**Instituto Tecnológico y de Estudios Superiores de Monterrey**



**Diseño de compiladores**

**Documentación del Proyecto Final**

Dibuja++

Francia Sugey Meléndez Hernández A00757592

Benjamín Salvador González Cárdenas A01190193

Monterrey, N.L., a 25 de Noviembre de 2015

**Indice**

**Descripción del proyecto**…………………………………………………………………2

Visión, Objetivos y Alcance del Proyecto…………………………………………2

Análisis de requerimientos y casos de uso generales…………………………..2

Descripción de los primeros Test Cases………………………………………….3

Descripción del proceso general seguido para el desarrollo …………………..3

**Descripción del lenguaje**…………………………………………………………………4

Nombre del lenguaje………………………………………………………………..4

Descripción genérica de las principales características del lenguaje………....4

Descripción de los errores que pueden ocurrir…………………………………..5

**Descripción del compilador**……………………………………………………………...5

Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo...5

Descripción del análisis del Léxico………………………………………………..5

Patrones de construcción…………………………………………………..5

Enumeración de los “tokens” del lenguaje y su código asociado……...6

Descripción del análisis de Sintaxis……………………………………………….7

Gramática Formal empleada para las estructuras sintácticas………...7

Descripción de Generación de Código intermedio y Análisis Semántico……..8

Código de operación y direcciones virtuales …………………………….8

Diagrama de Sintaxis con las acciones correspondientes……………...9

Descripción de cada una de las acciones semánticas y de código…..13

Tabla de consideraciones semánticas ………………………………….13

Descripción detallada del proceso de Administración de Memoria…………..14

Especificación gráfica de cada estructura de datos usada………....…14

**Descripción de la máquina virtual**…………………………………………………….15

Equipo de cómputo, lenguaje y utilerías especiales usadas………………….15

Descripción detallada del proceso de Administración de Memoria…………..15

Especificación gráfica de cada estructura de datos usada……………15

Asociación hecha entre las direcciones virtuales y las reales………..15

**Pruebas del funcionamiento del lenguaje**………………………………………...…16

**Listados del proyecto**……………………………………………………………………18

**Descripción del proyecto**

1. **Visión, Objetivos y Alcance del Proyecto**

El proyecto tiene como visión crear un lenguaje gráfico: **Dibuja++**, el cual acerque el aprendizaje y conocimiento de la programación a los jóvenes de una forma didáctica y divertida. De tal manera que ellos puedan comprender la lógica que se encuentra en la programación de una forma más visual. Así mismo por medio del proyecto buscamos fortalecer nuestras habilidades de programación durante el desarrollo.

El principal objetivo del lenguaje es que sea simple y visual en donde por medio de nuestro lenguaje se puedan crear dibujos y obtener un resultado visual de la codificación.

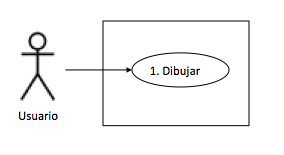
1. **Análisis de requerimientos y casos de uso generales**

**Palabras reservadas:** “programa”, “funcion”, “princ”, “regresa”, “entero”, “booleano”, “flotante”, “void”, “global”, “local”, “si”, “sino”, “mientras”, “para”, “cuadrado”, “circulo”, “arco”, “triangulo”, “poligono”, “linea”, “contorno”, “relleno”, “texto”, “grosor”, “xy”, “rotacion”.

**Id’s:** Palabras que representan variables o nombres de funciones.

**Constantes:** numéricas, booleanos, flotantes

**Casos de uso:**



|  |  |
| --- | --- |
| Nombre del caso de uso: | Dibujar |
| Descripción: | El usuario da instrucciones al compilador en forma de código del dibujo que desea crear, el compilador ejecuta las instrucciones y despliega de forma gráfica el dibujo brindado por el usuario. |
| Precondiciones: | Que el usuario indique en forma de código el dibujo que desea crear. |
| Poscondiciones: | Se despliega el dibujo. |
| Flujo de eventos principal: | 1. El usuario introduce el código. 2. El compilador, ejecuta las instrucciones del código. 3. Se despliega de forma gráfica el dibujo deseado. |
| Excepciones: | 2.1 En caso de que exista un error sintáctico, no se desplegará el dibujo y se notificará al usuario el error. |

1. **Descripción de los primeros Test Cases**

* Comprobar la declaración de variables y/o funciones.
* Comprobar operaciones aritméticas como suma, resta, multiplicación y división.
* Comprobar los operadores condicionales como >, <, >=, <=, == y !=.
* Comprobar la llamada a las funciones en el princ.
* Comprobar la funcionalidad de los ciclos.
* Comprobar la funcionalidad de arreglos.

1. **Descripción del proceso general seguido para el desarrollo**

|  |  |  |
| --- | --- | --- |
| **Bitácora Dibuja** | | |
| Fecha | Contenido Esperado | Avance real |
| 08/10/2015 | Léxico y Sintaxis | El léxico se encuentra listo, la sintaxis le falta detallar más. |
| 11/10/2015 | Tablas de variables y directorio de procedimientos. | Se hizo una hashtable para las variables, se creó el init, put y falta el get. Y en el archivo yacc se agregó el directorio de procedimientos. |
| 16/10/2015 | Semántica y código para expresiones | Se terminó la hashtable más hizo falta desarrollar el cubo semántico. |
| 21/10/2015 | Estatus secuenciales: asignación, lectura, escritura, etc. | Se terminó el cubo semántico. |
| 31/10/2015 | Funciones, Mapa de memoria de ejecución | Se hizo el mapa de memoria y se inició a trabajar en el avail. |
| 07/11/2015 | Máquina virtual: Expresiones aritméticas y estatutos secuenciales | Se continuó trabajando con el avail e iniciamos a trabajar las expresiones aritméticas en la máquina virtual |
| 14/11/2015 | Generación de código de arreglos.  Máquina virtual: estatutos condicionales | Se hizo modificaciones en el lex, yacc y avance para la máquina virtual. |
| 20/11/2015 | Máquina virtual. Primera parte de documentación | Se hicieron modificaciones al avail y se continuó la máquina virtual. |
| 25/11/2015 | Entrega final | Finalización y entrega del proyecto |

Para el desarrollo del proyecto realizamos juntas semanales en donde repartimos el trabajo y mantuvimos la comunicación para resolver dudas y discutir los aspectos del proyecto. Para el control de versiones utilizamos la herramienta GitHub en donde cada integrante iba subiendo su avance. Dicha herramienta fue de gran ayuda para mantener el orden en la integración del proyecto.

**Descripción del lenguaje**

1. **Nombre del lenguaje**

Dibuja ++

1. **Descripción genérica de las principales características del lenguaje.**

* El lenguaje es un output gráfico en el cual se pueden crear figuras geométricas como rectángulos, triángulos, círculos y líneas, y con ellos poder hacer diferentes dibujos. Así mismo se podrá definir distintos atributos de las figuras, tal como el color de relleno, color de contorno, grosor de la línea y posición de las figuras.
* No se podrán mezclar los tipos de datos al realizar operaciones aritméticas o de asignación.
* Los nombres de las variables deben ser distintos para cada una de ellas, es decir, no se deben repetir.
* Los operadores tendrán una jerarquía en el orden de aplicación, en donde se dará mayor prioridad a los paréntesis, luego a la multiplicación y división, y después a la suma y resta.
* Las variables pueden o no tener su asignación al momento de declararlas, en caso de no tenerla, se deberá realizar antes de llamar a la variable para su implementación.
* El final de línea en el código lo determina un punto y coma.

1. **Descripción de los errores que pueden ocurrir, tanto compilación como ejecución**

**Compilación:**

* Repetir nombres de variables y/o funciones.
* No declarar una variable y/o función.
* Llamar a una función con menos parámetros de los declarados.
* Utilizar una variable o función con diferente tipo al declarado (ej. declarar un string y querer utilizar la variable como tipo entero).
* No definir el tipo al momento de declarar una variable y/o función.

**Ejecución:**

* Que se cicle una función recursiva o un ciclo.
* Cuando se hace una división entre 0.
* Acceso de memoria inválido.

**Descripción del compilador**

1. **Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo**

**Equipo de cómputo**: Computadoras MacBook con OS 10.9.5 y 10.10

**Lenguaje:** Python

**Utilerias:** Ply

1. **Descripción del análisis del Léxico**
   1. **Patrones de construcción**

Palabra reservada : clave que utiliza el parser

'si' : 'SI',

'sino' : 'SINO',

'programa' : 'PROG',

'var' : 'V',

'princ' : 'PRINC',

'entero' : 'ENT',

'flotante' : 'FLOT',

'void' : 'VOID',

'global' : 'GL',

'funcion': 'FUNCION',

'rectangulo' :'REC',

'fondo' :'Fondo',

'triangulo' :'TRI',

'circulo' : 'CIR',

'cuadrado' : 'SQ',

'poligono' : 'POL',

'angulo' : 'A',

'arco' : 'ARC',

'linea' : 'LS',

'repetir' : 'RE',

'regresar' : 'RT',

'texto' : 'LA',

'grosor':'GROSOR',

'contorno':'CONTORNO',

'relleno':'RELLENO',

'xy':'XY',

* 1. **Enumeración de los “tokens” del lenguaje y su código asociado**

t\_ignore = ' \t\n\r'

t\_P = r'&'

t\_IG = r'='

t\_MA = r'>'

t\_ME = r'<'

t\_LTH = r'<='

t\_MTH = r'>='

t\_CD = r'!='

t\_SUM = r'\+'

t\_RES = r'-'

t\_MUL = r'\\*'

t\_DIV = r'/'

t\_PC = r';'

t\_C = r'\,'

t\_LB = r'\{'

t\_RB = r'\}'

t\_AP = r'\('

t\_CP = r'\)'

t\_AC = r'\['

t\_CC = r'\]'

t\_VALI = r'\d+'

t\_VALF = r'\d+\.\d+'

t\_STR = r'\'.\*\''

t\_CI = r'=='

1. **Descripción del análisis de Sintaxis**
   1. **Gramática Formal empleada para representar las estructuras sintácticas**

prog : PROG prog1 princ AC locales Bloque CC

princ : PRINC

locales : var

prog1 : prog2 prog3

prog2 : globales

prog3 : Funciones prog3 | vacia

Funciones : Fun1 varFunciones Bloque

Fun1 : fBloque fID func

fBloque : FUNCION FTipo

FTipo : VOID | tipo

func : fun1 fun2

fun2 : C fun1 fun2

fun1 : tipo arreglo ID arrD

arreglo : P | vacia

arrD : exp | vacia

Regresar : RT Regresar2 PC

Regresar2 : exp | vacia

Bloque : Bloque3

Bloque3 : Estatuto Bloque3 | vacia

globales : glob var | vacia

glob : GL

varFunciones : var

var : V var11 | vacia

var11 : tipo var1 var11 | vacia

var1 : varSalvar var2 var23 var3

var3 : varSalvar var2 var23 var3 | vacia

varSalvar : ID

var23 : IG var4 | vacia

var2 : var21 | LB var22 | vacia

var4 : exp | exp exp var41| exp var42

var41 : exp exp var41 | vacia

var42 : exp var42 | vacia'

var21 : exp

var22 : exp

Estatuto : Objeto | Condicion | Asignacion | Ciclo | Regresar

tipo : ENT | FLOT

expresion : exp ex2

ex2 : ex3 exp

exp : Termino exprog2

exprog2 : exp4 exprog3 exp | exp4 vacia

exp4 : vacia

exprog3 : SUM | RES

Termino : Factor Termino2

Termino2 : Termino4 Termino3 Termino | Termino4 vacia

Termino4 : vacia

Termino3 : MUL | DIV

Factor : Factor2 exp | Factor4

Factor2 : (

Factor4 : varCte | faID Factor5

faID : ID

Factor5 : [ exp ] | { exp } | Llamada | vacia

varCte : VALI | VALF

Condicion : SI ( expresion condicion2 Bloque condicion3

condicion2 : )

ondicion3 : vacia | con3 Bloque

con3 : SINO

Asignacion2 : Asigna | PC

Asigna : = AsignaT

AsignaT : exp PC | lAsigna

Asignacion : faID Factor5 Asignacion2

Llamada : funEra func2 )

funEra : (

func2 : func4 func3 | vacia

func3 : C func4 func3 | vacia

Ciclo : RE Cicloprog3 Bloque

Cicloprog3 : varCte

Objeto : Cuadrado | Triangulo | Poligono | Linea | uno\_par | Arco | Texto

position : XY ( exp , exp );

Cuadrado : REC ( exp , exp p\_relleno );

p\_relleno : C Fondo | vacia

Poligono : POL ( idList p\_relleno );

Triangulo : TRI ( [ exp , exp ], [ exp , exp ], [ exp, exp] p\_relleno);

uno\_par : CIR ( exp p\_relleno); | SQ ( exp p\_relleno);

Linea : LS (idList);

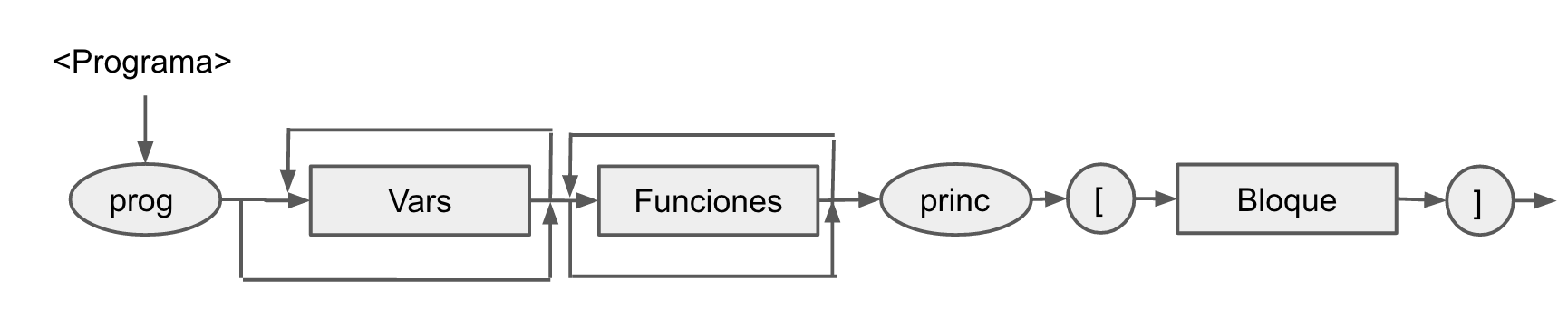
idList : ID

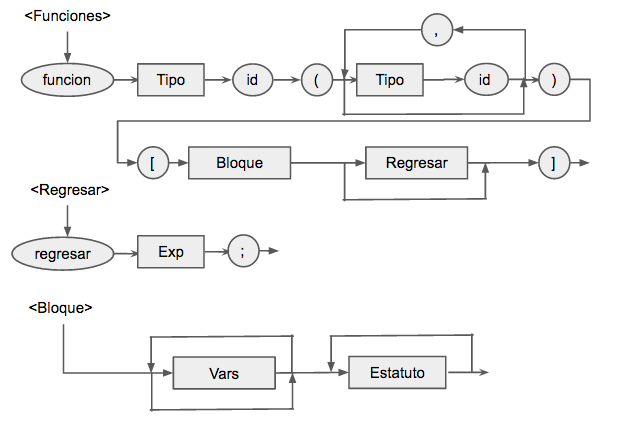
Arco : ARC ( exp p\_relleno);

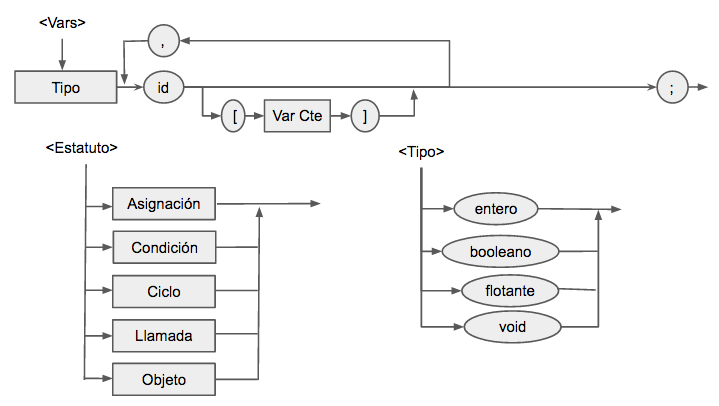
1. **Descripción de Generación de Código intermedio y Análisis Semántico**
   1. **Código de operación y direcciones virtuales asociadas a los elementos del código**

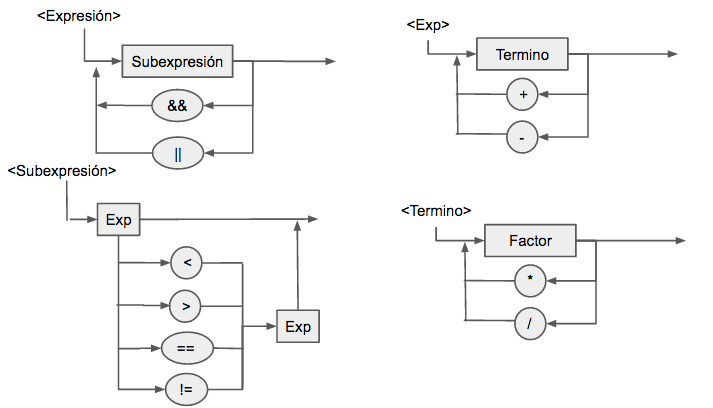
|  |  |
| --- | --- |
| Código de operación | Código asociado |
| suma | + |
| resta | - |
| multiplicar | \* |
| dividir | / |
| menor que | < |
| menor igual | <= |
| mayor que | > |
| mayor igual | >= |
| diferente | != |
| igual que | == |
| goto | GOTO |
| gotof | GOTOF |
| endprog | ENDPROG |
| endproc | ENDPROC |
| gosub | GOSUB |
| parametro | PARAM |
| era | ERA |
| return | RETURN |
| dim arreglo | DIM |
| dim matriz | DIMC |
| apuntador dir | DIR |
| apuntador dir cons | DIRC |
| asigna | 101 |
| texto | 208 |
| linea | 206 |
| cuadrado | 201 |
| triangulo | 202 |
| circulo | 203 |
| arco | 207 |
| poligono | 205 |
| relleno | 209 |
| color contorno | 301 |
| color relleno | 302 |
| grosor | 304 |

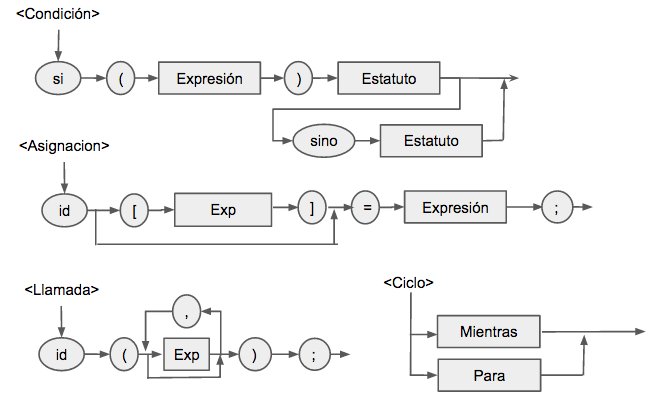
* 1. **Diagrama de Sintaxis con las acciones correspondientes**

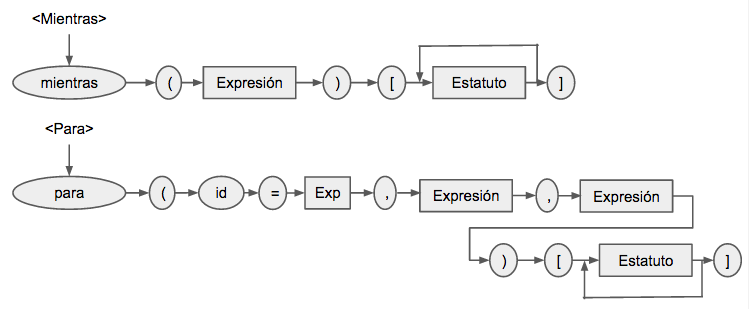


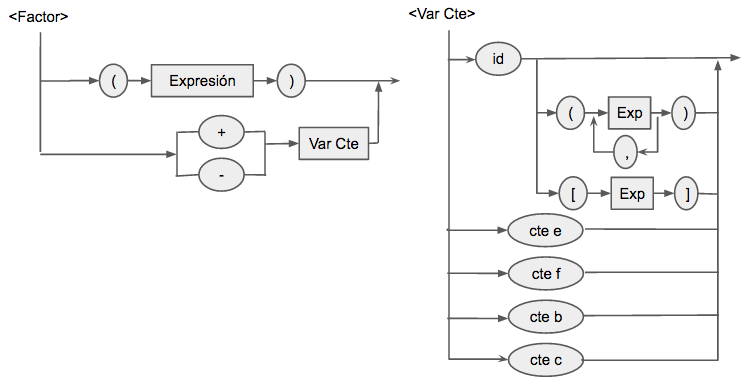


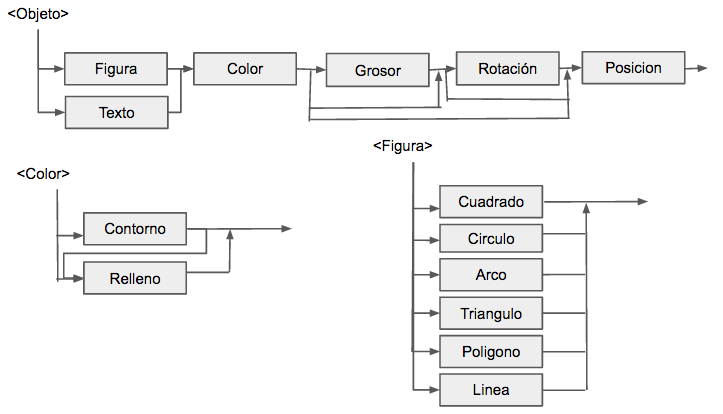


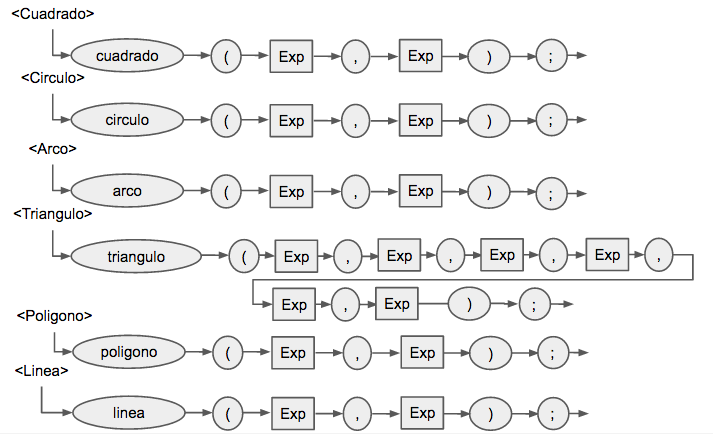


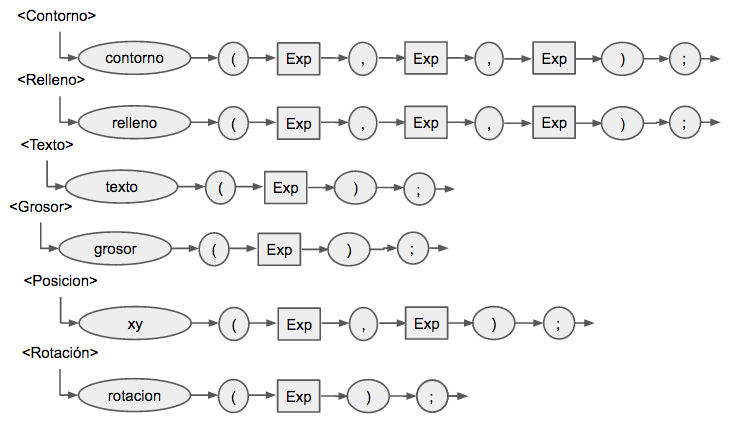












* 1. **Descripción de cada una de las acciones semánticas y de código**

**Relleno.-** recibe como parámetros expresiones que representan el código de color RGB y regresa el color para el objeto.

**Figura.-** recibe como parámetro expresiones que determinan el tipo de figura que se quiere dibujar.

**Vars Declaradas.-** recibe como parámetro la dirección virtual y determina si una variable está declarada en el bloque.

**Grosor.-** recibe como parámetro una expresión con la cual se determina el ancho de la línea de contorno de la figura.

**Posición.-** recibe como parámetro dos expresiones que indican la posición en X y Y que va a tener la figura.

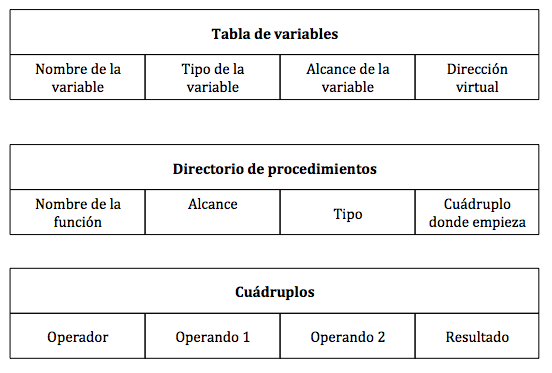
**Tipo variable.-** se determina el tipo de variable.

**Polígono.**- recibe como parámetro expresiones y crea un polígono.

* 1. **Tabla de consideraciones semánticas**

|  |  |  |  |
| --- | --- | --- | --- |
| Operador | Tipo 1 | Tipo 2 | Resultado |
| + | entero | entero | entero |
|  | entero | flotante | flotante |
|  | flotante | entero | flotante |
|  | flotante | flotante | flotante |
| - | entero | entero | entero |
|  | entero | flotante | flotante |
|  | flotante | entero | flotante |
|  | flotante | flotante | flotante |
| \* | entero | entero | entero |
|  | entero | flotante | flotante |
|  | flotante | entero | flotante |
|  | flotante | flotante | flotante |
| / | entero | entero | entero |
|  | entero | flotante | flotante |
|  | flotante | entero | flotante |
|  | flotante | flotante | flotante |
| < | entero | entero | bool |
|  | entero | flotante | bool |
|  | flotante | entero | bool |
|  | flotante | flotante | bool |
| <= | entero | entero | bool |
|  | entero | flotante | bool |
|  | flotante | entero | bool |
|  | flotante | flotante | bool |
| > | entero | entero | bool |
|  | entero | flotante | bool |
|  | flotante | entero | bool |
|  | flotante | flotante | bool |
| >= | entero | entero | bool |
|  | entero | flotante | bool |
|  | flotante | entero | bool |
|  | flotante | flotante | bool |
| != | entero | entero | bool |
|  | entero | flotante | bool |
|  | flotante | entero | bool |
|  | flotante | flotante | bool |
| == | entero | entero | bool |
|  | entero | flotante | bool |
|  | flotante | entero | bool |
|  | flotante | flotante | bool |

1. **Descripción detallada del proceso de Administración de Memoria usado en la compilación**
   1. **Especificación gráfica de cada estructura de datos usada**



**Descripción de la máquina virtual**

1. **Equipo de cómputo, lenguaje y utilerías especiales usadas**

**Equipo de cómputo**: Computadoras MacBook con OS 10.9.5 y 10.10

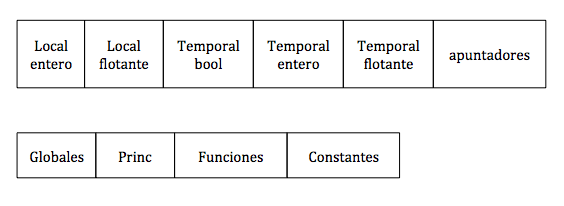
**Lenguaje:** Python

**Utilerias:** Tkinter

1. **Descripción detallada del proceso de Administración de Memoria en ejecución (Arquitectura)**
   1. **Especificación gráfica de cada estructura de datos usada**

Direcciones virtuales, dirección de inicio para cada bloque:

|  |  |
| --- | --- |
| **Variable** | **Dirección** |
| str | 1000 |
| enteros | 2000 |
| flotantes | 3000 |
| booleanos temporales | 4000 |
| enteros temporales | 5000 |
| flotantes temporales | 6000 |
| apuntadores | 7000 |
| globales | 10000 |
| princ | 20000 |
| funciones | 30000 |
| constantes | 40000 |



* 1. **Asociación hecha entre las direcciones virtuales (compilación) y las reales (ejecución)**

En la clase del administrador de memoria, el método de obtenerValor es el encargado de asociar las direcciones virtuales con las reales utilizando el cuádruplo y su dirección correspondiente. Esta función checa que la resta de las condiciones de cada bloque sea menor a 10000, si es asi hace otra resta según la dirección de cada bloque para determinar a que rango pertenece la direccion y asi saber el tipo de variable según al bloque que corresponda.

**Pruebas del funcionamiento del lenguaje**

**Factorial Recursivo:**

programa inicio del programa

funcion entero Factorial(entero F){ se declara función factorial

rectangulo(F+10,9,fondo); dibuja rectangulo

si(F < 0){ ciclo, si se cumple la condición F<0

regresar 9;

} sino { si no se cumple la condición

Factorial(F-1); llamada recursiva de la función

}

}

princ inicia programa princ (main)

[

{

posicion(250,250); se determina la posición para dibujar

relleno(255, 255, 78); se determina el color de la figura

grosor(4); grosor del contorno

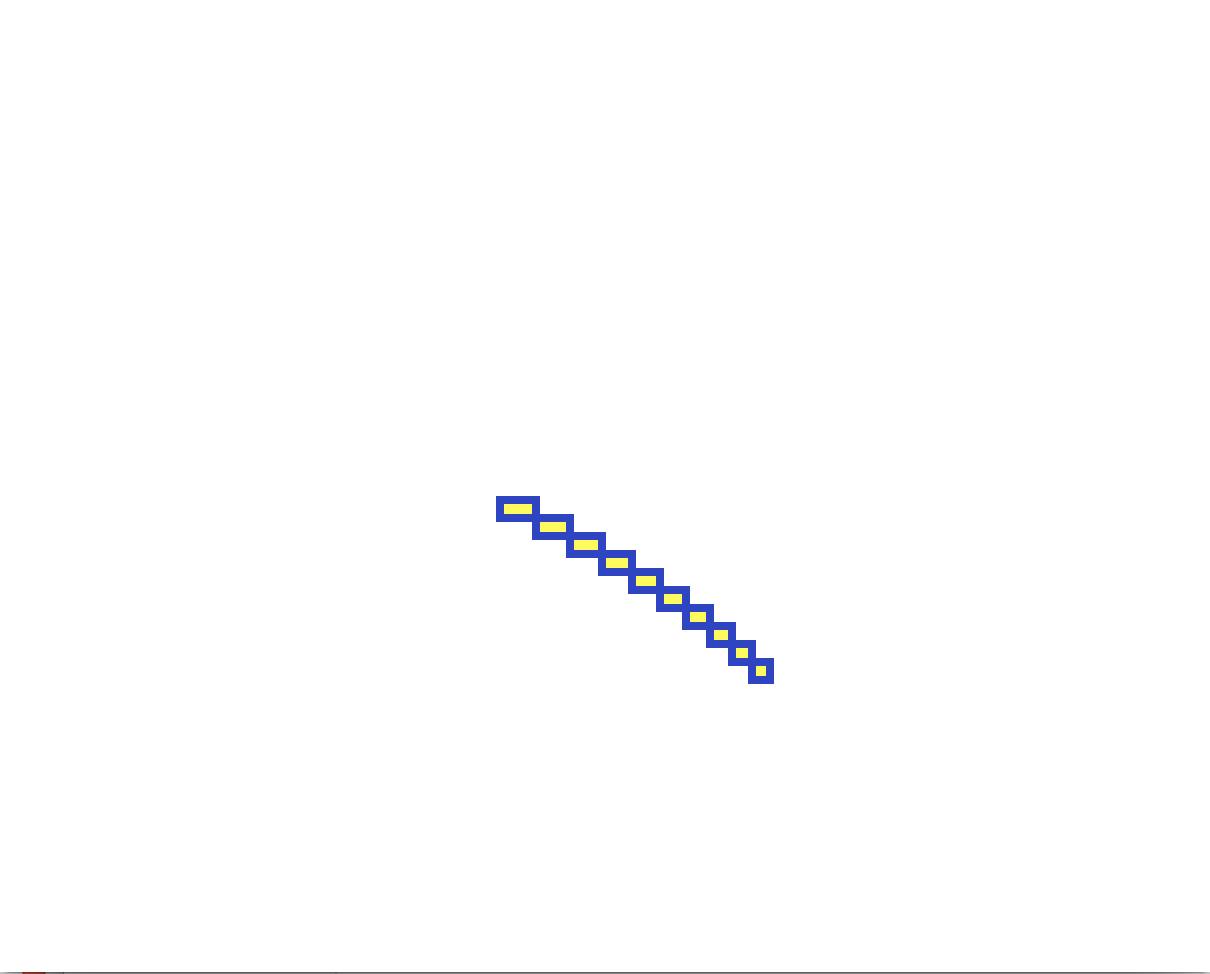
contorno(35, 45, 178); color del contorno

Factorial(8); se llama a la función Factorial

}

]

**Resultado**:



**Search:**

programa inicio

funcion entero search(entero &d[6], entero number, entero c) función

var entero pos, cont = 0; se declaran variables

{

c = c + 1;

repetir c{ ciclo, se repite c veces

si(d[cont] == number){ condición, si entonces..

pos = cont;

}

cuadrado(d[cont], fondo); dibuja un cuadro

cont = cont + 1;

}

regresar pos; regresa un valor

}

princ inicia princ (main)

[

var entero x[10] = [3.4,33, 19,21, 21, 4, 5, 7.2, 51,6]; declaración de vars

flotante h=9;

{

contorno(255,230,51); defina color de línea de contorno

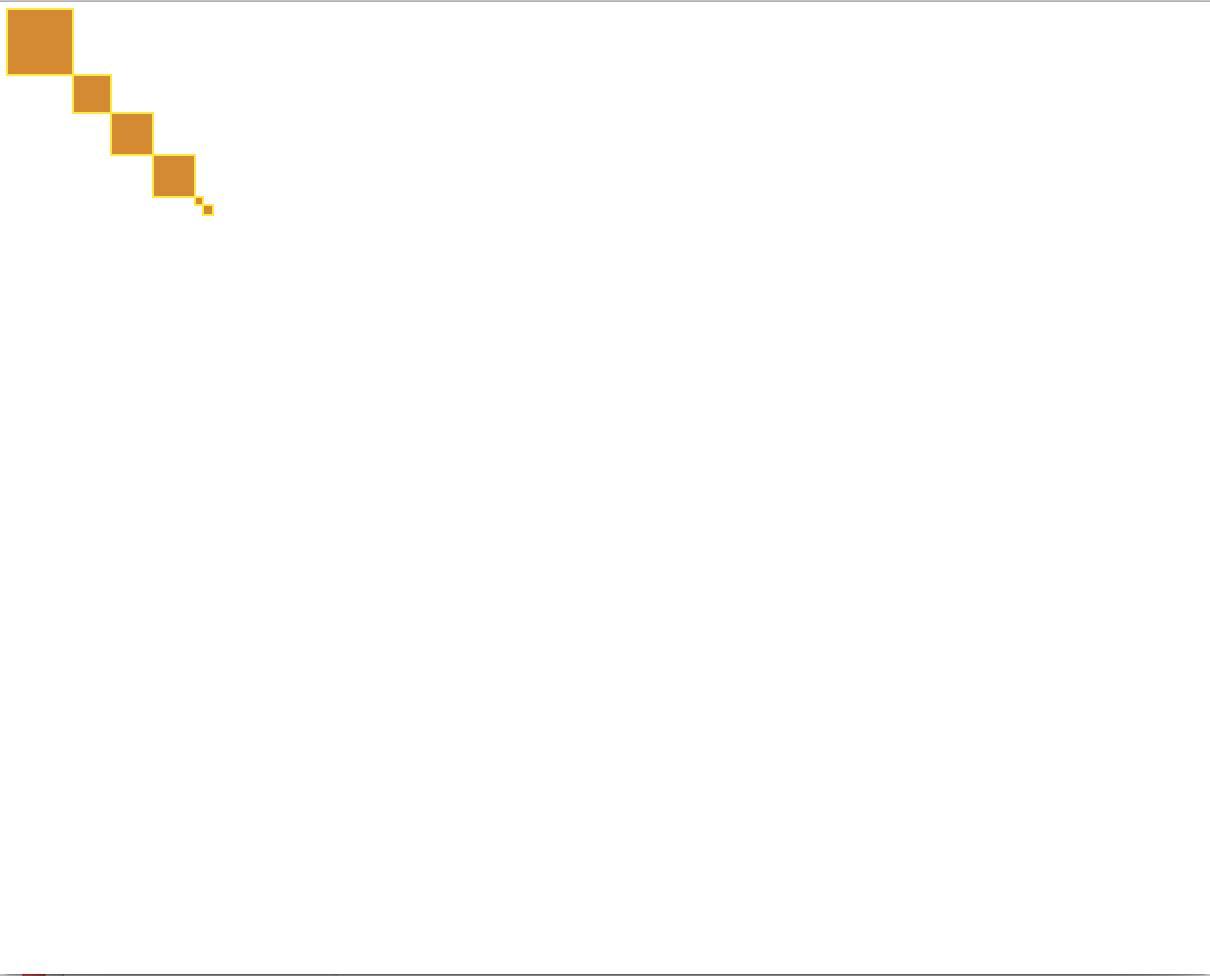
relleno(200,120,40); define color de figura

h = search(x, 4, 6); se asigna a h el resultado de la función

}

]

Resultado:



**Listados del proyecto**

yacc. py - generación de código de la gramática para dibujar un rectangulo

#Objeto----------------------------------------------------------------

def p\_Objeto(p):

'''Objeto : Figura

| Color

| Posicion

| Grosor'''

def p\_Figura(p):

'''Figura : Rectangulo

| Triangulo

| Poligono

| Linea

| CUADCIR

| Arco

| Texto'''

def p\_Posicion(p):

'''Posicion : XY AP exp C exp CP PC'''

avail.append\_quad\_dos(307)

def p\_Color(p):

'''Color : CONTORNO AP exp C exp C exp CP PC

| RELLENO AP exp C exp C exp CP PC'''

if(p[1] == 'contorno'):

fond = 301

elif(p[1] == 'relleno'):

fond = 302

avail.append\_quad\_tres(fond)

def p\_Grosor(p):

'''Grosor : GROSOR AP exp CP PC '''

avail.append\_quad\_uno(304)

def p\_Rectangulo(p):

'''Rectangulo : REC AP exp C exp fondo CP PC'''

avail.append\_quad\_dos(201)

def p\_fondo(p):

'''fondo : C Fondo

| vacia'''

if(len(p) == 3):

sC = [209, -1, -1, 1]

else:

sC = [209, -1, -1, -1]

avail.append\_quad(sC)

Instrucciones de la máquina virtual para dibujar un rectangulo

#funcion que crea un rectangulo

def rectangulo():

global quadActual, relleno, colorRell, colorCont, gros

x = memoria.getValor('41000')

y = memoria.getValor('41001')

x2 = x + memoria.getValor(quad[quadActual][1])

y2 = y + memoria.getValor(quad[quadActual][2])

if(relleno):

w.create\_rectangle(x, y, x2, y2, fill=colorRell, outline=colorCont, width=gros)

else:

w.create\_rectangle(x, y, x2, y2, fill='', outline=colorCont, width=gros )

memoria.escribeValor('41000', x2)

memoria.escribeValor('41001', y2)

quadActual += 1

**Instituto Tecnológico y de Estudios Superiores de Monterrey**



**Diseño de compiladores**

**Manual de usuario**

Dibuja++

Francia Sugey Meléndez Hernández A00757592

Benjamín Salvador González Cárdenas A01190193

Monterrey, N.L., a 25 de Noviembre de 2015

**Introducción**

El lenguaje Dibuja++ es un output gráfico y está enfocada a crear dibujos a través del código, de manera que al ejecutar el programa, el compilador regresa el dibujo del cual se dieron las instrucciones.

**Características generales del lenguaje:**

* El lenguaje es un output gráfico en el cual se pueden crear figuras geométricas como rectángulos, triángulos, círculos y líneas, y con ellos poder hacer diferentes dibujos. Así mismo se podrá definir distintos atributos de las figuras, tal como el color de relleno, color de contorno, grosor de la línea y posición de las figuras.
* No se podrán mezclar los tipos de datos al realizar operaciones aritméticas o de asignación.
* Los nombres de las variables deben ser distintos para cada una de ellas, es decir, no se deben repetir.
* Los operadores tendrán una jerarquía en el orden de aplicación, en donde se dará mayor prioridad a los paréntesis, luego a la multiplicación y división, y después a la suma y resta.
* Las variables pueden o no tener su asignación al momento de declararlas, en caso de no tenerla, se deberá realizar antes de llamar a la variable para su implementación.
* El final de línea en el código lo determina un punto y coma.

**Estructura básica de un programa:**

Ejemplo:

programa ← Inicio del programa

funcion entero uno(entero n){ ← Función adicional

rectangulo(n+1,9,fondo);

}

princ ← Función princ del programa (aquí se indica lo que

[ se va a ejecutar)

{

posicion(250,250); ← Instrucciones del programa.

relleno(255, 255, 78);

grosor(4);

contorno(35, 45, 178);

uno(5);

}

]

**Tipos de variables:**

* flotante
* entero

**Funciones predefinidas:**

Rectangulo (lado x, lado y, fondo)

Cuadrado (lado)

Triangulo ([lado, angulo], [lado, angulo], [lado, angulo], fondo)

Poligono (lista de puntos, fondo)

Linea (puntos)

Arco (angulo, fondo)

Texto (texto)

Posicion (posX, posY)

Contorno (entero, entero, entero) - los parámetros representan el código de color en RGB.

Relleno (entero, entero, entero) - los parámetros representan el código de color en RGB.

Grosor (tamaño)

**Estructuras**

**Funciones:**

funcion entero uno(entero n){ ← se define el tipo, nombre y los parámetros

rectangulo(n+1,9,fondo); ← se definen las instrucciones

}

**Ciclos:**

repite 9 { ← se define la cantidad de veces que se repite

rectangulo(n+1,9,fondo); ← instrucción

}

**Condiciones:**

si(n<0){ ← si (condición)

regresar 1; ← instrucción

} sino { ← en caso de que no se cumpla

regresar 2; ← instrucción

}