```
18 : < 5100 9002 12001
19 : gotoF 12001 None 51
20 : = 14001 None 5101
21 : - 5102 5100 9003
22 : - 9003 14002 9004
23 : < 5101 9004 12002
24 : gotoF 12002 None 48
25 : ver 5101 0 100
26 : + 5101 5000 13001
27 : + 5101 14002 9005
28 : ver 9005 0 100
29 : + 9005 5000 13002
30 : > 13001 13002 12003
31 : gotoF 12003 None 45
32 : ver 5101 0 100
33 : + 5101 5000 13003
34 : = 13003 None 5103
35 : ver 5101 0 100
36 : + 5101 5000 13004
37 : + 5101 14002 9006
38 : ver 9006 0 100
39 : + 9006 5000 13005
40 : = 13005 None 13004
41 : + 5101 14002 9007
42 : ver 9007 0 100
43 : + 9007 5000 13006
44 : = 5103 None 13006
45 : + 5101 14002 9008
46 : = 9008 None 5101
47 : goto None None 21
48 : + 5100 14002 9009
49 : = 9009 None 5100
50 : goto None None 17
```

```
51 : print None None 16001
52 : print None None 16002
53 : = 14001 None 5100
54 : < 5100 5102 12004
55 : gotoF 12004 None 62
56 : ver 5100 0 100
57 : + 5100 5000 13007
58 : print None None 13007
59 : + 5100 14002 9010
60 : = 9010 None 5100
61 : goto None None 54
62 : endprogram None None None
```

Resultado

```
unsorted
8 67 98 97 71 2 88 8 53 86 92 32 77 90 64 94 17 53 49 44 51 57 18 86
23 24 87 62 26 82 1 83 19 94 89 71 0 92 56 50 45 56 51 92 95 84 93 7
6 95 45 25 84 73 37 35 97 9 72 43 39 64 78 75 69 78 45 73 62 57 35
35 52 17 15 72 65 36 65 5 98 19 30 76 17 52 52 89 14 74 61 71 22 87
83 15 96 62 57 7
sorted
0 1 2 5 6 7 7 8 8 9 14 15 15 17 17 17 18 19 19 22 23 24 25 26 30 32
35 35 35 36 37 39 43 44 45 45 45 49 50 51 51 52 52 52 53 53 56 56 57
57 57 61 62 62 62 64 64 65 65 67 69 71 71 71 72 72 73 73 74 75 76 77
78 78 82 83 83 84 84 86 86 87 87 88 89 89 90 92 92 92 93 94 94 95 95
96 97 97 98 98
```

Manual de Usuario Lenguaje Luyer

Proyecto para la materia de diseño de compiladores del semestre Agosto-Diciembre 2022.

Setup

Luyer es un lenguaje hecho en Python, por lo que este se necesita para ejecutarlo.

Además, ya con python instalado, se necesitan descargar e instalar dos librerías, LARK y NumPy. Para instalarla ejecuta los siguiente en tu terminal.

`pip install lark` `pip install numpy`

Después clona este repositorio

`git clone https://github.com/Luyer74/Compilador`

Sintaxis y ejecución

La estructura de un programa hecho en Luyer es `Globales -> Funciones -> Main`

Variables

Las variables en Luyer se pueden declarar localmente o globalmente. Como se vió en la estructura anterior, las globales son lo primero. Las variables locales se declaran al inicio de main o de cada función para ser accesadas ahí mismo. Las variables se pueden declarar sin valor

`int : i`

o con valor asignado

`int : i = 0`

Condiciones

Las condiciones siguen la sintaxis estándar

```
if (i != 0){
  out(i);
} else{
```

```
    out("fin");
  }
```

Ciclos

Se cuentan con ciclos de tipo while

```
while (i < 100){
  out(i);
  i = i + 1;
}
```

Funciones

Para funciones, se sigue el siguiente formato

```
func int : suma(int : x, int : y){
  ret x + y;
}
```

Variables con Dimensiones

Luyer maneja variables de N dimensiones.

```
int : a[10, 5, 2];
```

Funciones estadísticas

Luyer cuenta con funciones auxiliares.

`fill(método, arreglo)`: Llena un arreglo con el método especificado.

- "zeros", arreglo: llena con ceros
- "ones", arreglo: llena con unos
- "arange", arreglo: llena con números de 0 al tamaño del arreglo en orden

`min(arreglo)` : Regresa el valor mínimo de un arreglo

`max(arreglo)` : Regresa el valor máximo de un arreglo

`mean(arreglo)` : Regresa el valor promedio de un arreglo

`std(arreglo)` : Regresa el valor de desviación estándar de un arreglo

`len(arreglo)` : Regresa la longitud de un arreglo

Ejecución

Para ejecutar un programa en Luyer, debemos reemplezar una parte del archivo `compilador.py`

En la sección de `input = open("./pruebas/array_test.ly", "r").read()` se reemplaza el archivo actual por tu propio archivo, el cual puede tener cualquier terminación.