Instituto Tecnológico de Estudios Superiores de Monterrey

Diseño de Compiladores

“Doodlr”

Fecha de Entrega:

Miércoles, 21 de Noviembre de 2018

Integrantes del Equipo:

Gala Stefanía Ramos Islas A00817135

André Alain Jiménez Díaz A01032006

Firmas:

Índice

DESCRIPCIÓN DEL PROYECTO

Propósito

El propósito de este proyecto es el poder crear un lenguaje que permita a los usuarios a las personas interactuar con una interfaz gráfica. Además, que el objetivo es enseñar a los usuarios algunas instrucciones básicas de la programación, además de practicar de una forma más dinámica teniendo una visualización más amigable de los resultados.

Objetivos

Con la ayuda del lenguaje de programación Python se nos ha facilitado el uso de y desarrollo de gráficos, ayudándonos con la ayuda de librería de Turtle implementada. El usuario sólo deberá escribir estatutos básicos para crear figuras geométricas. El código permitirá que el usuario pueda crear código un poco más complejo dependiendo del aprendizaje que lleva el usuario. El programa mostrará figuras diferentes según el código que ingresen.

Alcance del Proyecto

El lenguaje creará un ambiente fácil y amigable en el cual los usuarios, fortalecerán sus conocimientos y habilidades de programación. El proyecto leerá un archivo de texto con todos los comandos del usuario y dará la representación gráfica.

Análisis de Requerimientos

R1. El lenguaje debe leer archivos de texto para extraer el código.

R2. El lenguaje debe aceptar tipos de valores: ‘Int’, ‘Float’ y ‘Booleano’.

R3. El lenguaje debe aceptar operaciones aritméticas: ‘+’, ‘-’, ‘*’, ‘/’, ‘pow^'

R4. El lenguaje debe aceptar operaciones lógicas.

R5. El lenguaje debe aceptar operaciones relacionadas.

R6. El lenguaje debe aceptar condiciones de flujo: ‘if’, ‘else’.

R7. El lenguaje debe aceptar ciclos tipo ‘while’.

R8. El lenguaje debe aceptar tipo de funciones ‘Int’, ‘Float’ y ‘Void’.

R9. El lenguaje debe aceptar la llamada de funciones iterativas y recursivas.

R10. El lenguaje debe aceptar llamadas predeterminadas de la librería de turtle.

R11. El lenguaje debe aceptar la declaración de arreglos de una dimensión.

R12. El lenguaje debe mostrar un output gráfico de los comandos del usuario.

Casos de Uso Generales

ID	UC1
Requerimientos cubiertos	R1, R12
Descripción	El usuario crea un archivo de texto con el código y lo compila con el lenguaje Doodlr.
Precondiciones	El código debe ser escrito según la estructura y reglas del lenguaje Doodlr.
Post condiciones	El compilador muestra los cuádruplos de código del usuario y muestra un output gráfico de las instrucciones.
Pasos de creación del caso de uso	> py Doodlr.py fibonacci.txt

Principales Test Cases

Los casos que el lenguaje Doodlr cubren operaciones aritméticas básicas, operaciones lógicas y relaciones; los casos de uso muestran un output gráfico del código ejecutado.
Se realizaron casos de uso para:

- Estatus básicos lineales.
- Factorial de “n” con iteración o recursividad.
- Fibonacci de “n” con iteración o recursividad.
- Orden de un arreglo.
- Búsqueda de un elemento en un arreglo.

Proceso General de Desarrollo

En el proceso de este proyecto ambos integrantes del equipo trabajamos por separado el las primeras 5 entregas del proyecto, posteriormente nos juntos como equipo e hicimos una comparación de estos avances, tomamos las mejores opciones, integramos componentes y continuamos con el desarrollo desde los cuádruplos.
En general este fue un proceso general del desarrollo de ambos.

- Primeramente, se crearon diagramas de sintaxis del lenguaje, junto con el lexer y parser, se comprobó la gramática y se creó una función que pudiera tener inputs de textos para compilar los archivos y probarlos.
- Se crearon listas para guardar las variables y tipos, se creó un archivo donde se crea un directorio de funciones que va almacenando las funciones y la tabla de variables para cada función. Se probaron que las listas tuvieran todas las variables de los archivos de prueba.
- Se creó un cubo semántico para comparar los tipos de datos en las operaciones aritméticas, lógicas y relaciones; el archivo recibe dos variables de tipo de datos y una operación.

- Se crearon pilas que guardan en orden de prioridad las operaciones y variables para la creación de cuádruplos.
- Se creó un archivo que crea los cuádruplos. Se continuó con la realización de los cuádruplos básicos de operaciones aritméticas y posteriormente los cuádruplos de condiciones, estos cuádruplos luego se guardan en una lista que contiene todos los cuádruplos.
- Se realizaron pruebas de los cuádruplos simples, ingresando casos de prueba para verificar que los cuádruplos eran correctos.
- Se comenzaron a crear los cuádruplos de funciones y se integraron con los demás cuádruplos.
- Se crearon funciones que asignaban una dirección a las variables y constantes. Además, se empezó a realizar un archivo donde se estable la memoria virtual.
- Se realizaron más pruebas y se comenzó a crear parte de la máquina virtual y conectarla con la memoria virtual.
- Se integró las funciones para output gráfico en maquina virtual y se realizaron pruebas fáciles, posteriormente se comenzaron a realizar pruebas requerida de factorial y Fibonacci.

Reflexiones Personales

Gala:
Este proyecto nos dio muchos retos pues tuvimos que crear desde cero un compilador, en lo personal me gusto saber todo lo que se necesita para lograr crear un lenguaje en un nivel muy básico. Creo que de lo mas difícil que maneje fue los cuádruplos ya que fue muy tardado conseguir la lógica para saber que valores tenia que conseguir, donde tenían que ir exactamente los puntos neurálgicos, etc. Fue un trabajo de mucho esfuerzo, pero definitivamente creo que sí aprendí mucho el manejo de cuádruplos, memoria y máquina virtual.

André:
Este proyecto ha resultado muy difícil y demandante en el poner nuestros conocimientos adquiridos durante el semestre para crear un compilador. La mayor dificultad que se me presentó en lo personal es el manejo y realización de la máquina virtual, al tener que tener comunicación con la memoria virtual y más cuando debíamos validar funciones que debían ser anidadas o recursivas. Fue un proyecto de mucho esfuerzo y muchas adversidades, pero considero que fue un proyecto con un buen resultado.

DESCRIPCIÓN DEL LENGUAJE

Nombre del lenguaje

Doodlr

Principales características del lenguaje

Doodlr es un lenguaje de output gráfico; se escriben bloques de código en un archivo de texto y el lenguaje lo transforma a figuras. El lenguaje maneja órdenes de códigos a través de funciones y llamadas a funciones, la estructura del lenguaje es la siguiente:

<<Program>> Variables Globales, Funciones y Main

El código debe ser escrito en ese orden estrictamente, pero las variables globales y funciones pueden ser de valor nulo.

Variables Globales = El usuario puede definir n variables como deseé, estas variables serán definidas como globales para usarse en todo el código.

Funciones = El usuario puede definir n funciones de tipo Int, Float, Bool o Void para utilizar en el código.

Main = El usuario define el comportamiento que desea para el programa donde puede llamar a funciones y variables globales.

Errores que podrían ocurrir en compilación o ejecución

1. La sintaxis del código y su estructura está incorrecta.
2. Definir los nombres de variables o funciones con palabras reservadas.
3. Definir un valor a una variable de tipo incompatibles.
4. Declarar una variable o función que ya existe.
5. Llamada de una función o variable no declarada.
6. Falta de declaración de parámetros o declarar más de los esperados.
7. Regresar un valor de tipo diferente al que se espera en una función.
8. Uso de expresiones no booleanas en estatutos de condición.
9. Definir un arreglo con diferentes tipos de valores.
10. No compilar y ejecutar el archivo de código correctamente.

DESCRIPCIÓN DEL COMPILADOR

Equipo de computo

MacBookPro - macOsHigh Sierra v. 10.13.6
Dell - windows 10

Lenguaje usado

Python 3.7

Librerías implementadas

ply 3.11 - https://pypi.org/project/ply/

re - https://docs.python.org/3/library/re.html

codecs - https://docs.python.org/2/library/codecs.html

os - https://docs.python.org/3/library/os.html

sys - https://docs.python.org/2/library/sys.html

turtle - https://docs.python.org/2/library/turtle.html

Análisis Léxico

TOKEN	EXPRESION REGULAR
ADD	+
SUBS	-
MULT	*
DIV	/
POW	^
MOD	%
EQUALTO	==
EQUALS	=
DIFF	!=
LESST	<
LESSTEQ	<=
GREATT	>
GREATTEQ	>=
PAROP	(
PARCLO	)
BRACKOP	{
BRACKCLO	}
CBRACKOP	[
CBRACKCLO	]
INT	r'[0-9]+'
FLOAT	r'[0-9]+\.[0-9]+'
ID	r'[a-zA-Z][a-zA-Z0-9]*'
COMA	,
PUNTOCOMA	;

PALABRAS RESERVADAS	
FUNCTION	Marca el inicio de una función
GLOBAL	Marca la declaración de variables globales
RETURN	Regresa valor
MAIN	Marca el inicio del main del programa
IF	Condicional
ELSE	Default condicional
WHILE	Ciclo while
TRUE	Verdadero
FALSE	Falso
INT	Número entero o función de tipo int
FLOAT	Número flotante o función de tipo float
BOOL	Operador lógico booleano que regresa TRUE o FALSE
VOID	Función de tipo void
AND	Intersección
OR	Unión
WRITE	Escribe el valor de la variable
READ	Lee el valor de la variable

Rule 0 S' -> programa
Rule 1 programa -> varGlobales declaraFunciones PR_main TO_BRACKOP mainBloque TO_BRACKCLO
Rule 2 varGlobales -> PR_global defVariables varGlobales
Rule 3 varGlobales -> empty
Rule 4 declaraFunciones -> PR_function defFuncion declaraFunciones
Rule 5 declaraFunciones -> empty
Rule 6 defFuncion -> decTipo ID agregaFuncion TO_PAROP decParametros TO_PARCLO TO_BRACKOP mainBloque TO_BRACKCLO
Rule 7 decParametros -> decTipo ID
Rule 8 decParametros -> decTipo ID TO_COMA decParametros
Rule 9 mainBloque -> funcCiclos mainBloque
Rule 10 mainBloque -> funcIf mainBloque
Rule 11 mainBloque -> defVariables mainBloque
Rule 12 mainBloque -> llamadaDeFunciones mainBloque
Rule 13 mainBloque -> funcIgual mainBloque
Rule 14 mainBloque -> funcWrite mainBloque
Rule 15 mainBloque -> funcRead mainBloque
Rule 16 mainBloque -> funcReturn mainBloque
Rule 17 mainBloque -> empty
Rule 18 funcWrite -> PR_write TO_PAROP ID TO_PARCLO TO_PuntoComa
Rule 19 funcRead -> PR_read TO_PAROP ID TO_PARCLO TO_PuntoComa
Rule 20 funcReturn -> PR_return TO_PAROP ID TO_PARCLO TO_PuntoComa
Rule 21 funcIgual -> ID OP_EQUALS defExpresiones TO_PuntoComa
Rule 22 funcIgual -> ID TO_CBRACKOP defExpresiones TO_CBRACKCLO OP_EQUALS defExpresiones TO_PuntoComa
Rule 23 funcCiclos -> PR_While TO_PAROP defExpresiones TO_PARCLO TO_BRACKOP mainBloque TO_BRACKCLO
Rule 24 funcIf -> PR_if TO_PAROP defExpresiones TO_PARCLO TO_BRACKOP mainBloque TO_BRACKCLO funcElse
Rule 25 funcElse -> PR_else TO_BRACKOP mainBloque TO_BRACKCLO
Rule 26 funcElse -> empty
Rule 27 decTipo -> PR_int
Rule 28 decTipo -> PR_float
Rule 29 decTipo -> PR_bool
Rule 30 decTipo -> PR_void
Rule 31 defVariables -> PR_int defVar1 TO_PuntoComa
Rule 32 defVariables -> PR_float defVar1 TO_PuntoComa
Rule 33 defVariables -> PR_bool defVar1 TO_PuntoComa
Rule 34 defVariables -> PR_void defVar1 TO_PuntoComa
Rule 35 defVar1 -> variable defVar2
Rule 36 defVar1 -> arreglo defVar2
Rule 37 defVar2 -> TO_COMA defVar1
Rule 38 defVar2 -> empty
Rule 39 variable -> ID

Rule 40 arreglo -> ID TO_CBRACKOP TO_INT TO_CBRACKCLO
Rule 41 defExpresiones -> decExpresion
Rule 42 defExpresiones -> decExpresion PR_and defExpresiones
Rule 43 defExpresiones -> decExpresion PR_or defExpresiones
Rule 44 decExpresion -> miniExp
Rule 45 decExpresion -> miniExp OP_EQUALTO miniExp
Rule 46 decExpresion -> miniExp OP_DIFF miniExp
Rule 47 decExpresion -> miniExp OP_LESST miniExp
Rule 48 decExpresion -> miniExp OP_LESSTEQ miniExp
Rule 49 decExpresion -> miniExp OP_GREATT miniExp
Rule 50 decExpresion -> miniExp OP_GREATTEQ miniExp
Rule 51 miniExp -> microExp
Rule 52 miniExp -> microExp OP_SUBS miniExp
Rule 53 miniExp -> microExp OP_ADD miniExp
Rule 54 microExp -> micromicroExp
Rule 55 microExp -> micromicroExp OP_MULT microExp
Rule 56 microExp -> micromicroExp OP_DIV microExp
Rule 57 microExp -> micromicroExp OP_MOD microExp
Rule 58 micromicroExp -> decSolucion
Rule 59 micromicroExp -> decSolucion OP_POW micromicroExp
Rule 60 decSolucion -> ID
Rule 61 decSolucion -> ID TO_CBRACKOP defExpresiones TO_CBRACKCLO
Rule 62 decSolucion -> TO_INT
Rule 63 decSolucion -> TO_FLOAT
Rule 64 decSolucion -> PR_true
Rule 65 decSolucion -> PR_false
Rule 66 decSolucion -> llamadaDeFunciones
Rule 67 decSolucion -> TO_PAROP defExpresiones TO_PARCLO
Rule 68 llamadaDeFunciones -> ID TO_PAROP decParamFuncs TO_PARCLO TO_PuntoComa
Rule 69 llamadaDeFunciones -> funcionesDibuja TO_PuntoComa
Rule 70 funcionesDibuja -> PR_circulo TO_PAROP ID TO_COMA ID TO_COMA ID
TO_PARCLO
Rule 71 funcionesDibuja -> PR_rectangulo TO_PAROP ID TO_COMA ID TO_COMA ID
TO_PARCLO
Rule 72 funcionesDibuja -> PR_espiral TO_PAROP ID TO_COMA ID TO_COMA ID
TO_PARCLO
Rule 73 funcionesDibuja -> PR_estrella TO_PAROP ID TO_COMA ID TO_COMA ID
TO_PARCLO
Rule 74 decParamFuncs -> ID
Rule 75 decParamFuncs -> ID TO_COMA decParamFuncs
Rule 76 decParamFuncs -> empty
Rule 77 empty -> <empty>

Código de operación:

Los códigos de operación se declaran a través de los cuádruplos. Los cuádruplos tienen la siguiente estructura:

<ind, estatuto, var1, var2, var3>

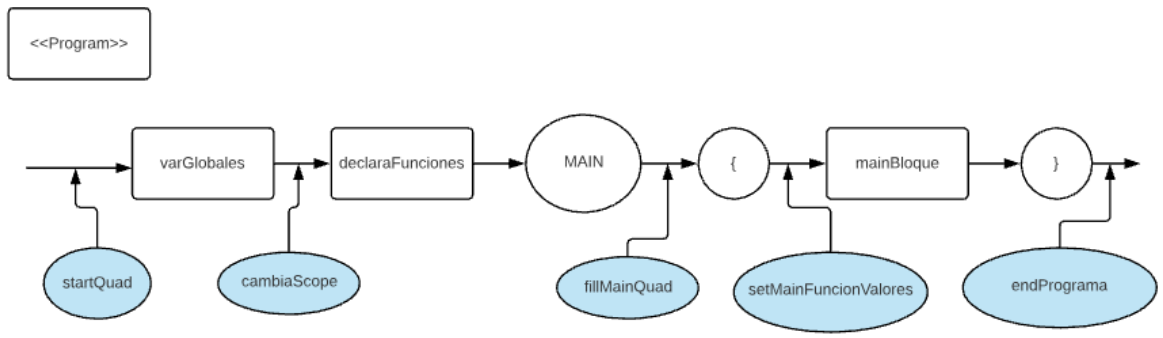
ind = Se refiere al índice en el cual se ubica el cuádruplo.
estatuto = Se refiere a la operación ya sea aritmética, lógica, condicional que se desea hacer.
var1 = Es el operando izquierdo de la operación.
var2 = Es el operando derecho de la operación.
var3 = Es la dirección de memoria donde se guardará el resultado de la operación.

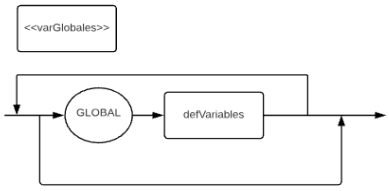
Las operaciones de los cuádruplos son:

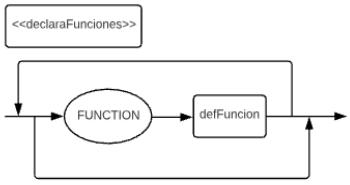
- Goto
- GotoFalse
- Era
- Gosub
- Param
- Return
- Endproc
- Write
- Read
- And
- Or
- Círculo
- Rectángulo
- Espiral
- Estrella
- SuperEstrella
- +
- -
- *
- /
- ^
- ==
- =
- !=
- >
- <
- <=
- >=

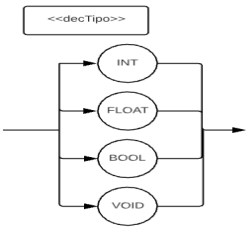
Diagramas de sintaxis

PN: 1.StarQuad, 2.cambiaScope, 3.fillMainQuad, 4.SetMainFuncionValores, 5. endProgram



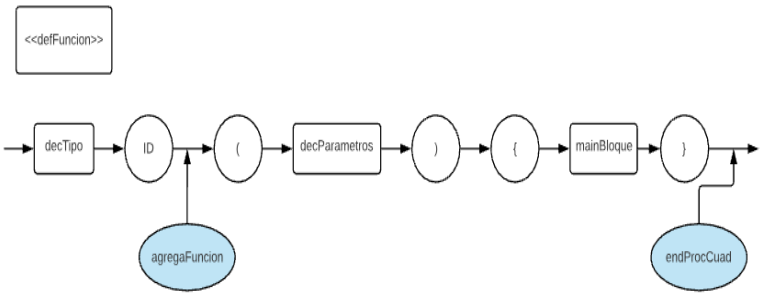


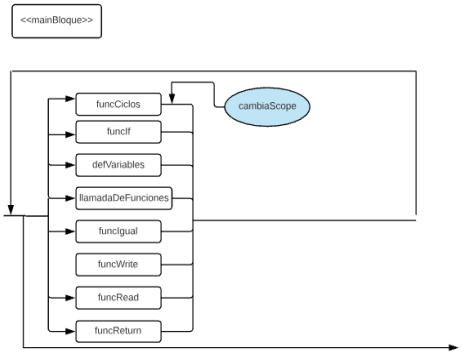




PN: 6. agregaFuncion, 7. endProc

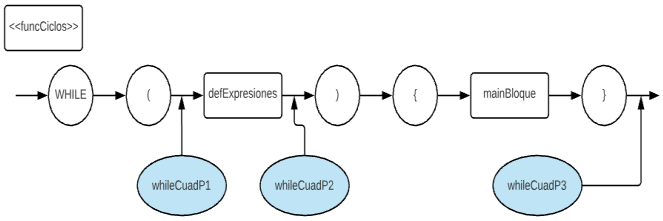
PN: 8.cambiaScope

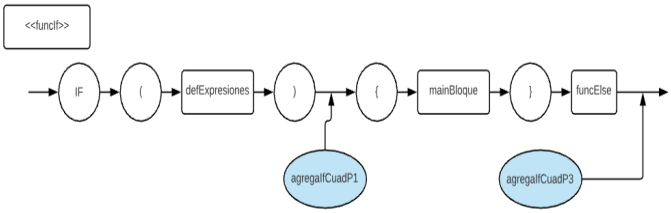




PN: 9.whileCuadP1, 10.whileCuadP2, 11.whileCuadP3

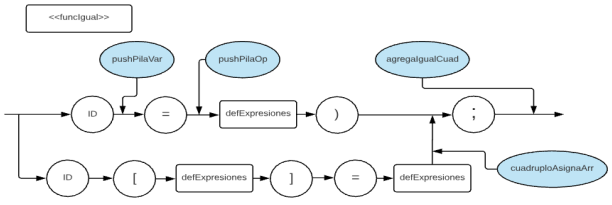
PN: 12.agregaIfCuadP1,

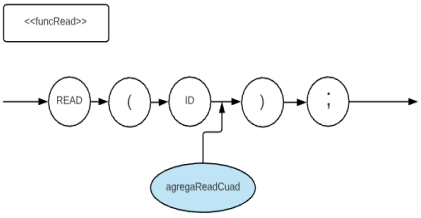




PN: 14.pushPilaVar, 15.PushPilaOp 16.agregaIgualCuad 17.cuadruploAsignaArr

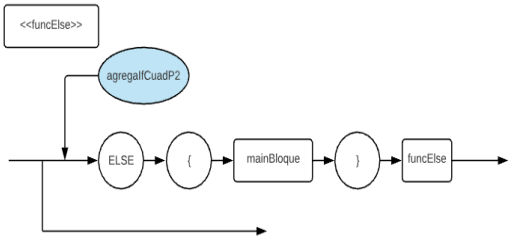
PN: 18.agregaReadCuad

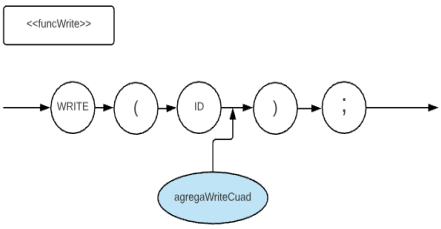




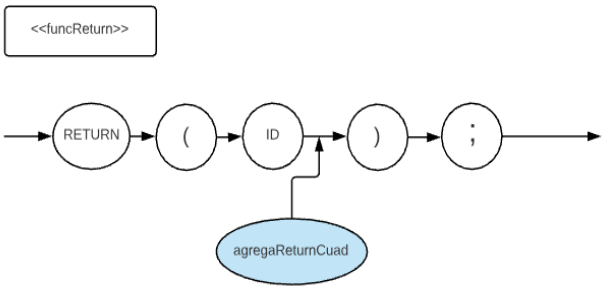
PN: 19. agregaIfCuaP2

PN: 20.agregaWriteCuad

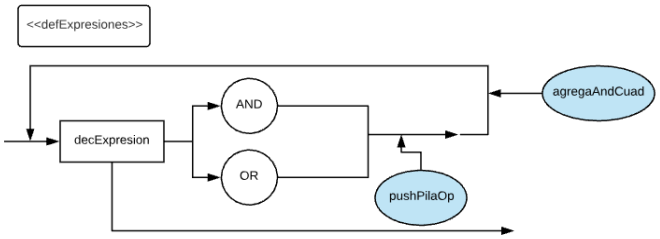




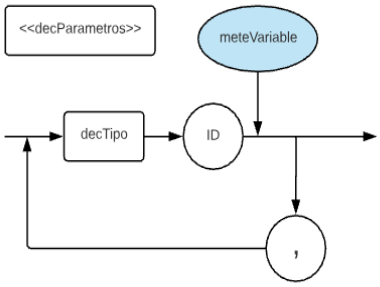
PN: 21. agregaReturnCuad



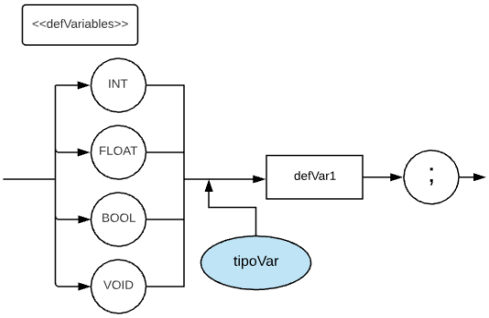
PN: 22.pushPilaOp, 23.agregaAndCuad



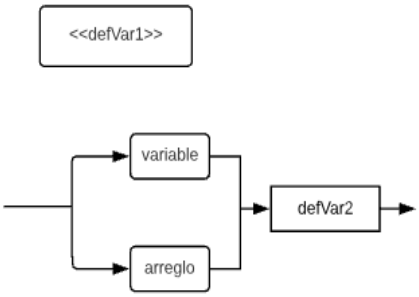
PN: 24.meteVariable



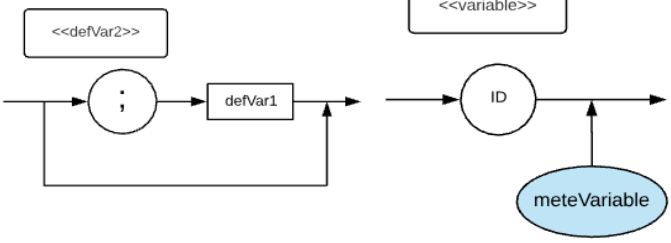
PN: 25.tipoVar

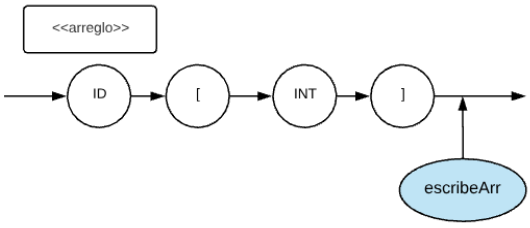


PN: 26. meteVariable

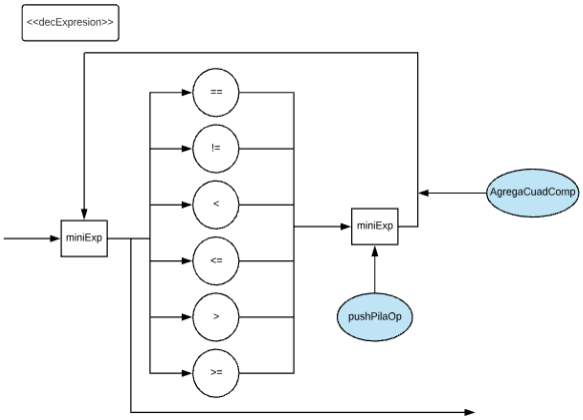


PN: 27.escribeArr

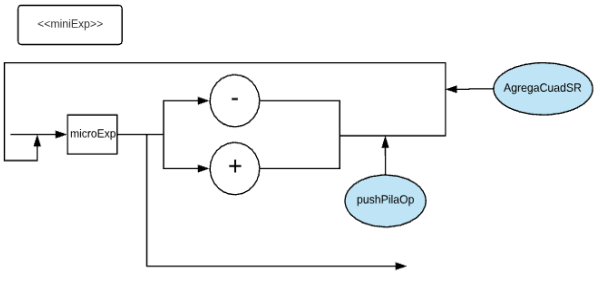




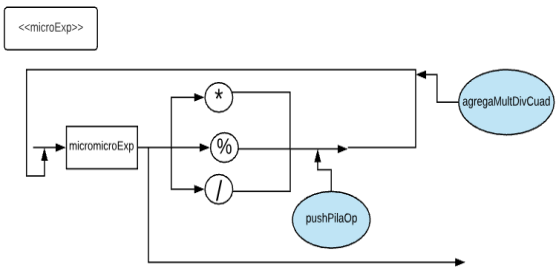
PN: 28. pushPilaOP, 29. agregaComparCuad



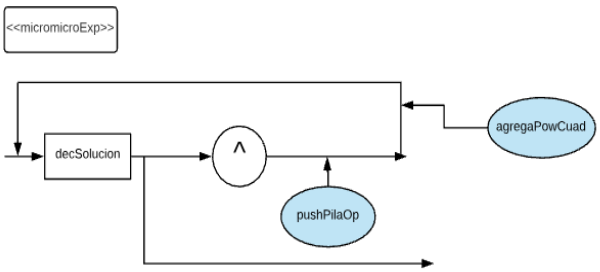
PN: 30.pushPilaOP, 31.



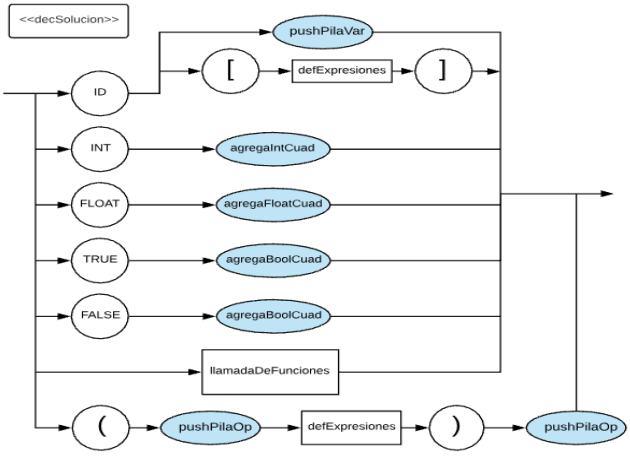
PN: 32.pushPilaOP, 33.agregaMultDivCuad



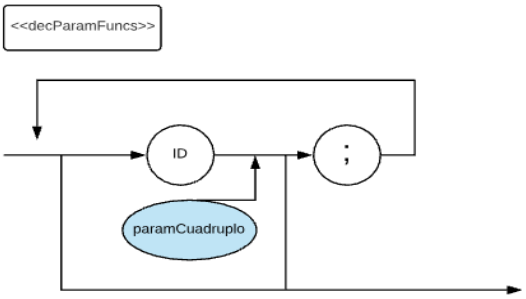
PN: 34.pushPilaOp, 35.agregaPowCuad



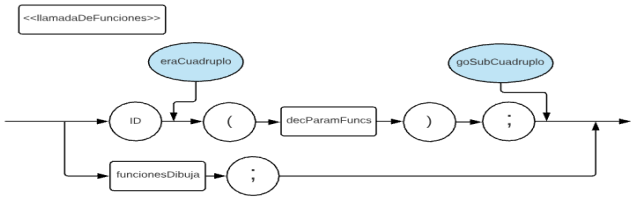
PN: 36. pushPilaVar, 37.agregaIntCuad 38.agregaFloatCuad 39.agregaBoolCuad, 40.agregaBoolCuad, 41.pushPilaOP, 42.pushPilaOp



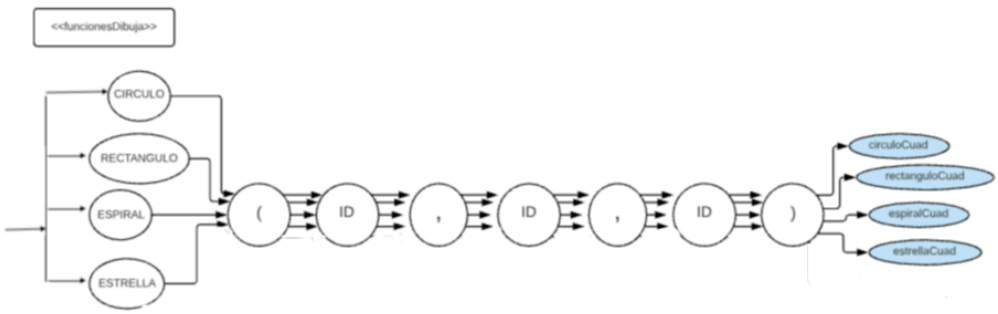
PN: 43.paramCuadruplo



PN: 44.eraCuadruplo, 45.goSubCuadruplo



PN: 46.circuloCuad, 47.rectanguloCuad, 48.espiralCuad, 49.estrellaCuad



Descripción de puntos neurálgicos

1. Crea un espacio de Goto para después ser definido por el Main.
2. Cambia el scope a local para definir el contenido de las funciones como local.
3. Llena la dirección pendiente del Goto.
4. Reinicializa las direcciones de las variables en 0.
5. Indica en el final programa con END.
6. Agrega la nueva función al directorio de funciones, reinicia las direcciones de variables a 0 y crea su tabla de variables.
7. Indica el fin de la función con EndProc.
8. Cambia el scope a local.
9. Guarda el índice donde inicia el while.
10. Crea un GotoFalse con la próxima dirección de una variable en ejecución.
11. Crea un Goto al while y llena la dirección con el índice que se guardo en la pila de saltos.
12. Crea un GotoFalse con la próxima dirección de una variable en ejecución.
13. Guarda la dirección que apunta al final de la condicional del if.
14. Guarda la variable en la pila de variables.
15. Guarda el operador (=) en la pila de operadores.
16. Pop() de variables y operador, saca sus direcciones y genera cuádruplo de igual y lo adjunta a los cuádruplos.
17. Pop() de variables y operador, saca sus direcciones y genera cuádruplo de igualdad de arreglos y lo adjunta a los cuádruplos.
18. Obtiene dirección de ID y genera cuádruplo de read.
19. Crea un Goto con el valor de la pila de saltos de la dirección donde acaba el if.
20. Obtiene dirección de ID y genera cuádruplo de write.
21. Obtiene dirección de ID y genera cuádruplo de return.
22. Guarda el operador (AND ó OR) en la pila de operadores.
23. Pop() de variables y operador, genera cuádruplo de And y lo adjunta a los cuádruplos.
24. Busca el ID en variables locales y globales, si no ha sido definida le asigna dirección de memoria y la guarda.
25. Guarda el valor del tipo de la variable en una pila.

26. Busca el ID en variables locales y globales, si no ha sido definida le asigna dirección de memoria y la guarda.
27. Busca el ID del arreglo en variables locales y globales, si no ha sido definida le asigna dirección de memoria y la guarda como arreglo.
28. Guarda el operador (<, >, >=, =<) en la pila de operadores.
29. Pop() de variables y operador, saca sus direcciones y genera cuádruplo de comparación y lo adjunta a los cuádruplos.
30. Guarda el operador (+ o -) en la pila de operadores.
31. Pop() de variables y operador, saca sus direcciones y genera cuádruplo de suma ó resta, depende del operador, y lo adjunta a los cuádruplos.
32. Guarda el operador (* o /) en la pila de operadores.
33. Pop() de variables y operador, saca sus direcciones y genera cuádruplo de multiplicación o división, depende del operador, y lo adjunta a los cuádruplos.
34. Guarda el operador (^) en la pila de operadores.
35. Pop() de variables y operador, saca sus direcciones, genera cuádruplo de pow y lo adjunta a los cuádruplos.
36. Guarda la variable en la pila de variables.
37. Guarda la variable en la tabla de variables de su función con una dirección de tipo Int.
38. Guarda la variable en la tabla de variables de su función con una dirección de tipo Float.
39. Guarda la variable en la tabla de funciones con una dirección de tipo Bool (true).
40. Guarda la variable en la tabla de funciones con una dirección de tipo Bool (false).
41. Guarda el operador (‘(‘) en la pila de operadores.
42. Guarda el operador (‘)‘) en la pila de operadores.
43. Obtiene dirección del ID, crea cuádruplo Param y lo manda.
44. Crea un cuádruplo de ERA con el ID de la función a llamar.
45. Crea un cuádruplo de GoSub con el ID de la función a llamar.
46. Obtiene la dirección de los valores ingresados y los manda en un cuádruplo de Circulo.
47. Obtiene la dirección de los valores ingresados y los manda en un cuádruplo de Rectángulo.
48. Obtiene la dirección de los valores ingresados y los manda en un cuádruplo de Espiral.
49. Obtiene la dirección de los valores ingresados y los manda en un cuádruplo de Estrella.

Tabla de consideraciones semánticas:

Para el caso de la semántica, realizamos un diccionario para el cubo con todas las combinaciones válidas posibles y sus resultados. Además de esto, implementamos una función que verifica que los 3 parámetros que son recibidos en el cubo existan dentro de él, sino podrá regresar un error por la gramática. A continuación, se muestra el cubo semántico en la forma de Operador izquierdo, Operando, Operador derecho y resultado final, sólo en los casos.

Operador Izq	Operando	Operador Der	Resultado
INT	+	INT	INT
INT	-	INT	INT
INT	*	INT	INT
INT	/	INT	INT
INT	>	INT	BOOL
INT	>=	INT	BOOL
INT	<	INT	BOOL
INT	<=	INT	BOOL
INT	=	INT	INT
INT	==	INT	BOOL
INT	!=	INT	BOOL
INT	^	INT	INT
INT	+	FLOAT	FLOAT
INT	-	FLOAT	FLOAT
INT	*	FLOAT	FLOAT
INT	/	FLOAT	FLOAT
INT	>	FLOAT	BOOL
INT	>=	FLOAT	BOOL
INT	<	FLOAT	BOOL
INT	<=	FLOAT	BOOL
INT	=	FLOAT	FLOAT
INT	==	FLOAT	BOOL
INT	!=	FLOAT	BOOL
INT	^	FLOAT	FLOAT
FLOAT	+	FLOAT	FLOAT
FLOAT	-	FLOAT	FLOAT
FLOAT	*	FLOAT	FLOAT
FLOAT	/	FLOAT	FLOAT
FLOAT	>	FLOAT	BOOL
FLOAT	>=	FLOAT	BOOL
FLOAT	<	FLOAT	BOOL
FLOAT	<=	FLOAT	BOOL
FLOAT	=	FLOAT	FLOAT
FLOAT	==	FLOAT	BOOL
FLOAT	!=	FLOAT	BOOL
FLOAT	^	FLOAT	FLOAT
BOOL	=	BOOL	BOOL
BOOL	==	BOOL	BOOL
BOOL	!=	BOOL	BOOL
BOOL	AND	BOOL	BOOL
BOOL	OR	BOOL	BOOL

Proceso de administración de memoria usada en la compilación

Se crearon diferentes direcciones de memoria donde se asigna una base mas el contador de variables. (INT, FLOAT, BOOL)

```
self.mGlobal = valoresmemoria(5500, 6500, 7500)
self.mLocales = valoresmemoria(8500, 9500, 10500)
self.mTemporales = valoresmemoria(11500, 12500, 13500)
self.mConstante = valoresmemoria(14500, 15500, 16500)
self.mTempArr = valoresmemoria(17500, 18500, 19500)
```

DESCRIPCIÓN DE LA MÁQUINA VIRTUAL

El equipo de computo, lenguaje y librerías son lo mismo usado en compilación.
En memoria las direcciones se dieron de la siguiente manera:

self.mGlobal = valoresmemoria(5500, 6500, 7500)
self.mLocales = valoresmemoria(8500, 9500, 10500)
self.mTemporales = valoresmemoria(11500, 12500, 13500)
self.mConstante = valoresmemoria(14500, 15500, 16500)
self.mTempArr = valoresmemoria(17500, 18500, 19500)

Las primeras direcciones son de tipo INT, las segundas tipo FLOAT, y las ultimas tipo BOOL.

PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE

FIBONACCI

- Código .txt
- Generación de código

```
MAIN {

INT n;
INT x1;
INT x2;
INT temp;
INT next;

n = 5;
x1 = 1;
x2 = 1;
temp = 1;
next = 0;

WHILE (temp <= n) {

  IF(temp == 1) {
        WRITE(x1);
   }

  IF(temp == 2) {
    WRITE(x2);
  }

  IF(temp > 2) {
    next = x1 + x2;
    x1 = x2;
    x2 = next;
   WRITE(next);
  }
  temp = temp + 1;
 }
}
```

```
0       Goto    None    None    1
1       =       14500   None    8500
2       =       14501   None    8501
3       =       14501   None    8502
4       =       14501   None    8503
5       =       14504   None    8504
6       <=      8503    8500    13500
7       GotoF   13500   None    None
8       GotoF   8504    None    24
9       ==      8503    14501   13501
10      GotoF   13501   None    12
11      Write   8501    None    None
1
12      ==      8503    14506   13502
13      GotoF   13502   None    15
14      Write   8502    None    None
1
15      >       8503    14506   13503
16      GotoF   13503   None    22
17      +       8501    8502    11500
18      =       11500   None    8504
19      =       8502    None    8501
20      =       8504    None    8502
21      Write   8504    None    None
2
22      +       8503    14501   11501
23      =       11501   None    8503
Archivo aprobado
```

```
Cuadruplos:
0       Goto    None    None    1
1       =       14500   None    8500
2       =       14501   None    8501
3       =       14501   None    8502
4       =       14501   None    8503
5       =       14504   None    8504
6       <=      8503    8500    13500
7       GotoF   13500   None    None
8       GotoF   8504    None    24
9       ==      8503    14501   13501
10      GotoF   13501   None    12
11      Write   8501    None    None
12      ==      8503    14506   13502
13      GotoF   13502   None    15
14      Write   8502    None    None
15      >       8503    14506   13503
16      GotoF   13503   None    22
17      +       8501    8502    11500
18      =       11500   None    8504
19      =       8502    None    8501
20      =       8504    None    8502
21      Write   8504    None    None
22      +       8503    14501   11501
23      =       11501   None    8503
24      END     None    None    None
```

ESTRELLA

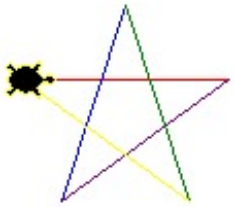
- Código .txt

```
MAIN  {
INT a;
INT b;
INT c;

a = 5;
b = 2;
c = 4;

ESTRELLA(a,b,c);
}
```

- Generación de código

```
0       Goto    None    None    1
1       =       14500   None    8500
2       =       14501   None    8501
3       =       14502   None    8502
4       Estrella        8500    8501    8502
Archivo aprobado

Cuadruplos:
0       Goto    None    None    1
1       =       14500   None    8500
2       =       14501   None    8501
3       =       14502   None    8502
4       Estrella        8500    8501    8502
5       END     None    None    None
```

- Output gráfico



ESPIRAL

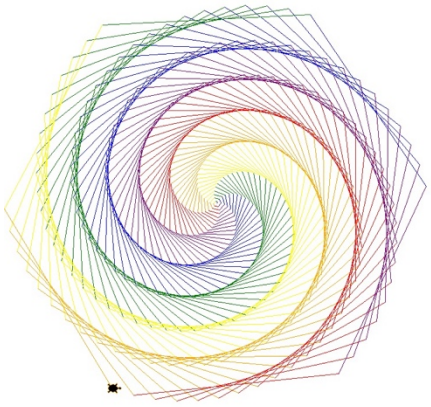
- Código .txt

```
FUNCION INT intento(INT n) {
INT F;
INT K;
INT P;

K = 6;
P = 7;
F = K + P;

RETURN (F);

}

MAIN  {
INT a;
INT b;
INT c;
INT d[10];
INT e[5];

a = 360;
b = 59;
c = 10;

e[3] = 4;
e[4] = 60;
e[5] = 5;

ESPIRAL(a,b,c);

intento(a);

IF (c == 2) {

}

}
```

- Generación de código

```
0       Goto    None    None    7
7       =       14502   None    8500
8       =       14503   None    8501
9       =       14504   None    8502
10      Espiral 8500    8501    8502
11      Era     1       None    None
12      Param   8500    None    param1
13      Gosub   1       None    None
1       =       14500   None    8502
2       =       14501   None    8503
3       +       8502    8503    11500
4       =       11500   None    8501
5       Return  8501    None    None
14      ==      8502    14511   13500
15      GotoF   13500   None    16
Archivo aprobado

Cuadruplos:
0       Goto    None    None    7
1       =       14500   None    8502
2       =       14501   None    8503
3       +       8502    8503    11500
4       =       11500   None    8501
5       Return  8501    None    None
6       EndProc None    None    None
7       =       14502   None    8500
8       =       14503   None    8501
9       =       14504   None    8502
10      Espiral 8500    8501    8502
11      Era     1       None    None
12      Param   8500    None    param1
13      Gosub   1       None    None
14      ==      8502    14511   13500
15      GotoF   13500   None    16
16      END     None    None    None
```

- Output gráfico



MANUAL DE USUARIO

Doodlr

Doodlr es un lenguaje de programación que ayuda a los usuarios aprender a programar en un nivel básico, en el cual puedes practicar con variables, arreglos, ifs, whiles y operaciones aritméticas, y logras ver un resultado con un output grafico con el que puedes jugar y crear diferentes figuras.

Requerimientos

Doodlr necesita tener la siguiente librería para compilar:

- Python (~v mínima 3) : https://www.python.org/download/releases/3.0/

Empieza a usar Doodlr

1. Entra a la siguiente liga: https://github.com/andrejimenez/Doodlr y descarga Doodlr como un .zip, descomprímelo en tu computadora en la ubicación en la que lo desees usar.
2. Crea tus archivos de códigos en un archivo .txt y ubícalos en la misma carpeta donde se ubique Doodlr.
3. Abre tu terminal y escribe:

    - python Doodlr.py NombreDeTuArchivo.txt

Programar en Doodlr.

VARIABLES

Las variables y arreglos se deben definir como: INT, FLOAT ó BOOL, de la siguiente manera:

- INT a;
- FLOAT b;
- BOOL c;
- INT a[5];
- a = 10;
- b = 24.5;
- c = TRUE;
- a[4] = 3;

Se deben definir individualmente por aparte y después ya realizar la asignación. Las variables se pueden definir de manera local (main y/ó funciones) o de manera global (en el principio del archivo).
Las variables deben tener nombres diferentes, para utilizar el mismo nombre la variable debe ser global. Las variables globales se declaran:

- GLOBAL INT a;

CONDICIONALES & CICLOS

Los ifs y whiles deben tener la siguiente estructura:

- IF (a == 7) {
  d = d + 1;
  }

- WHILE (a > 5) {
  d = d + 1;
  a = a - 1;
  }

- IF (a == 7) {
  d = d + 1;
  }
  ElSE {
  d = d + 2;
  }

FUNCIONES & LLAMADAS

Las funciones se declaran antes del Main y su estructura es la siguiente:

- FUNCION VOID nombreFuncion (INT a)
  {
  a = a +1;
  nombreFuncion (a) ;
  }

Todo programa debe tener una función Main donde se pueden establecer variables locales y mandar a llamar funciones. Su estructura es la siguiente:

- MAIN
  {
  BOOL x;
  X = TRUE;
  miFuncion (x);
  WRITE (x);
  }

OUTPUT GRAFICO

En Doodlr puedes crear diferentes figuras, tales como un circulo, rectángulo, espiral y estrella. Una ventaja de Doodlr es que puedes jugar con los valores que estableces para estas figuras y ver como cambia e incluso crear nuevas.

Para crear todas las figuras se toman tres parámetros, para los cuales debes definir primero su valor y luego asignar la variable en la sintaxis de la figura.

Ejemplo:

varA = 100;
varB = 20;
varC = 3;

Circulo:

- CIRCULO (varA, varB, varC);

- varA hace referencia al radio.
- varB hace referencia al ancho.
- varC hace referencia al color.

Espiral:

- ESPIRAL (varA, varB, varC);

- varA hace referencia al rango.
- varB hace referencia al angulo.
- varC hace referencia al color.

Rectángulo:

- RECTANGULO (varA, varB, varC);

- varA hace referencia al largo.
- varB hace referencia al alto.
- varC hace referencia al color.

Estrella:

- ESTRELLA (varA, varB, varC);

- varA hace referencia a los vertices.
- varB hace referencia a los pasos.
- varC hace referencia a largo.

Color:

Para las figuras circulo, rectángulo y espiral:

1 = “Rojo” 2 = “Morado” 3 = “Azul” 4 = “Amarillo” 5 = “Naranja”

Solo para espiral (opción extra):

10 = “Todos los colores”