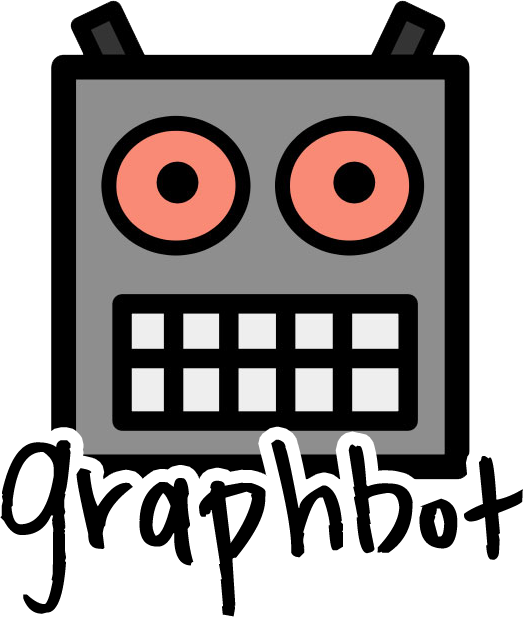
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY CAMPUS MONTERREY



Fernando Mendoza Ramírez A00515576

Claudia Graciela Hernández Rivera A01033904

Campus Monterrey

Maestras Titulares: Ing. Elda Quiroga

Curso: Compiladores

Monterrey, Nuevo León, a 7 de mayo de 2014

**a) Descripción del Proyecto**

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

**Visión:**

Diseñar un lenguaje gráfico que pueda ser utilizado como una herramienta de aprendizaje para jóvenes sin nociones de programación. El lenguaje permitirá que los usuarios aprendan los fundamentos de la lógica de programación de una manera divertida al utilizar una línea de comandos que irá dibujando líneas, figuras y formas en un canvas conforme el usuario vaya ejecutando los comandos.

**Objetivos:**

El lenguaje Graphbot tiene como objetivos:

* Ser un lenguaje de fácil aprendizaje para su trabajo con niños y jóvenes
* Enseñar los principales conceptos de la lógica de programación mediante el soporte para manejo de listas, ciclos, variables, entre otros
* Desarrollar habilidades metacognitivas en los niños o jóvenes mediante la presentación de retos intelectuales que puedan ser resueltos con el lenguaje

**Alcance del Proyecto:**

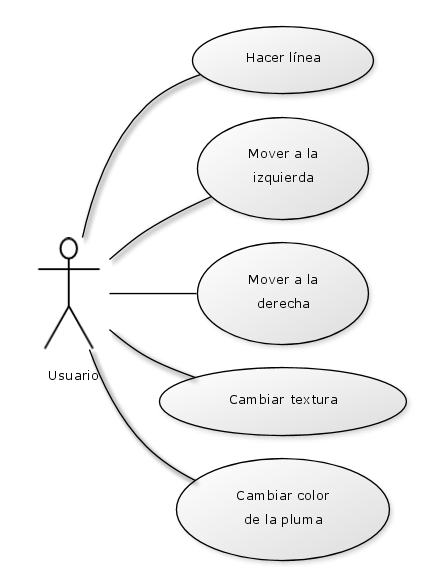
El lenguaje tiene como objetivo ser implementado en escuelas primarias para alentar a los jóvenes en el área de programación y fomentar el gusto por las ciencias computacionales. Se busca que sea un lenguaje atractivo que capture el interés de los niños para que quieran continuar programando. El mostrarles directamente en pantalla el resultado de los comandos que ellos mismos han programado, los incentivará a querer continuar utilizando el lenguaje.

* **Análisis de Requerimientos y Casos de Uso Generales**

**Requerimientos Principales**

* Manejo de Ciclos (*for, while)*
* Implementación de condicionales (*if)*
* Manejo de variables tipo lista
* Uso de módulos y funciones
* Salida gráfica
* Implementación de línea de comandos y ejecución de programas dentro de una aplicación

**Casos de Uso Generales**



**1. Hacer línea**

El usuario hace una línea mediante el comando *move* y una distancia especificada.

**2. Mover a la izquierda**

El usuario mueve al robot a la izquierda al utilizar el comando *leftturn* seguido de una cantidad en grados.

**3. Mover a la derecha**

El usuario mueve al robot a la izquierda al utilizar el comando *rightturn* seguido de una cantidad en grados.

**4. Cambiar textura**

El usuario cambia el fondo de la aplicación utilizando el comando *setbackgroundtxt* seguido del número de textura que quiera utilizar.

**5. Cambiar color de la pluma**

El usuario cambia el color de la pluma (comienza en negro) utilizando el comando *setcolor* seguido de los tres valores R,G,B del color al que quiere cambiar.

* **Descripción de los principales Test Cases**

espiral.gbot

Programa que dibuja una espiral. El programa comienza asignando la primera textura a la aplicación y después declara una variable i con valor inicial 10.0 Después utilizando esa variable entra a un ciclo for que aumenta la variable i en una cantidad de 10.0 cada vez que completa una iteración y lo repite hasta llegar a la cantidad 110.0. En cada iteración del for, se dibuja una línea con distancia i y se hace una vuelta a la derecha en 90º.

figuras.gbot

Programa que dibuja un asterisco y 4 octagonos alrededor. El programa comienza declarando la función *octagon* y declara una variable i que después es utilizada dentro de un ciclo for que se repite 8 veces y realiza líneas con una distancia de 50.0 para después girar a la derecha en 45º. Se declara otra función llamada *asterisk* que salva una variable j y la utiliza en un ciclo for que repite 20 veces y realiza una línea con distancia de 80.0, luego se dirige hacia abajo y vuelve a moverse en 80.0 para terminar dando una vuelta a la derecha en 18º. Finalmente llega el programa *main* que asigna la segunda textura al canvas, luego cambia el color de la pluma a blanco, llama la función *asterisk*, luego guarda otra variable i y repite la función *octagon* cuatro veces sobre el canvas.

heart.gbot

Programa que dibuja un corazón. El programa declara una función llamada heart que tiene dos parámetros x y. Al inicio de la función declara una variable i que utilizará dentro de un ciclo while. A continuación entra al while que se ejecutará dos veces y comienza asignando la posición del robot de acuerdo a los valores x y que se le pasaron y a partir de ese punto comienza a dibujar. Declara 3 variables que son utilizadas en un triple for. El primer for se repite 8 veces y gira a la derecha o la izquierda en 45º de acuerdo al resultado del if. El segundo for, se repite 6 veces e incluye otro for que se repite 90 veces y mueve el robot 1 unidad de distancia para después girar a la derecha o a la izquierda en 2º de acuerdo al resultado del if. Al acabar este for, se gira 90º a la derecha o la izquierda dependiendo del resultado del if y termina. La variable i se actualiza y se evalúa en el while. Al terminar el ciclo, dibuja las dos líneas que complementan el corazón. El programa main asigna la textura 3 (la libreta) a la pantalla, después cambia de color a rojo y finalmente manda llamar la función heart con párametros x y y de 0.0.

* **Descripción del proceso general seguido para el desarrollo del proyecto**

**Semana 2 (Marzo 10 – 14)**

* Resolución de conflictos
  + - 75 reduce/reduce
    - 5 shift/Reduce
* Directorio de procedimientos
* Inserción de procedimientos por declaración
* Tablas de Variables
* Inserción a tablas de variables por declaración y asignación
* Verificación de uso de variables inexistentes
* Creación de Makefile

**Semana 3 (Marzo 17 – 21)**

* Generación de Cuádruplos
  + - Expresiones Aritméticas
    - Asignación, Lectura (Comandos)
    - Comparadores
* Impresión a archivo de texto
* Revaluación ciclos y condiciones

**Semana 4 (Marzo 25 – 29)**

* Generación de Cuádruplos
  + - If
    - While
    - For

**Semana 5 (Abril 31 – 4)**

* Codificar comandos
* Funciones/Módulos
* Manejo de Variables Tipo Lista (tabla de variables, memoria virtual)
* Corregir estructura de datos utilizada para la tabla de variables y directorio de procedimientos
* Corregir cuando se mandan llamar dos funciones definidas por el usuario seguidas

**Semana 6 (Abril 7 – 11)**

* Implementar diseño de memoria
* Codificar comandos
* Convertir constantes numéricas en direcciones virtuales
* Codificar comparadores y operadores aritméticos
* Convertir saltos en direcciones virtuales
* Convertir variables en direcciones virtuales
* Convertir temporales en direcciones virtuales
* Cálculo de tamaño de funciones

**Semana Santa (Abril 15 – 19)**

* Cargar memoria en ejecución
* Cargar constantes
* Ejecución de funciones aritméticas

**Semana 7 (Abril 21 – 25)**

* Manejo de Memorias
* Ejecución de Estatutos Condicionales
* Instrucciones *move*, *rightturn* y *leftturn* listas

**Semana 8 (Abril 28 – 2)**

* Implementación de nuevos comandos
* Show
* Hide
* PenUp
* PenDown
* RightTurn
* LeftTurn
* Integración con WX Widgets (GUI)
* Implementación de comentarios
* Primera Versión de Documentación

**Principales aprendizajes logrados:**

**Fernando Mendoza:**

“*Durante el prcoa adasjdjasbksjdkajsdnkjasndkajsds*”

**Claudia Hernández:**

“*Definitivamente, la mejor manera de comprender los conceptos vistos en la clase de Compiladores fue el aplicarlos en un proyecto real. De lo contrario, los conceptos quedan como alusiones abstractas. El desarrollo de este proyecto me ayudó a aterrizar dichos conceptos a bits y bytes y a finalmente entender todo el proceso que hay detrás de un compilador.* ”

**b) Descripción del Lenguaje**

* **Nombre del Lenguaje**

Graphbot

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

Graphbot es un lenguaje que permite crear programas y funciones que tienen salida gráfica. Mediante sus diferentes tipos de comandos, Graphbot permite la creación de líneas, figuras y dibujos en pantalla. La línea de comandos permite evaluar de manera automática en pantalla la salida de los comandos, mientras que el *text area* permite la creación de programas que se ejecutarán al ser llamados desde línea de comandos. Graphbot permite la creación de infinidad de dibujos gracias a sus ciclos, condicionales y demás opciones que brindan variedad a la forma de dibujar sobre el canvas.

* **Descripción de los errores que pueden ocurrir, tanto en compilación como ejecución**
  + **Errores de Compilación:**
* *Función ya declarada:* La función ya se encuentra en el directorio de procedimientos.
* *Variable no declarada:* La variable no ha sido declarada y se intenta utilizar dentro del programa.
* *Función no declarada:* La función no ha sido declarada y se intenta utilizar dentro del programa.
* *Cantidad de parámetros errónea en función:* Durante la llamada a una función, la cantidad de parámetros es diferente a los que se declararon en la misma.
* *Variable es de tipo lista y no puede ser usada en una expresión:* La variables tipo lista no pueden ser utilizadas en expresiones aritméticas
* *Variable no es de tipo lista:* Las variables tipo lista pueden ser *usada únicamente dentro de los ciclos.*
* *BANG, error de parser: syntax error en la línea:* Cualquier tipo de error relacionado con léxico no reconocido, constantes que no sean flotantes, entre otros.

**c) Descripción del Compilador**

* **Equipo de Cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto**

El lenguaje se desarrolló en computadoras portátiles con OS X Versión 10.9.1. La herramienta Flex y Bison fue utilizada para el desarrollo del compilador. Se utilizó además tanto para el compilador como para la máquina virtual la versión 4.2.1 de g++.

* **Descripción del Análisis de Léxico**
  + **Patrones de Construcción y Enumeración de Tokens**

|  |  |
| --- | --- |
| **Expresión Regular** | **Token** |
| move | RW\_MOVE |
| main | RW\_MAIN |
| about | RW\_ABOUT |
| setpos | RW\_SETPOS |
| setx | RW\_SETX |
| sety | RW\_SETY |
| setcolor | RW\_SETCOLOR |
| setpensize | RW\_SETPENSIZE |
| setbackground | RW\_SETBACKGROUND |
| setbackgroundtxt | RW\_SETBACKGROUNDTXT |
| show | RW\_SHOW |
| hide | RW\_HIDE |
| clean | RW\_CLEAN |
| home | RW\_HOME |
| getcolorR | RW\_GETCOLORR |
| getcolorG | RW\_GETCOLORG |
| getcolorB | RW\_GETCOLORB |
| getpensize | RW\_GETPENSIZE |
| getx | RW\_GETX |
| gety | RW\_GETY |
| function | RW\_FUNCTION |
| save | RW\_SAVE |
| end | RW\_END |
| playmusic | RW\_PLAYMUSIC |
| stopmusic | RW\_STOPMUSIC |
| penup | RW\_PENUP |
| pendown | RW\_PENDOWN |
| rightturn | RW\_RIGHTTURN |
| leftturn | RW\_LEFTTURN |
| program | RW\_PROGRAM |
| if | RW\_IF |
| for | RW\_FOR |
| while | RW\_WHILE |
| [ | OP\_BRACKET |
| ] | CL\_BRACKET |
| ( | OP\_PAR |
| ) | CL\_PAR |
| , | COMMA |
| == | EQUAL |
| != | NOT\_EQUAL |
| >= | GREAT\_EQ\_THAN |
| <= | LESS\_EQ\_THAN |
| < | LESS\_THAN |
| > | GREAT\_THAN |
| + | ADD |
| - | SUB |
| \* | TIMES |
| / | DIV |
| // | COMMENTS |
| /\* | MULTI\_COMMENTS |
| [[:alpha:]][[:alnum:]|\_[:alnum:]]\* | ID |
| [[:digit:]]+"."[[:digit:]]+ | FLOAT |

* **Descripción del Análisis de Sintaxis**

<GRAPHBOT> GRAPHBOT’ PROGRAM

<GRAPHBOT’> GRAPHBOT’ FUNCTION

<GRAPHBOT’> ε

<FUNCTION> function id FUNCTION’ FUNCIONES end

<FUNCTION’> VAR FUNCTION’

<FUNCTION’> ε

<VAR> id VAR’

<VAR’> , VAR

<VAR’> ε

<PROGRAM> program id FUNCIONES end

<FUNCIONES> COMANDOS FUNCIONES’

<FUNCIONES> FOR FUNCIONES’

<FUNCIONES> WHILE FUNCIONES’

<FUNCIONES> CONDICIÓN FUNCIONES’

<FUNCIONES’> FUNCIONES FUNCIONES’

<FUNCIONES’> ε

<COMANDOS> COMANDO

<COMANDOS> COMANDO\_RETURN

<COMANDOS> COMANDO1 EXPRESIÓN

<COMANDOS> COMANDO3 EXPRESIÓN EXPRESIÓN EXPRESIÓN

<COMANDOS> setpos EXPRESIÓN EXPRESIÓN

<COMANDOS> save id VARIABLE

<COMANDOS> LLAMADA\_FUNC

<COMANDO> show

<COMANDO> hide

<COMANDO> clean

<COMANDO> home

<COMANDO> playmusic

<COMANDO> stopmusic

<COMANDO> about

<COMANDO> penup

<COMANDO> pendown

<COMANDO\_RETURN> getcolorR

<COMANDO\_RETURN> getcolorG

<COMANDO\_RETURN> getcolorB

<COMANDO\_RETURN> getpensize

<COMANDO\_RETURN> getx

<COMANDO\_RETURN> gety

<COMANDO1> move

<COMANDO1> rightturn

<COMANDO1> leftturn

<COMANDO1> setx

<COMANDO1> sety

<COMANDO1> setpensize

<COMANDO1> setbackgroundtxt

<COMANDO3> setcolor

<COMANDO3> setbackground

<FOR> for [ id , EXPRESION, EXPRESION ] LISTA

<CONDICIÓN> if EXPRESIÓN LISTA

<WHILE> while EXPRESIÓN LISTA

<LISTA> [ LISTA’ ]

<LISTA’> EXPRESIÓN | COMANDOS LISTA’’

<LISTA’’> , LISTA’

<LISTA’’> ε

<EXPRESIÓN> EXP EXPRESIÓN’ <EXPRESIÓN’> COMPARADOR EXP

<EXPRESIÓN’> ε

<EXP> TÉRMINO EXP’

<EXP’> + | - EXP

<EXP’> ε

<TÉRMINO> FACTOR TÉRMINO’

<TÉRMINO’> \* | / TÉRMINO

<TÉRMINO’> ε

<FACTOR> ( EXPRESIÓN )

<FACTOR> VARCTE

<VARCTE> id

<VARCTE> float

<COMPARADOR> <

<COMPARADOR> >

<COMPARADOR> !=

<COMPARADOR> ==

<COMPARADOR> <=

<COMPARADOR> >=

<LLAMADA\_FUNC> id LLAMADA\_FUNC’

<LLAMADA\_FUNC’> ( LLAMADA\_FUNC’’)

<LLAMADA\_FUNC’’> EXPRESIÓN LLAMADA\_FUNC’’

<LLAMADA\_FUNC’’> ε

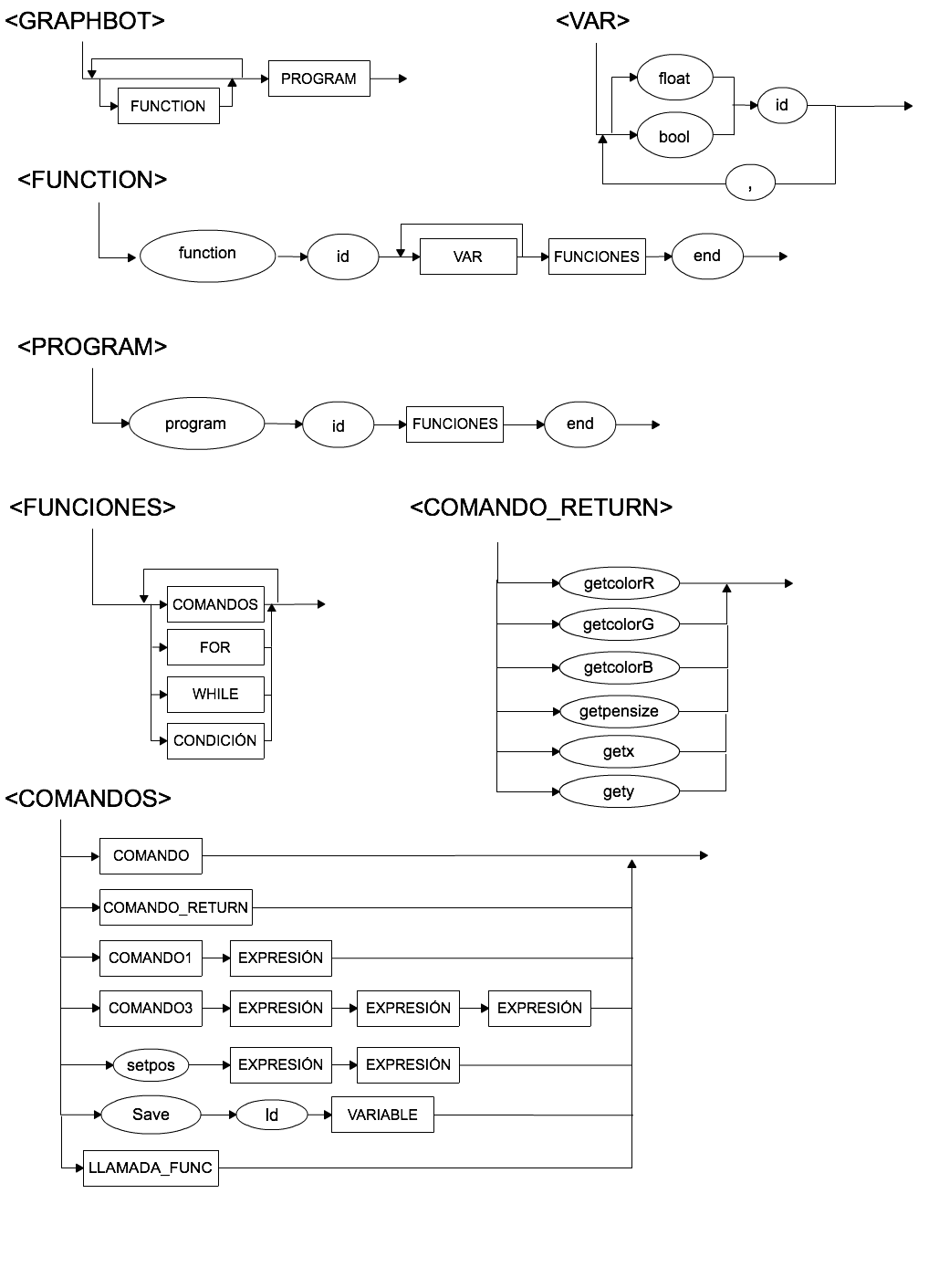
* **Descripción del Código Intermedio y Análisis Semántico**
  + **Código de operación y direcciones virtuales asociadas a los elementos del código**

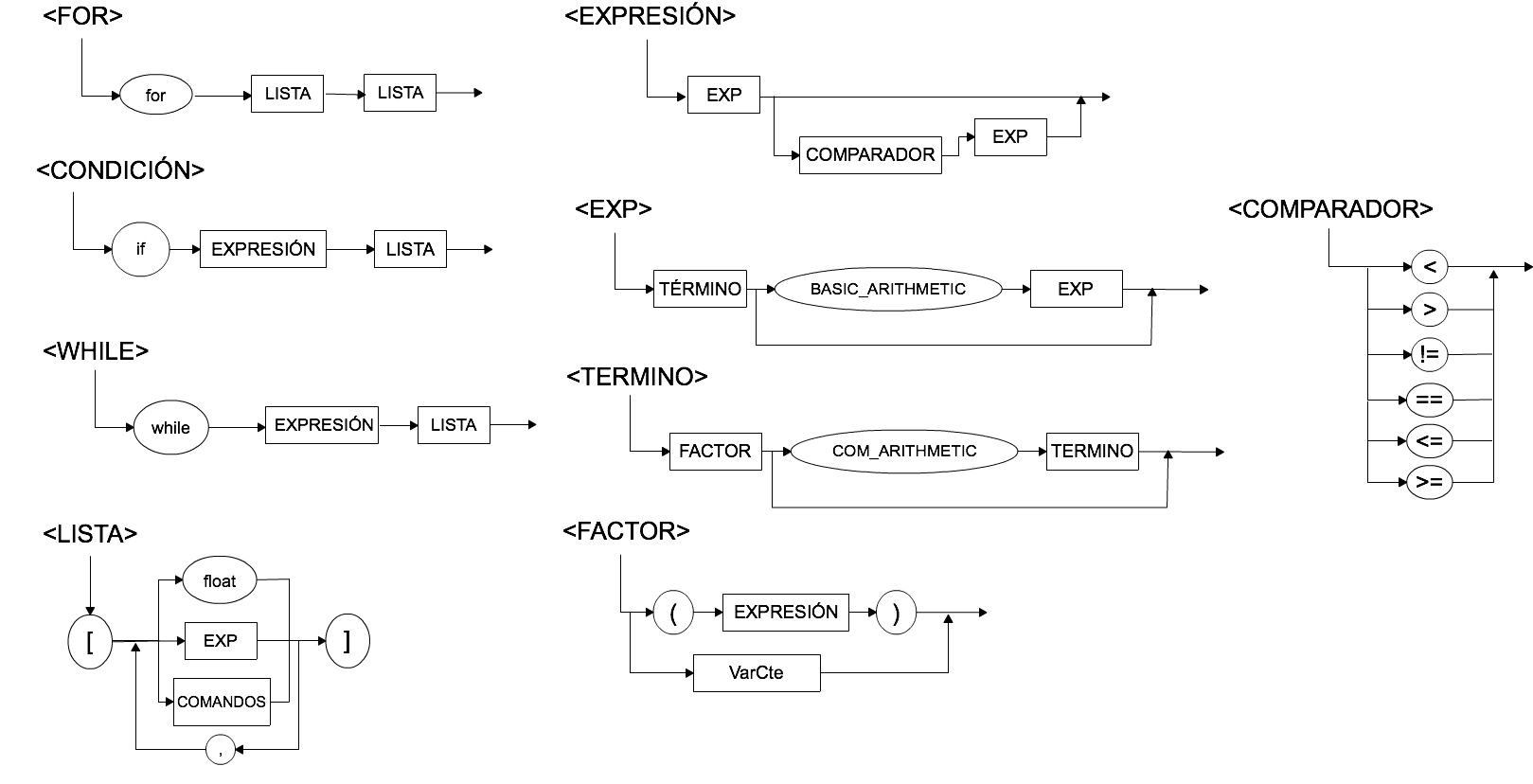
0001 - 0999 : constantes saltos  
1000 - 1999 : constantes numéricas  
2000 - 2999 : temporales  
3000 - 3999 : variables flotantes  
4000 - 4999 : variables enteras  
5000 - 5042 : comandos

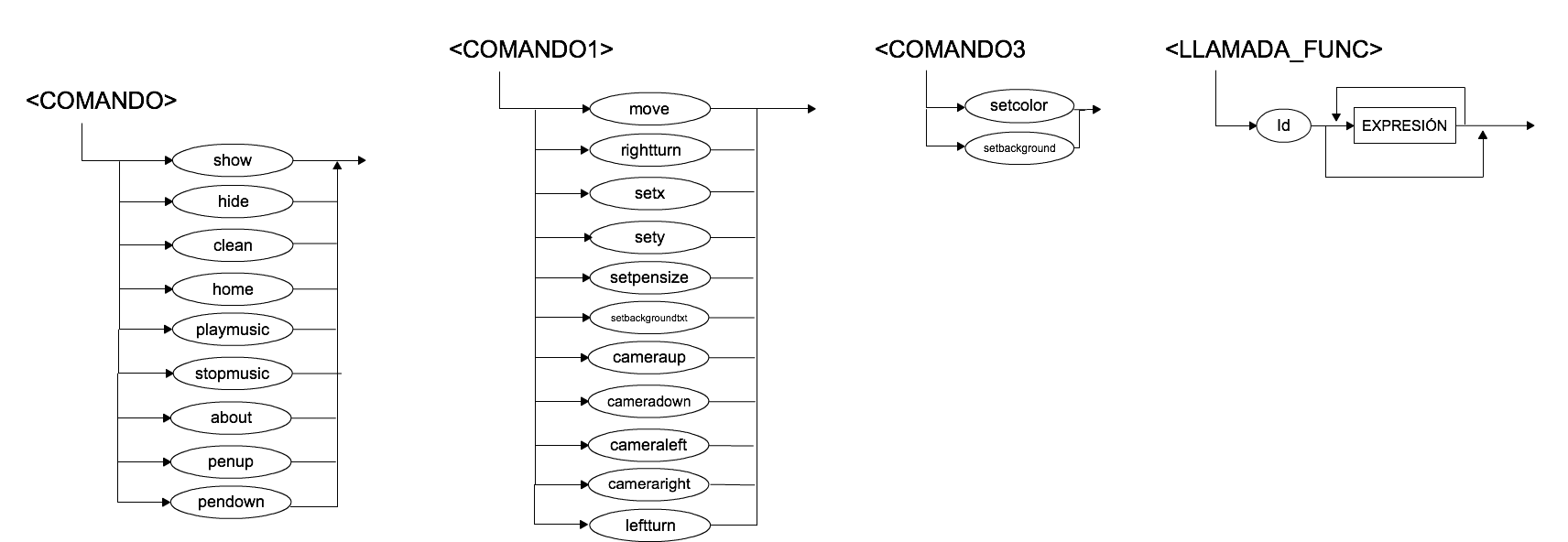
**Codificación de comandos**

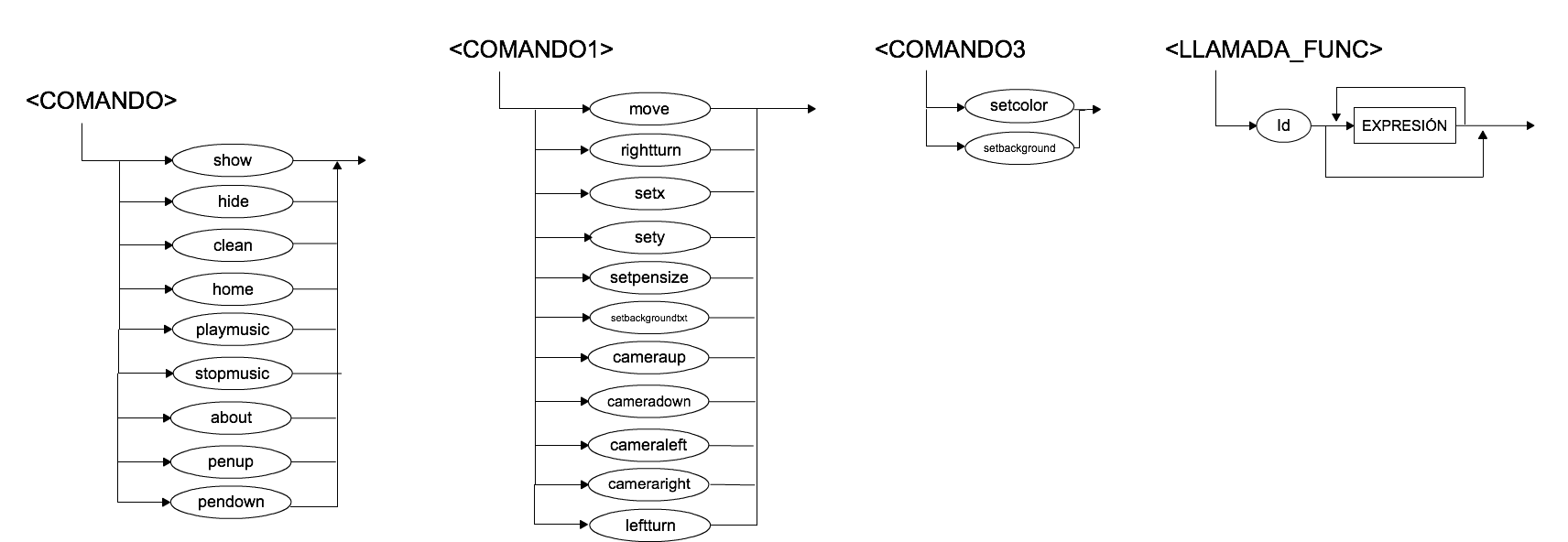
5000 : show  
5001 : hide  
5002 : clean  
5003 : home   
5004 : getcolorr  
5005 : getcolorg  
5006 : getcolorb  
5007 : getpensize  
5008 : getx  
5009 : gety  
5010 : stopmusic  
5011 : playmusic  
5012 : move  
5013 : about  
5014 : setx  
5015 : sety  
5016 : setpensize  
5017 : penup  
5018 : pendown  
5019 : rightturn  
5020 : leftturn  
5021 : setbackgroundtxt  
5022 : save  
5023 : setpos  
5024 : setcolor  
5025 : setbackground  
5026 : =  
5027 : >  
5028 : <  
5029 : !=  
5030 : >=  
5031 : <=  
5032 : +  
5033 : -  
5034 : \*  
5035 : /  
5036 : goto  
5037 : gotoF  
5038 : param  
5039 : era  
5040 : retorno función  
5041 : retorno lista  
5042 : gosub

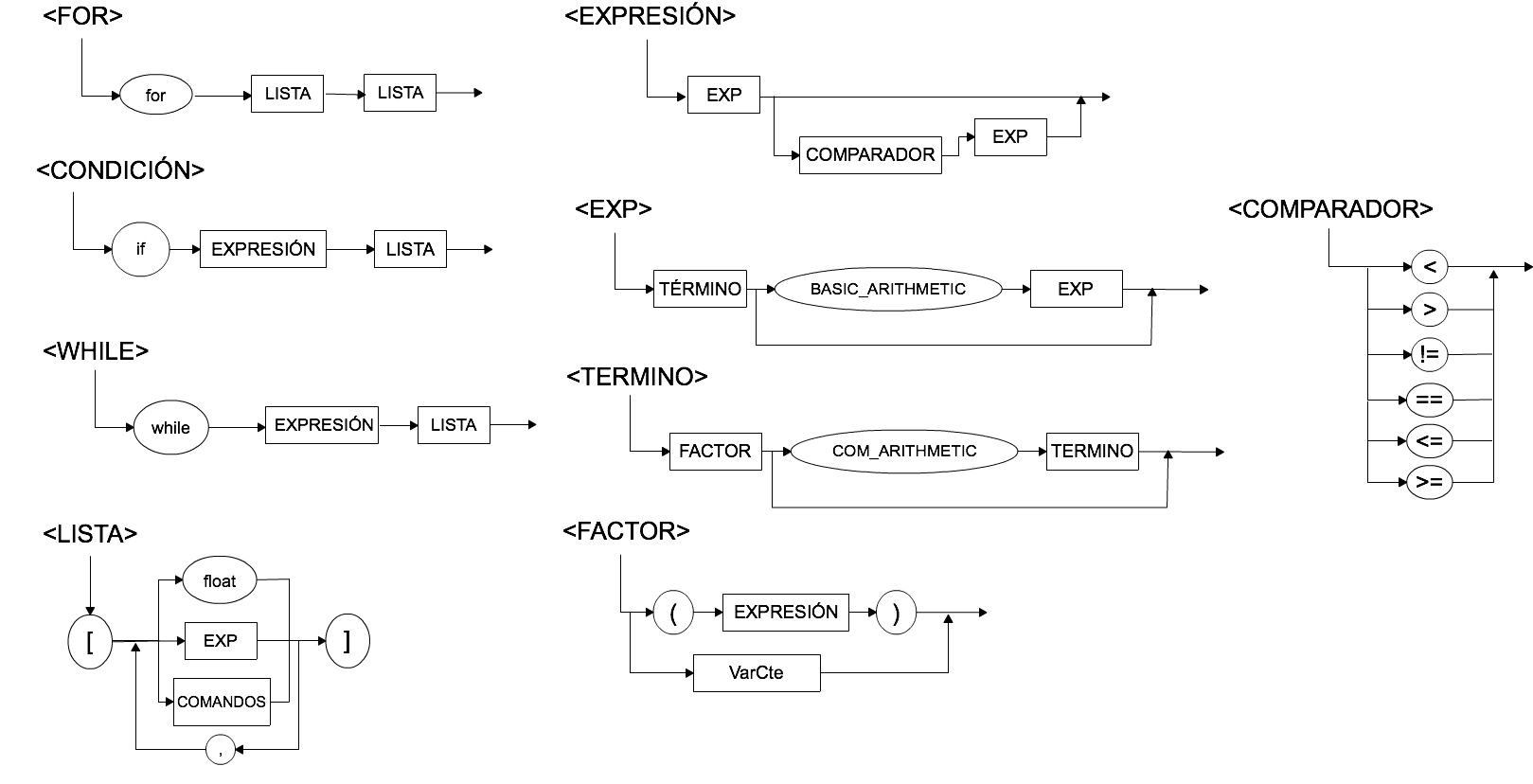
* + **Diagramas de sintaxis**

****

****

****

****

****

* + **Breve Descripción de cada una de las acciones semánticas y de código**

Graphbot únicamente utiliza dos tipos de variables: variables tipo flotante y variables tipo lista. Las acciones semánticas son para verificar que no se utilicen variables tipo lista en expresiones y variables tipo flotante en ciclos. No contamos con conversiones y equivalencias de tipo ya que solo manejamos dos tipos de variables indistintas.

Para cada procedimiento se comprueba el número de parámetros permitidos y se compara con los parámetros enviados durante la llamada a función. De ser diferente, genera un error en compilación.

* **Descripción detallada del proceso de Administración de Memoria usado en la compilación**

dirProc.h

El directorio de procedimientos se manejó mediante la clase llamada dirProcs. Se construyó una estructura de datos en C llamada procs (procedimientos) que contiene los campos id, nombre, dirI (dirección inicial), temp (número de temporales), numParam (número de parámetros), varFlotante (número de variables flotantes), varLista (número de variables tipo lista) y tv (su respectiva tabla de variables). La clase implementa una hash table de procedimientos, además de contener distintos métodos de lectura y escritura, los cuáles son enlistados a continuación:

dirProcs();

int hash(int &id);

int rehash(int &id);

int add\_proc(procs &d);

bool find\_proc(string &name);

int num\_Param(string &name);

int num\_varLista(string &name);

int num\_varFlotante(string &name);

int get\_dirI(string &name);

int get\_tmp(string &name);

void output\_proc();

tablaVar.h

Para la construcción de las tablas de variables se utilizó otra clase llamada tablaVariable. De igual manera se realizó una estructura de datos llamada data que contiene la información que debe tener la tabla de variables, es decir, id, nombre, tipo, dirV (dirección virtual), dirI (dirección inicial) y tam (tamaño). La clase implementa también una hash table donde se almacenan las variables y su información para su posterior anexo a un procedimiento. Los métodos de lectura y escritura de nuestra clase tablaVariable son los siguientes:

tablaVariables();

int hash(int &id);

int rehash(int &id);

int add\_var(data &d);

int remove\_var(string &name);

void remove\_all();

int getid\_var(string &name);

bool find\_var(string &name);

int find\_type(string &name);

int getdirI(string &name);

int getdirV(string &name);

int get\_tam(string &name);

void output();

**Representación Gráfica**

**Directorio de Procedimientos**

**Tabla de   
  
Variables**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **id** | **nombre** | **dirI** | **tmp** | **numParam** | **varFlotante** | **varLista** | **tv** |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **id** | **nombre** | **tipo** | **dirV** | **dirI** | **tam** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Generador.h

La creación de cuádruplos se hace mediante el uso de la clase Generador. La clase cuenta con las variables temporalActual, variablesFlotanteActual, variableListaActual y cont\_cuadruplos. Dichas variables ayudan al manejo de la asignación de temporales, variables y cuádruplos. Además, la clase Generador cuenta con tres pilas privadas para el manejo de saltos, operadores y operandos. Se implementaron métodos privados para la manipulación de pilas ya que en C, la estructura stack tiene dos métodos diferentes para sacar el último elemento de una pila y otro para borrarlo. En la clase Generador se implementó que ambas acciones se hagan al mismo tiempo con un solo método.

Los métodos públicos son todos orientados a la escritura de los cuádruplos en un archivo de texto que llamamos “CodigoInt.txt”. Los métodos públicos más relevantes de la clase Generador son los siguientes:

void generaEraMain(int flotantes, int listas, int temporales)

void rellena(int salto, int donde)

void start(int i)

void param(int param)

void gosub(int dir)

void era(int param, int varf, int varl, int tmp)

El método más importante es el de start. Dicho método contiene un switch que dependiendo del caso imprime en el archivo de texto su respectivo cuádruplo. Dicho método hace uso de las pilas privadas y de acuerdo a lo visto en clase va sacando los operandos y operadores de las pilas para imprimir su contenido en el archivo. Los casos se dividen en los siguientes:

1. Cuádruplos de + ó –
2. Cuádruplos de \* ó /
3. Cuádruplos para comandos de 1 parámetro
4. Cuádruplos para comandos de 2 parámetros
5. Cuádruplos para comandos de 3 parámetros
6. Cuádruplos sin parámetros que no regresan nada
7. Cuádruplos sin parámetros que regresan algo
8. Cuádruplos de comparadores
9. Cuádruplos para GotoF
10. Cuádruplos para Goto
11. Cuádruplos para GotoF de fors (caso especial)
12. Aumenta variable de control (manejo de for’s)
13. Cuádruplo para retorno de funciones
14. Cuádruplos para Goto (manejo de while’s)
15. Cuádruplos para retorno de listas

Para el manejo de los Goto’s y GotoF’s se utilizó un carácter de escape que designamos con el símbolo de #. El método rellena, busca en el archivo cada símbolo # y lo remplaza con el último elemento en la pila de saltos.

Los demás métodos generaEraMain, param, era, gosub, reciben parámetros específicos una vez contabilizados el número de variables tipo flotante, tipo lista, las temporales, los parámetros para así poder generar los cuádruplos.

Globalizador.h

La clase Globalizador básicamente se hace cargo del manejo de constantes. Cada vez que se encuentra una constante se manda llamar a esta clase que le asigna una dirección virtual dentro de 1000 a 1999. Asimismo cuenta con un método que una vez terminada la compilación del programa Graphbot, imprime en un archivo llamado constNum.txt toda la tabla de constantes con sus respectivas direcciones virtuales para que pueda ser leído en ejecución.

En resumen, la administración de memoria en compilación se va generando para cada función y para el programa principal main mediante el guardado de la cantidad de variables y/o parámetros que necesitará para su desarrollo. Utilizamos las variables *params*, *varsFlotante* y *varsLista* para ir contando cuantos parámetros, variables tipo flotante y variables tipo lista necesitamos reservar. Una vez terminada la declaración de la función se agrega esta información al directorio de procedimientos y al término de la compilación se pasa esta información al archivo de código intermedio para su mas tarde ejecución en la máquina virtual.

**d) Descripción de la Máquina Virtual**

* **Equipo de Computo**

La máquina virtual se desarrolló en computadoras portátiles con OS X Versión 10.9.1. Se utilizó además tanto para el compilador como para la máquina virtual la versión 4.2.1 de g++ además de la librería de gráficos OpenGL, la librería SOIL para la carga de texturas, la librería OpenAL para los sonidos y la librería WX Widgets para la implementación de la línea de comandos y la ejecución de programas.

* **Descripción detallada del proceso de Administración de Memoria en ejecución**

Cargador.h

La clase Cargador se encarga de guardar las constantes leídas del archivo constNum.txt en una estructura global para su fácil acceso dentro de la máquina virtual.

Memoria.h

La clase Memoria contiene 3 listas para temporales, variables de tipo flotantes y variables de tipo lista. Al comienzo de la ejecución, se crea una nueva memoria para el programa main de acuerdo a los parámetros asignados en el código intermedio. Si dentro del main se manda llamar funciones, una vez que se llama cada función se crea una nueva memoria y la anterior es guardada dentro de una pila de memorias. La nueva memoria es asignada como memoria actual y se deshecha una vez que se haya terminado de ejecutar la función. Al terminar, se saca de la pila de memorias, la memoria anterior y se le asigna a la memoria actual.

La clase memoria cuenta con métodos para asignar y acceder a las variables tomando en cuenta sus respectivos offsets, además de contar con métodos para crear y borrar memoria.

**Memoria**

|  |  |  |
| --- | --- | --- |
| Temporales | Variables Flotantes | Variables Lista |
|  |  |  |

|  |
| --- |
|  |
| … |
|  |

4999

4000

3999

3000

2000

|  |
| --- |
|  |
| … |
|  |

|  |
| --- |
|  |
| … |
|  |

2999

Memoria4

Memoria3

Memoria2

Memoria1

**Pila de Memorias**

MemoriaActual

Cuadruplos.h

La clase Cuadruplos, en ejecución, lee todos los cuádruplos almacenados en el archivo CodigoInt.txt y los acomoda en una estructura que divide cada cuádruplo en los siguientes datos: cuádruplo (número del cuádruplo), comando (codificación del comando), opdo1, opdo2 y opdo3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **cuáduplo** | **comando** | **opdo1** | **opdo2** | **opdo3** |
|  |  |  |  |  |
|  |  |  |  |  |

**Representación Gráfica**

**Cuádruplos**

**e) Pruebas del Funcionamiento del Lenguaje**

* **Codificación de la pruebas**

espiral.gbot

// Programa que dibuja una espiral en pantalla

program main

setbackgroundtxt 1.0 // Utiliza textura graffiti

save i 10.0

for [i, 110.0, 10.0][

move i, // Mueve el robot una distancia i

rightturn 90.0 // Vuelta a la derecha

]

end

* **Resultados por el código intermedio y por la ejecución**

**CodigoInt.txt**

11

1. 5039 1 0 2

2. 5036 3

3. 5021 1000

4. 5022 1001 3000

5. 5028 3000 1002 2000

6. 5037 2000 12

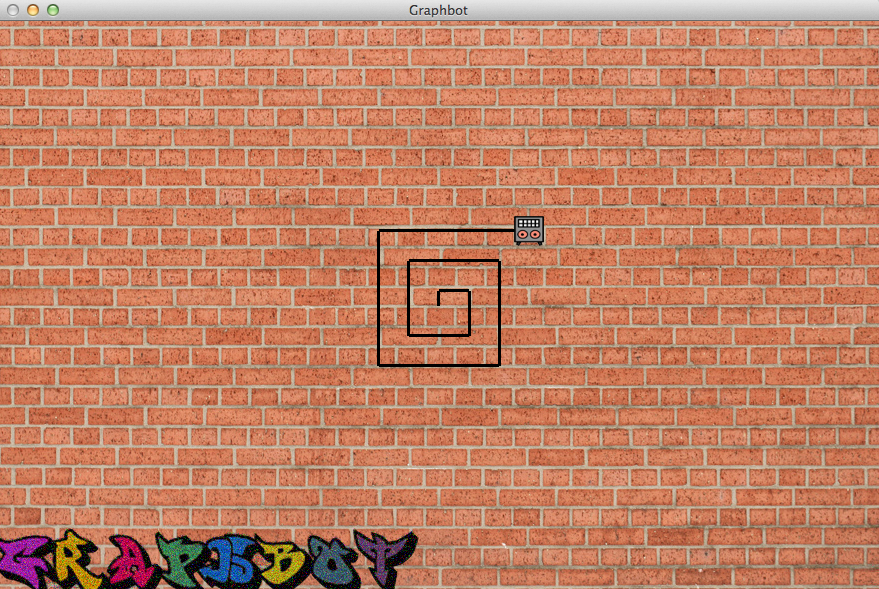
7. 5012 3000

8. 5019 1003

9. 5032 1001 3000 2001

10. 5022 2001 3000

11. 5036 5

**Ejecución**

**Codificación de la prueba**

figuras.gbot

// Dibuja un octagono

function octagon

save i 0.0

for [i, 8.0, 1.0][

move 50.0,

rightturn 45.0

]

end

// Dibuja un asterisco

function asterisk

save j 0.0

for [j, 20.0, 1.0][

move 80.0,

rightturn 180.0,

move 80.0,

rightturn 18.0  
 ]

end

program main

setbackgroundtxt 2.0 // Utiliza textura de pizarrón

setcolor 1.0 1.0 1.0

asterisk

save i 0.0

for[i, 4.0, 1.0][

octagon,

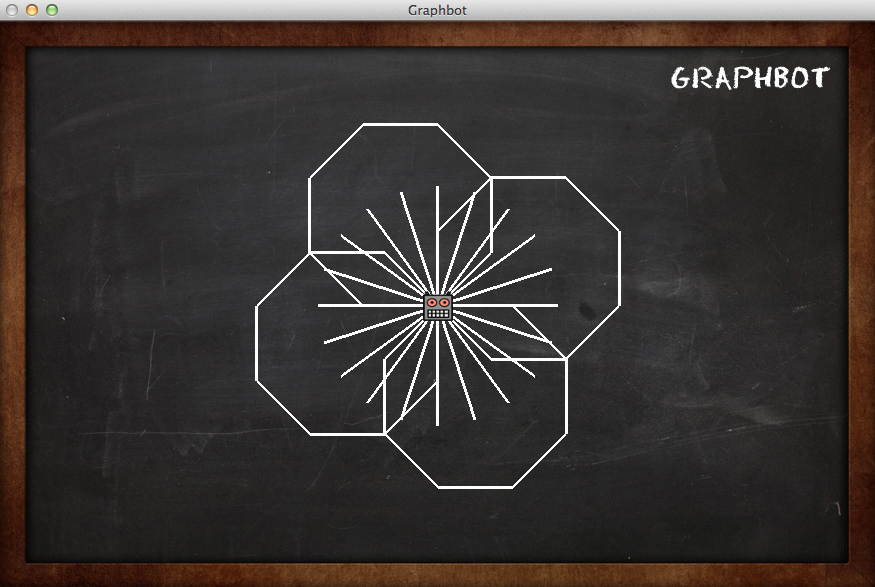
rightturn 90.0  
]

end

* **Resultados por el código intermedio y por la ejecución**

**CodigoInt.txt**

35  
1. 5039 1 0 2  
2. 5036 23  
3. 5022 1000 3000  
4. 5028 3000 1001 2000  
5. 5037 2000 11  
6. 5012 1003  
7. 5019 1004  
8. 5032 1002 3000 2001  
9. 5022 2001 3000  
10. 5036 4  
11. 5040  
12. 5022 1000 3000  
13. 5028 3000 1005 2000  
14. 5037 2000 22  
15. 5012 1006  
16. 5019 1007  
17. 5012 1006  
18. 5019 1008  
19. 5032 1002 3000 2001  
20. 5022 2001 3000  
21. 5036 13  
22. 5040  
23. 5021 1009  
24. 5024 1002 1002 1002  
25. 5039 1 0 2  
26. 5042 12  
27. 5022 1000 3000  
28. 5028 3000 1010 2000  
29. 5037 2000 36  
30. 5039 1 0 2  
31. 5042 3  
32. 5019 1011  
33. 5032 1002 3000 2001  
34. 5022 2001 3000  
35. 5036 28

**Ejecución**

* **Codificación de la prueba**

// Función que dibuja un corazón dadas las coordenadas x y

function heart x, y

save i 0.0

while i < 2.0 [

home,

setpos x y,

save ocho 0.0,

save seis 0.0,

save nov 0.0,

for[ocho, 8.0, 1.0][

if i == 0.0 [rightturn 45.0],

if i == 1.0 [leftturn 45.0],

for[seis, 6.0, 1.0][

for[nov, 90.0, 1.0][

move 1.0,

if i == 0.0 [rightturn 2.0],

if i == 1.0 [leftturn 2.0]

],

if i == 0.0 [rightturn 45.0],

if i == 1.0 [leftturn 45.0]

]

],

save i i+1.0

]

// Dibuja las dos líneas debajo de las 2 curvas que

// hacen el corazón

leftturn 135.0

move 60.0

leftturn 92.0

move 60.0

hide

end

program main

setbackgroundtxt 3.0 // Asigna la textura 3 (Libreta)

setpensize 5.0 // Hace más gruesa la línea

setcolor 1.0 0.0 0.0 // Color rojo

heart(0.0 0.0) // Dibuja corazón al centro

setcolor 1.0 0.0 1.0 // Color rosado

heart(0.0-200.0 100.0) // Dibuja corazón a la izquierda

setcolor 1.0 1.0 0.0 // Color amarillo

heart(200.0 100.0) // Dibuja corazón a la derecha

home

rightturn 180.0

penup

move 30.0

rightturn 180.0

show

end

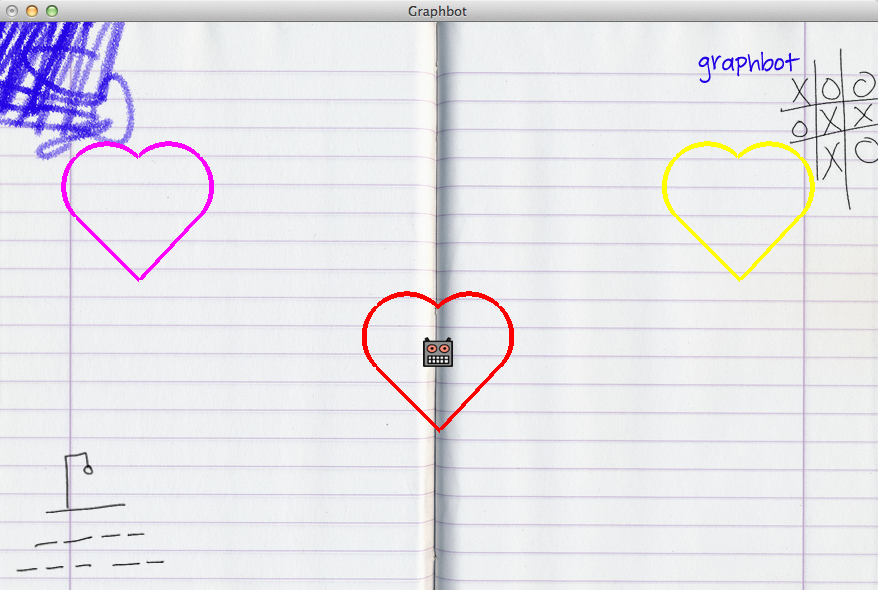
* **Resultados por el código intermedio y por la ejecución**

**CodigoInt.txt**

77  
1. 5039 0 0 1  
2. 5036 54  
3. 5022 1000 3002  
4. 5028 3002 1001 2000  
5. 5037 2000 48  
6. 5003  
7. 5023 3001 3000  
8. 5022 1000 3003  
9. 5022 1000 3004  
10. 5022 1000 3005  
11. 5028 3003 1002 2001  
12. 5037 2001 45  
13. 5026 3002 1000 2002  
14. 5037 2002 16  
15. 5019 1004  
16. 5026 3002 1003 2003

17. 5037 2003 19  
18. 5020 1004  
19. 5028 3004 1005 2004  
20. 5037 2004 42  
21. 5028 3005 1006 2005  
22. 5037 2005 33  
23. 5012 1003  
24. 5026 3002 1000 2006  
25. 5037 2006 27  
26. 5019 1001  
27. 5026 3002 1003 2007  
28. 5037 2007 30  
29. 5020 1001  
30. 5032 1003 3005 2008  
31. 5022 2008 3005  
32. 5036 21  
33. 5026 3002 1000 2009  
34. 5037 2009 36  
35. 5019 1004  
36. 5026 3002 1003 2010  
37. 5037 2010 39  
38. 5020 1004  
39. 5032 1003 3004 2011  
40. 5022 2011 3004  
41. 5036 19  
42. 5032 1003 3003 2012  
43. 5022 2012 3003  
44. 5036 11  
45. 5032 3002 1003 2013  
46. 5022 2013 3002  
47. 5036 4  
48. 5020 1007  
49. 5012 1008  
50. 5020 1009  
51. 5012 1008  
52. 5001  
53. 5040  
54. 5021 1010  
55. 5016 1011  
56. 5024 1003 1000 1000  
57. 5039 6 0 14  
58. 5038 1000 3000  
59. 5038 1000 3001  
60. 5042 361. 5024 1003 1000 1003  
62. 5033 1000 1012 2000  
63. 5039 6 0 14  
64. 5038 1013 3000  
65. 5038 2000 3001  
66. 5042 3  
67. 5024 1003 1003 1000  
68. 5039 6 0 14  
69. 5038 1013 3000  
70. 5038 1012 3001  
71. 5042 3  
72. 5003  
73. 5019 1014  
74. 5017  
75. 5012 1015  
76. 5019 1014  
77. 5000

**Ejecución**

****

**f) Listados Documentados del Proyecto**

**graphbot.y**

83 //Grammar rules

84 graphbot:

85 graph programa {

86 // Imprime el directorio de procedimientos con sus respectivas tablas de variables

87 directorio.output\_proc();

88 globalizador.toFile();

89 cout<<"Compilación Exitosa"<<endl;

90 }

91 ;

92

93

94 graph: /\*empty\*/

95 | funcion graph

96 ;

97

Al acabar el proceso de compilación y llegar a esta parte de la gramática, el compilador imprime un mensaje de éxito y además incluye el directorio de procedimientos con sus respectivas tablas de variables.

210 | RW\_SAVE ID variable {

211 //cout<<"Matched <SAVE> a ID: "<<$2<<endl;

212 string id = $2;

213 // Si la misma variable se vuelve a asignar, se borra la anterior y se mete la nueva

214 if(tv.find\_var(id)) {

215 tvar.dirV = tv.getdirV(id);

216 tv.remove\_var(id);

217 varsFlotante--;

218 }

219 // Agrega una variable a la tabla de variables

220 tvar.nombre = id;

221

222 if(tvar.tipo == 0) {

223 varsFlotante++;

224 //genera direccion virtual para variable de tipo flotante

225 if(tvar.dirV == 0)

226 tvar.dirV = generador.variablesDeAvail(true);

227 //genera cuadruplo save flotante

228 generador.pushPOper(5022);

229 generador.pushPilaO(tvar.dirV);

230 generador.start(4);

231 }else if(tvar.tipo == 1){

232 varsLista++;

233 //genera direccion virtual para variable tipo lista

234 if(tvar.dirV == 0)

235 tvar.dirV = generador.variablesDeAvail(false);

236 //genera cuadruplo save lista

237 generador.rellena\_save(generador.popPSaltos(), tvar.dirV);

238 }

239 //agrega variable a tabla de variables

240 tv.add\_var(tvar);

241 tvar.dirV = 0;

242

243 }

Al recibir el comando *save* encargado de guardar variables, primero se verifica si la variable ya había sido asignada con anterioridad y de ser así, borra la variable anterior, mantiene su misma dirección virtual y sobrescribe su valor. Si no encuentra la variable en la tabla y le asigna una dirección virtual nueva. En la línea 222, verifica si la variable es de tipo flotante, para posteriormente en la línea 228 hacer push en la POper (pila de operadores) de la codificación del comando *save* y en la PilaO (pila de operandos) la dirección virtual asignada a esa variable. La línea 230 utiliza el método start de nuestra clase Generador con parámetro 4, para la generación de cuádruplos con 2 parámetros, es decir, el caso *save*. La línea 240 agrega la variable a su tabla de variables.

589 void yyerror(const char \*s) {

590 cout << "BANG, error de parser: "<<s<<" en la línea "<<yylineco<< endl;

591 // might as well halt now:

592 exit(-1);

593 }

594

595 // Función que maneja los errores

596 void errores(int i, string val) {

597 switch (i) {

598 // La función ya se encuentra en el directorio de procedimientos

599 case 1:

600 cout << "Función "<< val << " ya declarada." << endl;

601 exit(-1);

602 break;

603

604 // La variable no ha sido declarada

605 case 2:

606 cout << "Variable "<< val << " no declarada." << endl;

607 exit(-1);

608 break;

609

610 // La función no esta declarada

611 case 3:

612 cout << "Función " << val << " no existe." << endl;

613 exit(-1);

614 break;

615

La función yyerror muestra en pantalla los errores de léxico o sintaxis que suceden en la pantalla y la línea donde se genera el error. La función en la línea 596 maneja un switch que contiene los distintos errores que pueden ocurrir en compilación tales como *Función ya declarada*, *Variable no declarada*, *Función x no existe.* Entre otros errores.

**Generador.h**

237 void start(int i){

238 ofstream fileout;

239 //fileout.open("MaquinaVirtual/Build/Products/Debug/CodigoInt.txt", ios::app);

240 fileout.open("CodigoInt.txt", ios::app);

241

242 switch(i) {

243 case 1:

244 // 5.- Si pop(POper) == '+' o '-'

245 if(!POper.empty() && PilaO.size() >=2){

246 int posible\_operador = POper.top();

247 if((posible\_operador == 5032) || (posible\_operador == 5033)){

248 int opdo2 = popPilaO();

249 int opdo1= popPilaO();

250 //sacar posible\_operador de la pila

251 POper.pop();

252 //genera(posible\_operador,opdo1,opdo2, "", 0);

253 fileout << cont\_cuadruplos << ". "<< posible\_operador << " " << opdo1 << " " << opdo2 <<" "<<tempDeAvail()<<endl;

254 //actualiza el contador de cuádruplos

255 cont\_cuadruplos++;

256 }

257 }

258 break;

259

En el caso 1 dentro del switch en la función start, se manejan los cuádruplos para las operaciones de sumas y restas. La línea 245 verifica que la pila de operadores no se encuentre vacía y que la pila de operandos sea mayor o igual a 2. La siguiente línea toma el primer operador de la pila de operadores (sin sacarlo) y checa que efectivamente se trate de una suma o una resta. De ser así, saca el operando 1 y operando 2 de la pila de operandos y también el operador de la pila de operador. En la línea 253 se imprime en el archivo CodigoInt.txt el cuádruplo de manera *num\_cuádruplo. operador operando1 operando2*. La línea 255 suma 1 al contador de cuádruplos.

**Memoria.h**

36 void Memoria::nueva(int temp, int flotantes, int listas){

37 Memoria::temporales = new float[temp];

38 Memoria::variablesFlotantes = new float[flotantes];

39 Memoria::variablesListas = new int[listas];

40 }

41

42 void Memoria::destruye(){

43 delete[] Memoria::temporales;

44 delete[] Memoria::variablesFlotantes;

45 delete[] Memoria::variablesListas;

46 }

La función nueva crea tres listas que manejan las temporales, las variables flotantes y las variables listas. En la línea 42 la función destruye libera la memoria de nuestro programa.

**Cuadruplos.h**

36 void Cuadruplos::set(int cuadruplo, int comando, int opdo1, int opdo2, int opdo3){

37 struct cuadruplo cuadAPonerEnLista;

38 cuadAPonerEnLista.cuadruplo = cuadruplo;

39 cuadAPonerEnLista.comando = comando;

40 cuadAPonerEnLista.opdo1 = opdo1;

41 cuadAPonerEnLista.opdo2 = opdo2;

42 cuadAPonerEnLista.opdo3 = opdo3;

43

44 listaDeCuadruplos[cuadruplo-1] = cuadAPonerEnLista;

45 }

La línea 37 muestra la estructura que se creó para el manejo de cuádruplos en ejecución. Dicha estructura guarda el número de cuádruplo, el comando, el operando 1, operando 2 y operando 3 que contenga el cuádruplo. Si en la sección de operandos no recibe nada, la estructura le asigna un 0.

**g) Bibliografía**

Harvey, B. (2008, 09 03). *Berkeley logo 6.0* . Retrieved from     http://www.cs.berkeley.edu/~bh/docs/html/usermanual.html

**MANUAL DE USUARIO**

**Graphbot 1.0**

**Gráficas**

*move*

move distancia

*donde distancia es un float, una expresión aritmética o una variable tipo flotante*

mueve al robot hacia adelante una distancia especificada

*rightturn*

rightturn grados

*donde grados es un float, una expresión aritmética o una variable tipo flotante*

mueve al robot a la derecha de acuerdo al ángulo especificado medido en grados

*leftturn*

leftturn grados

*donde grados es un float, una expresión aritmética o una variable tipo flotante*

mueve al robot a la izquierda de acuerdo al ángulo especificado medido en grados

*setpos*

setpos X Y

*donde X y Y son un float, una expresión aritmética o una variable tipo flotante*

mueve al robot a una posición absoluta en el canvas, el comando recibe como entrada las coordenadas X y Y

*setx*

*setx coorx*

*donde coorx es un float, una expresión aritmética o una variable tipo flotante*

mueve al robot horizontalmente de su posición antigua a una nueva coordenada X

*sety*

*sety coory*

*donde coory es un float, una expresión aritmética o una variable tipo flotante*

mueve al robot verticalmente de su posición antigua a una nueva coordenada Y

*setcolor*

*setcolor r g b*

*donde r, g, y b son un float, una expresión aritmética o una variable tipo flotante*

establece un color a la pluma de acuerdo al color generado por los tres valores flotantes que le siguen, estos siendo valores entre 0.0 y 1.0

*setpensize*

*setpensize tamaño*

*donde tamaño es un float, una expresión aritmética o una variable tipo flotante*

establece el espesor de la pluma, la entrada es un número entero

*setbackground*

*setbackground r g b*

*donde r, g, y b son un float, una expresión aritmética o una variable tipo flotante*

pinta el fondo del canvas de acuerdo al color generado por los tres valores flotantes que le siguen, estos siendo valores entre 0.0 y 1.0

*setbackgroundtxt*

*setbackgroundtxt textura*

*donde textura es un float [1.0, 2.0, 3.0, 4.0]*

pinta el fondo del canvas de acuerdo a la textura elegida

*show*

vuelve al robot visible

*hide*

vuelve al robot invisible

*clean*

borra todo lo dibujado en el canvas

*home*

regresa al robot al centro del canvas

**Queries**

*getcolorr*

regresa el valor de R

*getcolorg*

regresa el valor de G

*getcolorb*

regresa el valor de B

*getpensize*

regresa el tamaño actual del espesor de la pluma

*getx*

regresa la coordenada x en la que se encuentra el robot

*gety*

regresa la coordenada y en la que se encuentra el robot

**Estructuras de Control**

*function*

*function nombrefunción parametros*

*comandos*

*end*

*donde parametros es una lista de ids, comandos es una lista de comandos o una variable del tipo lista de comandos*

genera una función local al programa

*save*

*save nombrevariable valor*

*donde valor es un float, una expresión aritmética, o una lista*

guarda una variable nueva con su respectivo nombre y valor

*for*

*for [var final incremento] instrucciones*

*donde instrucciones es una lista de comandos o una variable del tipo lista de comandos*

repite una serie de instrucciones de acuerdo al valor inicial establecido, el incremento y el valor límite

*if*

*if condición instrucción*

*donde instrucción es una lista de comandos o una variable del tipo lista de comandos*

ejecuta una instrucción si y solo si la condición establecida es verdadera

*while*

*while condición instrucciones*

*donde instrucciones es una lista de comandos o una variable del tipo lista de comandos*

ejecuta una instrucción continuamente siempre y cuando la condición establecida sea verdadera

**Misceláneas**

*playmusic*

*playmusic*

reproduce música durante la ejecución de la aplicación

*stopmusic*

*stopmusic*

detiene la música

*penup*

*penup*

levanta la pluma del canvas y permite no dibujar en el mismo mientras se ejecutan otras instrucciones

*pendown*

*pendown valor*

apunta la pluma al canvas y permite dibujar en el mismo mientras se ejecutan otras instrucciones

*about*

*about*

muestra información de los creadores de Graphbot