



Proyecto de Compiladores

Lienzo

Héctor Ricardo Méndez Sordia A01195770

Graciela García Díaz A01195698

4 de mayo de 2016
Ing. Elda Quiroga

Héctor Ricardo Méndez Sordia

Graciela García Díaz

Monterrey, Nuevo León, México

Tabla de Contenido

Proyecto de Compiladores	1
Documentación	4
DESCRIPCIÓN DEL PROYECTO.....	4
Visión, Objetivos y Alcance del Proyecto.....	4
Análisis de Requerimientos y Casos de Uso generales.....	4
Descripción de los principales Test Cases	5
Descripción del proceso general seguido para el desarrollo del proyecto.....	5
DESCRIPCIÓN DEL LENGUAJE:.....	7
Nombre del lenguaje	7
Principales características del lenguaje.....	7
Posibles errores en compilación y ejecución	7
DESCRIPCIÓN DEL COMPILADOR.....	7
Equipo de cómputo, lenguaje y utilerías especiales utilizadas	7
Descripción del Análisis de Léxico	8
Descripción del Análisis de Sintaxis.....	9
Descripción de Generación de Código Intermedio y Análisis Semántico	12
Proceso de Administración de Memoria usado en la compilación.	21
DESCRIPCIÓN DE LA MÁQUINA VIRTUAL	22
Equipo de cómputo, lenguaje y utilerías especiales usadas	22
Descripción detallada del proceso de Administración de Memoria en ejecución	22
PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE	22
Prueba de ordenamiento de arreglos	22
Prueba Fibonacci.....	24
Prueba de dibujo #1	26
Prueba de dibujo #2	27
Prueba de dibujo #3	28
Prueba de dibujo #4	29
Prueba de dibujo #5	30
LISTADOS PERFECTAMENTE DOCUMENTADOS DEL PROYECTO	31

Manual de Usuario.....	63
Sobre la instalación de herramientas requeridas	63
Python.....	63
Antlr.....	63
Sobre la compilación	63
Mi primer programa en Lienzo	63
Convenciones de léxico	63
Comentarios	63
Identificadores	64
Palabras reservadas.....	64
Diseño de un programa	65
Expresiones	65
Condiciones	65
Ciclos	66
Funciones	66
Ejemplo de programa sin output gráfico	66
Ejemplo de programa con output gráfico	67

Documentación

DESCRIPCIÓN DEL PROYECTO

Visión, Objetivos y Alcance del Proyecto

Visión

El objetivo de este proyecto es facilitarle a los jóvenes de enseñanza media el proceso de aprendizaje de la programación. Se desarrolló un lenguaje gráfico amigable y sencillo, de nombre Lienzo, que consiste de elementos propios a la programación, tales como ciclos y condiciones, pero que además cuenta con una pluma virtual con la que se puede dibujar libremente. Así, la salida será algo con lo que están familiarizados.

Los usuarios de este lenguaje serán jóvenes de prepa o secundaria que no tengan nociones de programación y que no dominen el idioma inglés. El lenguaje será parecido al español. De este modo, abrimos las oportunidades a los jóvenes hispanófonos que no dominan el idioma inglés.

Objetivo

El objetivo del lenguaje Lienzo es permitir la creación de gráficos simples mediante instrucciones sencillas de ejecutar. Lienzo será natural de alto nivel y sus palabras reservadas parecidas al español. Además, será imperativo y se ejecutará en una computadora de propósito general.

Lienzo servirá para darles oportunidad de aprender a programar a los jóvenes de secundaria que no dominan el inglés. Permitirá dibujar figuras geométricas bidimensionales sencillas, tales como cuadrados, círculos, etc.. y darles animación, lo cual hará sin duda divertido el proceso de aprendizaje.

Análisis de Requerimientos y Casos de Uso generales

Los requerimientos del lenguaje son:

- Estar en español
- Ser un lenguaje compacto
- Ser un lenguaje parecido al lenguaje natural, pero al mismo tiempo parecido a los lenguajes de 3era generación
- Ser un lenguaje de dibujo gráfico

Casos de uso:

- Soportar las instrucciones secuenciales
- Soportar las llamadas a funciones y recursividad
- Soportar operaciones aritméticas básicas
- Soportar ciclos y saltos condicionales
- Permitir dibujar simulando un lápiz y colores

Descripción de los principales Test Cases

Los casos de uso utilizados fueron los siguientes:

- Factorial recursivo e iterativo
- Fibonacci recursivo e iterativo
- Ordenamiento de un arreglo
- Búsqueda lineal de un arreglo
- Diversas formas gráficas

Descripción del proceso general seguido para el desarrollo del proyecto

El proyecto fue desarrollado en Github con el fin de tener un control de versiones de todo el proyecto. Cada semana se produjo un avance y se subió a la plataforma de Blackboard junto con una descripción del mismo. A continuación se muestran dichas descripciones con sus fechas:

Bitácora

3/3/16 8:36 PM

Primer Avance

Scanner y Parser en ANTLR funcionando al 100%

3/10/16 11:47 AM

Segundo Avance

Ya se modificó la gramática y se crearon las tablas de variables que validan si las variables están repetidas (incluyendo funciones). También se creó otro programa de prueba, el primero sí funciona y el segundo no. El directorio de funciones ya está funcionando.

3/20/16 9:42 PM

Tercer Avance

Cuadros funcionando al 100% con expresiones, declaraciones, asignaciones, lectura, escritura

Funciones son capaces de marcar errores si los argumentos son incorrectos.

Cubo semántico completado.

4/1/16 6:35 PM

Cuarto Avance

En este avance implementamos los cuádruplos de IF, IF ELSE y WHILE.

También arreglamos algunas cosas de la implementación, como los retornos de las funciones y estilo de sintaxis.

4/9/16 10:19 PM

Quinto Avance

Cuádruplos de funciones implementadas y funcionando al 100%.

4/16/16 10:26 PM

Sexto Avance

Fase 1 de la máquina virtual ya quedó implementada. Se traducen de cuádruplos a código meta las siguientes cosas: operaciones aritméticas, booleanas y secuenciales.

4/24/16 8:46 PM

Septimo Avance

Fase 2 de la máquina virtual: ciclos y condicionales, print, write, read, draw with Turtle
Código fuente: soporta también todo tipo de funciones y arreglos

5/2/16 10:45 PM

Ultimo Avance

Compilador funcionando al 100%
Documentacion 50% completa.

Reflexiones Individuales

Héctor Ricardo Méndez

Este proyecto fue muy enriquecedor porque me hizo pensar como trabajo un compilador tras bambalinas. El proceso es complejo, hay muchos factores y elementos a considerar, pero estoy orgulloso de que este proyecto haya sido un éxito. El compilador funciona 100% a la perfección. Como posible mejora, se puede hacer más eficiente el proceso de traducción. A medida que íbamos avanzando, nos dimos cuenta que se podían eficientizar ciertas cosas, sin embargo, consideramos que ya era tarde para mejorarlas.

Graciela García

Este proyecto involucró un proceso de inmersión y compromiso total por más de siete semanas seguidas. Considero que lo más enriquecedor de este proyecto fue la dedicación que le tuvimos, puesto que la mayoría de las veces nos encontrábamos arreglando algo no por que se requiriera si no porque queríamos entregar un proyecto de buena calidad. En lo personal estoy muy satisfecha con nuestro trabajo y considero que el único factor que tuvimos en nuestra contra fue el tiempo, y que con un poco más de ello hubiera ayudado a que implementáramos más funcionalidad.

DESCRIPCIÓN DEL LENGUAJE:

Nombre del lenguaje

Lienzo

Principales características del lenguaje

Lienzo es un lenguaje de programación basado en el idioma español que puede ser utilizado como un lenguaje de programación básico, con condiciones, ciclos y funciones, pero el mismo tiempo puede utilizarse para dibujar sobre un lienzo.

Principales características semánticas

Los tipos que maneja Lienzo son: numérico, mensaje, Las operaciones aritméticas únicamente podrán realizarse con expresiones de tipo numéricas.

Habrà jerarquía de operaciones. Primero se evaluarán los paréntesis, luego las multiplicaciones y divisiones de izquierda a derecha, y finalmente las sumas y restas de izquierda a derecha.

Las funciones no podrán tener el mismo nombre que las variables.

Si una hay un parámetro que se haya pasado por referencia, entonces el argumento debe estar ligado a una zona de memoria, es decir, debe ser forzosamente una variable.

Solamente puede haber arreglos de figuras. No puede haber arreglos de enteros o condiciones o colores dentro de la sección animación.

Posibles errores en compilación y ejecución

Los errores que pueden ocurrir en compilación son los siguientes:

- No se encontró Python
- No se encontró Antlr
- No se encontró el archivo a compilar

Los errores que pueden ocurrir durante la ejecución son los siguientes:

- Errores que notificará el compilador

DESCRIPCIÓN DEL COMPILADOR

Equipo de cómputo, lenguaje y utilerías especiales utilizadas

Para desarrollar este proyecto se utilizó un equipo con Windows 10, Python y Antlr.

Descripción del Análisis de Léxico

Los tokens asociados al lenguaje son los siguientes:

WS : [\t\r\n]+ -> skip
ROJO : 'rojo'
VERDE : 'verde'
AMARILLO : 'amarillo'
AZUL : 'azul'
BLANCO : 'blanco'
NEGRO : 'negro'
MORADO : 'morado'
NARANJA : 'naranja'
CAFE : 'cafe'
GRIS : 'gris'
COLOR : 'color'
LIENZO : 'lienzo'
EQUALS : '='
TAMANO : 'tamano'
POR : 'por'
DE : 'de'
EN : 'en'
MOVER : 'mover'
ADELANTE : 'adelante'
ATRAS : 'atras'
GIRAR : 'girar'
DERECHA : 'derecha'
IZQUIERDA : 'izquierda'
LEVANTAR : 'levantar'
BAJAR : 'bajar'
PLUMA : 'pluma'
DIBUJO : 'dibujo'
DORMIR : 'dormir'
MIENTRAS : 'mientras'
REGRESAR : 'regresar'
QUE : 'que'
SI : 'si'
SINO : 'sino'
TEXTO : 'texto'
LEER : 'leer'
BOLEANO : 'booleano'
NUMERO : 'numero'
ESCRIBIR : 'escribir'
IMPRIMIR : 'imprimir'
NADA : 'nada'

BOOLEAN_CONSTANT : 'verdadero' | 'falso'
 MODIFICABLE : 'modificable'
 INTEGRAL_CONSTANT : [0-9]+
 NUMERIC_CONSTANT : [0-9]+.'[0-9]+
 STRING_CONSTANT : '"' ~('"') * '"'
 ID : [A-Za-z][A-Za-z0-9]*

Descripción del Análisis de Sintaxis

/* converted on Tue May 3, 2016, 21:30 (UTC-05) by antlr_3-to-w3c v0.37 which is Copyright (c)
 2011-2016 by Gunther Rademacher <grd@gmx.net> */

```

program ::= declaracion* ( tamanoLienzo? colorLienzo? | colorLienzo? tamanoLienzo? )
funcion* instruccion_aux* EOF
colorLienzo
    ::= COLOR DE LIENZO '=' color ';'
color ::= ROJO
    | VERDE
    | AMARILLO
    | AZUL
    | BLANCO
    | NEGRO
    | MORADO
    | NARANJA
    | CAFE
    | GRIS
tamanoLienzo
    ::= TAMANO DE LIENZO '=' ss_expresion POR ss_expresion ';'
declaracion
    ::= declaracion_variable
    | declaracion_arreglo
declaracion_variable
    ::= tipo ID '=' ss_expresion ';'
declaracion_arreglo
    ::= tipo '[' INTEGRAL_CONSTANT ']' ID ';'
funcion ::= tipoFunc ID '(' ( parametro ( ',' parametro ) * ) ? ')' '{' cuerpo '}'
tipoFunc ::= tipo
    | NADA
parametro
    ::= tipo MODIFICABLE? ID
cuerpo ::= declaracion* instruccion_aux*
bloque_instrucciones
    ::= instruccion_aux
    | '{' instruccion_aux* '}'
  
```

```

instruccion_aux
    ::= ( asignacion | llamadaFuncionPredefinida | llamadaFuncion | regresar ) ';'
        | condicional
        | mientrasQue
regresar ::= REGRESAR ss_expresion
llamadaFuncionPredefinida
    ::= lectura
        | escritura
        | imprimir
        | imprimir_no_ln
        | mover_adelante
        | mover_atras
        | girar_izquierda
        | girar_derecha
        | cambio_color
        | subir_pluma
        | bajar_pluma
        | velocidad_pluma
        | posicion_x_pluma
        | posicion_y_pluma
lectura ::= LEER ID
escritura
    ::= ESCRIBIR ss_expresion
imprimir_no_ln
    ::= IMPRIMIR SIN SALTO ss_expresion
imprimir ::= IMPRIMIR ss_expresion
mover_adelante
    ::= MOVER ADELANTE ss_expresion
mover_atras
    ::= MOVER ATRAS ss_expresion
girar_derecha
    ::= GIRAR DERECHA ss_expresion
girar_izquierda
    ::= GIRAR IZQUIERDA ss_expresion
subir_pluma
    ::= LEVANTAR PLUMA
bajar_pluma
    ::= BAJAR PLUMA
cambio_color
    ::= COLOR DE PLUMA '=' color
velocidad_pluma
    ::= VELOCIDAD DE PLUMA '=' ss_expresion
posicion_x_pluma
    ::= POSICION_X DE PLUMA '=' ss_expresion

```

```

posicion_y_pluma
    ::= POSICION_Y DE PLUMA '=' ss_expresion
asignacion
    ::= ID ( '[' ss_expresion ']' )? '=' ss_expresion
tipo    ::= TEXTO
        | BOLEANO
        | NUMERO
condicional
    ::= SI ss_expresion bloque_instrucciones ( SINO bloque_instrucciones )?
mientrasQue
    ::= MIENTRAS QUE ss_expresion bloque_instrucciones
llamadaFuncion
    ::= ID '(' ( ss_expresion ( ',' ss_expresion )* )? ')'
ss_expresion
    ::= s_expresion ( ( '&' | '|' ) s_expresion )*
s_expresion
    ::= expresion ( ( '==' | '!=' | '>' | '<' | '>=' | '<=' ) expresion )*
expresion
    ::= termino ( ( '+' | '-' ) termino )*
termino ::= factor ( ( '*' | '/' | '%' ) factor )*
factor  ::= '!'? factor_aux
        | '!'? ( NUMERIC_CONSTANT | INTEGRAL_CONSTANT )
        | STRING_CONSTANT
factor_aux
    ::= ID ( '[' ss_expresion ']' )?
        | BOOLEAN_CONSTANT
        | llamadaFuncion
        | '(' ss_expresion ')'

<?TOKENS?>

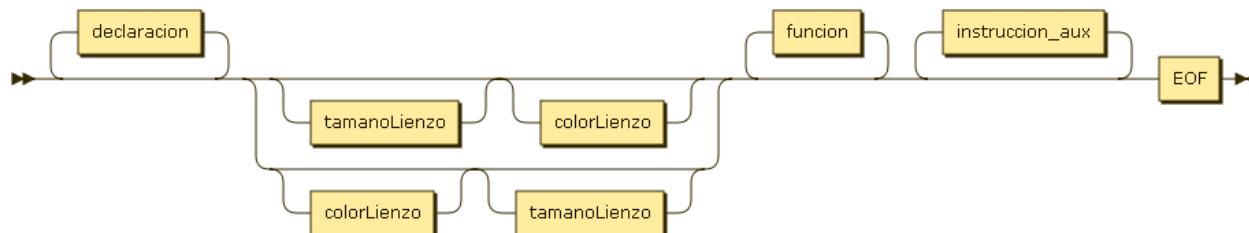
EOF    ::= $

```

Descripción de Generación de Código Intermedio y Análisis Semántico

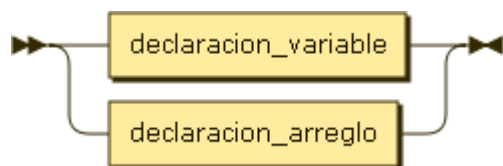
A continuación se mostrarán los diagramas de sintaxis utilizados para crear el lenguaje Lienzo.

Creación del programa



El programa consiste de lo siguiente: primeramente la declaración de variables globales y esta parte es opcional. Posteriormente si se va a hacer uso del lienzo se puede modificar su tamaño y color en cualquier orden. Seguido a esto están las declaraciones de funciones, y cuando estas acaban siguen un conjunto de instrucciones que conforman el programa principal

Declaración

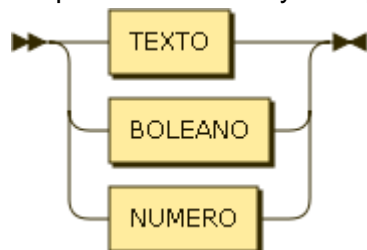


Las declaraciones pueden ser de dos tipos, ya sea que declaren una variable atómica o que declaren un arreglo de variables

Declaración de variables atómicas

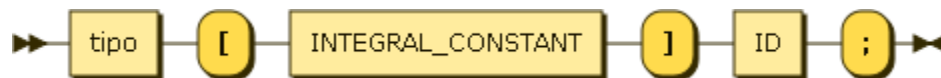


Las variables atómicas se declaran como se expresa en el diagrama anterior. Primero se indica el tipo de la variable y este puede pertenecer a las siguientes categorías:



Posteriormente se le asigna un ID, o identificador único a la variable. Este debe ser alfanumérico y no contener caracteres especiales. La declaración de variables atómicas posteriormente se encuentra con un signo de “=” que representa a la asignación. Del lado derecho del signo de igualdad se encuentra “ss_expresión” que representa al conjunto de expresiones que puede ser asignado a una variable.

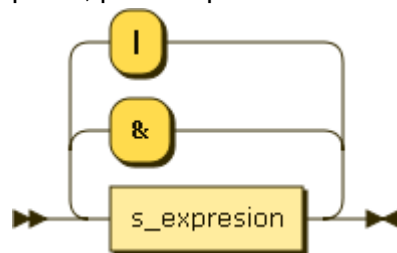
Declaración de arreglos



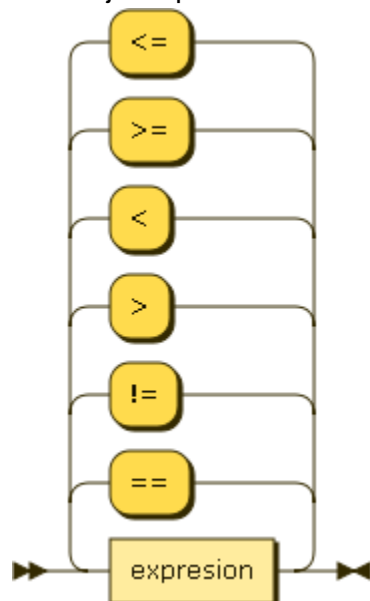
Los arreglos se declaran de manera similar a las variables atómicas, sólo que entre el tipo de variable y el identificador se encuentran unos brackets "[" que encierran el tamaño del arreglo. Este tamaño debe ser igual a una constante entera.

Expresiones

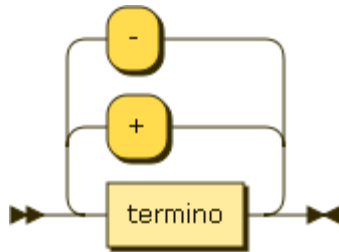
Las expresiones abarcan varios tipos y estos pueden constituirse de cadenas de caracteres, expresiones aritméticas o inclusive expresiones booleanas. El siguiente diagrama muestra la "ss_expresion" cuya función es separar la lógica booleana de la lógica aritmética y el texto plano, puesto que se evalúan diferente a estos últimos dos.



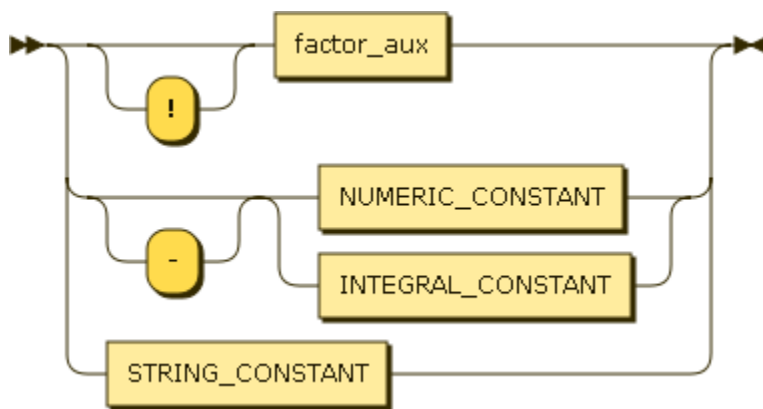
Esta separación que se muestra fue hecha con el fin de establecer una jerarquía de operadores, y a su vez, "s_expresion" también tiene distintos operadores que comparten una misma jerarquía:



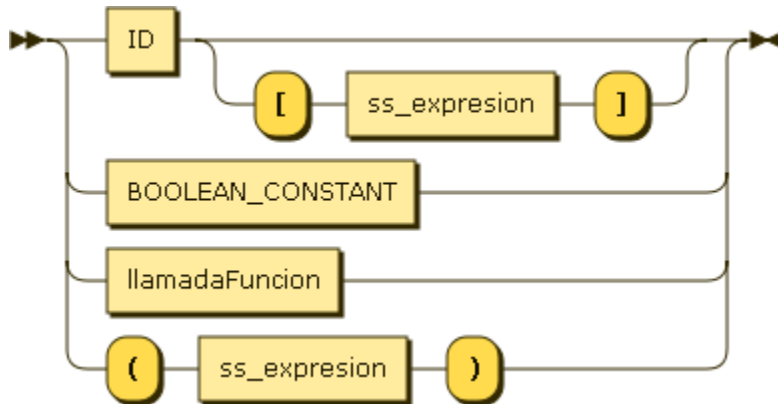
Posteriormente se cuenta con "expresion" que alberga a los diferentes operadores aritméticos. Estos también fueron separados de acuerdo al orden de importancia que tengan en la jerarquía de operadores establecida:



En el diagrama anterior se muestran las operaciones de '+' y '-' que son las de menor prioridad, acompañadas por el recuadro "término". El siguiente diagrama muestra a este último, cuyo contenido es un poco más complicado de lo que se ha visto hasta ahora. Esto se debe a que se maneja la negación de variables (con '!') y se manejan los índices de los arreglos con INTEGRAL_CONSTANT, que representan las posiciones de las variables a modificar.



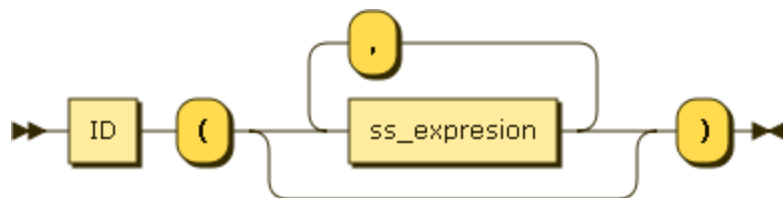
Factores



El diagrama anterior muestra cómo son los factores en Lienzo. Un factor puede involucrar una llamada a un lugar específico de un arreglo, a una constante con valor booleano, una llamada a una función o una expresión entre paréntesis.

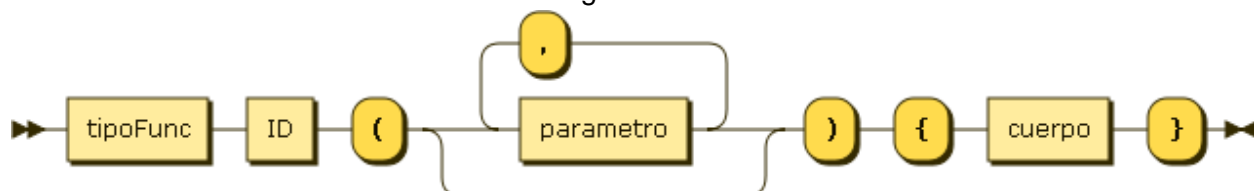
Funciones

Las llamadas a funciones se ven de la siguiente manera:

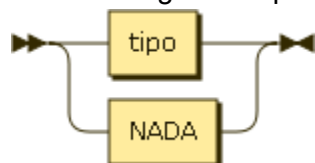


Las funciones se tratan como variables cuando se quiere acceder a ellas es necesario nombrarlas por su identificador, sin embargo también hay que utilizar paréntesis, que pueden o no contener múltiples parámetros (que pueden llevar la forma de una expresión).

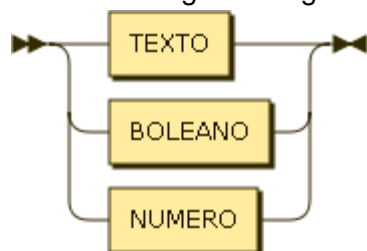
La declaración de una función se ve de la siguiente manera:



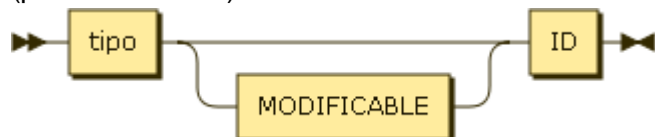
En este diagrama “tipoFunc” representa los tipos de datos que puede regresar una función:



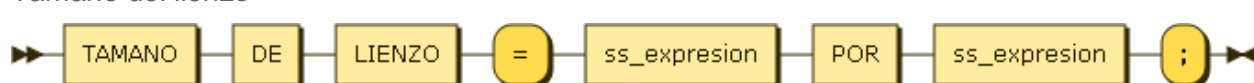
Las funciones pueden o no tener un tipo, cuando se utiliza la palabra “NADA” significa que la función no regresa ningún valor. Los tipos de datos son los siguientes:



Las funciones pueden o no llevar parámetros, y su sintaxis es de la siguiente forma, donde la palabra MODIFICABLE indica que es un parámetro por referencia y no por valor (predeterminado).



Tamaño del lienzo



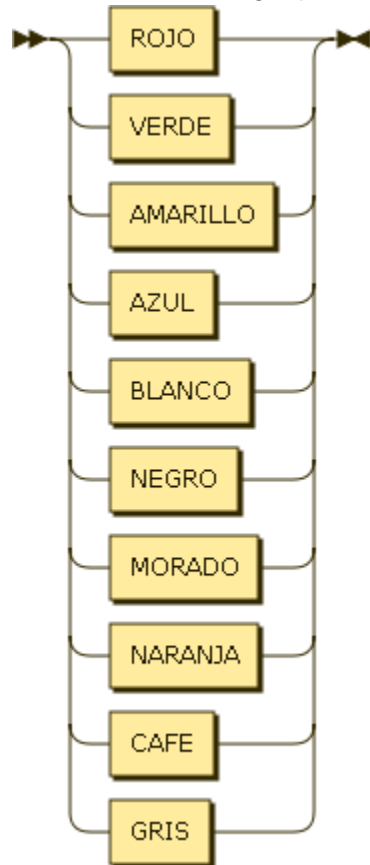
La instrucción anterior se utiliza para modificar el tamaño del lienzo, o la pantalla donde se va a dibujar. Las dimensiones del tamaño, ancho por altura, pueden estar escritas en forma de expresiones aritméticas.

Color del lienzo

Para cambiar el color predeterminado del lienzo (que usualmente es blanco) se utiliza la siguiente instrucción:

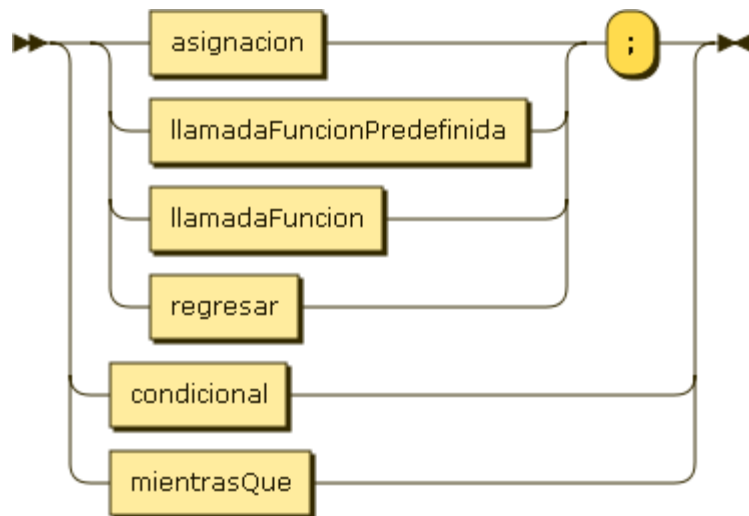


Los colores que se pueden utilizar son los siguientes, y forman parte de las palabras reservadas del lenguaje:

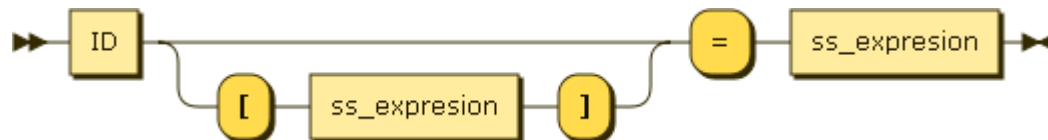


Instrucciones

Las instrucciones, que conforman en su mayoría al programa principal, se ven de la siguiente manera, cada instrucción termina con un “;”:



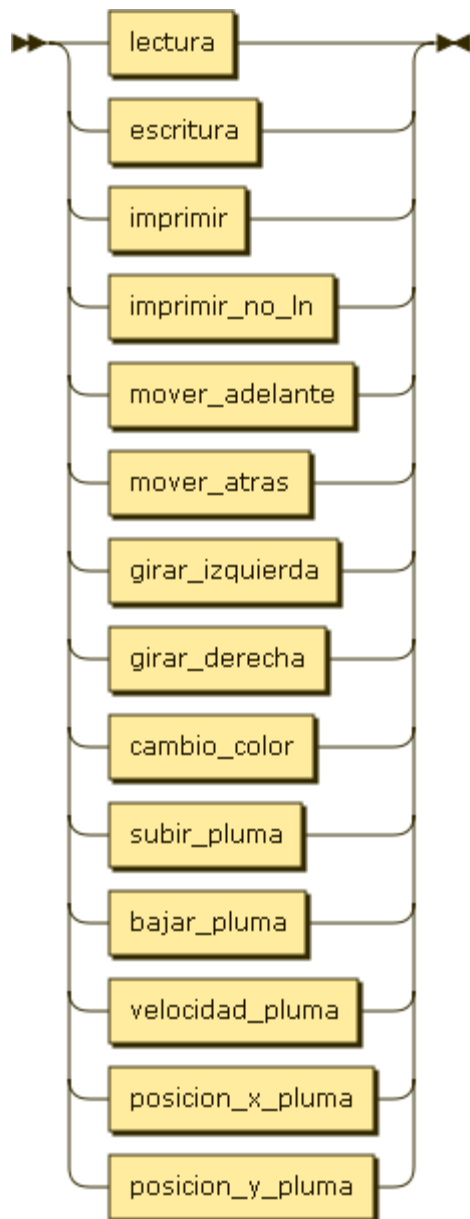
Asignaciones



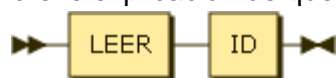
Las asignaciones se utilizan para asignar valores a una variable, ya sea de origen atómico o con múltiples casillas como un arreglo

Llamadas a funciones predefinidas

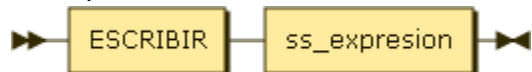
Las funciones predefinidas son aquellas que ya forman parte del lenguaje Lienzo y se encuentran dentro del siguiente listado:



A continuación se encuentra cada uno de los diagramas de las funciones predefinidas y una breve explicación de qué hace cada una.



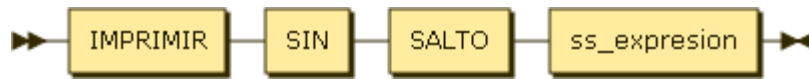
Sirve para leer una variable de la línea de comandos.



Sirve para escribir un texto dentro del lienzo.



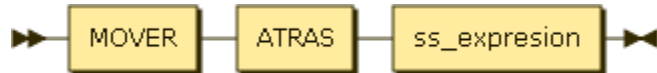
Sirve para imprimir un mensaje en la terminal.



Sirve para imprimir un mensaje en la terminal, pero cuando lo imprime no hace salto de línea.



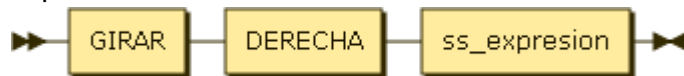
Sirve para mover adelante la pluma una cantidad delimitada por una expresión.



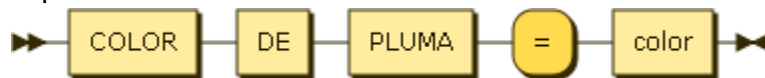
Sirve para mover la pluma hacia atrás una cantidad delimitada por una expresión.



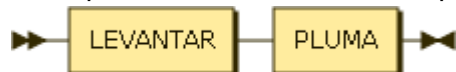
Sirve para girar la pluma hacia la derecha una cantidad de grados delimitada por una expresión.



Sirve para girar la pluma hacia la izquierda una cantidad de grados delimitada por una expresión.



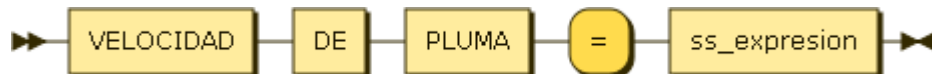
Sirve para cambiar el color de la pluma.



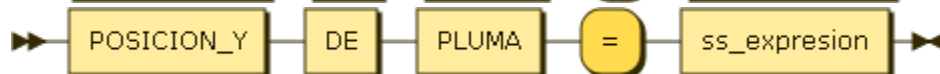
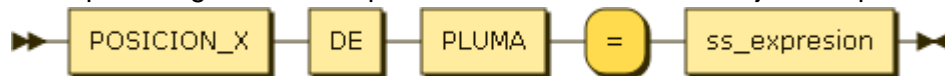
Instrucción que levanta la pluma, causando que las instrucciones subsecuentes no pinten en el lienzo.



Instrucción que baja la pluma, permitiendo que las instrucciones subsecuentes pinten en el lienzo.



Sirve para asignarle una expresión a la velocidad de dibujo de la pluma.



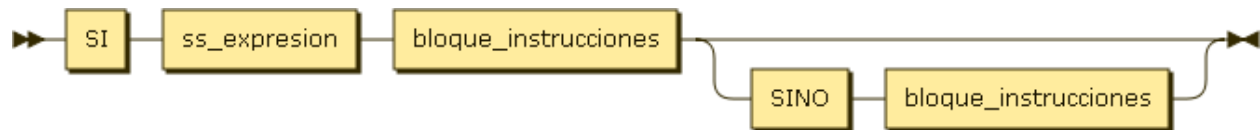
Sirven para cambiar la posición X o Y de la pluma una cierta cantidad delimitada por una expresión.

Regresar (valor de retorno)



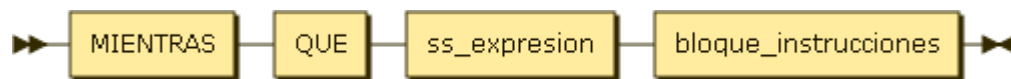
El “regresar” se utiliza para regresar un valor, que puede estar o no en la forma de una expresión, de una función que tiene un tipo asociado.

Condicional



La condicional se usa para ejecutar un bloque de instrucciones siempre y cuando se cumpla una condición. La condicional tiene dos formas de escribirse, una es con SI seguido de una expresión y un bloque de instrucciones. Si se cumple la expresión entonces se ejecutan las instrucciones. Sin embargo, como parte opcional se puede utilizar un SINO después del bloque de instrucciones del SI y este sirve para indicar que se ejecute el siguiente bloque de instrucciones en caso de que no se cumpla la condición del SI.

Mientras que (ciclos)



El ciclo en el lenguaje Lienzo se maneja con la instrucción “mientras que”, seguida de una expresión que si se evalúa verdadera hace que se ejecute un bloque de instrucciones. En caso de que sí se cumpla la expresión y que se ejecute el bloque de instrucciones, se vuelve a evaluar la expresión y así sucesivamente hasta que la expresión se deje de satisfacer.

Fin del archivo



Cuando las instrucciones se terminan se llega al fin del archivo y el programa está listo para mandarse a ejecutar.

Proceso de Administración de Memoria usado en la compilación.

Para administrar la memoria primeramente se fueron procesando las instrucciones en lenguaje Lienzo y guardándose en cuádruplos. La clase de cuádruplos es la siguiente:

```
class Cuadрупlos:
    def __init__(self):
        self.listaCuadрупlos = []
        self.pilaSaltos = []

    def getCuadрупlos(self):
        return self.listaCuadрупlos

    def addCuadрупlo(self, functionName, operator, op1, op2, t=None, generateT=True):
        if not t and generateT:
            t = TemporalRegister(functionName)
        self.listaCuadрупlos.append([operator, op1, op2, t])
        return t

    def editCuadрупlo(self, indice, t):
        self.listaCuadрупlos[indice][3]=t

    def last(self):
        return len(self.listaCuadрупlos)-1

    def pushPilaSaltos(self, indice):
        self.pilaSaltos.append(indice)

    def popPilaSaltos(self):
        return self.pilaSaltos.pop()

    def current(self):
        return len(self.listaCuadрупlos)
```

La clase Cuádruplos cuenta con una lista en python donde se guarda cada cuádruplo. Los cuádruplos consisten de varios campos en donde se guardan las variables y operaciones que se utilizan en cada línea de código. Cada cuádruplo tiene asociada una dirección de memoria, y esta se maneja en la clase MemoryRegister. Los cuádruplos en la lista de cuádruplos

MemoryRegister es nada más una clase genérica que representa un espacio de memoria en la máquina virtual. En sí, solamente es una administración de contadores. Estos contadores sirven para saber a qué casilla acceder en la estructura de ejecución de la máquina virtual

DESCRIPCIÓN DE LA MÁQUINA VIRTUAL

Equipo de cómputo, lenguaje y utilerías especiales usadas

Se utilizó el mismo equipo de cómputo que en el compilador. La librería adicional que se utilizó se llama Python Turtle y es para poder hacer gráficos con Python..

Descripción detallada del proceso de Administración de Memoria en ejecución

Para administrar la memoria en ejecución se utilizó una lista que simulaba ser una pila. Básicamente, el último elemento de la pila es el ámbito global, y los demás elementos son los ámbitos locales de las llamadas a las funciones que todavía no terminan de ejecutarse. Cada elemento de la pila es una tupla de dos: el primer elemento es una lista con los registros de memoria locales (o globales, si estamos en el ámbito global), y el segundo van los temporales. De esta forma se trabaja con las llamadas a funciones.

En el caso de arreglos, cada registro es una pareja. El primer elemento es el arreglo en si y el segundo es el índice al cual hay que acceder. Para los parámetros por referencia, se utiliza un acercamiento similar, de modo que se accede al registro original de la variable.

PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE

Para probar el funcionamiento del lenguaje se utilizaron varias de las siguientes pruebas. A continuación se muestra tanto su descripción como su resultado.

Prueba de ordenamiento de arreglos

En el siguiente programa se muestra un código donde se genera un arreglo, se ordena e imprime utilizando el algoritmo de burbuja. Se puede observar también el uso de comentarios dentro del código, delimitados por “//”.

Código

```
numero[10] arreglo;

nada imprimirArreglo() {
    numero i = 0;
    mientras que (i < 10) {
        imprimir sin salto arreglo[i];
        imprimir sin salto " ";
        i = i + 1;
    }
    imprimir "";
}

nada llenarArreglo() {
    numero i = 0;
```

```

    mientras que (i < 10) {
        arreglo[i] = 10 - i;
        i = i + 1;
    }
}

numero hola() {
    regresar 3;
}

nada ordenarArreglo() {
    numero i = 0;
    numero j = 1;
    numero temp = 0;

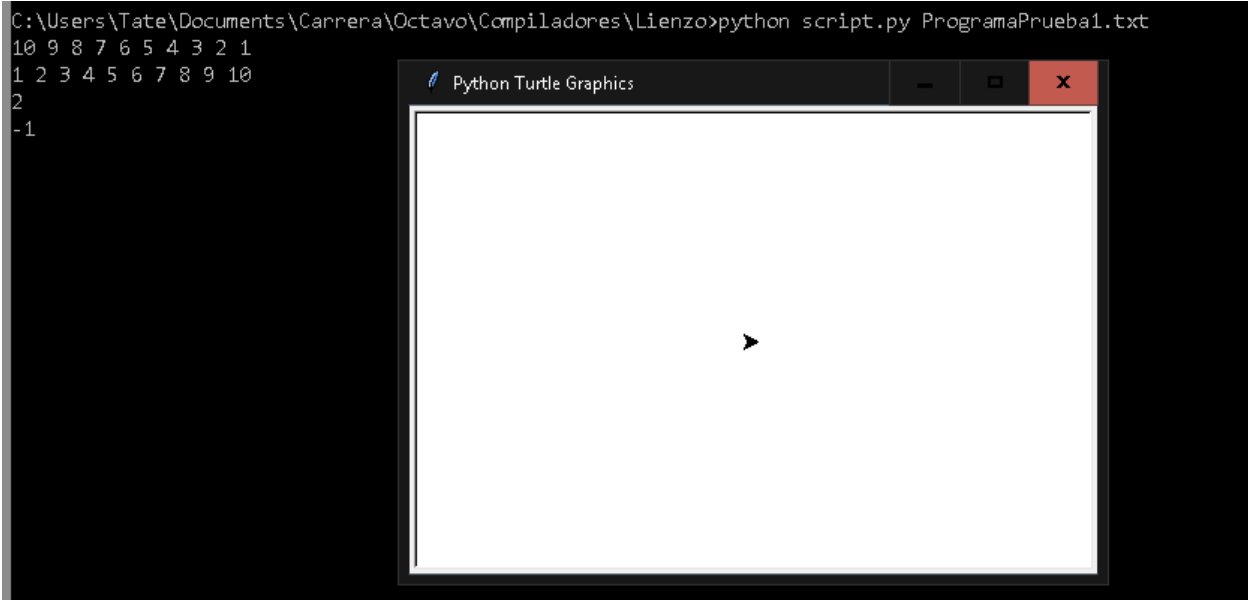
    mientras que (i < 10) {
        j = i + 1;
        mientras que (j < 10) {
            si (arreglo[i] > arreglo[j]) {
                temp = arreglo[i];
                arreglo[i] = arreglo[j];
                arreglo[j] = temp;
            }
            j = j + 1;
        }
        i = i + 1;
    }
}

// HOLA
// BYE
numero buscar(numero x) {
    numero i = 0;
    booleano encontrado = falso;
    mientras que (!encontrado & i < 10) {
        si (arreglo[i] == x) {
            regresar i;
        }
        i = i + 1;
    }
    regresar -1;
}

llenarArreglo();
imprimirArreglo();
ordenarArreglo();
imprimirArreglo();
imprimir(buscar(hola()));
imprimir(buscar(11));

```

Comprobación



Prueba Fibonacci

La siguiente prueba consiste de construir la serie de fibonacci tanto de manera iterativa como recursiva en el lenguaje Lienzo.

Código

```
numero fibonacciIterativo(numero x) {
```

```
    numero[100] aux;
    numero i = 2;
```

```
    aux[0] = 1;
    aux[1] = 1;
```

```
    mientras que (i <= x) {
        aux[i] = aux[i - 1] + aux[i - 2];
        i = i + 1;
    }
```

```
    regresar aux[x];
}
```

```
numero factorialIterativo(numero x) {
```

```
    numero acum = 1;
    numero i = 2;
```

```
    mientras que (i <= x) {
        acum = acum * i;
        i = i + 1;
    }
```

```
    regresar acum;
```



```

}

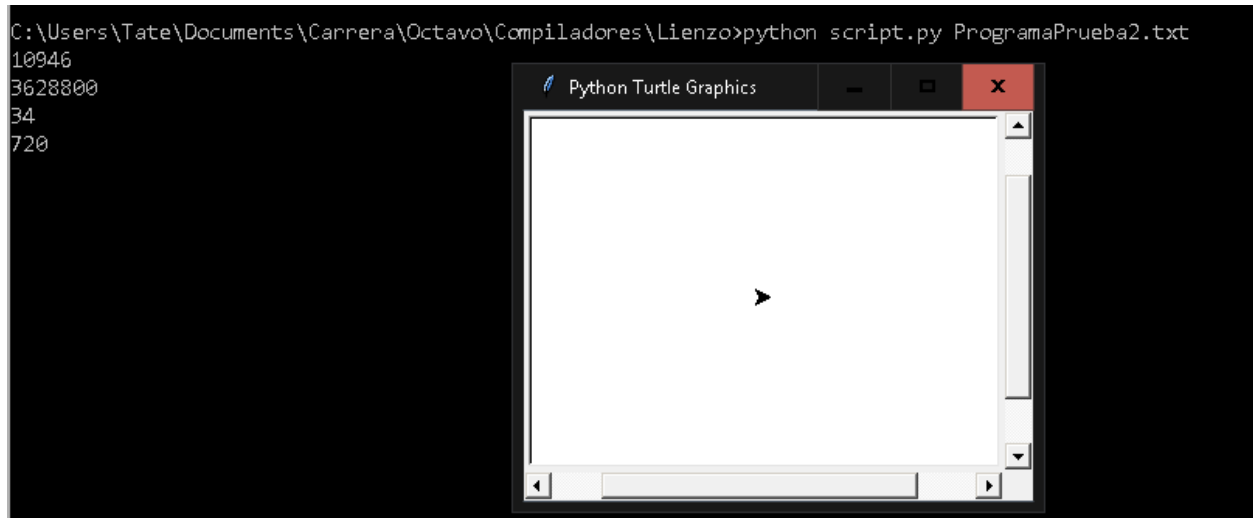
numero fibonacciRecursivo(numero i) {
    si i < 2
        regresar 1;
    regresar fibonacciRecursivo(i - 1) + fibonacciRecursivo(i - 2);
}

numero factorialRecursivo(numero k) {
    si k < 2
        regresar 1;
    regresar k * factorialRecursivo(k - 1);
}

imprimir fibonacciRecursivo(20);
imprimir factorialRecursivo(10);
imprimir fibonacciIterativo(8);
imprimir factorialIterativo(6);

```

Comprobación



```

C:\Users\Tate\Documents\Carrera\Octavo\Compiladores\Lienzo>python script.py ProgramaPrueba2.txt
10946
3628800
34
720

```

Prueba de dibujo #1

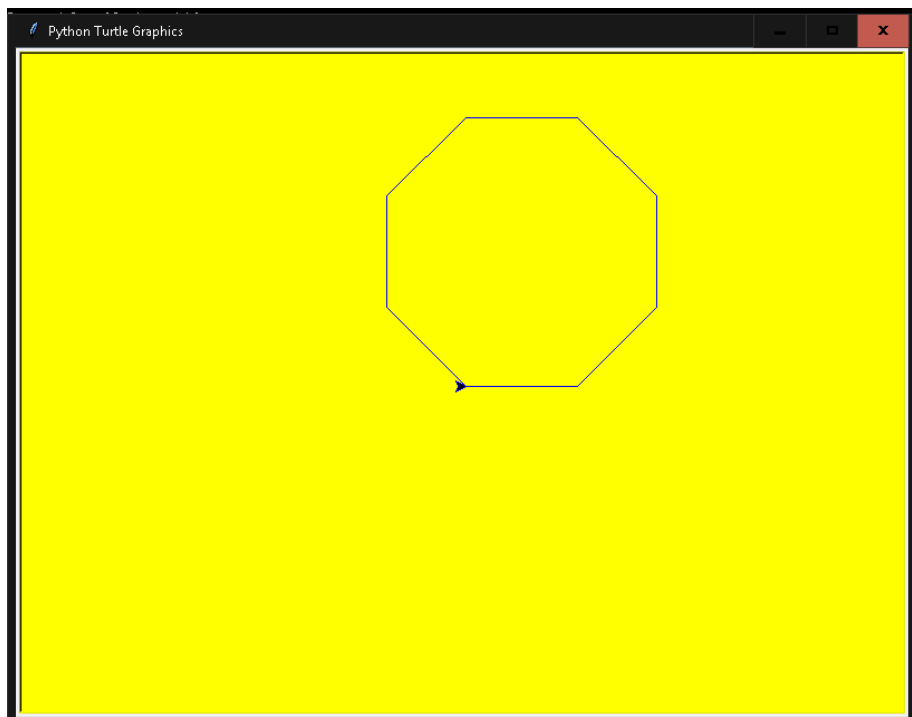
En la siguiente prueba se dibuja un hexágono utilizando el lenguaje Llenzo. También se define el tamaño de la pantalla y el color de la misma.

Código

```
color de lienzo = amarillo;  
tamano de lienzo = 800 por 600;
```

```
velocidad de pluma = 10;  
color de pluma = azul;  
mover adelante 100;  
girar izquierda 45;  
mover adelante 100;  
girar izquierda 45;  
mover adelante 100;  
girar izquierda 45;  
mover adelante 100;  
girar izquierda 45;  
mover adelante 100;  
girar izquierda 45;  
mover adelante 100;  
girar izquierda 45;  
mover adelante 100;  
girar izquierda 45;  
mover adelante 100;  
girar izquierda 45;
```

Comprobación



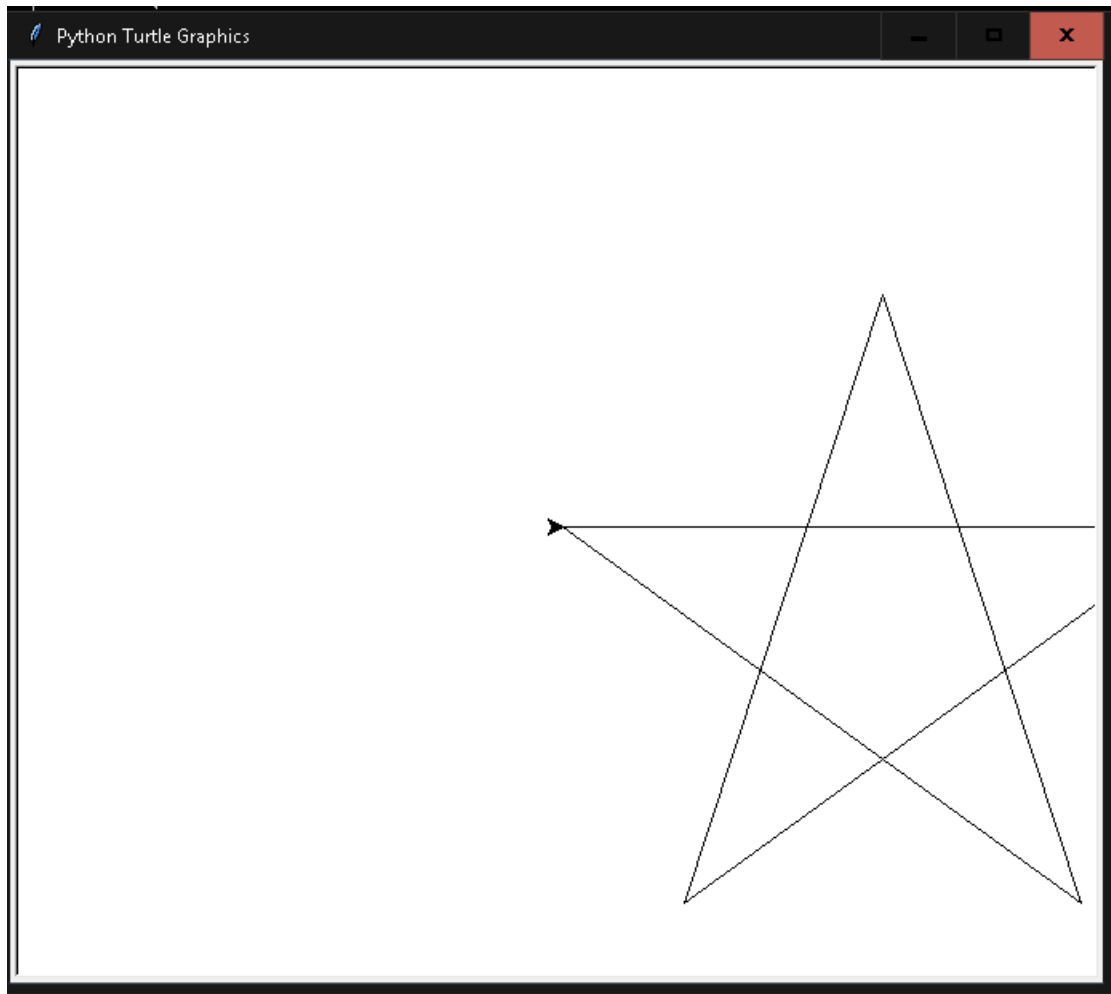
Prueba de dibujo #2

En la siguiente prueba también se hace uso de la pluma, pero en esta ocasión se utiliza un ciclo para moverla.

Código

```
numero i = 0;  
  
mientras que (i < 5) {  
    mover adelante 400;  
    girar derecha 144;  
    i = i + 1;  
}
```

Comprobación



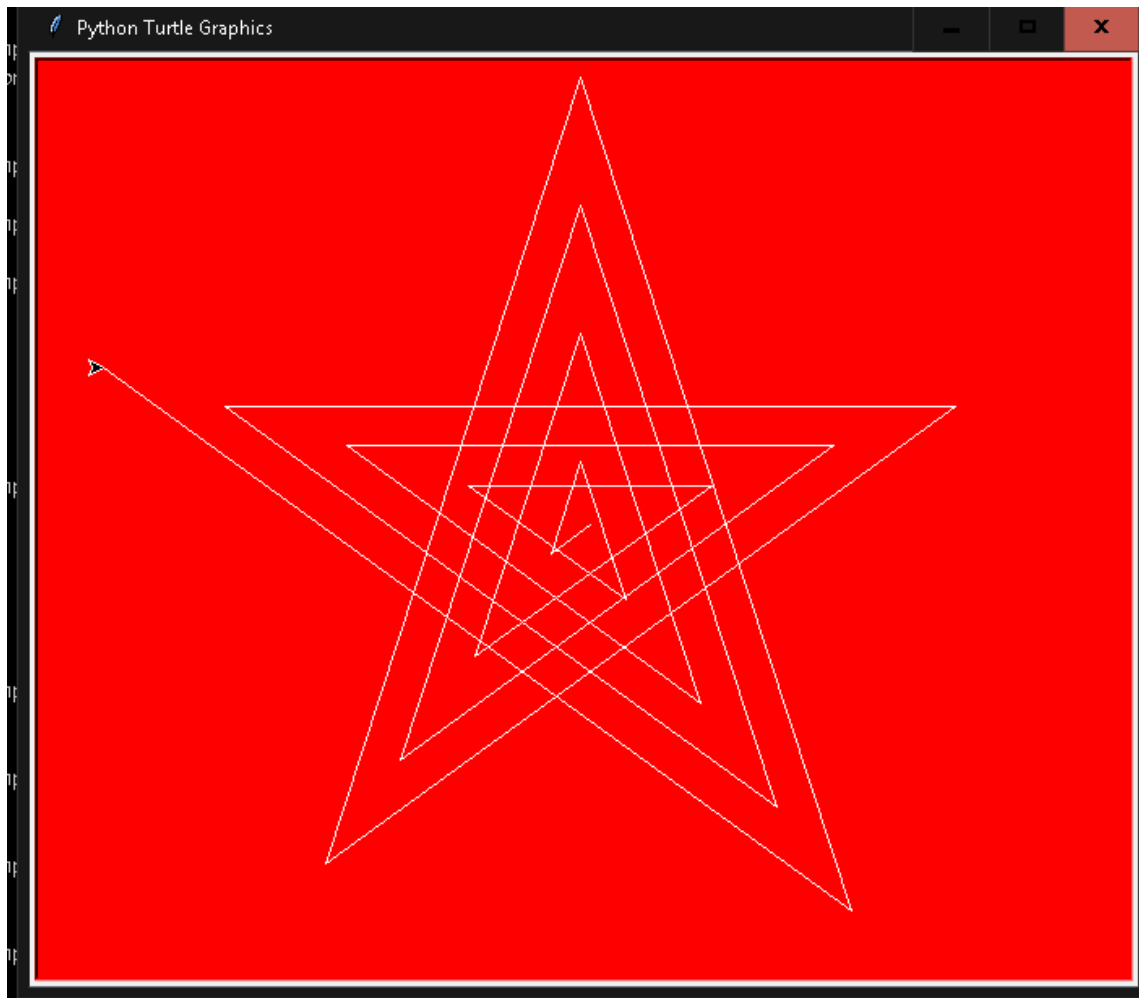
Prueba de dibujo #3

A continuación se muestra otra prueba utilizando la pluma y un ciclo. En esta ocasión también se cambió el color de la pluma.

Código

```
numero i = 0;  
  
color de lienzo = rojo;  
color de pluma = blanco;  
  
mientras que (i < 20) {  
    mover adelante i * 30;  
    girar derecha 144;  
    i = i + 1;  
}
```

Comprobación



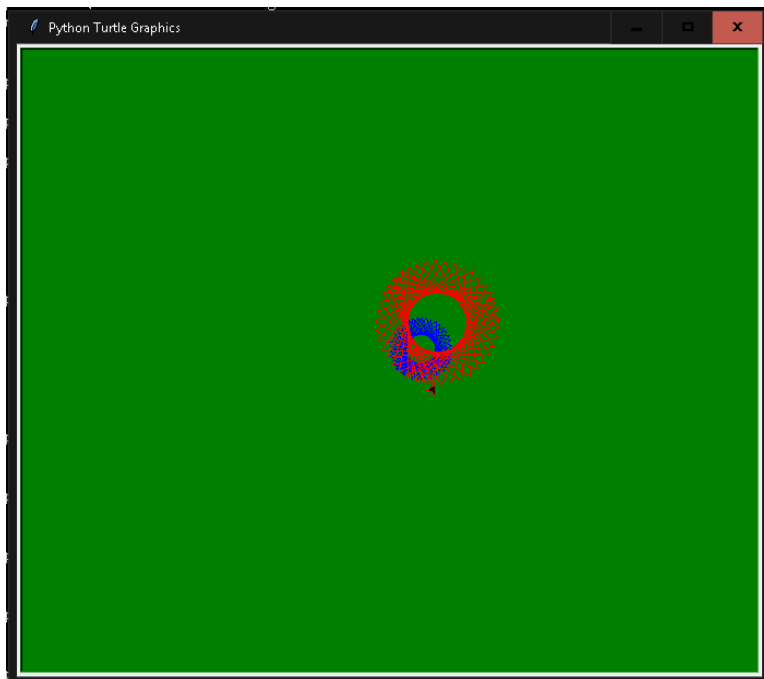
Prueba de dibujo #4

En el siguiente programa se juega con los distintos colores de la pluma, se dibuja una figura con un color y posteriormente otra con otro color. También se modifica la velocidad de la pluma.

Código

```
numero i = 0;  
color de lienzo = verde;  
  
velocidad de pluma = 10;  
color de pluma = azul;  
  
mientras que (i < 50) {  
    mover adelante 50;  
    girar izquierda 123;  
    i = i + 1;  
}  
  
color de pluma = rojo;  
i = 0;  
mientras que (i < 50) {  
    mover adelante 100;  
    girar izquierda 123;  
    i = i + 1;  
}
```

Comprobación



Prueba de dibujo #5

En la siguiente prueba se experimenta con la pluma de lienzo, esta vez se hace uso de la instrucción *levantar pluma* que permite al programador moverla sin pintar sobre el lienzo, y *bajar pluma* para que ésta vuelva a pintar.

Código

```
numero i = 0;

color de lienzo = amarillo;

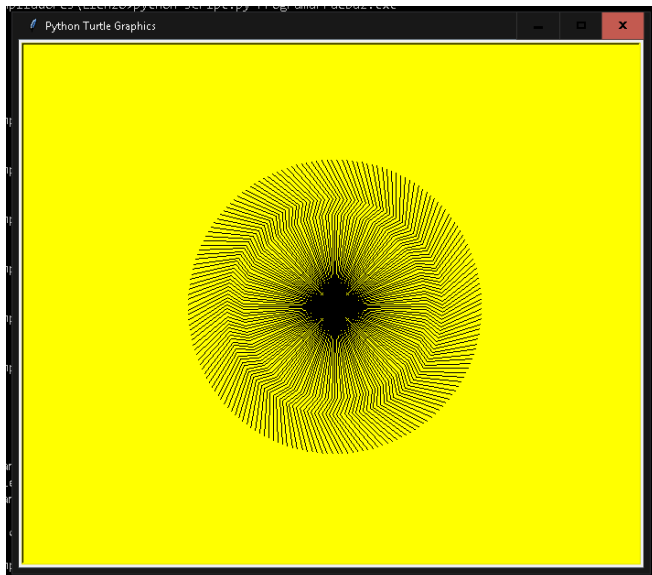
velocidad de pluma = 0;

mientras que (i < 180) {
    mover adelante 100;
    girar derecha 30;
    mover adelante 20;
    girar izquierda 60;
    mover adelante 50;
    girar derecha 30;

    levantar pluma;
    posicionX de pluma = 0;
    posicionY de pluma = 0;
    bajar pluma;

    girar derecha 2;
    i = i + 1;
}
```

Comprobación



LISTADOS PERFECTAMENTE DOCUMENTADOS DEL PROYECTO

Lienzo.g4 : la gramatica

grammar Lienzo;

```
options {  
    language=Python3;  
}
```

```
@header {  
from namespace import NamespaceTable  
from collections import defaultdict  
from MemoryRegister import MemoryRegisters  
from cuadрупlos import *  
import sys  
import VM
```

```
namespaceTable = NamespaceTable()  
currentFunctionName = ""  
currentTipoFunc = ""  
hasReturn = False
```

```
memoryregisters = MemoryRegisters()  
cuadрупlos = Cuadрупlos()
```

```
BOLEANO = "booleano"  
TEXTO = "texto"  
NUMERO = "numero"
```

```
cubo = {}  
cubo[BOLEANO] = {}  
cubo[TEXTO] = {}  
cubo[NUMERO] = {}
```

```
# Booleanos  
cubo[BOLEANO]['+'] = defaultdict(lambda: None, {})  
cubo[BOLEANO]['-'] = defaultdict(lambda: None, {})  
cubo[BOLEANO]['*'] = defaultdict(lambda: None, {})  
cubo[BOLEANO]['/'] = defaultdict(lambda: None, {})  
cubo[BOLEANO]['%'] = defaultdict(lambda: None, {})  
cubo[BOLEANO]['<'] = defaultdict(lambda: None, {})  
cubo[BOLEANO]['>'] = defaultdict(lambda: None, {})  
cubo[BOLEANO]['<='] = defaultdict(lambda: None, {})  
cubo[BOLEANO]['>='] = defaultdict(lambda: None, {})  
cubo[BOLEANO]['=='] = defaultdict(lambda: None, {BOLEANO : BOLEANO})  
cubo[BOLEANO]['!='] = defaultdict(lambda: None, {BOLEANO : BOLEANO})  
cubo[BOLEANO]['&'] = defaultdict(lambda: None, {BOLEANO : BOLEANO})  
cubo[BOLEANO]['|'] = defaultdict(lambda: None, {BOLEANO : BOLEANO})
```

```

# Mensajes
cubo[TEXTO]['+'] = defaultdict(lambda: None, {TEXTO : TEXTO})
cubo[TEXTO]['-'] = defaultdict(lambda: None, {})
cubo[TEXTO]['*'] = defaultdict(lambda: None, {NUMERO : TEXTO})
cubo[TEXTO]['/'] = defaultdict(lambda: None, {})
cubo[TEXTO]['%'] = defaultdict(lambda: None, {})
cubo[TEXTO]['<'] = defaultdict(lambda: None, {TEXTO : BOLEANO})
cubo[TEXTO]['>'] = defaultdict(lambda: None, {TEXTO : BOLEANO})
cubo[TEXTO]['<='] = defaultdict(lambda: None, {TEXTO : BOLEANO})
cubo[TEXTO]['>='] = defaultdict(lambda: None, {TEXTO : BOLEANO})
cubo[TEXTO]['=='] = defaultdict(lambda: None, {TEXTO : BOLEANO})
cubo[TEXTO]['!='] = defaultdict(lambda: None, {TEXTO : BOLEANO})
cubo[TEXTO]['&'] = defaultdict(lambda: None, {})
cubo[TEXTO]['|'] = defaultdict(lambda: None, {})

# Numero
cubo[NUMERO]['+'] = defaultdict(lambda: None, {NUMERO : NUMERO})
cubo[NUMERO]['-'] = defaultdict(lambda: None, {NUMERO : NUMERO})
cubo[NUMERO]['*'] = defaultdict(lambda: None, {NUMERO : NUMERO, TEXTO : TEXTO})
cubo[NUMERO]['/'] = defaultdict(lambda: None, {NUMERO : NUMERO})
cubo[NUMERO]['%'] = defaultdict(lambda: None, {NUMERO : NUMERO})
cubo[NUMERO]['<'] = defaultdict(lambda: None, {NUMERO : BOLEANO})
cubo[NUMERO]['>'] = defaultdict(lambda: None, {NUMERO : BOLEANO})
cubo[NUMERO]['<='] = defaultdict(lambda: None, {NUMERO : BOLEANO})
cubo[NUMERO]['>='] = defaultdict(lambda: None, {NUMERO : BOLEANO})
cubo[NUMERO]['=='] = defaultdict(lambda: None, {NUMERO : BOLEANO})
cubo[NUMERO]['!='] = defaultdict(lambda: None, {NUMERO : BOLEANO})
cubo[NUMERO]['&'] = defaultdict(lambda: None, {})
cubo[NUMERO]['|'] = defaultdict(lambda: None, {})

def num(s):
    try:
        return int(s)
    except ValueError:
        return float(s)

def error(linea, mensaje):
    print("Error: linea", linea, ":", mensaje)
    sys.exit(0)
}

program:
    declaracion* ((tamanoLienzo)? (colorLienzo)? | (colorLienzo)? (tamanoLienzo)?)
    {
        cuadрупlos.addCuadрупlo("", GOTO, None, None, None, False)
        cuadрупlos.pushPilaSaltos(cuadрупlos.last())
    } funcion*
    {
        cuadрупlos.editCuadрупlo(cuadрупlos.popPilaSaltos(), cuadрупlos.current())
    } instruccion_aux* EOF
    {

```



```

cuadрупlos.addCuadрупlo("", END, None, None, None, False)
VM.executeVM(namespaceTable.getDirProc(), cuadрупlos.getCuadрупlos())
}
;

colorLienzo:
    COLOR DE LIENZO '=' color ';'
{
    cuadрупlos.addCuadрупlo("", CANVAS_COLOR, $color.start.text, None, None, False)
}
;

color:
    ROJO
    | VERDE
    | AMARILLO
    | AZUL
    | BLANCO
    | NEGRO
    | MORADO
    | NARANJA
    | CAFE
    | GRIS
;

tamanoLienzo:
    TAMANO DE LIENZO '=' largo=ss_expresion POR ancho=ss_expresion ';'
{
    if $largo.type != NUMERO:
        error($largo.start.line, "Largo del lienzo debe ser una expresion entera")
    elif $ancho.type != NUMERO:
        error($ancho.start.line, "Ancho del lienzo debe ser una expresion entera")
    else:
        cuadрупlos.addCuadрупlo("", CANVAS_SIZE, $largo.valor, $ancho.valor, None, False)
}
;

declaracion:
    declaracion_variable | declaracion_arreglo
;

declaracion_variable:
    tipo ID
{
    if namespaceTable.idAlreadyTaken($ID.text, currentFunctionName):
        error($ID.line, ": Identificador " + $ID.text + " ya fue declarado")
} '=' ss_expresion ';'
{
    if $ss_expresion.type != $tipo.text:
        error($ID.line, "Variable " + $ID.text + " es de tipo " + $tipo.text)
    else:

```

```

        namespaceTable.addVariable($ID.text, $tipo.text, currentFunctionName)
        idcontent = memoryregisters.createMemoryRegister($ID.text, currentFunctionName,
$tipo.text)
        cuadрупlos.addCuadрупlo(currentFunctionName, ASSIGN, $ss_expresion.valor, None,
idcontent, False)
    }
    ;

declaracion_arreglo:
    tipo '[' INTEGRAL_CONSTANT ']' ID ';'
{
if namespaceTable.idAlreadyTaken($ID.text, currentFunctionName):
    error($ID.line, ": Identificador " + $ID.text + " ya fue declarado")
else:
    length = num($INTEGRAL_CONSTANT.text)
    if length == 0:
        error($ID.line, ": La longitud del arreglo " + $ID.text + " debe ser mayor a 0")
    else:
        namespaceTable.addArray($ID.text, $tipo.text, length, currentFunctionName)
        idcontents = memoryregisters.createMemoryRegisterForArray($ID.text, $tipo.text,
length, currentFunctionName)
        valorInicial = 0
        if $tipo.text == BOLEANO:
            valorInicial = False
        elif $tipo.text == TEXTO:
            valorInicial = ""
        for i in idcontents:
            cuadрупlos.addCuadрупlo(currentFunctionName, ASSIGN, valorInicial, None, i, False)
    }
    ;

funcion:
    tipoFunc ID
{
global currentFunctionName
global currentTipoFunc
currentFunctionName = $ID.text
currentTipoFunc = $tipoFunc.text
if not namespaceTable.addFunction(currentFunctionName, $tipoFunc.text,
cuadрупlos.current()):
    error($ID.line, "Funcion " + $ID.text + " ya fue declarada")
else:
    mr = memoryregisters.newFunction(currentFunctionName, $tipoFunc.text)
    valorInicial = 0
    if $tipoFunc.text == BOLEANO:
        valorInicial = False
    elif $tipoFunc.text == TEXTO:
        valorInicial = ""
    cuadрупlos.addCuadрупlo("", ASSIGNFUNC, valorInicial, None, mr, False)
}
'(' (parametro (' parametro*)? ')' '{' cuerpo '}')

```

```

{
global hasReturn
if currentTipoFunc != "nada" and not hasReturn:
    error($ID.line, "Funcion " + $ID.text + " debe tener valor de retorno")
else:
    hasReturn = False
cuadрупlos.addCuadрупlo(currentFunctionName, RET, None, None, None, False)
currentFunctionName = ""
currentTipoFunc = ""
}
;

tipoFunc:
    tipo | NADA
;

parametro:
    tipo MODIFICABLE? ID
{
modificable = False
if $MODIFICABLE.text:
    modificable = True
if not namespaceTable.addParameter(currentFunctionName, $ID.text, $tipo.text, modificable):
    error($ID.line, "Parametro " + $ID.text + " ya fue declarado")
else:
    memoryregisters.createMemoryRegister($ID.text, currentFunctionName)
}
;

cuerpo:
    declaracion* instruccion_aux*
;

bloque_instrucciones:
    instruccion_aux | '{' instruccion_aux* '}'
;

instruccion_aux:
    ((
        asignacion
        | llamadaFuncionPredefinida
        | llamadaFuncion
        | regresar
    ) ';' ) | condicional | mientrasQue
;

regresar:
    REGRESAR ss_expression
{
global hasReturn
hasReturn = True

```

```

if currentTipoFunc == "nada":
    error($REGRESAR.line, "Funcion " + currentFunctionName + " no debe tener valor de
retorno")
elif $ss_expresion.type != currentTipoFunc:
    error($REGRESAR.line, "Funcion " + currentFunctionName + " tiene valor de retorno de
tipo incorrecto. Se esperaba un " + currentTipoFunc)
else:
    cuadрупlos.addCuadрупlo(currentFunctionName, RETURN, $ss_expresion.valor, None,
None, False)
    cuadрупlos.addCuadрупlo(currentFunctionName, RET, None, None, None, False)
}
;

```

llamadaFuncionPredefinida:

```

lectura
| escritura
| imprimir
| imprimir_no_In
| mover_adelante
| mover_atras
| girar_izquierda
| girar_derecha
| cambio_color
| subir_pluma
| bajar_pluma
| velocidad_pluma
| posicion_x_pluma
| posicion_y_pluma
;

```

lectura:

```

LEER ID
{
idcontent = memoryregisters.getMemoryRegister($ID.text, currentFunctionName)
cuadрупlos.addCuadрупlo(currentFunctionName, READ, None, None, idcontent)
}
;

```

escritura:

```

ESCRIBIR ss_expresion
{
cuadрупlos.addCuadрупlo(currentFunctionName, WRITE, $ss_expresion.valor, None, None,
False)
}
;

```

imprimir_no_In:

```

IMPRIMIR SIN SALTO ss_expresion
{

```

```

cuadрупlos.addCuadрупlo(currentFunctionName, PRINT, $ss_expresion.valor, None, None,
False)
}
;

imprimir:
    IMPRIMIR ss_expresion
{
cuadрупlos.addCuadрупlo(currentFunctionName, PRINTLN, $ss_expresion.valor, None, None,
False)
}
;

mover_adelante:
    MOVER ADELANTE ss_expresion
{
cuadрупlos.addCuadрупlo(currentFunctionName, FORWARD, $ss_expresion.valor, None,
None, False)
}
;

mover_atras:
    MOVER ATRAS ss_expresion
{
cuadрупlos.addCuadрупlo(currentFunctionName, BACKWARD, $ss_expresion.valor, None,
None, False)
}
;

girar_derecha:
    GIRAR DERECHA ss_expresion
{
cuadрупlos.addCuadрупlo(currentFunctionName, RIGHT, $ss_expresion.valor, None, None,
False)
}
;

girar_izquierda:
    GIRAR IZQUIERDA ss_expresion
{
cuadрупlos.addCuadрупlo(currentFunctionName, LEFT, $ss_expresion.valor, None, None,
False)
}
;

subir_pluma:
    LEVANTAR PLUMA
{
cuadрупlos.addCuadрупlo(currentFunctionName, PENUP, None, None, None, False)
}
;

```

```

bajar_pluma:
    BAJAR PLUMA
{
    cuadрупlos.addCuadрупlo(currentFunctionName, PENDOWN, None, None, None, False)
}
    ;

cambio_color:
    COLOR DE PLUMA '=' color
{
    cuadрупlos.addCuadрупlo(currentFunctionName, COLOR_CHANGE, $color.text, None, None,
False)
}
    ;

velocidad_pluma:
    VELOCIDAD DE PLUMA '=' ss_expresion
{
    if $ss_expresion.type == NUMERO:
        cuadрупlos.addCuadрупlo(currentFunctionName, SETSPEED, $ss_expresion.valor,
None, None, False)
    else:
        error($ss_expresion.start.line, "La velocidad debe ser un numero")
}
    ;

posicion_x_pluma:
    POSICION_X DE PLUMA '=' ss_expresion
{
    if $ss_expresion.type == NUMERO:
        cuadрупlos.addCuadрупlo(currentFunctionName, PEN_POSX, $ss_expresion.valor,
None, None, False)
    else:
        error($ss_expresion.start.line, "La coordenada x debe ser un numero")
}
    ;

posicion_y_pluma:
    POSICION_Y DE PLUMA '=' ss_expresion
{
    if $ss_expresion.type == NUMERO:
        cuadрупlos.addCuadрупlo(currentFunctionName, PEN_POSY, $ss_expresion.valor,
None, None, False)
    else:
        error($ss_expresion.start.line, "La coordenada y debe ser un numero")
}
    ;

asignacion:
    ID (arr='[' indice=ss_expresion ']')? '=' lhs=ss_expresion

```

```

{
# regresa el tipo, no importa si es arreglo o variable
idType = namespaceTable.getVariableType($ID.text, currentFunctionName)
if not $arr:
    if not idType:
        error($ID.line, "Variable " + $ID.text + " no ha sido declarada")
    elif $lhs.type != idType:
        error($ID.line, "Variable " + $ID.text + " es de tipo " + idType)
    else:
        idcontent = memoryregisters.getMemoryRegister($ID.text, currentFunctionName)
        cuadрупlos.addCuadрупlo(currentFunctionName, ASSIGN, $lhs.valor, None, idcontent)
else:
    if not idType:
        error($ID.line, "Arreglo " + $ID.text + " no ha sido declarado")
    elif $lhs.type != idType:
        error($ID.line, "Arreglo " + $ID.text + " es de tipo " + idType)
    else:
        cuadрупlos.addCuadрупlo(currentFunctionName, CHECK_BOUNDS, $indice.valor,
namespaceTable.getArrayLength($ID.text, currentFunctionName), None, False)
        idcontent = memoryregisters.getArrayMemoryRegister($ID.text, $indice.valor,
currentFunctionName)
        cuadрупlos.addCuadрупlo(currentFunctionName, ASSIGN, $lhs.valor, None, idcontent)
}
;

tipo:
    TEXTO
    | BOLEANO
    | NUMERO
;

condicional:
    SI ss_expresion
{
if $ss_expresion.type != BOLEANO:
    error($ss_expresion.start.line, "el estatuto 'si' necesita una booleano")
else:
    cuadрупlos.addCuadрупlo(currentFunctionName, GOTO, $ss_expresion.valor, None,
None, False)
    cuadрупlos.pushPilaSaltos(cuadрупlos.last())
}
    bloque_instrucciones
    (SINO
{
cuadрупlos.addCuadрупlo(currentFunctionName, GOTO, None, None, None, False)
cuadрупlos.editCuadрупlo(cuadрупlos.popPilaSaltos(), cuadрупlos.current())
cuadрупlos.pushPilaSaltos(cuadрупlos.last())
}
    bloque_instrucciones
)?
{

```

```

cuadрупlos.editCuadрупlo(cuadрупlos.popPilaSaltos(),cuadрупlos.current())
}
;

mientrasQue:
    MIENTRAS QUE
    {
cuadрупlos.pushPilaSaltos(cuadрупlos.current())
    }
        ss_expresion
    {
if $ss_expresion.type != BOLEANO:
    error($ss_expresion.start.line, "el estatuto 'mientras que' necesita una boleano")
else:
    cuadрупlos.addCuadрупlo(currentFunctionName,
GOTO,$ss_expresion.valor,None,None,False)
    cuadрупlos.pushPilaSaltos(cuadрупlos.last())
    }
        bloque_instrucciones
    {
pop1 = cuadрупlos.popPilaSaltos()
pop2 = cuadрупlos.popPilaSaltos()
cuadрупlos.addCuadрупlo(currentFunctionName, GOTO,None,None,pop2,False)
cuadрупlos.editCuadрупlo(pop1,cuadрупlos.current())
    }
;

llamadaFuncion returns [valor,type]:
    ID
    {
functionType = namespaceTable.getFunctionType($ID.text)
if not functionType:
    error($ID.line, ": llamada a funcion " + $ID.text + " inexistente")
    $valor = None
else:
    $type = None if functionType == "nada" else functionType
    cuadрупlos.addCuadрупlo(currentFunctionName, ERA, $ID.text, None, None, False)
    $valor = memoryregisters.getMemoryRegister($ID.text, "")
    }
    '('
    {
k = 0
    }
    (ss_exp1=ss_expresion
    {
if namespaceTable.argumentAgree($ID.text, k, $ss_exp1.text, $ss_exp1.type):
    cuadрупlos.addCuadрупlo(currentFunctionName, PARAM, $ss_exp1.valor,
namespaceTable.parameterReference($ID.text, k), None, False)
else:
    error($ss_exp1.start.line, ": argumento #" + k + "no concuerda con el parametro
esperado")

```



```

k += 1
}

(',' ss_exp2=ss_expresion
{
if namespaceTable.argumentAgree($ID.text, k, $ss_exp2.text, $ss_exp2.type):
    cuadрупlos.addCuadрупlo(currentFunctionName, PARAM, $ss_exp2.valor,
namespaceTable.parameterReference($ID.text, k), None, False)
else:
    error($ss_exp1.start.line, ": argumento #" + k + "no concuerda con el parametro
esperado")
k += 1
}
)*)? paren=')'
{
amountOfParameters = namespaceTable.getParameterAmount($ID.text)
if k != amountOfParameters:
    error($paren.line, "Se esperaban" + amountOfParameters + " parametros, se recibieron
" + k)
else:
    cuadрупlos.addCuadрупlo(currentFunctionName, GOSUB,
namespaceTable.getDireccionInicio($ID.text), None, None, False)
}
;

ss_expresion returns [type,valor]:
    s_exp1=s_expresion
{
$type = $s_exp1.type
$val = $s_exp1.valor
}
    (op=('&' | '|') s_exp2=s_expresion
{
tipo = cubo[$type][$op.text][$s_exp2.type]
if not tipo:
    error(op.line, ": operador " + $op.text + " no puede ser aplicado a " + $type + " y a " +
$s_exp2.type)
else:
    namespaceTable.addTemporal(currentFunctionName, tipo)
    $valor = cuadрупlos.addCuadрупlo(currentFunctionName, $op.text, $valor,
$s_exp2.valor)
$type = tipo
}
    )*
;

s_expresion returns [type,valor]:
    exp1=expresion
{
$type = $exp1.type
$val = $exp1.valor
}

```

```

}
    (
        (op=('==' | '!=' | '>' | '<' | '>=' | '<=') exp2=expresion)
    {
        tipo = cubo[$type][$op.text][$exp2.type]
        if not tipo:
            error($op.line, ": operador " + $op.text + " no puede ser aplicado a " + $type + " y a " +
                $exp2.type)
        else:
            $valor = cuadрупlos.addCuadрупlo(currentFunctionName, $op.text,$valor,$exp2.valor)
            namespaceTable.addTemporal(currentFunctionName, tipo)
        $type = tipo
    }
    )*
;

expresion returns [type,valor]:
    term1=termino
    {
        $type = $term1.type
        $valor = $term1.valor
    }
    (
        (op=('+'|'-') term2=termino)
    {
        tipo = cubo[$type][$op.text][$term2.type]
        if not tipo:
            error($op.line, ": operador " + $op.text + " no puede ser aplicado a " + $type + " y a " +
                $term2.type)
        else:
            $valor = cuadрупlos.addCuadрупlo(currentFunctionName, $op.text,$valor,$term2.valor)
            namespaceTable.addTemporal(currentFunctionName, tipo)
        $type = tipo
    }
    )*
;

termino returns [type,valor]:
    factor1=factor
    {
        $type = $factor1.type
        $valor = $factor1.valor
    }
    (op=('*'|'/'|'%') factor2=factor
    {
        tipo = cubo[$type][$op.text][$factor2.type]
        if not tipo:
            error($op.line, ": operador " + $op.text + " no puede ser aplicado a " + $type + " y a " +
                $factor2.type)
        else:
            $valor = cuadрупlos.addCuadрупlo(currentFunctionName, $op.text,$valor,$factor2.valor)

```

```

        namespaceTable.addTemporal(currentFunctionName, tipo)
$type = tipo
    }
    )*
    ;

factor returns [type,valor]:
    (neg='!'? factor_aux)
{
$type = $factor_aux.type
if $neg.text and $factor_aux.type != BOLEANO:
    error($neg.line, "operador " + $neg.text + " no puede ser aplicado a " + $factor_aux.type)
    $type = None
else:
    if $neg.text:
        $valor = cuadрупlos.addCuadрупlo(currentFunctionName, $neg.text, $factor_aux.valor,
None)
        namespaceTable.addTemporal(currentFunctionName, $type)
    else:
        $valor = $factor_aux.valor
} | (neg='-'? n=(NUMERIC_CONSTANT|INTEGRAL_CONSTANT))
{
$type = NUMERO

if $neg.text:
    $valor = cuadрупlos.addCuadрупlo(currentFunctionName, MINUS, 0, num($n.text))
    namespaceTable.addTemporal(currentFunctionName, $type)
else:
    $valor = num($n.text)
} | STRING_CONSTANT
{
$type = TEXTO
$valor = $STRING_CONSTANT.text[1:-1]
}
;

factor_aux returns [type,valor]:
    ID (arr='[' ss_expresion ''])?
{
# no importa si es arreglo o variable, regresa el tipo correcto
$type = namespaceTable.getVariableType($ID.text, currentFunctionName)
if not $arr:
    if $type:
        $valor = memoryregisters.getMemoryRegister($ID.text, currentFunctionName)
    else:
        $valor = None
    error($ID.line, "variable " + $ID.text + " no ha sido declarada")
else:
    if $type:

```

```

        cuadрупlos.addCuadрупlo(currentFunctionName, CHECK_BOUNDS,
$ss_expresion.valor, namespaceTable.getArrayLength($ID.text, currentFunctionName), None,
False)
        $valor = memoryregisters.getArrayMemoryRegister($ID.text, $ss_expresion.valor,
currentFunctionName)
        else:
            $valor = None
            error($ID.line, "Arreglo " + $ID.text + " no ha sido declarado")
    } | BOOLEAN_CONSTANT
    {
        $type = BOLEANO
        $valor = True if $BOOLEAN_CONSTANT.text == 'verdadero' else False
    } | If=llamadaFuncion
    {
        functionType = $llamadaFuncion.type
        $type = functionType if functionType != "nada" else None
        $valor = cuadрупlos.addCuadрупlo(currentFunctionName, ASSIGN, $If.valor, None)
    } | '(' ss_expresion ')'
    {
        $type = $ss_expresion.type
        if $type:
            $valor = $ss_expresion.valor
        else:
            $valor = None
    }
    ;

```

```

WS : ('/' ~('\n'|\r)* '\r'? '\n'
      | '/' '*' ~ '*' '/'
      | [\t\r\n]+) -> skip ; // skip spaces, tabs, newline

```

```

ROJO : 'rojo' ;
VERDE : 'verde' ;
AMARILLO : 'amarillo' ;
AZUL : 'azul' ;
BLANCO : 'blanco' ;
NEGRO : 'negro' ;
MORADO : 'morado' ;
NARANJA : 'naranja' ;
CAFE : 'cafe' ;
GRIS : 'gris' ;
COLOR : 'color' ;
VELOCIDAD : 'velocidad' ;
LIENZO : 'lienzo' ;
EQUALS : '=' ;
TAMANO : 'tamano' ;
POR : 'por' ;
DE : 'de' ;
EN : 'en' ;
MOVER : 'mover' ;
ADELANTE : 'adelante' ;

```

ATRAS : 'atras' ;
 GIRAR : 'girar' ;
 DERECHA : 'derecha' ;
 IZQUIERDA : 'izquierda' ;
 LEVANTAR : 'levantar' ;
 BAJAR : 'bajar' ;
 PLUMA : 'pluma' ;
 MIENTRAS : 'mientras' ;
 REGRESAR : 'regresar' ;
 QUE : 'que' ;
 SIN : 'sin' ;
 SALTO : 'salto' ;
 SI : 'si' ;
 SINO : 'sino' ;
 TEXTO : 'texto' ;
 LEER : 'leer' ;
 POSICION_X : 'posicionX' ;
 POSICION_Y : 'posicionY' ;
 BOLEANO : 'booleano' ;
 NUMERO : 'numero' ;
 ESCRIBIR : 'escribir' ;
 IMPRIMIR : 'imprimir' ;
 NADA : 'nada' ;
 BOOLEAN_CONSTANT : 'verdadero' | 'falso' ;
 MODIFICABLE : 'modificable' ;
 INTEGRAL_CONSTANT : '[0-9]+' ;
 NUMERIC_CONSTANT : '[0-9]+' '.' '[0-9]+' ;
 STRING_CONSTANT : '"' ~('"') * '"' ;
 ID : '[A-Za-z][A-Za-z0-9]*' ;

Cuadрупlos.py

```

from MemoryRegister import *

# operaciones que soporta la maquina virtual
CANVAS_SIZE = "CANVAS_SIZE"
CANVAS_COLOR = "CANVAS_COLOR"
SETSPEED = "SET_SPEED"
READ = "READ"
WRITE = "WRITE"
PRINTLN = "PRINTLN"
PRINT = "PRINT"
PENUP = "PENUP"
PENDOWN = "PENDOWN"
FORWARD = "FORWARD"
BACKWARD = "BACKWARD"
LEFT = "LEFT"
RIGHT = "RIGHT"
COLOR_CHANGE = "COLOR_CHANGE"
GOTO = "GOTO"
  
```

```

END = "END"
RETURN = "RETURN"
ERA = "ERA"
PARAM = "PARAM"
GOSUB = "GOSUB"
RET = "RET"
PLUS = "+"
MINUS = "-"
TIMES = "*"
DIVIDE = "/"
MODULO = "%"
ASSIGN = "="
ASSIGNFUNC = "_=__"
EQUALS = "=="
NOT_EQUALS = "!="
LESS_THAN = "<"
GREATER_THAN = ">"
LESS_THAN_EQUAL = "<="
GREATER_THAN_EQUAL = ">="
AND = "&"
OR = "|"
NOT = "!"
CHECK_BOUNDS = "CHB"
PEN_POSX = "PEN_POSX"
PEN_POSY = "PEN_POSY"
END = "END"

```

```

class Cuadрупlos:

```

```

    def __init__(self):
        self.listaCuadрупlos = []
        self.pilaSaltos = []

```

```

    """Regresa la lista de cuadрупlos para que sea ejecutada por la maquina virtual"""

```

```

    def getCuadрупlos(self):
        return self.listaCuadрупlos

```

```

    """agrega un cuadрупlo a la lista de cuadрупlos"""

```

```

    def addCuadрупlo(self, functionName, operator, op1, op2, t=None, generateT=True):
        if not t and generateT:
            t = TemporalRegister(functionName)
        self.listaCuadрупlos.append([operator, op1, op2, t])
        return t

```

```

    """edita un cuadрупlos que quedo pendiente"""

```

```

    def editCuadрупlo(self, indice, t):
        self.listaCuadрупlos[indice][3]=t

```

```

    """regresa el indice del ultimo cuadрупlo"""

```

```

    def last(self):
        return len(self.listaCuadрупlos)-1

```

```
"""anade el indice de un cuadruplo a la lista de cuadruplos"""
def pushPilaSaltos(self, indice):
    self.pilaSaltos.append(indice)

"""elimina el ultimo elemento de la pila saltos"""
def popPilaSaltos(self):
    return self.pilaSaltos.pop()

"""regresa el indice actual de la lista de cuadruplos"""
def current(self):
    return len(self.listaCuadruplos)
```

MemoryRegister.py

'''clase de registros de memoria. administra la memoria que se genera para los cuádruplos'''
class MemoryRegisters:

```
    def __init__(self):
        self.registers = {}
        self.registers[""] = {}
        self.nextLocalCounter = {}

    '''se agrega una nueva funcion y se establece el contador de dicha funcion a 0'''
    def newFunction(self, nameOfFunction, typeOfFunction):
        self.registers[nameOfFunction] = {}
        self.nextLocalCounter[nameOfFunction] = 0
        return self.createMemoryRegister(nameOfFunction, "", typeOfFunction)

    '''se crea un registro de memoria para una variable en una funcion'''
    def createMemoryRegister(self, nameOfVariable, nameOfFunction,
typeOfVariable=None):
        register = None
        if nameOfFunction == "":
            register = GlobalRegister(typeOfVariable)
        else:
            register = LocalRegister(self.nextLocalCounter[nameOfFunction], typeOfVariable)
            self.nextLocalCounter[nameOfFunction] += 1
            self.registers[nameOfFunction][nameOfVariable] = register
        return register

    '''se crea un registro falso para las variables de referencia'''
    def createRefRegister(self, nameOfVariable, typeOfVariable, nameOfFunction):
        self.registers[nameOfFunction][nameOfVariable] =
RefRegister(self.nextLocalCounter[nameOfFunction], typeOfVariable)
        self.nextLocalCounter[nameOfFunction] += 1
        return self.registers[nameOfFunction][nameOfVariable]

    '''se crea un registro para un arreglo'''
    def createMemoryRegisterForArray(self, nameOfArray, typeOfArray, length,
nameOfFunction):
        registers = []
        if nameOfFunction == "":
            for i in range(0, length):
                registers.append(GlobalRegister(typeOfArray))
        else:
            for i in range(0, length):
                registers.append(LocalRegister(self.nextLocalCounter[nameOfFunction],
typeOfArray))
            self.nextLocalCounter[nameOfFunction] += 1
            self.registers[nameOfFunction][nameOfArray] = registers
        return registers

    '''regresa el registro de memoria asociado a una variable'''
```



```

def getMemoryRegister(self, nameOfVariable, nameOfFunction):
    if nameOfVariable in self.registers[nameOfFunction]:
        return self.registers[nameOfFunction][nameOfVariable]
    else:
        return self.registers[""][nameOfVariable]

    """regresa el registro de memoria asociado a un arreglo"""
def getArrayMemoryRegister(self, nameOfArray, index, nameOfFunction):
    if nameOfArray in self.registers[nameOfFunction]:
        return (self.registers[nameOfFunction][nameOfArray], index)
    else:
        return (self.registers[""][nameOfArray], index)

"""clase para representar un registro falso donde se guardan las variables por referencia"""
class RefRegister:
    def __init__(self, next, tipo=None):
        self.counter = next
        self.type = tipo

    def __repr__(self):
        return 'lr' + str(self.counter)

"""clase para representar un registro temporal donde se guardan los temporales"""
class TemporalRegister:

    nextCounter = {}

    def __init__(self, nameOfFunction):
        if nameOfFunction not in TemporalRegister.nextCounter:
            TemporalRegister.nextCounter[nameOfFunction] = 0
        self.counter = TemporalRegister.nextCounter[nameOfFunction]
        TemporalRegister.nextCounter[nameOfFunction] += 1

    def __repr__(self):
        return 't' + str(self.counter)

"""clase para guardar los registros globales que estan disponibles para todas las funciones"""
class GlobalRegister():
    c = 0

    def __init__(self, tipo = None):
        self.counter = GlobalRegister.c
        self.tipo = tipo
        GlobalRegister.c += 1

    def __repr__(self):
        return 'g' + str(self.counter)

"""clase para guardar los registros locales"""
class LocalRegister():

```

```
def __init__(self, next, tipo = None):  
    self.counter = next  
    self.type = tipo  
  
def __repr__(self):  
    return 'I' + str(self.counter)
```

Namespace.py

```
import re
BOLEANO = "booleano"
TEXTO = "texto"
NUMERO = "numero"
NADA = "nada"

class Variable:
    """ Una variable tiene nombre y tipo"""
    def __init__(self,nameOfVariable,typeOfVariable,ref):
        self.name=nameOfVariable
        self.type=typeOfVariable
        self.reference=ref

"""clase donde se guardan los atributos de un arreglo en especifico"""
class Array:
    def __init__(self, nameOfArray, typeOfArray, lengthOfArray):
        self.name = nameOfArray
        self.type = typeOfArray
        self.length = lengthOfArray

class Parameter:
    """ Una parametro tiene nombre, tipo y referencia"""
    def __init__(self,nameOfVariable,typeOfVariable,ref):
        self.name=nameOfVariable
        self.type=typeOfVariable
        self.reference=ref

class Function:
    """Una funcion tiene nombre, tipo y variables, que pueden ser variables o parametros"""
    def __init__(self,nameOfFunction,typeOfFunction,dirInicio):
        self.name = nameOfFunction
        self.type = typeOfFunction
        self.variables = []
        self.parameters = []
        self.arrays = []
        self.functionVariables = []
        self.direccionInicio = dirInicio
        self.tipos = {NUMERO: 0, BOLEANO: 0, TEXTO: 0}

    """Recibe los atributos de una variable, la crea y agrega a la funcion"""
    def addVariable(self,nameOfVariable,typeOfVariable,ref):
        auxiliar = Variable(nameOfVariable,typeOfVariable,ref)
        self.addVariableObject(auxiliar)

    """Recibe los atributos de un parametro, lo crea y agrega a la funcion"""
    def addParameter(self,nameOfVariable,typeOfVariable,ref):
        auxiliar = Parameter(nameOfVariable,typeOfVariable,ref)
        self.parameters.append(auxiliar)
```

```

"""Agrega directamente una variable ya creada (objeto tipo variable) a la funcion"""
def addVariableObject(self,auxiliar):
    self.variables.append(auxiliar)
    self.addType(auxiliar.type)

def addFunctionVariableObject(self, auxiliar):
    self.functionVariables.append(auxiliar)

"""Agrega directamente un parametro ya creado (objeto tipo parametro) a la funcion"""
def addParameterObject(self,auxiliar):
    self.parameters.append(auxiliar)

def addArrayObject(self, auxiliar):
    self.arrays.append(auxiliar)

"""Busca el nombre de una variable dentro de la funcion"""
def searchVariable(self,variablename):
    for f in self.variables + self.parameters + self.arrays:
        if variablename == f.name:
            return True
    return False

def searchParameter(self, parameterName):
    for f in self.parameters:
        if f.name == parameterName:
            return True
    return False

"""Regresa el tipo del nombre de la variable dada"""
def getType(self,variablename):
    for f in self.variables + self.parameters + self.arrays:
        if variablename == f.name:
            return f.type
    return None

"""regresa la longitud de uno de sus arreglos"""
def getArrayLength(self, arrayName):
    for a in self.arrays:
        if a.name == arrayName:
            return a.length
    return 0

def addType(self,type):
    self.tipos[type]+=1;

```

```

class NamespaceTable:

```

```

    def __init__(self):
        self.tabla = {}
        self.tabla[""] = Function("", NADA, 0)

```

```

'''regresa el directorio de procedimientos'''
def getDirProc(self):
    return self.tabla

'''Metodo que agrega una funcion a la tabla de funciones.
Regresa True si la operacion fue exitosa, False si no.'''
def addFunction(self, nameOfFunction, typeOfFunction, dirInicio):
    if nameOfFunction in self.tabla or self.tabla[""].searchVariable(nameOfFunction):
        return False
    else:
        auxiliar = Function(nameOfFunction,typeOfFunction,dirInicio)
        # Nombre del parametro, Tipo del parametro, Booleano Referencia True Valor False
        self.tabla[nameOfFunction]=auxiliar
        if typeOfFunction != NADA:
            self.tabla[""].addFunctionVariableObject(Variable(nameOfFunction,
typeOfFunction, False))
        return True

'''Metodo que agrega un parametro a la funcion'''
def addParameter(self, nameOfFunction, parameterName, typeOfParameter,
modificable):
    if self.tabla[nameOfFunction].searchParameter(parameterName):
        return False
    else:
        self.tabla[nameOfFunction].addParameter(parameterName, typeOfParameter,
modificable)
        return True

'''Metodo que agrega una variable a la tabla de variables en el ambito de la funcion
dada.
Regresa True si la operacion fue exitosa, False si no.'''
def addVariable(self, nameOfVariable, typeOfVariable, nameOfFunction):
    if self.tabla[nameOfFunction].searchVariable(nameOfVariable) or
self.tabla[""].searchVariable(nameOfVariable):
        return False
    else:
        auxiliar=Variable(nameOfVariable,typeOfVariable,False)
        self.tabla[nameOfFunction].addVariableObject(auxiliar)
        return True

'''Metodo que agrega un arreglo al directorio de variables'''
def addArray(self, nameOfArray, typeOfArray, lengthOfArray, nameOfFunction):
    auxiliar = Array(nameOfArray, typeOfArray, lengthOfArray)
    self.tabla[nameOfFunction].addArrayObject(auxiliar)

'''Metodo que se encarga de actualizar tamaño cuando hay variables temporales'''
def addTemporal(self,nameOfFunction,typeOfVariable):
    if self.functionExists(nameOfFunction):
        self.tabla[nameOfFunction].addType(typeOfVariable)
    else:
        return False;

```

```

    """Metodo que regresa la longitud del arreglo que se usa dentro de cierta funcion"""
    def getArrayLength(self, nameOfArray, nameOfFunction):
        answer = self.tabla[nameOfFunction].getArrayLength(nameOfArray)
        if answer == 0:
            answer = self.tabla[""].getArrayLength(nameOfArray)
        return answer

    """Regresa true si la variable ya fue declarada (si ya existe dentro de la tabla de
    variables en el ambito de la funcion dada)"""
    def idAlreadyTaken(self, nameOfVariable, nameOfFunction):
        return self.tabla[nameOfFunction].searchVariable(nameOfVariable) or
        self.tabla[""].searchVariable(nameOfVariable)

    """Metodo que busca una funcion en la tabla de funciones"""
    def functionExists(self, name):
        return name in self.tabla

    """Metodo que regresa el tipo de una variable que se usa dentro de una funcion. Regresa
    None si la variable no existe"""
    def getVariableType(self, nameOfVariable, nameOfFunction):
        if self.tabla[nameOfFunction].searchVariable(nameOfVariable):
            return self.tabla[nameOfFunction].getType(nameOfVariable)
        elif self.tabla[""].searchVariable(nameOfVariable):
            return self.tabla[""].getType(nameOfVariable)
        else:
            return None

    """regresa el tipo de la funcion, si existe. Si no existe, regresa None"""
    def getFunctionType(self, nameOfFunction):
        if self.functionExists(nameOfFunction):
            return self.tabla[nameOfFunction].type
        else:
            return None

    """regresa la cantidad de parametros que tiene definidos una funcion"""
    def getParameterAmount(self, nameOfFunction):
        return len(self.tabla[nameOfFunction].parameters)

    """regresa si el parametro es por referencia. True si si, False si no"""
    def parameterReference(self, nameOfFunction, k):
        return self.tabla[nameOfFunction].parameters[k].reference

    """Metodo que checa si los argumentos concuerdan"""
    def argumentAgree(self, nameOfFunction, argumentPos, argumentName,
    argumentType):
        pattern = re.compile("[A-Za-z]+$")
        if(argumentPos >= len(self.tabla[nameOfFunction].parameters)):
            return False
        else:

```

```

if self.tabla[nameOfFunction].parameters[argumentPos].type != argumentType:
    return False
else:
    """Por referencia"""
    if self.tabla[nameOfFunction].parameters[argumentPos].reference:
        "Hacer match para regexp por ref"
        if not pattern.match(argumentName):
            return False
        return True

"""regresa el indice del cuadruplo a partir del cual inicia una funcion"""
def getDireccionInicio(self,nameOfFunction):
    return self.tabla[nameOfFunction].direccionInicio

```

Vm.py

```
import turtle
from cuadрупlos import *
from MemoryRegister import *
import sys

# variables a usar
BOOLEAN = 0
NUMBER = 1
STRING = 2
TEMPORAL_REGISTER = 3
GLOBAL_REGISTER = 4
LOCAL_REGISTER = 5
pila = [[],[]]

GLOBAL = 0

GLOBALS = 0
LOCALS = 0
TEMPORALS = 1

'''metodo que traduce el color del espanol al ingles'''
def translateColor(color):
    translatedcolor="white"

    if color == "rojo":
        translatedcolor="red"
    elif color == "azul":
        translatedcolor="blue"
    elif color == "verde":
        translatedcolor="green"
    elif color == "amarillo":
        translatedcolor="yellow"
    elif color == "naranja":
        translatedcolor="orange"
    elif color == "blanco":
        translatedcolor="white"
    elif color == "negro":
        translatedcolor="black"
    elif color == "violeta":
        translatedcolor="violet"
    elif color == "cafe":
        translatedcolor="brown"
    elif color == "gris":
        translatedcolor="gray"

    return translatedcolor

'''guarda en el registro de memoria apuntado por variable el valor de la respuesta, dentro del
ambiente de ejecucion
```



```

(el tope de la pila"
def store(variable, respuesta):
    if isinstance(variable, tuple):
        store(variable[0][Valor(variable[1])], respuesta)
    elif Tipo(variable) == GLOBAL_REGISTER:
        while len(pila[GLOBAL][GLOBALS]) <= variable.counter:
            pila[GLOBAL][GLOBALS].append(None)
            pila[GLOBAL][GLOBALS][variable.counter] = respuesta
        else:
            current = len(pila) - 1
            if Tipo(variable) == LOCAL_REGISTER:
                while len(pila[current][LOCALS]) <= variable.counter:
                    pila[current][LOCALS].append(None)
                registro = pila[current][LOCALS][variable.counter]
                if isinstance(registro, list) and isinstance(registro[0], int):
                    pila[registro[0]][0][registro[1]] = respuesta
                    pila[current][LOCALS][variable.counter][2] = respuesta
                else:
                    pila[current][LOCALS][variable.counter] = respuesta
            elif Tipo(variable) == TEMPORAL_REGISTER:
                while len(pila[current][TEMPORALS]) <= variable.counter:
                    pila[current][TEMPORALS].append(None)
                pila[current][TEMPORALS][variable.counter] = respuesta

```

```

""metodo principal de la maquina virtual""
def executeVM(dirProc, listaCuadрупlos):
    global pila

    functionRegisters = []
    nextContext = None
    jumpStack = []
    showScreen = False

    screen = turtle.Screen()
    tortuga = turtle.Turtle()

    i = 0
    c = listaCuadрупlos[i]
    op = c[0]
    # mientras que el cuadрупlo no sea el FIN
    while op != END:

        valor1 = c[1]
        valor2 = c[2]
        variable = c[3]

        if op == GOTO and not Valor(valor1):
            i = Valor(variable);

        elif op == GOTO:

```

```

i = Valor(variable)

elif op == GOSUB:
    jumpStack.append(i + 1)
    pila.append(nextContext)
    i = Valor(valor1)

elif op == RET:
    i = jumpStack.pop()
    pila.pop()
    functionRegisters.pop()

else:
    if op == ASSIGN:
        store(variable, Valor(valor1))

    if op == ASSIGNFUNC:
        functionRegisters.append(variable)
        store(functionRegisters[len(functionRegisters)-1], Valor(valor1))

    elif op == PLUS:
        store(variable, Valor(valor1) + Valor(valor2))

    elif op == MINUS:
        store(variable, Valor(valor1) - Valor(valor2))

    elif op == TIMES:
        store(variable, Valor(valor1) * Valor(valor2))

    elif op == DIVIDE:
        store(variable, Valor(valor1) / Valor(valor2))

    elif op == MODULO:
        store(variable, Valor(valor1) % Valor(valor2))

    elif op == AND:
        store(variable, Valor(valor1) and Valor(valor2))

    elif op == OR:
        store(variable, Valor(valor1) or Valor(valor2))

    elif op == NOT:
        store(variable, not Valor(valor1))

    elif op == EQUALS:
        store(variable, Valor(valor1) == Valor(valor2))

    elif op == NOT_EQUALS:
        store(variable, Valor(valor1) != Valor(valor2))

    elif op == LESS_THAN:

```

```

        store(variable, Valor(valor1) < Valor(valor2))

elif op == GREATER_THAN:
    store(variable, Valor(valor1) > Valor(valor2))

elif op == LESS_THAN_EQUAL:
    store(variable, Valor(valor1) <= Valor(valor2))

elif op == GREATER_THAN_EQUAL:
    store(variable, Valor(valor1) >= Valor(valor2))

elif op == READ:
    aux = input()
    if variable.tipo == "texto":
        store(variable, aux)
    elif variable.tipo == "booleano":
        if aux == 'verdadero':
            aux = True
        else:
            aux = False
        store(variable, aux)
    else:
        try:
            aux = int(aux)
            store(variable, aux)
        except ValueError:
            try:
                aux = float(s)
                store(variable, aux)
            except:
                print("Error: se esperaba un ", variable.tipo)
                sys.exit(0)

elif op == WRITE:
    tortuga.write(Valor(valor1), True)

elif op == PRINTLN:
    print(Valor(valor1))

elif op == PRINT:
    print(Valor(valor1), end="")

elif op == CHECK_BOUNDS:
    v1 = Valor(valor1)
    v2 = Valor(valor2)
    if v1 >= v2:
        print("Error: intentando acceder al indice", v1, ": el arreglo es de solo", v2,
"elementos.")
        sys.exit(0)

elif op == ERA:

```

```

        nextContext = ([], [])

    elif op == PARAM:
        if not valor2:
            nextContext[0].append(Valor(valor1))
        elif Tipo(valor1) == LOCAL_REGISTER:
            nextContext[0].append([len(pila)-1, valor1.counter, Valor(valor1), valor1])
        else:
            nextContext[0].append([0, valor1.counter, Valor(valor1), valor1])

    elif op == RETURN:
        store(functionRegisters[len(functionRegisters)-1], Valor(valor1))

    else:
        showScreen = True

        if op == CANVAS_SIZE:
            screen.setup(Valor(valor1), Valor(valor2))

        elif op == CANVAS_COLOR:
            translatedcolor = translateColor(Valor(valor1))
            #valor1 es el color en string, pero esta en espanol, cambiar a ingles
            screen.bgcolor(translatedcolor)

        elif op == FORWARD:
            tortuga.forward(Valor(valor1))

        elif op == BACKWARD:
            tortuga.backward(Valor(valor1))

        elif op == LEFT:
            tortuga.left(Valor(valor1))

        elif op == RIGHT:
            tortuga.right(Valor(valor1))

        elif op == PENUP:
            tortuga.penup()

        elif op == PENDOWN:
            tortuga.pendown()

        elif op == COLOR_CHANGE:
            translatedcolor = translateColor(Valor(valor1))
            tortuga.pencolor(translatedcolor)

        elif op == SETSPEED:
            velocidad = Valor(valor1)
            if 0 > velocidad or velocidad > 10:
                print("Error: intentando poner velocidad de", velocidad, ": la velocidad solo puede
ser entre 0 y 10")

```

```

        tortuga.speed(velocidad)

        elif op == PEN_POSX:
            tortuga.setx(Valor(valor1))

        elif op == PEN_POSY:
            tortuga.sety(Valor(valor1))

    i += 1
    # aumentar ir al siguiente cuadruplo

    c = listaCuadрупlos[i]
    op = c[0]

    if showScreen:
        screen.mainloop()

'''metodo que regresa el tipo del valor que esta en el cuadruplo, dependiendo de la instancia
que es'''
def Tipo(valor1):
    if isinstance(valor1, bool):
        return BOOLEAN
    elif isinstance(valor1, float):
        return NUMBER
    elif isinstance(valor1, int):
        return NUMBER
    elif isinstance(valor1, str):
        return STRING
    elif isinstance(valor1, TemporalRegister):
        return TEMPORAL_REGISTER
    elif isinstance(valor1, GlobalRegister):
        return GLOBAL_REGISTER
    elif isinstance(valor1, LocalRegister):
        return LOCAL_REGISTER
    elif isinstance(valor1, list) and isinstance(valor1[0], int):
        return Tipo(valor1[3])
    else:
        return Tipo(valor1[0][Valor(valor1[1])])

'''metodo que lee la pila de ejecucion y regresa el valor del registro'''
def Valor(valor1):
    global pila
    if isinstance(valor1, tuple):
        return Valor(valor1[0][Valor(valor1[1])])
    else:
        aux = Tipo(valor1)
        if aux == GLOBAL_REGISTER:
            return pila[GLOBAL][GLOBALS][valor1.counter]
        else:
            current = len(pila) - 1
            if aux == LOCAL_REGISTER:

```

```
    registro = pila[current][LOCALS][valor1.counter]
    if isinstance(registro, list) and isinstance(registro[0], int):
        return registro[2]
    else:
        return registro
elif aux == TEMPORAL_REGISTER:
    return pila[current][TEMPORALS][valor1.counter]
else:
    return valor1
```

Manual de Usuario

A continuación se presenta un manual que detalla las diferentes características del lenguaje Lienzo, acompañado de ejemplos donde se muestra cómo se emplea.

Sobre la instalación de herramientas requeridas

Lienzo utiliza las siguientes herramientas adicionales: Python y Antlr. A continuación se explica cómo instalar ambas

Python

Lienzo está basado en el lenguaje de programación Python. Para instalar Python se debe acudir a la siguiente liga: <https://www.python.org/downloads/>, seleccionar Python 3.5.1 e instalarlo en el equipo de cómputo a utilizar..

Antlr

El compilador de Lienzo está basado en la herramienta Antlr. Para instalar Antlr se debe acudir a la siguiente liga: <http://wwwantlr.org/download/antlr-4.5.2-complete.jar> e instalar en el directorio C: del equipo de cómputo a utilizar.

Sobre la compilación

Para compilar un programa hecho en Lienzo se deben ejecutar las siguientes instrucciones:

```
java -jar C:\antlr-4.5.2-complete.jar -Dlanguage=Python3 Lienzo.g4
python script.py NombredelArchivo
```

Mi primer programa en Lienzo

A continuación se muestra cómo se escribiría el famosísimo “Hello World” en el lenguaje Lienzo.

Imprimir “Hello World!”

Convenciones de léxico

Comentarios

Este lenguaje cuenta con comentarios que se pueden emplear utilizando ‘//’ y ‘/* */’. Los siguientes son ejemplos de cómo se utilizan:

```
//Este es un comentario  
/*Este tambien  
es un comentario*/
```

Identificadores

Variables numéricas

Las variables numéricas llevan la palabra “numero” y pueden consistir de números enteros o decimales. Ejemplo: número n = 4;

Variables booleanas

Las variables booleanas llevan la palabra “booleano” y pueden llevar consigo el valor de falso o verdadero. Ejemplo: booleano e = verdadero;

Variables de texto

Las variables de texto llevan la palabra “texto” y pueden llevar consigo cualquier valor alfanumérico. Ejemplo: texto bienvenida = “Bienvenidos todos”;

Palabras reservadas

Las siguientes palabras no se pueden usar como nombres de variables o funciones porque ya están siendo utilizadas por el lenguaje Lienzo:

'rojo' , 'verde' , 'amarillo' , 'azul' , 'blanco' , 'negro' , 'morado' , 'naranja' , 'cafe' , 'gris' , 'color' ,
'lienzo' , 'tamano' , 'por' , 'de' , 'en' , 'mover' , 'adelante' , 'atras' , 'girar' , 'derecha' , 'izquierda' ,
'levantar' , 'bajar' , 'pluma' , 'dibujo' , 'dormir' , 'mientras' , 'regresar' , 'que' , 'si' , 'sino' , 'texto' ,
'leer' , 'booleano' , 'numero' , 'escribir' , 'imprimir' , 'nada' , 'verdadero' , 'falso' , 'modificable'

Diseño de un programa

Los programas escritos en lenguaje lienzo tienen el siguiente orden:

Declaración de variables
Declaración de funciones
Programa principal

Las variables globales se declaran al inicio, y no es hasta después de ser declaradas que se puede hacer uso de ellas. Dentro de las funciones se sigue la misma lógica, primero se declaran las variables a utilizar y posteriormente ya se puede hacer uso de ellas. El programa principal le procede a las funciones, y este no está encerrado en ninguna función, son líneas de código sueltas.

Expresiones

A continuación se muestran algunos ejemplos de expresiones escritas en lenguaje Lienzo:

Aritméticas

```
variable = 1 + 4;
```

Booleanas

```
variable = falso;
```

Condiciones

Estas expresiones se pueden combinar para formar parte de condiciones. Aquí se muestran algunas de ellas:

Condicional si

```
si (variable - 4 > 0){  
    variable=variable+1;  
}
```

Condicional si no

```
si (variable - 4 > 0){  
    variable=variable+1;  
} sino{  
    variable=7;  
}
```

Ciclos

Las condiciones se pueden utilizar para formar ciclos que se ejecuten una y otra vez dependiendo de la condición base que contengan.

Mientras que

```
mientras que(b>5){  
    b=b-1;  
    imprimir b;  
}
```

Funciones

Las funciones en Lienzo también tienen identificadores únicos, el siguiente es un ejemplo de una función llamada “hola” que regresa un valor de tipo numérico

```
numero hola() {  
    regresar 3;  
}
```

Ejemplo de programa sin output gráfico

Este programa genera un arreglo de 10 números y posteriormente los imprime.

```
numero[10] arreglo;
```

```
nada imprimirArreglo() {  
    numero i = 0;  
    mientras que (i < 10) {  
        imprimir sin salto arreglo[i];  
        imprimir sin salto " ";  
        i = i + 1;  
    }  
    imprimir "";  
}
```

```
nada llenarArreglo() {  
    numero i = 0;  
    mientras que (i < 10) {  
        arreglo[i] = 10 - i;  
        i = i + 1;  
    }  
}
```

```
llenarArreglo();  
imprimirArreglo();
```

Ejemplo de programa con output gráfico

Este programa imprime varias figuras en forma de estrellas dentro de un lienzo rojo.

```
numero i = 0;
```

```
color de lienzo = rojo;
```

```
color de pluma = blanco;
```

```
mientras que (i < 20) {  
    mover adelante i * 30;  
    girar derecha 144;  
    i = i + 1;  
}
```