



Tecnológico de Monterrey

Campus Monterrey

Diseño de Compiladores

Ing. Elda Quiroga González

Ing. Héctor Ceballos Cancino

Documentación Patito++

Rafael Serna Salazar
A01282298

Héctor de Luna Pámanes
A01282272

Descripción del proyecto	2
Propósito, objetivos y alcance del proyecto	2
Análisis de requerimiento y casos de uso generales	2
Descripción de los principales test cases	4
Descripción del proceso general	4
Descripción del lenguaje	7
Nombre del lenguaje	7
Descripción de las principales características del lenguaje	7
Errores que pueden ocurrir	7
Descripción del compilador	8
Equipo de cómputo, lenguaje y utilerías especiales usadas	8
Descripción del análisis del léxico	8
Descripción del análisis de sintaxis	9
Descripción de generación de código intermedio y análisis semántico	12
Descripción del proceso de administración de memoria	12
Descripción de la máquina virtual	18
Librerías especiales	18
Proceso de administración de memoria en ejecución	18
Pruebas del funcionamiento del lenguaje	19
Listados perfectamente documentados del proyecto	23
Commits	54

Descripción del proyecto

Propósito, objetivos y alcance del proyecto

El propósito de desarrollar el compilador fue para poder entender mejor cómo es que se desarrolla un lenguaje de programación, aprender esto lograría llegar a un mejor entendimiento del desarrollo de otras aplicaciones y a aprovechar el uso de los compiladores y lenguajes ya existentes. El objetivo principal es poder desarrollar las habilidades necesarias para construir un compilador desde cero y pasar la materia. El alcance es poder crear un lenguaje funcional, que demuestre nuestros conocimientos del curso.

Análisis de requerimiento y casos de uso generales

Requerimientos

- Se pueden declarar variables globales o locales de tipo INT, FLOAT, CHAR, BOOL, o bien variables dimensionadas de los tipos anteriores (arreglos y matrices de máximo dos dimensiones)
- Se puede declarar funciones con parámetros que regresan un valor de los tipos anteriores o no (VOID)
- Se tienen estatutos de decisión (IF)
- Se tienen estatutos de repetición condicional (WHILE)
- Se tienen estatutos de repetición no-condicional (FOR)
- Se permite lectura y escritura (READ, PRINT)
- Se tienen los operadores aritméticos, de comparación y relacionales tradicionales de Java o C: +, -, *, /, ==, |, &, >, <, >=, <=
- Se tienen operadores especiales para matrices
 - j : genera la transpuesta
 - ? : genera la matriz inversa
 - \$: genera el determinante de una matriz

Casos de uso

1	Correr Programa
Actor	Usuario
Precondiciones	Archivo de texto con formato .ppp
Resultado	Se ejecuta el programa
Flujo	1. Se convierte texto a tokens 2. Se hace el parse con los tokens al programa 3. Se verifica sintaxis y semántica correcta, se generan los cuádruplos 4. Máquina virtual ejecuta los cuádruplos
Flujo Alternativo	3. Sintaxis, semántica incorrectos, se muestra error al usuario

2	Verificar sintaxis
Actor	Compilador
Precondiciones	Archivo de texto con formato .ppp con algún error de sintaxis
Resultado	Se muestra error al usuario
Flujo	<ol style="list-style-type: none"> 1. Se convierte texto a tokens 2. Se hace el parse con los tokens al programa 3. Sintaxis, semántica incorrectos, se muestra error al usuario
Flujo Alternativo	N/A

3	Asignar un arreglo a una matriz
Actor	Usuario
Precondiciones	Arreglo y matrices declarados en el archivo .ppp
Resultado	Error, dado que los tipos no son compatibles
Flujo	<ol style="list-style-type: none"> 1. Se llega hasta el cuádruplo de asignación 2. Se verifica que se haga la asignación 3. Se muestra error al usuario
Flujo Alternativo	N/A

4	Validación del índice de arreglos y matrices
Actor	Compilador
Precondiciones	Se define arreglo o matriz con un valor en el índice
Resultado	Se guarda el valor del arreglo en esa posición en la memoria virtual
Flujo	<ol style="list-style-type: none"> 1. Se verifica la sintaxis 2. Se ejecuta el cuádruplo de verificación 3. Se ejecuta el resto de los cuádruplos
Flujo Alternativo	2. El cuádruplo de verificación regresa error por fuera de rango

5	Se ejecuta una operación
Actor	Compilador
Precondiciones	Se tiene en el código una operación aritmética
Resultado	Se regresa el resultado de la operación
Flujo	<ol style="list-style-type: none"> 1. Se lee la operación del código 2. Se guarda la operación en un cuádruplo 3. Se regresa el resultado de la operación
Flujo Alternativo	N/A

Descripción de los principales test cases

- ejemplo.ppp

Demuestra la capacidad del lenguaje de una manera general. Incluye declaración de *int*, *float*, matrices y arreglos, *print*, *read*, funciones *void*, funciones *int*, además de una función factorial, los ciclos mientras (*while*) y desde (*for*), calcula el determinante y trabaja con algunas operaciones aritméticas y de comparación básicas.

- fiboyfact.ppp

Demuestra la capacidad del lenguaje en cuanto al cálculo de las funciones factoriales y de Fibonacci. Ambas las trabaja de forma cíclica o con recursividad.

- findsort.ppp

Ejemplo de la implementación de un *sort* en Patito++.

- matrix.ppp

Muestra algunas operaciones básicas de matrices y las otras especializadas en Patito++ (i, ?, \$).

Descripción del proceso general

El proceso que llevamos a cabo para el desarrollo del proyecto fue por medio de repartición de las tareas y módulos a desarrollar. Entre los dos definimos la gramática del lenguaje, además de la arquitectura general, a medida que fue avanzando el proyecto y fue subiendo de dificultad el desarrollo, nos dividimos las tareas por parte del avance que tocaba cada semana, por ejemplo: uno hacia el desarrollo de la tabla de variables, otro la memoria virtual. Además de luego explicar al otro lo que fue desarrollado, y cómo esto interactuaría con el resto del compilador, su función principal y donde y como se usaría.

Bitácoras:

Semana 1:

El avance de esta semana era crear las funciones de sintaxis, este avance si funciona, con el ejemplo del documento original, además nosotros lo modificamos para probar diferentes eventos que podrían pasar y tratamos de probar como si estuviéramos ya programando. Ahorita se marcan los errores en la línea y columna, diciendo qué se encontró y que se esperaba como regla siguiente. Ahorita podríamos mejorar algunas ambigüedades que tenemos, pero son muy pocas, dos reglas, entonces podríamos analizar esas dos y arreglarlas para el siguiente avance.

Semana 2:

La semana del 13 al 20 de abril trabajamos en el avance en cuanto a la tabla de variables y el directorio de funciones, tuvimos una ligera curva de aprendizaje en cuanto a Lark pero si logramos implementar ambas tablas. Creemos ya estar

preparados para seguir trabajando en el siguiente avance planeado. Seguimos usando el test del avance pasado y seguimos con los mismos resultados positivos.

Semana 3:

Durante la semana del 20 al 27 de abril agregamos la implementación de los cuádruplos y la tabla semántica, que en nuestro caso optamos por hacerlo en forma de clase en lugar de tabla. Con las funciones sirviendo para regresar el tipo deseado del resultado de la operación. Los cuádruplos están desarrollados por medio de la implementación de diferentes vectores, por ejemplo el de operadores, brincos, variables y tipos. En general esta semana el trabajo salió bien, todo funciona como se espera hasta el momento, y vamos al día con los avances.

Semana 4:

La semana del 27 de abril al 4 de mayo trabajamos agregando la generación de código de los estatutos condicionales, ya funcionan nuestros ciclos while, for y la condición if else. Ya confirmamos que se generan correctamente los cuádruplos ligados a todas las operaciones de nuestro programa test y los saltos ya están implementados también. Nos sentimos bien en cuanto al avance que hemos tenido en nuestro proyecto hasta la fecha. Vamos al día con las entregas.

Semana 5:

En este avance hicimos los quadriples de las funciones, fue algo simple, ya que tenemos la base de los otros quads como referencia, entonces no nos tomo mucho tiempo, también hicimos refactor de código repetido, además de avanzar la memoria virtual y la asignación de las casillas, que no se encuentra en este avance. Nos faltan algunos detalles que tenemos que tenemos en cuenta, pero en los próximos avances los cambiaremos para que no se quedan detrás.

Semana 6:

Esta entrega agregamos la maquina virtual, actualmente resuelve las expresiones además de hacer los GOTO y GOTO, nos faltan otros quads normales y los quads de funciones además de los arreglos y matrices. Necesitamos verificar lo que nos falta para no quedarnos atrás en cosas que no acabamos bien en su momento.

Semana 7:

La semana del 19 al 26 nuestro avance fue principalmente la generación de código para los arreglos y las instrucciones en la máquina virtual. Igual que las semanas anteriores seguimos con el avance al día. El compilador hasta el momento funciona como debería, los quadriplos se generan de manera correcta, al igual que la tabla de variables. Esperamos terminar el proyecto este fin de semana y comenzaremos a construir la documentación a partir de mañana.

Semana 8:

La semana del 26 al 31 de mayo trabajamos en los últimos detalles del compilador, la generación del código y máquina virtual con algunos test cases que se van a pedir en la revisión, para este punto el compilador está completo y funcional. Además

comenzamos el desarrollo de la documentación del proyecto, pero todavía le falta gran parte. No tenemos duda que para el miércoles que se tiene la entrega final tendremos todo listo.

Reflexiones:

Reflexión de Héctor:

Fue interesante aprender todo lo que lleva hacer un compilador, podría ser una tarea sencilla pero llega a ser una muy extensa al momento de hacerla. Es un claro ejemplo de aplicaciones de estructuras de datos y como nos llegan a servir para hacer muchos problemas complicados a problemas con soluciones rápidas, claro que no me imagino como lenguajes como C pueden hacer lo mismo sin las herramientas que contamos nosotros en este proyecto, ya que nos ahorramos tiempo en desarrollo por herramientas como python que nos dejan libremente manipular objetos y datos sin mucho problema. Encontré eficiente la manera que un compilador trabaja, por cómo trata de tomar ventaja de todas las fases para que las próximas lleguen a ser más fáciles y simples.

Reflexión de Rafael:

A lo largo del proyecto siento que aprendí mucho. Al principio el proyecto me parecía una tarea muy difícil y complicada, pero con el apoyo de las clases y en especial de mi compañero, poco a poco pude sentirme más tranquilo y mas confiado con mis habilidades. Agradezco a Héctor que siempre presionó sobre mantener el proyecto a tiempo, pues al final creo que sí hizo más sencillo el desarrollo, o al menos más organizado. Creo ya comprender qué es lo que hace realmente un compilador y espero este aprendizaje me sirva para poder utilizar los ya existentes de mejor manera y más eficiente. En general me gustó el curso, fue retador en momentos, pero el resultado final es muy gratificante.

Descripción del lenguaje

Nombre del lenguaje

Patito Plus Plus (Patito++) (.ppp)

Descripción del principales características del lenguaje

El lenguaje es simple pero cumple con todas las funcionalidades básicas de una buena forma, define variables, declara funciones, tiene lectura y escritura, además de ciclos, decisiones también tiene manipulación de elementos no atómicos (arreglos de hasta dos dimensiones).

Errores que pueden ocurrir

Los errores probables de compilación podrían ser:

- Declaración de variables con sintaxis incorrecta
- Cantidad incorrecto de parámetros en las funciones
- Declaración errónea de arreglos
- Falta de ; al final de cada línea
- Error al cerrar o abrir paréntesis o corchetes

Los errores probables de ejecución podrían ser:

- Error de rango al acceder elementos de arreglos
- Dividir entre 0
- Llamar a funciones de manera equívoca
- Límite de memoria al declarar demasiadas variables o arreglos

Patito++ cuenta con varios tipos de errores que se presentan cuando existe algún problema en el código:

- Error al hacer operación
- Error en cuanto a tamaños de las matrices o arreglos
- Error con operaciones exclusivas a las matrices
- Error en cuanto una matriz que debe ser cuadrada
- Error al intentar combinar una matriz y un arreglo
- Error llamando tipo de parámetros de forma incorrecta
- Error al no regresar un valor bool en un IF

Descripción del compilador

Equipo de cómputo, lenguaje y utilerías especiales usadas

Patito Plus Plus fue desarrollado en Python, con Lark para la generación de tokens, se utilizó Numpy para las operaciones de matrices.

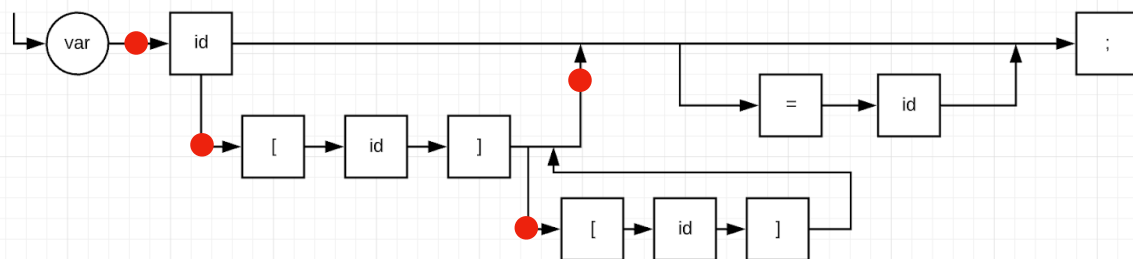
Descripción del análisis del léxico

[a-z][A-Z][0-9]*	ID
/[-+]?[0-9]*\.[0-9]+([eE][-+]?[0-9]+)?/	NUMBER
/([+-]?[1-9]\d* 0)/	INTEGER
"True" "False"	BOOLEAN
(LPAREN
)	RPAREN
{	LBRACE
}	RBRACE
[LBRAKE
]	LBRAKE
;	SEMI
:	COLON
=	ASSIGN
+	PLUS
-	MINUS
*	TIMES
/	DIVIDE
<	LESS

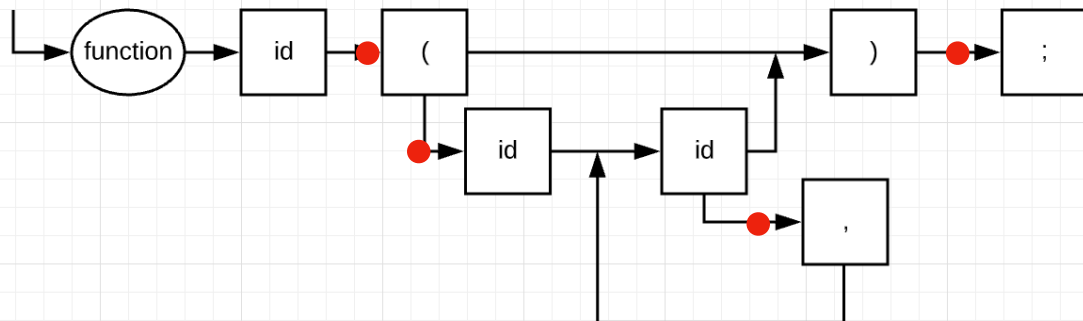
>	GRATER
==	EQUAL
.	DOT
&	AND
	OR
"!=" "<>"	DIFF
\$	SIGN
i	EXCLD
?	QUES
"%%%" /(. \n \\r)+/	COMMENT

Descripción del análisis de sintaxis

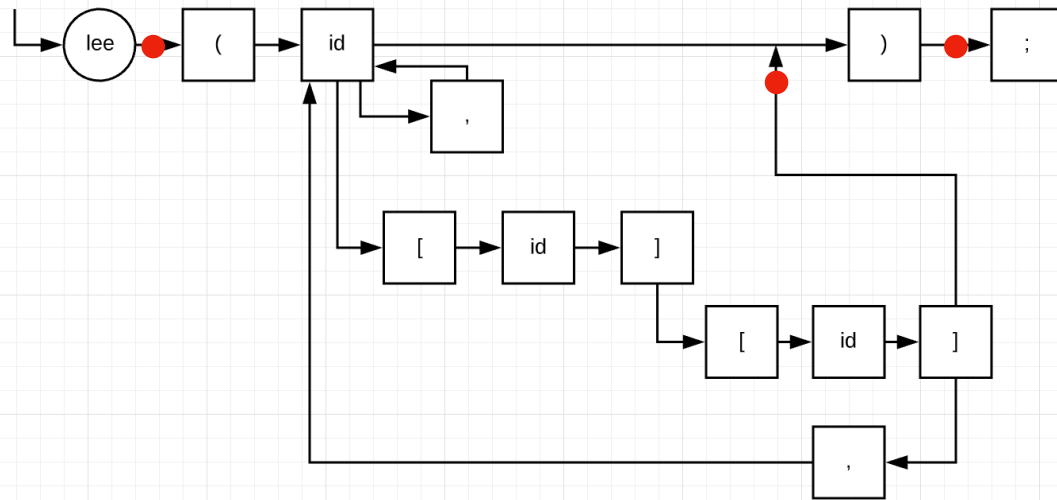
<dec-var>



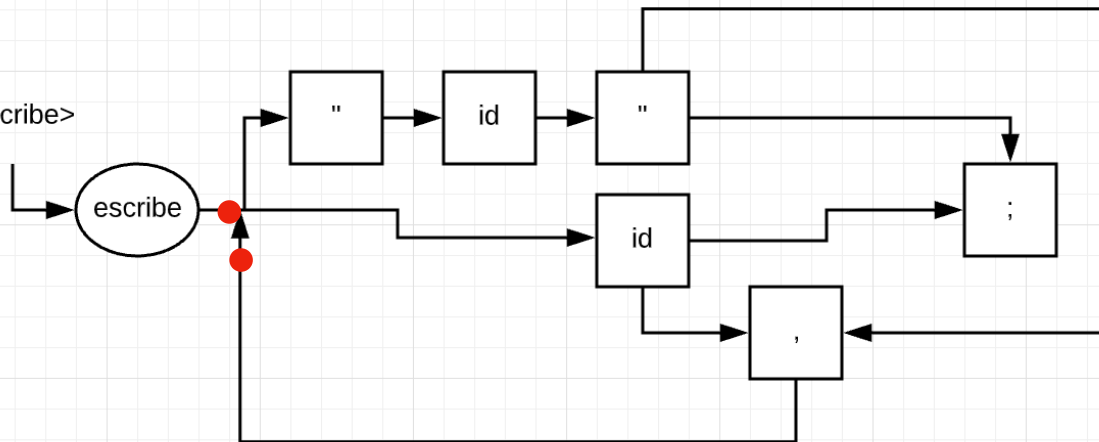
<dec-function>



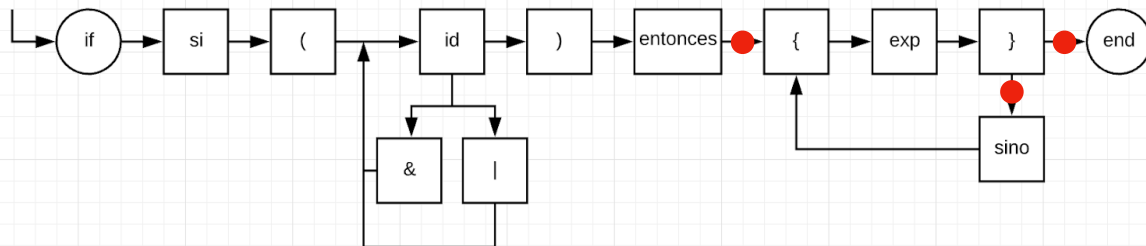
<lee>



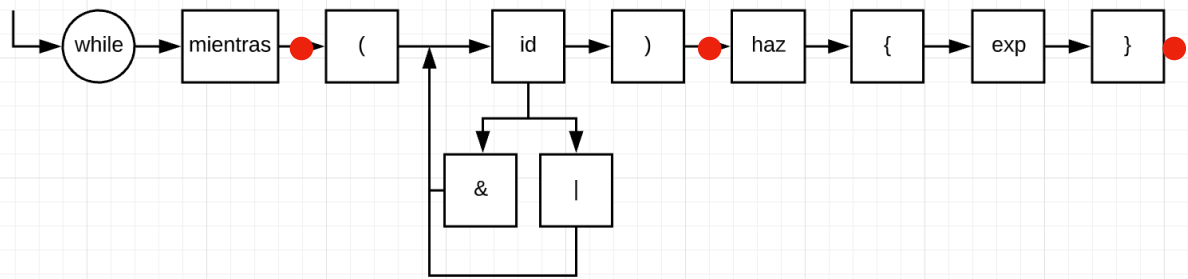
<escribe>



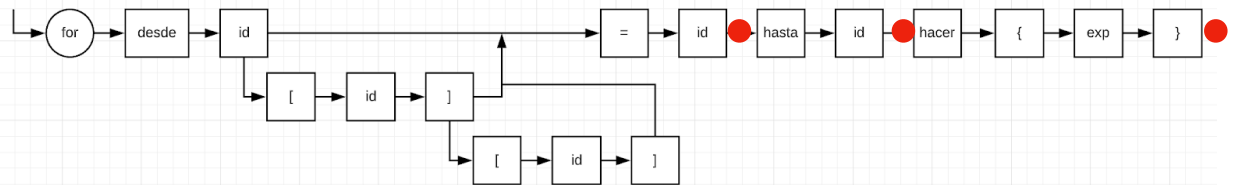
<if>



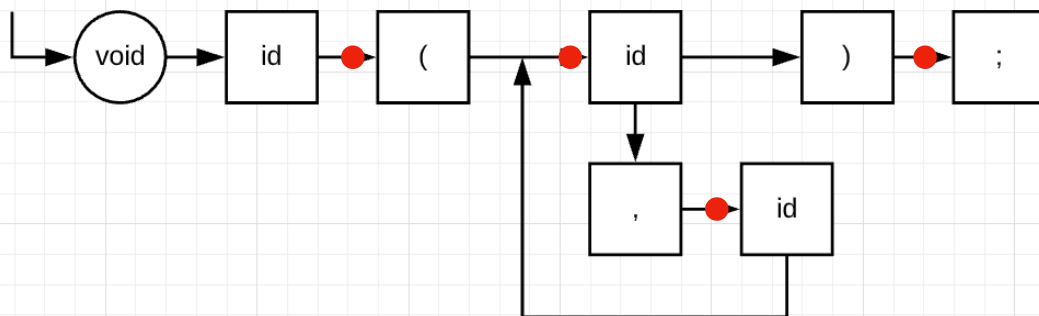
<while>



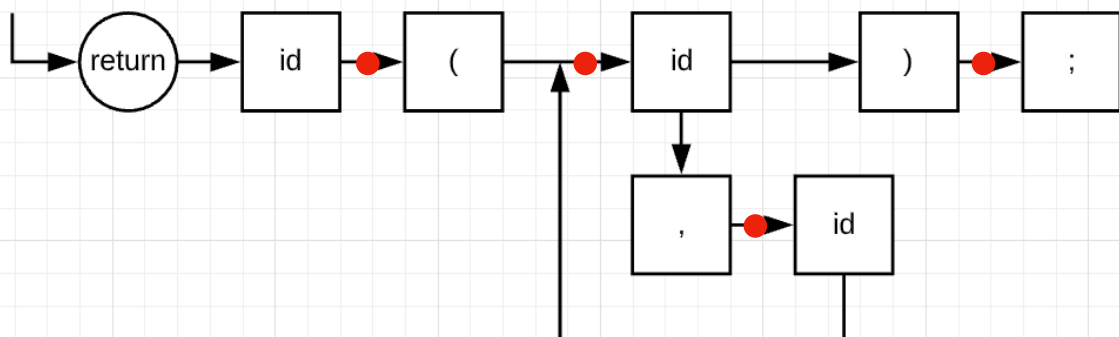
<for>

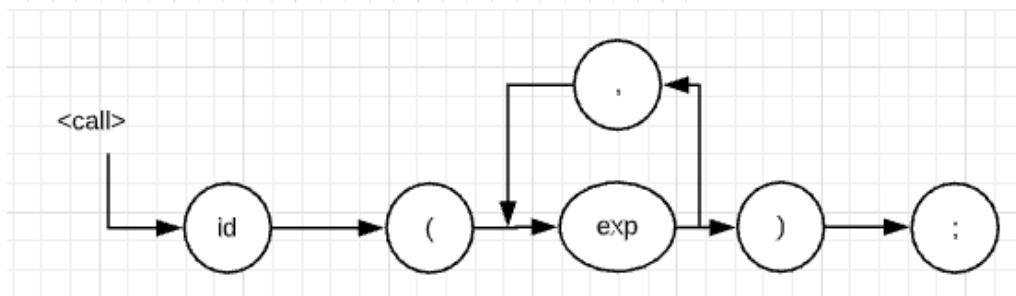
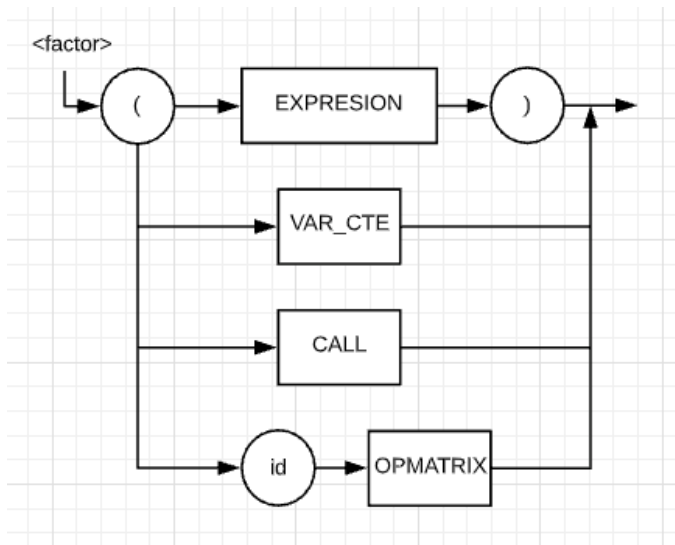
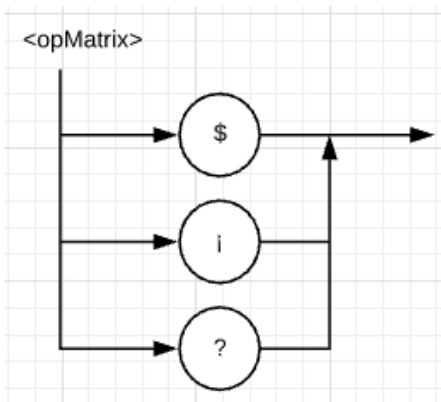
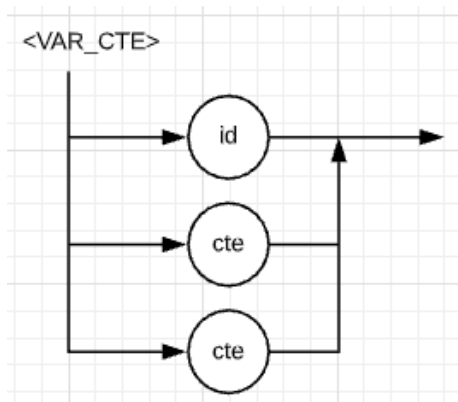
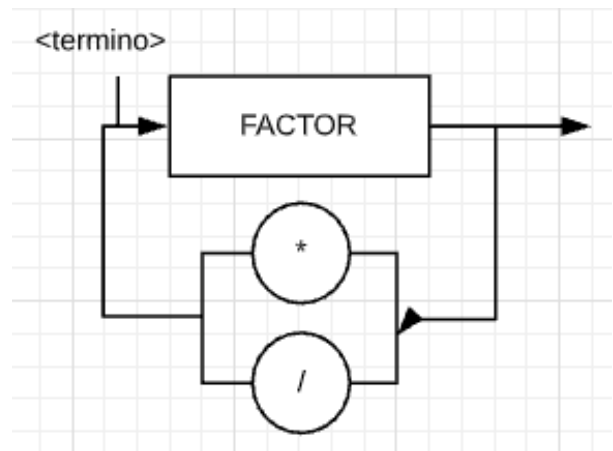
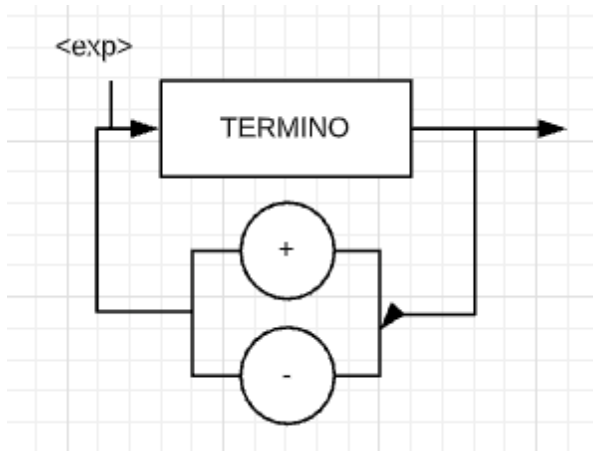


<void-func>



<return-func>





Descripción de generación de código intermedio y análisis semántico

El código intermedio de Patito++ funciona por medio de la generación de cuádruplos que la máquina virtual procesará, todos los cuádruplos tienen el formato de:

[operador, operandolzquierdo, operandoDerecho, resultado]

Las operaciones posibles a realizar en un cuádruplo son:

- +
- -
- *
- /
- >
- <
- >=
- <=
- ==
- !=
- AND
- OR
- Goto
- GotoF
- print
- read
- \$
- i
- ?
- ===M
- ==M
- +++
- ---
- ***
- ///
- ++
- --
- **
- //
- SIZE
- ERA
- Ver
- GOSUB
- PARAMETER
- ENDFunc
- return

Descripción del proceso de administración de memoria

La memoria se administra con los siguientes rangos:

Globales		
Int	0	999
Float	1000	1999
Char	2000	2999
Bool	3000	3999
Locales		

Int	4000	4999
Float	5000	5999
Char	6000	6999
Bool	7000	7999
Temporales		
Int	8000	8999
Float	9000	9999
Char	10000	10999
Bool	11000	11999
Constantes		
Int	12000	12999
Float	13000	13999
Char	14000	14999
Bool	15000	15999
Pointers		
Int	1.6×10^{19}	$1.6 \times 10^{20} - 1$
Float	1.6×10^{20}	$1.6 \times 10^{21} - 1$
Char	1.6×10^{21}	$1.6 \times 10^{22} - 1$
Bool	1.6×10^{22}	$1.6 \times 10^{23} - 1$

```

programa patito;
var
int x, resul;

funcion int fibo (int x)
var int y;
{
    si (x > 1) entonces {
        y = fibo(x - 1);
        regresa(x * y);
    }
    regresa(1);
}

principal () {
    x = 5;
    resul = fibo(x);
    escribe(resul);
}

```

Ejemplo del código intermedio con Fibonacci sencillo:

Cuádruplos generados:

```
0 ['Goto', None, None, 13]
1 ['>', 4000, 12000, 11000]
2 ['GotoF', 11000, None, 11]
3 ['ERA', 'fibo', None, None]
4 ['-', 4000, 12000, 8000]
5 ['PARAMETER', 8000, None, 1]
6 ['GOSUB', None, None, 'fibo']
7 ['=', 4002, None, 2]
8 ['=', 4001, None, 4002]
9 ['*', 4000, 4001, 8001]
10 ['return', None, None, 8001]
11 ['return', None, None, 12000]
12 ['ENDFunc', None, None, None]
13 ['=', 0, None, 12001]
14 ['ERA', 'fibo', None, None]
15 ['PARAMETER', 0, None, 1]
16 ['GOSUB', None, None, 'fibo']
17 ['=', 3, None, 2]
18 ['=', 1, None, 3]
19 ['print', None, None, 1]
```

Estos cuádruplos representan cada una de las operaciones a desarrollarse durante la ejecución del código, desde los saltos hasta las operaciones.

Cuádruplo 1:

- Operador: comparación mayor que
- OperandoIzquierdo: variable en la dirección de memoria virtual 4000 (entero local)
- OperandoDerecho: variable en la dirección de memoria virtual 12000 (entero constante)
- Resultado: variable en la dirección de memoria virtual 11000 (bool temporal)

Con los elementos anteriores se llega a un resultado temporal que el siguiente cuádruplo usará para seguir compilando el programa.

Tabla de variables globales

__global__				
type				
vars		VOID		
		x		
	type		int	
	value		0	
	dir		0	
	dim		0	
	arrList		[]	
		resul		
	type		int	
	value		0	
	dir		1	
	dim		0	
	arrList		[]	
		fibo		
	type		int	
	value		0	
	dir		2	
	dim		0	
	arrList		[]	

La tabla de variables muestra cómo globalmente se tienen guardados 3 variables diferentes empezando en la posición absoluta 0. Se registra la función *fibo*, pues en esa dirección de memoria se va a guardar el retorno de la misma.

Tabla de variables locales

fibo			
type	int		
vars	x		
		type	int
		value	0
		dir	4000
		dim	0
		arrList	[]
	y		
		type	int
		value	0
		dir	4001
		dim	0
		arrList	[]
params			
	0	var	x
		type	int
	params_size		
	1		
	local_size		
	1		
	quad_count		
	1		

La tabla de variables locales funciona igual que la global, solo que aquí se guardan las que están declaradas en la función *fibo*. También incluye la sección de params, donde se guarda la cantidad, tipo y nombre de los parámetros de la función

Tabla de constantes

__cte__			
type	VOID		
vars	1		
		type	int
		value	1
		dir	12000
	5		
		type	int
		value	5
		dir	12001

La tabla de constantes guarda todos los valores constantes en direcciones específicas de memoria, con la intención de manipular todo por medio de direcciones de memoria virtual en los cuádruplos.

Clase semántica

Optamos por definir la semántica por medio de una clase donde se recibe el operador y los dos tipos a trabajarse y se regresa el tipo del resultado correspondiente

```
def result(self, left, right, op):
    if op == "+" or op == "-":
        if left == "float" and right == "int":
            return "float"
        elif left == "int" and right == "float":
            return "float"
        elif left == "int" and right == "int":
            return "int"
        elif left == "float" and right == "float":
            return "float"
        elif left == "char" and right == "char":
            return "char"
    else:
        return False
```

El código anterior es el ejemplo de cuando el operador es + o - de todos los casos de tipos que se puedan encontrar, además está desarrollado el código para los demás operadores.

Descripción de la máquina virtual

Librerías especiales

Para desarrollar la máquina virtual se utilizó la librería Numpy, la cual fue de gran utilidad para trabajar las operaciones de matrices.

Proceso de administración de memoria en ejecución

El manejo de memoria se utiliza principalmente stacks, para ambas memorias local y temporal.

Existen tres funciones desarrolladas en la máquina virtual:

- *getVarValue*: la función cumple con la tarea de regresar la dirección virtual actual de la variable pedida por alguna operación ejecutada en *runMachine*
- *setVarValue*: la función cumple con la tarea de asignar una dirección virtual a alguna variable deseada, además de agregarla a la tabla de variables correspondiente, ya sea global, local, temporal o pointer.
- *runMachine*: esta función cumple con la tarea de ejecutar las operaciones que se incluyen al principio de los cuádruplos, además de usar *getVarValue* y *setVarValue* para mantener un registro correcto de las variables en todo el programa.

Pruebas del funcionamiento del lenguaje

Ejemplo 1 fiboyfact.ppp:

```
programa patito;
var
int i, j, k, x, f, z, y;
char c ,l , m;
int A[2][2];
float B[2][2];

funcion void inicia(int z)
var
int h, i;
{
    i = 5;
    mientras (i > 0) haz{
        escribe(i);
        i = i-1;
    }
}

funcion int fact(int x)
var int y;
{
    si (x > 1) entonces {
        y = fact(x - 1);
        regresa(x * y);
    }
    regresa(1);
}

funcion int fibo(int x)
{
    si (x == 1 | x == 0) entonces {
        regresa(x);
    } sino {
        regresa(fibo(x - 1) + fibo(x - 2));
    }
}

%% Comment
principal()
{
    x = 10;
    f = 1;
    desde (i = 1) hasta (i < x) haz {
        f = f + f * i;
    }
}
```

```

    escribe(f);
    escribe(fact(10));
    l = "h" * 5;
    escribe(l);
    i = 0;
    k = 0;
    z = 0;
    y = 1;
    mientras (i < x) haz {
        z = k + y;
        k = y;
        y = z;
        i = i + 1;
    }
    escribe(k);
    escribe(fibo(10));
}

```

Resultado:

```

|Rafaels-MacBook-Pro:patitoplusplus rafaelserna$ python3 analizador_lark.py ./test/fiboyfact.ppp
3628800
3628800
hhhhh
55
55

```

Ejemplo 2 (findsort.ppp)

```

programa patito;
var
int A[5];
int B[2];
int i, j, xtmp, ytmp, n;

funcion int find(int x, int size)
var int i;
{
    i = 0;
    mientras (i <= size) haz {
        si (A[i] == x) entonces {
            regresa(i);
        }
        i = i + 1;
    }
    regresa(-1);
}

principal()
{
    i = 0;
    mientras (i < 5) haz {

```

```

    A[i] = 23;
    i = i + 1;
}
A[4] = 5;
escribe(find(5, 5));

A[0] = 30;
A[1] = 23;
A[2] = 12;
A[3] = 1;
A[5] = 22;
i = 0;
j = 0;
escribe(A);
mientras (i < 5 - 1) haz {
    j = 0;
    mientras (j < (5 - i - 1)) haz {
        si (A[j] > A[j + 1]) entonces {
            xtmp = A[j];
            A[j] = A[j + 1];
            A[j + 1] = xtmp;
        }
        j = j + 1;
    }
    i = i + 1;
}
B[0] = 2;
n = B[0];
escribe(B[0]);
A[B[0]] = 300;
escribe(A[B[0]]);
escribe(A);
}

```

Resultado:

```

[Rafaels-MacBook-Pro:patitoplusplus rafaelserna$ python3 analizador_lark.py ./test/findsort.ppp
4
30 23 12 1 5
2
300
1 5 300 23 30

```

Ejemplo 3 (matrix.ppp)

```

programa patito;
var
int A[3][5];
int B[3][5];
int C[2][2];
int i, j, xtmp, ytmp, n;

```

```

principal()
{
    B[1][1] = 10;
    A[2][2] = B[1][1];
    escribe(A[2][2]);
    escribe(B[1][1]);

    i = 0;
    mientras (i < 3) haz {
        j = 0;
        mientras (j < 5) haz {
            A[i][j] = 3;
            B[i][j] = 2;
            j = j + 1;
        }
        i = i + 1;
    }
    escribe(A * B);
    escribe(A + B);
    escribe(A - B);
    C[0][0] = 4;
    C[0][1] = 7;
    C[1][0] = 2;
    C[1][1] = 6;
    escribe("Determinante", C$);
    escribe("Inversa", C?);
    escribe("Transpuesta", Ci);
    A = B;
    escribe(A);
}

```

Resultado:

```

[Rafaels-MacBook-Pro:patitoplusplus rafaelserna$ python3 analizador_lark.py ./test/matrix.ppp
10
10
6 6 6 6 6
6 6 6 6 6
6 6 6 6 6
5 5 5 5 5
5 5 5 5 5
5 5 5 5 5
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
Determinante
10.000000000000002
Inversa
0.6000000000000001 -0.7000000000000001
-0.2 0.4
Transpuesta
4 2
7 6
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2

```


Listados perfectamente documentados del proyecto

Quadruple.py: clase donde se define la estructura de los cuádruplos

```
class Quadruple:
    def __init__(self, op, leftop, rightop, result):
        self.op = op
        self.quad = [op, leftop, rightop, result]

    def getQuad(self):
        return self.quad
```

Semantic.py: clase donde se definen las reglas semánticas

```
class Semantic:
    def __init__(self):
        self.table = [[]]

    def result(self, left, right, op):
        if op == "+" or op == "-":
            if left == "float" and right == "int":
                return "float"
            elif left == "int" and right == "float":
                return "float"
            elif left == "int" and right == "int":
                return "int"
            elif left == "float" and right == "float":
                return "float"
            elif left == "char" and right == "char":
                return "char"
            else:
                return False
        elif op == "*":
            if left == "float" and right == "int":
                return "float"
            elif left == "int" and right == "float":
                return "float"
            elif left == "int" and right == "int":
                return "int"
            elif left == "float" and right == "float":
                return "float"
            elif left == "char" and right == "int":
                return "char"
            elif left == "int" and right == "char":
                return "char"
            else:
                return False
```

```

elif op == "/":
    if left == "float" and right == "int":
        return "float"
    elif left == "int" and right == "float":
        return "float"
    elif left == "int" and right == "int":
        return "float"
    elif left == "float" and right == "float":
        return "float"
    else:
        return False
elif op == ">" or op == "<" or op == "==":
    if left == "int" and right == "int":
        return "bool"
    elif left == "int" and right == "float":
        return "bool"
    elif left == "float" and right == "int":
        return "bool"
    else:
        return False
elif op == "AND" or op == "OR":
    if left == "bool" and right == "bool":
        return "bool"
    else:
        return False
elif op == "$":
    if left == "float" and right == "float":
        return "float"
    elif left == "int" and right == "int":
        return "float"
    else:
        return False
elif op == "?":
    if left == "float" and right == "float":
        return "float"
    elif left == "int" and right == "int":
        return "float"
    else:
        return False
elif op == "i":
    if left == "float" and right == "float":
        return "float"
    elif left == "int" and right == "int":
        return "int"
    else:
        return False
elif op == "return":
    return left == right
elif op == "print":
    return True

```

Stack.py: clase donde se define la estructura de datos más usada en el compilador

```
class Stack:
    def __init__(self, toPrint):
        self.toPrint = toPrint
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.insert(0,item)
        if self.toPrint:
            print("Push", self.items)

    def pop(self):
        popped = self.items.pop(0)
        if self.toPrint:
            print("Pop", self.items)
        return popped

    def peek(self):
        if self.toPrint:
            print("Peek", self.items)
        if self.size() > 0:
            return self.items[0]
        return None

    def size(self):
        return len(self.items)

    def clean(self):
        self.items.clear()

    def print(self):
        print(self.items)
```

Memoriavirtual.py: clase donde se define la memoria virtual, se consiguen direcciones y se asignan también

```
# Valor de pointer
pointerMin = 160000*1000000000000000
# Crear dirs para semantica
class MemoriaVirtual:
    def __init__(self):
        self.memVirtual = {
            "global": {"int": -1, "float": 999, "char": 1999, "bool": 2999},
            "local": {"int": 3999, "float": 4999, "char": 5999, "bool": 6999},
            "temp": {"int": 7999, "float": 8999, "char": 9999, "bool": 10999},
            "cte": {"int": 11999, "float": 12999, "char": 13999, "bool": 14999},
```

```

        "pointer": {"int": pointerMin - 1, "float": pointerMin - 1 + 1000,
"char": pointerMin - 1 + 2000, "bool": pointerMin - 1 + 3000}}

    def getAddress(self, indice, scope):
        if scope != False:
            self.memVirtual[indice][scope] = self.memVirtual[indice][scope] + 1
            return self.memVirtual[indice][scope]

    def setNextAddress(self, indice, scope, value):
        if scope != False:
            self.memVirtual[indice][scope] = self.memVirtual[indice][scope] +
value - 1

class Memory:
    # Se hace init con la memoria vacia
    def __init__(self, inicial):
        self.arrInt = {}
        self.arrFloat = {}
        self.arrChar = {}
        self.arrBool = {}
        self.valInicial = inicial

    def setVar(self, address, variable):
        address = address - self.valInicial
        if address >= 0 and address < 1000:
            self.arrInt[address] = int(variable)
        elif address >= 1000 and address < 2000:
            self.arrFloat[address - 1000] = float(variable)
        elif address >= 2000 and address < 3000:
            self.arrChar[address - 2000] = variable
        elif address >= 3000 and address < 4000:
            self.arrBool[address - 3000] = variable
        else:
            return False

    def getVar(self, address):
        address = address - self.valInicial
        if address >= 0 and address < 1000:
            if address in self.arrInt:
                return int(self.arrInt[address])
            else:
                return -1
        elif address >= 1000 and address < 2000:
            address = address - 1000
            if address in self.arrFloat:
                return float(self.arrFloat[address])
            else:
                return -1
        elif address >= 2000 and address < 3000:
            return self.arrChar[address - 2000]
        elif address >= 3000 and address < 4000:
            return self.arrBool[address - 3000]
        else:

```

```

        return False

    def printInt(self):
        print(self.arrInt)

    def printBool(self):
        print(self.arrBool)

    def printFloat(self):
        print(self.arrFloat)

    def clearMemory(self):
        self.arrInt = {}
        self.arrFloat = {}
        self.arrChar = {}
        self.arrBool = {}

```

virtualMachine.py: se trabajan todas las operaciones de los cuádruplos, además de trabajarse todas las direcciones virtuales

```

from MemoriaVirtual import *
from Stack import *
import numpy as np

# La memoria init para el pointer
pointerMin = 160000*10000000000000000

# Se hacen las memorias iniciales
# Para local y tmp se tienen stacks de memoria
globalMem = Memory(0)
cteMem = Memory(12000)
pointMem = Memory(pointerMin)
funcNames = Stack(False)
quadPos = Stack(False)
memStack = Stack(False)
tmpStack = Stack(False)
tmpMem = Memory(8000)
tmpStack.push(tmpMem)

# Se hacen las operaciones
ops = {
    "+": (lambda a,b: a+b),
    "-": (lambda a,b: a-b),
    "*": (lambda a,b: a*b),
    "/": (lambda a,b: a/b),
    "AND": (lambda a,b: a and b),
    "OR": (lambda a,b: a or b),
    ">": (lambda a,b: a > b),
    ">=": (lambda a,b: a >= b),
    "==" : (lambda a,b: a == b),
    "<": (lambda a,b: a < b),
    "!=": (lambda a,b: a != b),

```

```

    "<=": (lambda a,b: a <= b)
}

def pretty(d, indent=0):
    for key, value in d.items():
        print('\t' * indent + str(key))
        if isinstance(value, dict):
            pretty(value, indent+1)
        else:
            print('\t' * (indent+1) + str(value))

# obtiene el valor de la variable en base al numero de dir
def getVarValue(dirVar):
    if dirVar >= pointerMin:
        currVar = pointMem.getVar(dirVar)
    elif dirVar >= 12000:
        currVar = cteMem.getVar(dirVar)
    elif dirVar >= 8000:
        tmpMem = tmpStack.peak()
        currVar = tmpMem.getVar(dirVar)
    elif dirVar >= 4000:
        localMem = memStack.peak()
        currVar = localMem.getVar(dirVar)
    elif dirVar >= 0:
        currVar = globalMem.getVar(dirVar)
    return currVar

# hace set al valor de la variable en base al numero de dir
def setVarValue(dirVar, value):
    if dirVar >= pointerMin:
        currVar = pointMem.setVar(dirVar, value)
    elif dirVar >= 12000:
        currVar = cteMem.setVar(dirVar, value)
    elif dirVar >= 8000:
        tmpMem = tmpStack.peak()
        currVar = tmpMem.setVar(dirVar, value)
    elif dirVar >= 4000:
        localMem = memStack.peak()
        currVar = localMem.setVar(dirVar, value)
    elif dirVar >= 0:
        currVar = globalMem.setVar(dirVar, value)
    return currVar

def runMachine(quadruples, functions):
    # Se crean las tablas para function global y las ctes que se definieron
    for currVar in functions["__cte__"]["vars"]:
        cteMem.setVar(functions["__cte__"]["vars"][currVar]["dir"], functions["__cte__"]["vars"][currVar]["value"])

    for currVar in functions["__global__"]["vars"]:
        globalMem.setVar(functions["__global__"]["vars"][currVar]["dir"], functions["__global__"]["vars"][currVar]["value"])

```

```

i = -1
# Para la op de arreglos
currentArraySize = 0
currentArrayWH = [0,0]
# Tmps para llamadas
newTmp = None
newMem = None
while i < len(quadruples):
    i = i + 1

    if i >= len(quadruples):
        break

    [op, leftDir, rightDir, resDir] = quadruples[i]

    # Guarda el resultado en la memoria
    # Si el valor es un pointer se vuelve a acceder para el resultado
    if op == "=":
        currVar = getVarValue(leftDir)
        if currVar > pointerMin - 1:
            leftDir = currVar
        resVar = getVarValue(resDir)
        if isinstance(resVar, int) and resVar > pointerMin - 1:
            resVar = getVarValue(resVar)
        setVarValue(leftDir, resVar)

    # Genera el resultado y lo guarda en la memoria
    # Si el valor es un pointer se vuelve a acceder para el resultado
    if op in ["+", "-", "/", "*", "<", ">", "=", "!", "<=", ">=", "AND", "OR"]:
        if leftDir > pointerMin - 1:
            leftVar = leftDir
        else:
            leftVar = getVarValue(leftDir)
            if isinstance(leftVar, int) and leftVar > pointerMin - 1:
                leftVar = getVarValue(leftVar)
        if rightDir > pointerMin - 1:
            rightVar = rightDir
        else:
            rightVar = getVarValue(rightDir)
            if isinstance(rightVar, int) and rightVar > pointerMin - 1:
                rightVar = getVarValue(rightVar)
        res = ops[op](leftVar, rightVar)
        setVarValue(resDir, res)

    # Genera el resultado y lo guarda en la memoria
    # Si el valor es un pointer se vuelve a acceder para el resultado
    if op in ["+", "-", "/", "*", "<", ">", "=", "!", "<=", ">=", "AND", "OR"]:
        if leftDir > pointerMin - 1:
            leftVar = leftDir
        else:
            leftVar = getVarValue(leftDir)
            if isinstance(leftVar, int) and leftVar > pointerMin - 1:
                leftVar = getVarValue(leftVar)

```

```

if rightDir > pointerMin - 1:
    rightVar = rightDir
else:
    rightVar = getVarValue(rightDir)
    if isinstance(rightVar, int) and rightVar > pointerMin - 1:
        rightVar = getVarValue(rightVar)
    res = ops[op](leftVar, rightVar)
    setVarValue(resDir, res)

# Cambia el i al valor del goto
if op == "Goto":
    i = resDir - 1

if op == "GotoF":
    resBool = getVarValue(leftDir)
    if resBool == False:
        i = resDir - 1

# se ejecuta la action de print
if op == "print":
    # Los arreglos y matrices se imprimen diferente
    if isinstance(resDir, int) and resDir > pointerMin - 1:
        size = currentArraySize
        resultPointer = resDir
        [w, h] = currentArrayWH
        index = 0
        cw = 0
        # Si es de una dim se imprime como lista
        if h == None:
            while cw < w:
                value = getVarValue(resDir + cw)
                print(value, end=' ')
                cw = cw + 1
            print("")
        else:
            # Se imprime como matrix
            # el off sirve para el salto de renglon
            off = 0
            while cw < w:
                j = 0
                while j < h:
                    value = getVarValue(resDir + j + cw + off)
                    print(value, end=' ')
                    j = j + 1
                off = off + 1
                cw = cw + 1
            print("")
        else:
            # Se imprime normal, si es un pointer al igual se busca el valor real
            # para momentos como b = 16000000000;
            val = getVarValue(resDir)
            if isinstance(val, int) and val > pointerMin - 1:
                val = getVarValue(val)

```



```

    print(val)
# Se lee el input y se guarda
if op == "read":
    value = input()
    setVarValue(resDir, value)

# Operacion para matrix
if op == "$":
    [w, h] = currentArrayWH
    # Se genera matrix temporal con los valores de tamano
    Matrix = [[0 for x in range(w)] for y in range(h)]
    M = leftDir
    cw = 0
    off= 0
    # Se guarda en la matrix tmp
    while cw < w:
        j = 0
        while j < h:
            value = getVarValue(leftDir + j + cw + off)
            Matrix[cw][j] = value
            j = j + 1
            off=off+1
            cw = cw + 1
    # Se saca la determinante
    res = np.linalg.det(Matrix)
    setVarValue(resDir, res)

if op == "i":
    [w, h] = currentArrayWH
    Matrix = [[0 for x in range(h)] for y in range(w)]
    M = leftDir
    cw = 0
    off=0
    while cw < w:
        j = 0
        while j < h:
            value = getVarValue(leftDir + j + cw + off)
            Matrix[cw][j] = value
            j = j + 1
            off=off+1
            cw = cw + 1
    # Se hace la transpuesta
    m = np.array(Matrix)
    res = m.T
    cw = 0
    j = 0
    off=0
    # Se guarda el resultado en la direction del pointer tmp
    while cw < h:
        j = 0
        while j < w:
            setVarValue(resDir + j + cw + off, res[cw][j])
            j = j + 1

```

```

        off=off+1
        cw = cw + 1

if op == "?":
    [w, h] = currentArrayWH
    Matrix = [[0 for x in range(h)] for y in range(w)]
    M = leftDir
    cw = 0
    off=0
    while cw < w:
        j = 0
        while j < h:
            value = getVarValue(leftDir + j + cw + off)
            Matrix[cw][j] = value
            j = j + 1
        off=off+1
        cw = cw + 1
    # Se saca la inversa
    m = np.array(Matrix)
    res = np.linalg.inv(m)
    cw = 0
    j = 0
    off = 0
    while cw < w:
        j = 0
        while j < h:
            setVarValue(resDir + j + cw + off, res[cw][j])
            j = j + 1
        off = off + 1
        cw = cw + 1

# Asignacion especial para matriz y lista
if op in ["==M", "=="M"]:
    size = currentArraySize
    leftPointer = leftDir
    resultPointer = resDir
    index = 0
    # Igual que la op pero se guarda en el res
    while index < size:
        resultPointerValue = getVarValue(resultPointer)
        setVarValue(leftPointer, resultPointerValue)
        index = index + 1
        leftPointer = leftPointer + 1
        resultPointer = resultPointer + 1

# Operaciones de matrices y arreglos
if op in ["+++", "---", "***", "///", "++"]:
    size = currentArraySize
    leftPointer = leftDir
    rightPointer = rightDir
    resultPointer = resDir
    index = 0
    # Matriz y lista se tratan igual en este caso, se genera la op por cada uno

```

```

while index < size:
    leftPointerValue = getVarValue(leftPointer)
    rightPointerValue = getVarValue(rightPointer)
    res = ops[op[0]](leftPointerValue, rightPointerValue)
    setVarValue(resultPointer, res)
    index = index + 1
    leftPointer = leftPointer + 1
    rightPointer = rightPointer + 1
    resultPointer = resultPointer + 1

# Se guardan las dimensiones
if op == "SIZE":
    currentArraySize = resDir
    currentArrayWH[0] = leftDir
    currentArrayWH[1] = rightDir

# Se generan las memorias proximas
if op == "ERA":
    # Para recursividad
    funcNames.push(leftDir)
    newMem = Memory(4000)
    newTmp = Memory(8000)
    # Se hacen init a las vars
    for currVar in functions[leftDir]["vars"]:
        newMem.setVar(functions[leftDir]["vars"][currVar]["dir"], functions[leftDir]["vars"][currVar]
["value"])

# Se verifica los rangos
if op == "Ver":
    target = getVarValue(leftDir)
    if target > pointerMin - 1:
        target = getVarValue(target)
    limInf = getVarValue(rightDir)
    limSup = getVarValue(resDir)
    if target <= limInf or target > limSup:
        print("Error en rango de indice", target, limInf, limSup)
        break

# Pasa la memoria al stack y se cambia la i, y se guarda su pos
if op == "GOSUB":
    quadPos.push(i)
    memStack.push(newMem)
    tmpStack.push(newTmp)
    i = functions[funcNames.peak()][ 'quad_count' ] - 1

# se hace init de cada param, en la mem nueva, que no esta en el stack todavia
if op == "PARAMETER":
    param = getVarValue(leftDir)
    nameVar = functions[funcNames.peak()][ 'params' ][resDir-1][ 'var' ]
    dirVar = functions[funcNames.peak()][ 'vars' ][nameVar][ 'dir' ]
    newMem.setVar(dirVar, param)

# Se borra la memoria y se regresa a la i original

```

```

if op == "ENDFunc":
    i = quadPos.pop()
    funcNames.pop()
    memStack.pop()
    tmpStack.pop()

# Se guarda el return en el valor, y se trata como un endfunc
if op == "return":
    dirToRes = functions['__global__']['vars'][funcNames.peek()]['dir']
    resValue = getVarValue(resDir)
    setVarValue(dirToRes, resValue)
    i = quadPos.pop()
    funcNames.pop()
    memStack.pop()
    tmpStack.pop()

```

Transformer.py: se encarga de convertir todos los tokens para luego mandarlo a la máquina virtual y trabajar los cuádruplos

```

from lark import Transformer, Tree
from Quadruple import *
from Stack import *
from Semantic import *
from MemoriaVirtual import *
from virtualMachine import *

# Se definen los stacks para operacion
memVirtual = MemoriaVirtual()
stackOp = Stack(False)
stackJumps = Stack(False)
stackVar = Stack(False)
# Stack para operaciones en arreglos/matrices
stackArrs = Stack(False)
stackArrName = Stack(False)
stackDim = Stack(False)
stackType = Stack(False)
# Init de tabla semantica
semTable = Semantic()
currParams = Stack(False)
# Stack para el trato de vars en el for loop
# Para el auto incremento de la variable
forloopvar = Stack(False)
quadruples = []
t_num = 0

# Para imprimir objetos y listas claramente
def pretty(d, indent=0):
    for key, value in d.items():
        print('\t' * indent + str(key))
        if isinstance(value, dict):
            pretty(value, indent+1)
        else:

```

```

        print('\t' * (indent+1) + str(value))

def prettyList(d, indent=0):
    count = 0
    for value in d:
        print(str(count) + " " + '\t' * indent + str(value))
        count = count + 1

# Generacion de quadruples de logica
def genQuadOpLog(cond):
    if stackOp.size() > 0:
        top = stackOp.peek()
        if top in cond:
            right_operand = stackVar.pop()
            right_type = stackType.pop()
            left_operand = stackVar.pop()
            left_type = stackType.pop()
            operator = stackOp.pop()
            result_type = semTable.result(left_type, right_type, operator)
            if result_type != False:
                posMemVirtual = memVirtual.getAddress('temp', result_type)
                quad = Quadruple(operator, left_operand, right_operand,
posMemVirtual)
                quadruples.append(quad.getQuad())
                stackVar.push(posMemVirtual)
                stackType.push(result_type)
            else:
                raise TypeError("ERROR: You can't perform that operation! 😞")

# Generacion de quadruples de expresion
def genQuadOpExp(cond):
    if stackOp.size() > 0:
        top = stackOp.peek()
        if top in cond:
            right_operand = stackVar.pop()
            right_type = stackType.pop()
            # Cuando son estas ops se genera el res asi mismo, el left y right
operand son iguales
            if top not in ["$", "?", "!"]:
                left_operand = stackVar.pop()
                left_type = stackType.pop()
            else:
                left_operand = right_operand
                left_type = right_type
            operator = stackOp.pop()
            result_type = semTable.result(left_type, right_type, operator)
            if result_type != False:
                posMemVirtual = 0
                # Se verifica que sea una op de arreglos o matrices
                if stackArrs.size() > 0:
                    leftArr = stackArrs.pop()
                    size = 1
                    # Si es una + - * se toma en cuenta el right

```

```

if stackArrs.size() > 0 and top not in ["$", "?", "i"]:
    rightArr = stackArrs.pop()
    # Se verifican que sea de la misma dim
    if len(leftArr['arrList']) != len(rightArr['arrList']):
        raise TypeError("ERROR: A list with a Matrix")
    index = 0
    # Se verifica que tengan el mismo numero de col y rows
    # Ademas de guardar el tamano de la matriz y num de cols rows
    while (index < len(leftArr['arrList'])):
        if leftArr['arrList'][index][0] != rightArr['arrList'][index][0]:
            raise TypeError("ERROR: Different sizes for operation 😞")
        size = size * leftArr['arrList'][index][0]
        index = index + 1
        operator = operator + operator[0]
else:
    if top in ["$", "?", "i"] and len(leftArr['arrList']) != 2:
        raise TypeError("ERROR: This operation can be only done in Matrix 😞")
    index = 0
    # Guarda el tamano y el row y col
    while (index < len(leftArr['arrList'])):
        size = size * leftArr['arrList'][index][0]
        index = index + 1
    dimtwo = None
    if len(leftArr['arrList']) > 1:
        dimtwo = leftArr['arrList'][1][0]
    # Se genera un quad size que contiene los tamanos de la matriz o lista
    quad = Quadruple("SIZE", leftArr['arrList'][0][0], dimtwo, size)
    quadruples.append(quad.getQuad())
    posMemVirtual = None
    if operator == "$":
        if leftArr['arrList'][0][0] != leftArr['arrList'][1][0]:
            raise TypeError("ERROR: Matrix must be square")
        posMemVirtual = memVirtual.getAddress("temp", result_type)
    else:
        arrsizes = leftArr['arrList']
        # En esta op se cambian las rows y cols para siguientes ops
        if operator == "i":
            tmp = leftArr['arrList'][0]
            leftArr['arrList'][0] = leftArr['arrList'][1]
            leftArr['arrList'][1] = tmp
        stackArrs.push({ 'dir': posMemVirtual, 'arrList': arrsizes })
        if stackOp.peek() != "[":
            posMemVirtual = memVirtual.getAddress("pointer", result_type)
            memVirtual.setNextAddress("pointer", result_type, size)
        else:
            posMemVirtual = memVirtual.getAddress("temp", result_type)
            memVirtual.setNextAddress("temp", result_type, size)
        else:
            posMemVirtual = memVirtual.getAddress('temp', result_type)
            quad = Quadruple(operator, left_operand, right_operand, posMemVirtual)
            quadruples.append(quad.getQuad())
            stackVar.push(posMemVirtual)

```

```

        stackType.push(result_type)
    else:
        print("error termino")

# Se ejecuta al finalizar la expresion, como = print etc
def genQuadEndExp(cond):
    if stackOp.size() > 0:
        top = stackOp.peak()
        if top in cond:
            right_operand = stackVar.pop()
            right_type = stackType.pop()
            operator = stackOp.pop()
            if stackVar.size() > 0 and top not in ["$", "?", "!"]:
                left_operand = stackVar.pop()
                left_type = stackType.pop()
                result_type = semTable.result(left_type, right_type, operator)
            else:
                left_operand = None
                result_type = True
            if result_type != False:
                if stackArrs.size() > 0:
                    leftArr = stackArrs.pop()
                    size = 1
                    if stackArrs.size() > 0:
                        size = 1
                        rightArr = stackArrs.pop()
                        if len(leftArr['arrList']) != len(rightArr['arrList']):
                            raise TypeError("ERROR: A list with a Matrix 🙄")
                        index = 0
                        while (index < len(leftArr['arrList'])):
                            if leftArr['arrList'][index][0] != rightArr['arrList'][index][0]:
                                raise TypeError("ERROR: Different sizes for operation 🙄")
                            size = size * leftArr['arrList'][index][0]
                            index = index + 1
                            operator = operator + operator[0]
                        operator = operator + "M"
                    else:
                        index = 0
                        while (index < len(leftArr['arrList'])):
                            size = size * leftArr['arrList'][index][0]
                            index = index + 1
                dimtwo = None
                if len(leftArr['arrList']) > 1:
                    dimtwo = leftArr['arrList'][1][0]
                quad = Quadruple("SIZE", leftArr['arrList'][0][0], dimtwo, size)
                quadruples.append(quad.getQuad())
                quad = Quadruple(operator, left_operand, None, right_operand)
                quadruples.append(quad.getQuad())
            else:
                raise TypeError("ERROR: Cant perform that operation 🙄")
        else:
            raise TypeError("ERROR: Exp ended too soon ", top)

```

```

else:
    raise TypeError("ERROR: Stack Op 0")

# Se generan los goto saltos
def genQuadGoto():
    if stackJumps.size() > 0:
        end = stackJumps.pop()
        quadruples[end][3] = len(quadruples)
    else:
        raise TypeError("ERROR: Goto Error, No jumps left")

class TransformerLark(Transformer):
    # Se hace init a las vars locales
    def __init__(self):
        self.functions = {}
        self.currType = ""
        self.currFunction = "__global__"
        self.currVar = ""
        self.currArr = ""
        self.currFuncCounter = 0
        self.dim = 1
        self.R = 0
        self.currNodes = []
        self.calledFunction = ""

    # Hacemos init a la function global y generamos el goto al main
    def program_id(self, args):
        self.currFunction = "__global__"
        self.functions["__global__"] = { 'type': 'VOID', 'vars': {} }
        self.functions["__cte__"] = { 'type': 'VOID', 'vars': {} }
        quad = Quadruple("Goto", None, None, None)
        quadruples.append(quad.getQuad())
        return Tree('program', args)

    # Al llegar a principal reemplazamos el goto del main
    def main(self, args):
        quadruples[0][3] = len(quadruples)
        self.currFunction = "__global__"

    # Se crea los datos de la function que se declaro ademas de hacer verificacion de que no exista
    def func_name(self, args):
        self.currFunction = args[0].value
        currParams.clean()
        self.currFuncCounter = 0
        if args[0] in self.functions:
            raise ValueError(args[0] + " already defined")
        else:
            self.functions[args[0]] = { 'type': self.currType, 'vars': {}, 'params': {} }
            if self.currType != "void":
                self.saveVar(args[0], "__global__")
        return Tree('func_name', args)

    # Agregamos los parametros a la function con el tipo

```



```

# Agregamos los parametros a la function con el tipo
def args_func(self, args):
    if currParams.size() > 0:
        top = currParams.pop()
        var = top["var"]
        typ = top["type"]
        self.functions[self.currFunction]['params'][self.currFuncCounter] = { 'var': var,
'type': typ }
        self.currFuncCounter = self.currFuncCounter + 1

# Guardamos el tamaño de los parametros
def end_func_decl(self, args):
    self.functions[self.currFunction]['params_size'] = self.currFuncCounter
    return Tree('end_func_decl', args)
# Se guarda el tamaño de las vars y el inicio de la func
def func(self, args):
    self.functions[self.currFunction]['local_size'] =
abs(len(self.functions[self.currFunction]['vars']) - self.functions[self.currFunction]
['params_size'])
    self.functions[self.currFunction]['quad_count'] = len(quadruples)
    return Tree('end_func_decl', args)

# Se genera el endfunc
def func_bloque(self, args):
    quad = Quadruple("ENDFunc", None, None, None)
    quadruples.append(quad.getQuad())

def gen_era(self, args):
    quad = Quadruple("ERA", self.calledFunction, None, None)
    quadruples.append(quad.getQuad())
    self.currFuncCounter = 0
    return Tree('gen_era', args)

# Se verifica que exista y que esa funcion exista
def call_name(self, args):
    if args[0].value not in self.functions:
        raise TypeError("ERROR: " + args[0].value + " cant be found!")
    self.calledFunction = args[0].value
    return Tree('call_name', args)

# Se crean los quads de parametros
def call_var(self, args):
    if stackVar.size() > 0:
        argument = stackVar.pop()
        typ = stackType.pop()
        param = self.functions[self.calledFunction]['params'][self.currFuncCounter]
        if param["type"] == typ:
            quad = Quadruple("PARAMETER", argument, None, self.currFuncCounter + 1)
            self.currFuncCounter = self.currFuncCounter + 1
            quadruples.append(quad.getQuad())
        else:
            raise TypeError("ERROR: Parameters type mismatch 🤔")
    return Tree('call_var', args)

```

```

# Se hace el gosub, ademas de guardar el resultado de la llamada en el stack de vars para
ops, y guardar el valor en un tmp
def call_end(self, args):
    if self.currFuncCounter < len(self.functions[self.calledFunction]['params']):
        raise TypeError("ERROR:" + self.currFuncCounter + " parameters given, expecting " +
len(self.functions[self.calledFunction]['params']))
    else:
        quad = Quadruple("GOSUB", None, None, self.calledFunction)
        quadruples.append(quad.getQuad())
        if self.functions[self.calledFunction]['type'] != "void":
            scope = "local"
            if self.currFunction == "__global__":
                scope = "global"
            mem = memVirtual.getAddress(scope, self.functions["__global__"]["vars"]
[self.calledFunction]["type"])
            res = self.functions["__global__"]["vars"][self.calledFunction]["dir"]
            quad = Quadruple("=", mem, None, res)
            quadruples.append(quad.getQuad())
            stackVar.push(mem)
            stackType.push(self.functions["__global__"]["vars"][self.calledFunction]
["type"])
        return Tree('call_end', args)

def return_val(self, args):
    self.currType = args[0].value
    return Tree('return_val', args)

# Se guarda la variable con los atts como tamaño, tipo etc
def saveVar(self, var, scope = ""):
    if scope == "":
        scope = self.currFunction
    if var in self.functions[scope]['vars']:
        raise ValueError(var + " already defined")
    else:
        if scope == "__global__":
            scopeMem = "global"
        else:
            scopeMem = "local"
        posMemVirtual = memVirtual.getAddress(scopeMem, self.currType)
        self.functions[scope]['vars'][var] = { 'type': self.currType, 'value': 0, 'dir':
posMemVirtual, 'dim': 0, 'arrList': [] }
        currParams.push({ "var": var, "type": self.currType })

def tipo(self, defType):
    self.currType = defType[0]
    return Tree('tipo', defType)

# Al momento de encontrar un id se empuja al stack de vars
# ademas de que si es arreglo se empuja
# a la lista de arreglos
def id(self, args):
    exists = self.findbyMem(args[0].value)
    self.currVar = args[0].value

```

```

arr = self.getArray(self.currVar)
sze = len(arr)
if sze > 0:
    stackArrs.push({ 'dir': exists, 'arrList': arr })
stackVar.push(exists)
stackType.push(self.findbyType(args[0]))
if sze > 0:
    stackArrName.push(self.currVar)
return Tree('id',args)

# Se guarda la variable nueva
def id_new(self, args):
    self.saveVar(args[0].value)
    self.currVar = args[0].value
    sze = len(self.getArray(self.currVar))
    if sze > 0:
        stackArrName.push(self.currVar)
    return Tree('id_new', args)

def lbrake(self, args):
    if self.R > 1:
        self.dim = self.dim + 1
    else:
        self.R = 1
        self.dim = 1
    self.getArray(self.currVar).append([0,0])
    return Tree('lbrake', args)

def arms(self, args):
    if stackArrs.size() > 0:
        stackArrs.pop()
    return Tree('arms', args)

# se guardan las dims en stack para acceder despues, y agarran los nodos actuales
def arrbrake(self, args):
    if stackVar.size() > 0:
        var = stackVar.pop()
        if len(self.getArray(stackArrName.peek())) > 0:
            self.dim = 1
            stackDim.push({ 'var': var, 'dim' : 1 })
            self.currNodes = self.getArray(stackArrName.peek()).copy()
            stackOp.push("[")
        return Tree('arrbrake', args)

# Se guarda el tamano del arreglo
def size(self, args):
    self.R = int(args[0].value) * self.R
    sze = len(self.getArray(self.currVar))
    self.getArray(self.currVar)[sze - 1][0] = int(args[0].value)
    return Tree('size', args)

# Se hace el quad de ver, ademas de los de acceso
def arrexpsize(self, args):

```

```

if stackOp.size() > 0:
    top = stackOp.peak()
    if top in ["["]:
        dimension = self.currNodes[0]
        var = stackVar.peak()
        cteDirZero = self.insertToCte(-1)
        cteDirDim = self.insertToCte(dimension[0])
        quad = Quadruple("Ver", var, cteDirZero, cteDirDim)
        quadruples.append(quad.getQuad())
        if len(self.currNodes) > 1:
            aux = stackVar.pop()
            posMemVirtual = memVirtual.getAddress('temp', "int")
            quad = Quadruple("*", aux, cteDirDim, posMemVirtual)
            quadruples.append(quad.getQuad())
            stackVar.push(posMemVirtual)
        if self.dim > 1:
            aux2 = stackVar.pop()
            aux1 = stackVar.pop()
            posMemVirtual = memVirtual.getAddress('temp', "int")
            quad = Quadruple("+", aux1, aux2, posMemVirtual)
            quadruples.append(quad.getQuad())
            stackVar.push(posMemVirtual)
    return Tree('arrexpsize', args)

# Cuando se encuentra otra dim se trata como tal
def lbrakesecund(self, args):
    self.dim = self.dim + 1
    currDim = stackDim.pop()
    currDim['dim'] = self.dim
    stackDim.push(currDim)
    self.currNodes.pop(0)
    return Tree('lbrakesecund', args)

# Al finalizar el arreglo se guarda el resultado del acceso mas pointer
def arr(self, args):
    if stackArrs.size() > 0:
        stackArrs.pop()
    if stackVar.size() > 0:
        aux1 = stackVar.pop()
        posMemVirtual2 = memVirtual.getAddress('temp', "int")
        posMemVirtual3 = self.findbyMem(stackArrName.pop())
        quad = Quadruple("+", aux1, posMemVirtual3, posMemVirtual2)
        stackVar.push(posMemVirtual2)
        quadruples.append(quad.getQuad())
        stackOp.pop()
        self.dim = 1

# cuando se decalra una variable, si es arreglo se crean el tamano de memroia
def decl_var(self, args):
    dimTmp = 0
    offset = 0
    sze = self.R
    last = len(self.getArray(self.currVar))

```

```

if last == 0:
    return Tree('decl_var', args)

posMemVirtual = memVirtual.getAddress("pointer", self.currType)
memVirtual.setNextAddress("pointer", self.currType, self.R)
self.functions[self.currFunction]['vars'][self.currVar]["dir"] = posMemVirtual
for node in self.getArray(self.currVar):
    mDim = self.R / node[0]
    self.R = mDim
    node[1] = mDim
    offset = offset + mDim
K = offset
self.getArray(self.currVar)[last - 1][1] = -K
self.dim = 1
self.R = 0

# Las siguientes funciones son para guardar los valores en ctes
def string(self, args):
    dirCte = self.findbyMemCte(args[0].value)
    if dirCte == -1:
        dirCte = memVirtual.getAddress('cte', 'char')
        slicedString = args[0].value[1:len(args[0].value) - 1]
        self.functions["__cte__"]['vars'][args[0].value] = { 'type': "char", 'value':
slicedString, 'dir': dirCte }
        stackVar.push(dirCte)
        stackType.push("char")
    return Tree('string', args)

def insertToCte(self, value):
    dirCte = self.findbyMemCte(value)
    if dirCte == -1:
        dirCte = memVirtual.getAddress('cte', 'int')
        self.functions["__cte__"]['vars'][value] = { 'type': "int", 'value': value, 'dir':
dirCte }
    return dirCte

def boolean(self, args):
    dirCte = self.findbyMemCte(args[0].value)
    if dirCte == -1:
        dirCte = memVirtual.getAddress('cte', 'bool')
        self.functions["__cte__"]['vars'][args[0].value] = { 'type': "bool", 'value':
args[0].value, 'dir': dirCte }
        stackVar.push(dirCte)
        stackType.push('bool')
    return Tree('bool', args)

def integer(self, args):
    dirCte = self.findbyMemCte(args[0].value)
    if dirCte == -1:
        dirCte = memVirtual.getAddress('cte', 'int')
        self.functions["__cte__"]['vars'][args[0].value] = { 'type': "int", 'value':
args[0].value, 'dir': dirCte }
        stackVar.push(dirCte)

```

```

        stackType.push('int')
        return Tree('integer', args)

def number(self, args):
    dirCte = self.findbyMemCte(args[0].value)
    if dirCte == -1:
        dirCte = memVirtual.getAddress('cte', 'float')
        self.functions["__cte__"]['vars'][args[0].value] = { 'type': "float", 'value':
args[0].value, 'dir': dirCte }
        stackVar.push(dirCte)
        stackType.push('float')
        return Tree('number', args)

# Las siguientes funciones agregan las ops al stack
def times_divide(self, args):
    stackOp.push(args[0].value)
    return Tree('times', args)

def plus_minus(self, args):
    stackOp.push(args[0].value)
    return Tree('plus', args)

def assign(self, args):
    stackOp.push("=")
    return Tree('equal', args)

def op4(self, args):
    stackOp.push("AND")
    return Tree('op4', args)

def op3(self, args):
    stackOp.push("OR")
    return Tree('op3', args)

def op5(self, args):
    stackOp.push(args[0].value)
    return Tree('op5', args)

def opmatrix(self, args):
    stackOp.push(args[0].value)
    return Tree('opmatrix', args)

# Se guarda donde se empezo el loop
def while_key(self, args):
    stackJumps.push(len(quadruples))
    return Tree('while_key', args)

def to(self, args):
    stackJumps.push(len(quadruples))
    return Tree('to', args)

def return_str(self, args):
    stackOp.push('return')

```

```

funcType = self.functions[self.currFunction]['type']
stackType.push(funcType)
return Tree('return_str', args)

# Fin de exp logica para generar el gotof, y se guarda donde se genero
def end_exp_log(self, args):
    if stackType.size() > 0:
        exp_type = stackType.pop()
        if exp_type == "bool":
            res = stackVar.pop()
            quad = Quadruple("GotoF", res, None, None)
            quadruples.append(quad.getQuad())
            stackJumps.push(len(quadruples) - 1)
        else:
            raise TypeError("ERROR: Logical operation error")
    return Tree('end_exp_log', args)

# Se genera el goto con el retorno, si se puede
# y reemplaza el gotof
def fin_bloque(self, args):
    end = stackJumps.pop()
    if stackJumps.size() > 0:
        ret = stackJumps.pop()
        quad = Quadruple("Goto", None, None, ret)
        quadruples.append(quad.getQuad())
    quadruples[end][3] = len(quadruples)
    return Tree('fin_bloque', args)

# Se genera los quads para el auto increment de las variables
# Se genera el goto para el loop y reemplaza el gotof
def fin_for_loop(self, args):
    aux = forloopvar.pop()
    posMemVirtual = memVirtual.getAddress('temp', "int")
    numone = self.insertToCte(1)
    quad = Quadruple("+", aux, numone, posMemVirtual)
    quadruples.append(quad.getQuad())
    quad = Quadruple("=", aux, None, posMemVirtual)
    quadruples.append(quad.getQuad())
    end = stackJumps.pop()
    if stackJumps.size() > 0:
        ret = stackJumps.pop()
        quad = Quadruple("Goto", None, None, ret)
        quadruples.append(quad.getQuad())
    quadruples[end][3] = len(quadruples)
    return Tree('fin_for_loop', args)

# Se guarda la variable que se modificara
def exp_end_for(self, args):
    quadsze = len(quadruples)
    currVar = quadruples[quadsze - 1][1]
    forloopvar.push(currVar)
    return Tree('exp_end_for', args)

```

```

# Se hace un goto generico
def if_bloque(self, args):
    genQuadGoto()
    return Tree('if_bloque', args)

# Se genera el gotof y se guarda
def if_key(self, args):
    if stackType.size() > 0:
        exp_type = stackType.pop()
        if exp_type == "bool":
            result = stackVar.pop()
            quad = Quadruple("GotoF", result, None, None)
            quadruples.append(quad.getQuad())
            stackJumps.push(len(quadruples) - 1)
        else:
            raise TypeError("ERROR: You cant do that in an IF, no boolean was returned")
    return Tree('if_key', args)

# Se hace el goto para el if y se reemplaza el pasado del if
def else_key(self, args):
    if stackJumps.size() > 0:
        end = stackJumps.pop()
        quad = Quadruple("Goto", None, None, None)
        quadruples.append(quad.getQuad())
        stackJumps.push(len(quadruples) - 1)
        quadruples[end][3] = len(quadruples)
    return Tree('else_key', args)

# Las ops para print y read
def print_exp(self, args):
    stackOp.push("print")
    stackType.push("print")
    return Tree('print_exp', args)

def read_emp(self, args):
    stackOp.push("read")
    stackType.push("read")
    return Tree("read_emp", args)

def comma(self, args):
    stackOp.push("print")
    stackType.push("print")
    return Tree('comma', args)

def comma_read(self, args):
    stackOp.push("read")
    stackType.push("read")
    return Tree("comma_read", args)

# Para hacer los end exp para print read
def print_exp_fin(self, args):
    genQuadEndExp(["print"])
    return Tree('print_exp_fin', args)

```



```

def id_finished(self, args):
    genQuadEndExp(['read'])
    return Tree('id_read', args)

# Genera quad logicos
def exp_comp(self, args):
    genQuadOpLog([">", "<", "!=", "==", "<=", ">="])
    return Tree('exp_comp', args)

def exp_log_and(self, args):
    genQuadOpLog(["AND"])
    return ('exp_log_or', args)

def exp_log_or(self, args):
    genQuadOpLog(["OR"])
    return ('exp_log_or', args)

# Se generan quads de exp
def termino(self, args):
    genQuadOpExp(['+', '-'])
    return ('termino', args)

def factor(self, args):
    genQuadOpExp(["*", "/", "$", "?", "i"])
    return ('factor', args)

def assign_var(self, args):
    genQuadEndExp(['='])

def return_exp(self, args):
    genQuadEndExp(['return'])

# Funciones helpers
def findbyType(self, var):
    if var in self.functions[self.currFunction]['vars']:
        return self.functions[self.currFunction]['vars'][var]['type'].value
    elif var in self.functions["__global__"]['vars']:
        return self.functions["__global__"]['vars'][var]['type'].value
    return False

def findbyValue(self, var):
    if var in self.functions[self.currFunction]['vars']:
        return self.functions[self.currFunction]['vars'][var]['value']
    elif var in self.functions["__global__"]['vars']:
        return self.functions["__global__"]['vars'][var]['value']
    return False

def findbyMem(self, var):
    if var in self.functions[self.currFunction]['vars']:
        return self.functions[self.currFunction]['vars'][var]['dir']
    elif var in self.functions["__global__"]['vars']:
        return self.functions["__global__"]['vars'][var]['dir']
    return -1

```

```

def findbyMemCte(self, var):
    if var in self.functions['__cte__']['vars']:
        return self.functions['__cte__']['vars'][var]['dir']
    return -1

def setbyValue(self, var, value):
    if var in self.functions[self.currFunction]['vars']:
        self.functions[self.currFunction]['vars'][var]['value'] = value
    elif var in self.functions["__global__"]['vars']:
        self.functions["__global__"]['vars'][var]['value'] = value
    return False

def getArray(self, var):
    if var in self.functions[self.currFunction]['vars']:
        return self.functions[self.currFunction]['vars'][var]['arrList']
    elif var in self.functions["__global__"]['vars']:
        return self.functions["__global__"]['vars'][var]['arrList']
    return []

# Se guardan las ops de lparen y se popean al final del arreglo
def lparen(self, args):
    stackOp.push(args[0].value)
    return Tree("lparen", args)

def rparen(self, args):
    stackOp.pop()
    return Tree("rparen", args)

# Se ejecuta al final del programa
def program(self, args):
    #prettyList(quadruples)
    #pretty(self.functions)
    runMachine(quadruples, self.functions)
    return Tree("program", args)

```

analizador_lark.py: el main del compilador, contiene la gramática y manda llamar el transformer, despliega los resultados.

```

from lark import Lark
from transformer import TransformerLark
import os
import sys
import codecs

# Se define la gramatica y los tokens que se usaran
calc_grammar = r"""
start: program

program_id : ID
program : PROGRAM program_id SEMI program_es
program_es : vars (exec_func func_bloque)* main_func

```

```

| bloque
exec_func: func
func_bloque : bloque
main_func : main LPAREN RPAREN bloque
main: MAIN
var : id arr?
arr: arrmexp (lbrakesecond arrexpsize rbrakearr)?
lbrakesecond: LBRAKE
decl_var : tipo id_new arrm? arrm?
id_new: ID
id: ID
var_id : decl_var lista_var? SEMI var_id?
lista_var : "," id_new arrm? arrm? lista_var?
vars : VAR var_id

arrm : lbrake size rbrake
arrmexp : arrbrake arrexpsize rbrakearr
rbrakearr: RBRAKE
arrexpsize : exp
arrbrake: LBRAKE
lbrake: LBRAKE
rbrake: RBRAKE
size: INTEGER | NUMBER

tipo : INT
| FLOAT
| LETRAS

return_val: VOID
| INT
| FLOAT
| LETRAS

args_func : decl_var
func : FUNCTION return_val func_name LPAREN args_func? ("," args_func)* end_func_decl
func_vars?
func_vars: vars
end_func_decl: RPAREN
func_name: ID

bloque : LBRACE estatuto* RBRACE

estatuto : asignacion
| condicion
| escritura
| while
| return
| read
| for_loop
| call semi_call
semi_call: SEMI

asignacion : var assign exp ended? -> assign_var

```

```

ended: SEMI

read: read_emp LPAREN id_read RPAREN read_end
id_read: id_finished read_comma?
id_finished: id
read_comma: comma_read id_read
comma_read: ","
read_emp : READ
read_end : SEMI

escritura : print_exp LPAREN escritura_exp RPAREN end_print
end_print : SEMI
print_exp : PRINT
escritura_exp: print_exp_fin escritura_exp_comma?
print_exp_fin: exp
escritura_exp_comma: comma escritura_exp
comma: ","

return: return_str LPAREN return_exp RPAREN SEMI
return_str: RETURN
return_exp: exp

condicion: IF exp if_key if_bloque
if_bloque: bloque else?
if_key: THEN
else: else_key bloque
else_key: ELSE

while: while_key exp end_exp_log fin_bloque
fin_bloque: bloque
end_exp_log: DO
while_key: WHILE

for_loop: for_key LPAREN asignacion exp_end_for to exp end_exp_log fin_for_loop
exp_end_for: RPAREN
fin_for_loop: bloque
to: TO
for_key: FROM

exp : termino op1?
op1 : plus_minus exp

plus_minus: PLUS_MINUS

termino : factor op2?
op2 : times_divide termino

times_divide: TIMES_DIVIDE

factor : boolean
        | var opmatrix?
        | call
        | number

```

```

    | string
    | integer
    | PLUS var_cte
    | MINUS var_cte
    | LPAREN exp_log_or RPAREN

opmatrix: SIGN
| QUES
| EXCLD

SIGN: "$"
EXCLD: "i"

string: STRING
boolean: BOOLEAN
call: call_name gen_era call_args? call_end
call_end: rparen
gen_era: lparen
call_name: ID
call_args: call_var call_args_comma?
call_var: exp
call_args_comma: "," call_args

var_cte : cte EXCL
| cte QUES
| cte

cte : integer
| number
integer: INTEGER
number: NUMBER

exp_log_or: exp_log_and op3?
op3: OR exp_log_or
exp_log_and: exp_comp op4?
op4: AND exp_log_and
exp_comp: log_exp_comp | exp
log_exp_comp: exp op5 exp
op5 : GREATER ASSIGN
| LESS ASSIGN
| LESS
| GREATER
| DIFF
| EQUAL

assign: ASSIGN
lparen: LPAREN
rparen: RPAREN

IF: "si"
THEN: "entonces"

```

```

ELSE: "sino"
PROGRAM: "programa"
PRINT: "escribe"
VAR: "var"
FLOAT: "float"
INT: "int"
LETRAS: "char"
BOOL: "bool"
FUNCTION: "funcion"
RETURN: "regresa"
WHILE: "mientras"
DO.10: "haz" | "hacer"
MAIN: "principal"
READ: "lee"
FROM: "desde"
TO: "hasta"
VOID: "void"

BOOLEAN.10: "True" | "False"
ID : WORD
NUMBER : /[+-]?[0-9]*\.[0-9]+([eE][+-]?[0-9]+)?/
PLUS: "+"
TIMES_DIVIDE: "*" | "/"
PLUS_MINUS: "+" | "-"
MINUS: "-"
ASSIGN: "="
TIMES: "*"
COLON: ":"
DIVIDE: "/"
SEMI: ";"
LESS: "<"
GREATER: ">"
EQUAL: "=="
DOT: "."
AND: "&"
OR: "|"
LPAREN: "("
RPAREN: ")"
LBRACE: "{"
RBRACE: "}"
LBRAKE: "["
RBRAKE: "]"
EXCL: "!"
QUES: "?"
DIFF: "!=" | "<>"
INTEGER: /[+-]?[1-9]\d*|0/
COMMENT: "%%" /(.|\\n|\\r)+/

%import common.WS_INLINE
%import common.NEWLINE
%import common.WORD
%import common.ESCAPED_STRING -> STRING
%ignore NEWLINE

```

```

    %ignore WS_INLINE
    %ignore COMMENT
    """

duck_parser = Lark(calc_grammar, parser='lalr', debug=True, transformer=TransformerLark())
duck = duck_parser.parse

def main():
    while True:
        try:
            s = input('> ')
        except EOFError:
            break
        print(duck(s))

def test(file):
    fp = codecs.open(file, 'r', 'utf-8')
    cadena2 = fp.read()
    fp.close()
    duck(cadena2).pretty

if __name__ == '__main__':
    test(sys.argv[1])







```

Commits





Commits on Jun 2, 2020

Minor changes RafaelSerna committed 2 hours ago	 c96bf7d	
cambio a pointers Hectordeluna committed 4 hours ago	 6ee8af7	













Commits on May 31, 2020

removed print Hectordeluna committed 2 days ago	 5f1aa1a	
Test cases matrix tuning Hectordeluna committed 2 days ago	 3565469	
Array fixes Hectordeluna committed 2 days ago	 bd6f6ac	



Commits on May 27, 2020

Merge branch 'master' of https://github.com/Hectordeluna/patitoplusplus Hectordeluna committed 6 days ago	 f193fcb	
matrix Operations Hectordeluna committed 6 days ago	 7b9a0e2	







Commits on May 26, 2020

agregar bitacora avance 7 RafaelSerna committed 7 days ago	 97cdb82	
Merge branch 'master' of https://github.com/Hectordeluna/patitoplusplus Hectordeluna committed 7 days ago	 978f1f5	
Op matriz cambios Hectordeluna committed 7 days ago	 891d5c5	
Added quad instructions in virtualMachine RafaelSerna committed 7 days ago	 c8d37f0	
cambios simples Hectordeluna committed 7 days ago	 9c4dec4	
Matrix operations Hectordeluna committed 7 days ago	 85f70d0	

Commits on May 25, 2020

Virtual machine, funcs, matrix, etc Hectordeluna committed 8 days ago	 5d67e61	
---	---	---

Commits on May 17, 2020

added ctes Hectordeluna committed 16 days ago	 0fe1564	
Modified Memory class RafaelSerna committed 16 days ago	 188d1ac	
Added Memory class RafaelSerna committed 16 days ago	 f7e57c2	

Commits on May 14, 2020	<div> Fixes <div> Hectordeluna committed 20 days ago </div> <div> c2a980a </div> </div>
Commits on May 13, 2020	<div> Added virtual memory table and quads temporals <div> RafaelSerna committed 20 days ago </div> <div> 660be8f </div> </div>
Commits on May 11, 2020	<div> Hacer chico todo <div> Hectordeluna committed 22 days ago </div> <div> 610ea13 </div> </div>
Commits on May 10, 2020	<div> fix <div> Hectordeluna committed 23 days ago </div> <div> 3b21b63 </div> </div> <div> Gen quads func <div> Hectordeluna committed 23 days ago </div> <div> 1fc8763 </div> </div>
Commits on May 9, 2020	<div> cambio <div> Hectordeluna committed 24 days ago </div> <div> ff1a73d </div> </div>
Commits on May 4, 2020	<div> Gramatica modificada para ifs fors e imprimir cuatruplos <div> RafaelSerna committed 29 days ago </div> <div> 1aa4eec </div> </div> <div> If else y for agregado quads <div> RafaelSerna committed 29 days ago </div> <div> 0ce2ae8 </div> </div>
Commits on May 3, 2020	<div> while agregado quads <div> Hectordeluna committed on May 3 </div> <div> 2cfa147 </div> </div>
Commits on Apr 27, 2020	<div> Added semantic table <div> Rafael Serna authored and Rafael Serna committed on Apr 27 </div> <div> 64f458a </div> </div>
Commits on Apr 26, 2020	<div> Se agrego resolver operaciones <div> Hectordeluna committed on Apr 26 </div> <div> ae4e036 </div> </div>
Commits on Apr 20, 2020	<div> Cambios <div> Hectordeluna committed on Apr 20 </div> <div> 53c6ba0 </div> </div> <div> Transformer <div> Hectordeluna committed on Apr 20 </div> <div> b5425f9 </div> </div> <div> Modified var table <div> Rafael Serna authored and Rafael Serna committed on Apr 20 </div> <div> a6f6a95 </div> </div> <div> Modified var table <div> Rafael Serna authored and Rafael Serna committed on Apr 20 </div> <div> 0121366 </div> </div> <div> Added variable table <div> Rafael Serna authored and Rafael Serna committed on Apr 20 </div> <div> dea9f7b </div> </div>
Commits on Apr 13, 2020	<div> Init <div> Hectordeluna committed on Apr 13 </div> <div> 1c53e2a </div> </div>