Tecnológico de Monterrey

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

Documentación Final

“QUETZAL”

Diseño de compiladores (Ago 19 Gpo 2)

Ing. Héctor Gibrán Ceballo Cancino
Ing. Elda Guadalupe Quiroga González

Lizzie Marielle Guajardo Mozo
A00818258

Alejandro González Valles
A00818616

Monterrey, N.L. México 27 de noviembre del 2019

Índice del Documento

Descripción del Proyecto **3**
Visión, Objetivos y Alcance del Proyecto 3
Visión 3
Objetivos 3
Alcance 3
Análisis de Requerimientos y Casos de Uso generales 4
Análisis de Requerimientos 4
Casos de Uso Generales 11
Descripción de los principales Test Cases 23
Descripción del proceso general 23
Proceso General 23
Bitácoras generales 23
Reflexiones 25

Descripción del Lenguaje **26**
Nombre del Lenguaje 26
Descripción genérica de las principales características del lenguaje 26
Listado de errores que pueden ocurrir, tanto en compilación como en ejecución 26

Descripción Del Compilador **28**
Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto. 28
Descripción del Análisis de Léxico. 28
Patrones de Construcción de los elementos principales 28
Enumeración de los "tokens" del lenguaje y su código asociado 29
Descripción del Análisis de Sintaxis 31
Gramática Formal empleada para representar las estructuras sintácticas 31
Descripción de Generación de Código Intermedio y Análisis Semántico 33
Código de operación y direcciones virtuales asociadas a los elementos del código 33
Diagramas de Sintaxis con las acciones correspondientes 34
Tabla de consideraciones semánticas 47
Descripción detallada del proceso de Administración de Memoria usado en la compilación. 48
Especificación gráfica de CADA estructura de datos usada (Dir.Funciones, Tablas de Var's, Cuádruplos, etc...) 48

Descripción de la Máquina Virtual **49**
Equipo de cómputo, lenguaje y utilerías especiales usadas (en caso de ser diferente que el compilador). 49
Descripción detallada del proceso de Administración de Memoria en ejecución (Arquitectura) 49
Especificación gráfica de CADA estructura de datos usada para manejo de scopes (Memoria Local, global, etc..) 49
Asociación hecha entre las direcciones virtuales (compilación) y las reales (ejecución) 50
Pruebas del Funcionamiento del Lenguaje 50
Incluir pruebas que "comprueben” el funcionamiento del proyecto 50
Codificación de la prueba (en su lenguaje) 50
Resultados arrojados por la generación de código intermedio y por la ejecución 55
Listados Perfectamente Documentados del Proyecto 67
Incluir comentarios de Documentación, es decir: para cada módulo, una pequeña explicación de qué hace, qué parámetros recibe, qué genera como salida y cuáles son los módulos más importantes que hacen uso de él. 67
Dentro de los módulos principales se esperan comentarios de Implementación, es decir: pequeña descripción de cuál es la función de algún estatuto que sea importante de ese módulo. 67

Manual de Usuario **72**

Documentación Final “Quetzal”

1. Descripción del Proyecto

A continuación se expondrá el propósito, objetivo general y alcance que constituyen la descripción del compilador "Quetzal" desarrollado por Alejandro González Valles y Lizzie Marielle Guajardo Mozo, alumnos de la materia "Diseño de compiladores Gpo" impartida durante el periodo académico Agosto Diciembre 2019.

1.1. Visión, Objetivos y Alcance del Proyecto

1.1.1. Visión

Nuestro proyecto tiene como visión aplicar todo el conocimiento obtenido a través de la carrera profesional para desarrollar una herramienta que sea capaz de realizar las tareas básicas de un lenguaje de programación vinculado a un paquete de utilidades para el manejo y manipulación de colores en archivos de imagen.

1.1.2. Objetivos

El objetivo del lenguaje Quetzal es proporcionar al usuario una forma sencilla y legible de programar imperativamente. Quetzal permite el manejo de tipos de datos básicos como enteros, números flotantes y strings, también de un tipo de dato especial para color en formato hexadecimal que, en conjunto con funciones especiales, permitirá manipular los pixeles de un archivo de imagen para producir versiones alternas según filtros de color, escala de grises o modificación directa de la matriz con el conjunto de valores hexadecimales.

La herramienta busca ser de utilidad para estudiantes principiantes de programación, en particular educación media y media superior, que se estén introduciendo a la rama de gráficas computacionales, de modo que el manejo de la herramienta no se vea comprometido aun si no cuentan con conocimiento previo sobre teoría de color.

1.1.3. Alcance

El proyecto está contemplado para ser implementado durante el periodo académico de Agosto-Diciembre 2019, aproximadamente 4 meses. La entrega final tiene fecha para el Miércoles 27 de noviembre, siendo los entregables: el compilador terminado, su máquina virtual, la documentación final y el manual de usuario.

Los elementos básicos que el Lenguaje de Programación Quetzal incluye son:

- Estatutos: Asignaciones, Condiciones (if), Ciclos (While), Lectura (Read), Escritura (Print)
- Expresiones Aritméticas y Lógicas.
- Módulos, incluyendo parámetros, variables locales y globales.

- Elementos estructurados: Arreglos de varias dimensiones

Haciendo uso de estos elementos, el lenguaje le permitirá al usuario crear y ejecutar funciones para modificar imagenes locales con el uso de arreglos y nuestra funciones especiales, las cuales se definirán posteriormente en el documento.

1.2. Análisis de Requerimientos y Casos de Uso generales

1.3. Análisis de Requerimientos

El lenguaje de Programación Quetzal cumple con los siguientes requerimientos

1. Definición y asignación de variables
    a. La definición de una variable debe cumplir con el formato: la palabra reservada “define” seguido del tipo de la variable y el nombre (id)
    b. Es posible asignar un valor a una variable directamente desde su asignación
    c. El nombre de una variable deberá de empezar con un letra minúscula seguida de caracteres en minúscula o mayúscula
    d. No se puede cambiar el tipo de una variable ya definida
    e. Una variable definida sin asignársele un valor toma un valor por defecto según su tipo.
    f. La definición de variables puede ser a nivel global (fuera de una función*)* o a nivel local (dentro de una función)
    g. Se pueden declarar varias variables del mismo tipo al mismo tiempo si estas son del mismo tipo, los ids se separan por comas “,”.
    h. No se pueden acceder variables locales fuera de la instancia en la que fue definida
    i. Se puede asignar un valor a una variable previamente definida invocando su nombre seguido del signo de igual “=” y el valor deseado, siempre y cuando coincida con el tipo de la variable.

```
define int iMyInt;
define color cRed, cWhite, cBlack;
define float fMyFloat=12.5;
```

2. Líneas de código delimitadas por el símbolo punto y coma “;”
3. Tipos de variables
    a. int: número entero positivo o negativo.
        i. Valor por defecto: 0
    b. float: número decimal positivo o negativo.
        i. Valor por defecto: 0.0
    c. bool: True o False.
        i. Valor por defecto: False
    d. color: un número hexadecimal de 6 dígitos con el formato “#FFFFFF”.
        i. Valor por defecto: #000000

e. tag: secuencia de caracteres.
    i. No puede ser declarado ni acepta que se le hagan operaciones.
    ii. Utilizable únicamente en prints y para funciones especiales.

4. Definición de funciones con retorno de variable o sin retorno (void)
    a. El encabezado de la función sigue el formato: palabra reservada *“func”* seguido del tipo de retorno, el nombre de la función y un par de paréntesis con los parámetros que se recibe.
    b. El uso de parámetros dentro de una función se hace dentro de paréntesis “(...)” siendo los parámetros definidos como “*tipo* nombre” y una coma “,” separándolos.
    c. Funciones que no reciben parámetros deberán incluir en el encabezado los paréntesis vacíos “()”.
    d. El retorno de la función se construye con la palabra reservada *“return”* seguido de la variable o el valor de retorno correspondiente al tipo de la función.
    e. Funciones de tipo void deberán incluir también la palabra definida “*return”* sin ninguna variable o valor de retorno.
    f. El contenido de la función deberá de estar contenido dentro de un par de llaves “{...}” después del encabezado de la función

```
func void hola(tag nombre, int edad)
{
    print(“Hola”,nombre,” de “,edad,“años” );
}
func int sum()
{
    return 1 + 1;
}
```

5. Llamada de funciones
    a. El llamado de la función se hace escribiendo el nombre de la función seguido de los parámetros, sin la necesidad de volver a especificar los tipos.
    b. Las llamadas de función con un tipo de retorno distinto a *void* deben de ser precedidos por una variable (id) que guarde el valor de retorno de la función (el tipo del id debe de corresponder con el tipo de retorno) seguido de un signo de igual “=” y el nombre de la función
    c. Como parámetro dentro de una llamada de función se puede usar una constante, una variable, una operación aritmética, otra llamada a una función o el acceso a una casilla de arreglo.
    d. No se permite la definición de variables o funciones como parámetro de una llamada de función.

e. Para poder realizar una llamada de función, la función debe de estar previamente definida
f. El número de parámetros así como el tipo de los parámetros usados deben de coincidir con la definición de los parámetros en la definición de la función.

```
x = nombreDeFuncionConRetorno(y,z);
nombreDeFuncionSinRetornoYSinParametros();
miFunción(1+1,a[2]);
```

6. Comentarios por bloque o por línea
   a. Comentarios por bloque se especifican con el carácter de barra oblicua seguido de un asterisco para abrir el comentario “/*” y para cerrarlos otro asterisco y una barra oblicua “*/”.

```
/* Esto es un comentario
de tipo bloque */
```

   b. Comentarios por línea se especifican con dos caracteres de barra oblicua continuos “//”

```
//Esto es un comentario de línea
```

7. Lectura
   a. Es posible leer valores desde la consola con la función predefinida de “read” la cual sigue el formato: palabra reservada “read” que recibe entre paréntesis “(...)” como parámetro la variable (id) en donde se guardará el valor leído.
   b. Sólo se puede leer una variable por cada “read”
   c. La variable ingresada como parámetro debe de estar previamente definida.
   d. El valor recibido debe de coincidir con el tipo de la variable del parámetro.

```
read(x);
```

8. Escritura
   a. Es posible escribir valores a la consola con la función predefinida de “print” la cual sigue el formato: palabra reservada “read” que recibe entre paréntesis “(...)” como parámetros las variables, constantes, tags, accesos a índices de arreglos o llamadas a funciones que regresen valores, separadas por comas “,”.
   b. Por defecto, el print hace un salto de línea al final de la impresión.

```
print(“Mi nombre es: ”,nombre, “tengo ”,edad[i], “años”)
“Mi nombre es Lizzie tengo 22 años”
```

c.

9. Operaciones aritméticas
    a. Es posible realizar operaciones aritméticas básicas de tipo suma (+), resta (-), multiplicación (*) y división (/).
    b. Los operandos de las operaciones aritméticas deben ser de tipos compatibles según se define en el cubo semántico.
    c. Los operandos deben de estar separados por un operador (Ej. 5 + 5)
    d. Tales operandos pueden ser constantes, variables, llamadas de funciones que retornan valores o accesos de índices a arreglos.

```
5 + 5
x - suma(4,3)
arr[5] * x
```

10. Operaciones Lógicas
    a. Es posible armar operaciones lógicas con el uso de los siguiente operadores: es igual a (==), mayor que (>), menor que (<), es diferente a (!=), mayor o igual a (>=), menor o igual a (<=), o (||), y (&&).
    b. Los operandos de las operaciones lógicas deben ser de tipos compatibles según se define en el cubo semántico.
    c. Los operandos deben de estar separados por un operador (Ej. 5 == 5)
    d. Tales operandos pueden ser constantes, variables, llamadas de funciones que retornan valores o accesos de índices a arreglos.

```
5 > 4 && x < 10
x != suma(4,3)
arr[5] <= x
```

11. Estatutos condicionales
    a. Los estatutos condiciones “Si” deben de seguir el siguiente formato: la palabra reservada “*if*” seguida de un par de paréntesis con una expresión lógica dentro “(Expresión Lógica)”, seguida de un par de llaves que contengan el código deseado “{ ... }”
    b. Opcionalmente, el estatuto “Si no” deberá de proseguir el estatuto “if” siguiendo el formato de: palabra reservada “*else*” seguida de un par de llaves que contengan el código deseado “{...}”

```
if(condición)
{
   Código
}
else
{
   Código
}
```

12. Ciclos

a. El ciclo “mientras” se formulan con la palabra reservada “while” seguida de un par de paréntesis con una expresión lógica dentro “(Expresión Lógica)”, seguida de un par de llaves que contengan el código deseado “{...}”

```
while(condición)
{
   Código
}
```

13. Elementos estructurados: Arreglos de varias dimensiones
    a. Se definen arreglos siguiendo la estructura: palabra reservada “*define*” seguida del *tipo* del arreglo, el *nombre* del arreglo y un par de corchetes “[XX]” que contienen dentro un número entero constante que representa el tamaño de la dimensión.
    b. El número de corchetes representa el número de dimensiones
    c. La asignación del tamaño de las dimensiones es estático (No se pueden usar variables).
    d. No se pueden inicializar arreglos
    e. No se pueden hacer operaciones directas con todo un arreglo.
    f. Se accede al elemento de un arreglo, con el fin de asignar o realizar operaciones, escribiendo el nombre del arreglo ya definido, seguido de pares de corchetes “[X]” correspondientes al número de dimensiones del arreglo. Cada corchete debe contener una constante, una variable, una llamada de función con retorno o un acceso a otro arreglo que representa la posición a acceder del arreglo, siempre y cuando no exceda el tamaño dado en la definición del arreglo.

```
define int arr[5][10];
print(arr[i][suma(1+2)]);
```

14. Funciones especiales
    a. El lenguaje permite el uso de las siguientes funciones especiales:
        i. void openImg(tag,color arr[][])
            1. Recibe como parámetro la ubicación de un archivo de una imagen y lo guarda en una variable de matriz de colores.
            2. 

{{#FFFFFF,#FFFFFF,#000000,#000000,#000000,#FFFFFF,#FFFFFF,#FFFFFF,#000000,#000000,#000000,#FFFFFF,#FFFFFF},
{#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF},
{#000000,#BF1E2E,#FFFFFF,#FFFFFF,#BF1E2E,#BF1E2E,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000},
{#000000,#BF1E2E,#FFFFFF,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000},
{#000000,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000},
{#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF},
{#FFFFFF,#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF,#FFFFFF},
{#FFFFFF,#FFFFFF,#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF,#FFFFFF,#FFFFFF},
{#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF},

{#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#000000,#BF1E2E,#000000,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF},
{#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#000000,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF}}

ii. void saveImg(tag,color arr[][])
    1. Recibe como parámetro una matriz de colores y lo guarda como imagen según la ruta de archivo especificada.

iii. void grayscale(color arr[][])
    1. Recibe una matriz de colores y la convierte la imagen a escala de grises.

iv. void colorReplace(color arr[][],color,color)
    1. Toma una matriz de colores y recibe dos colores como parámetros, el primero será recoloreado por el segundo.

v. void colorFilter(color arr[][],color)
    1. Recibe una matriz de colores y un color. Esta función convierte en escala de gris todos los colores de la matriz que no sean iguales al segundo parámetro.

vi. void edgeDetection(color arr[][])
    1. Toma una matriz de colores. Esta función detecta automáticamente los bordes de los principales elementos de una imagen y los delinea.


vii. void saveScaleImg(color arr[][],int, int)

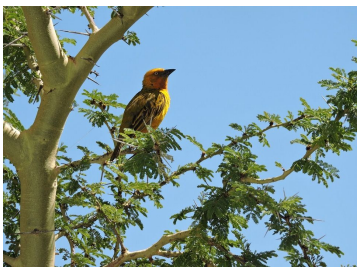
1. Toma como parámetro una matriz de colores y dos números flotantes. La imagen será escalada en x según el valor del segundo parámetro (Ej. 2 = 200%) y en y según el valor del tercer parámetro. Un número negativo escalará la imagen según la magnitud y la invertirá de forma horizontal o vertical según sea el caso.


viii. void getColorPalette(color arr[][],color arr[],int)

1. Función que recibe como parámetro una matriz de colores, un arreglo de una dimensional de colores y un número entero n. Se regresa un arreglo con n colores


ix. void colorMatchImage(color arr[][], color arr[][])

1. Función que recibe como parámetro dos matrices de colores, siendo la primer matriz la imagen fuente y la segunda la imagen de referencia. Esta función manipula los píxeles de la imagen fuente de modo que el histograma de pixeles concuerde con el histograma de la imagen de referencia.


x. void encodeSteganography(color arr[][], tag)
   1. Esta función se basa en la técnica de Esteganografía, la cual oculta un mensaje dentro de una imagen. La función recibe una imagen y un mensaje de texto y retorna la imagen con el código guardado. A simple vista la imagen no parece ser diferente a la original.

xi. void decodeSteganography(color arr[][])
   1. Esta función recibe una imagen modificada con esteganografía para obtener el mensaje encriptado que contenga. Recibe una matriz de colores e imprime el mensaje oculto

1.4. Casos de Uso Generales

ID	1.1
Nombre	Definición de variable (id)
Descripción	La definición de una variable debe cumplir con el formato: la palabra reservada “define” seguido del tipo de la variable y el nombre (id)
Precondición	Ninguna
Postcondición	Variable definida
Flujo de eventos	El usuario escribe la palabra define El usuario escribe el tipo de la variable El usuario escribe el id (primer caracter letra minúscula) El lenguaje guarda la variable en memoria
Flujo Alternativo	3. El nombre ya existe, el lenguaje despliega el mensaje: “Variable '" + newVar + "' already defined"
Variaciones	3.1.1. El usuario escribe una coma “,” 3.1.2. El usuario escribe el nombre de otro id a definir 3.2.1 El usuario asigna un valor a la variable (vease Caso de Uso 1.2)

ID	1.2
Nombre	Asignación de variable (id)
Descripción	Se puede asignar un valor a una variable previamente definida invocando su nombre seguido del signo de igual “=” y el valor deseado, siempre y cuando coincida con el tipo de la variable.
Precondición	Variable previamente definida
Postcondición	Valor de la variable actualizado
Flujo de eventos	1. El usuario manda a llamar el nombre de la variable a asignar un valor 2. El usuario escribe un signo de igual “=” 3. El usuario escribe un valor constante, de otra variable, de una llamada de función o acceso al elemento de un arreglo con retorno para asignarle a la variable 4. El lenguaje actualiza el valor de la variable
Flujo Alternativo	3. El valor asignado no es compatible con el tipo definido de la variable. El lenguaje despliega el mensaje: "Operand '{asigned}'({asignedType}) is incompatible with operand '{result}'({resultType}) with operator: {op}"

ID	2.1
Nombre	Escritura de Expresión
Descripción	Líneas de código delimitadas por el símbolo punto y coma “;”
Precondición	Ninguna
Postcondición	Línea de código añadida
Flujo de eventos	1. El usuario escribe una expresión como nueva línea de código siguiendo las reglas de gramática 2. El usuario concluye la expresión con un “;” 3. El lenguaje de programación reconoce la expresión

ID	4.1
Nombre	Definición de funciones con retorno de variable o sin retorno (void)
Descripción	El encabezado de la función sigue el formato: palabra reservada *“func”* seguido del tipo de retorno, el nombre de la función y un par de paréntesis con los parámetros que se recibe. El uso de parámetros dentro de una función se hace dentro de paréntesis “(...)” siendo los parámetros definidos como “*tipo* nombre” y una coma “,” separándolos.
Precondición	Ninguna
Postcondición	Función definida
Flujo de eventos	1. El usuario escribe la palabra “func”, el tipo de retorno y el nombre de la función 2. El usuario escribe los parámetros que la función recibe entre paréntesis siguiendo el formato: tipo, nombre de la variable y separados entre comas “,” 3. El usuario escribe el código de la función dentro de un par de llaves “{}” 4. El usuario escribe dentro del código la palabra “return” y el valor de retorno correspondiente al tipo de la función 5. El lenguaje guarda la función en memoria
Flujo Alternativo	1. El nombre de la función ya existe. El lenguaje alerta que ese nombre ya está ocupado. 4. El tipo del valor de retorno no coincide con el valor de retorno de la función, el lenguaje muestra el error correspondiente.
Variaciones	1. La función no tiene valor de retorno, se escribe “void” como tipo de retorno. 4. La función no tiene valor de retorno, se escribe “return” sin ninguna variable o valor de retorno.

ID	5.1
Nombre	Llamadas de función
Descripción	El llamado de la función se hace escribiendo el nombre de la función seguido de los parámetros, sin la necesidad de volver a especificar los tipos.
Precondición	Función previamente definida
Postcondición	Función llamada y ejecutada
Flujo de eventos	1. El usuario escribe el nombre de la función a llamar con los parámetros a pasar a la función dentro de paréntesis y separados por paréntesis. 2. El lenguaje ejecuta la función
Flujo Alternativo	1.1. Si el nombre de la función no está definido previamente, el lenguaje despliega el error. 1.2. Si el número de parámetros dados no corresponde con el número de parámetros definidos, el lenguaje despliega el error. 1.3 Si el tipo de uno de los parámetros no corresponde al tipo definido, el lenguaje marca el error correspondiente.
Variación	1. Si la función tiene un valor de retorno no void, el nombre de la función deberá ser precedido por una variable que reciba el valor de retorno y un signo de igual “=”. El lenguaje guarda el retorno de la función en la variable

ID	6.1
Nombre	Comentarios por bloque o por línea
Descripción	Comentarios utilizados para omitir partes del código intencionalmente
Precondición	Ninguna
Postcondición	Porción de código omitida
Flujo de eventos	1. El usuario escribe dos barras oblicuas (“//”) seguido de la línea de código a ignorar 2. El lenguaje ignora la línea
Variación	1. El usuario escribe una barra oblicua y un asterisco (“/*”) antes de una porción de código a ignorar seguido de un asterisco y una barra oblicua (“*/”)

ID	7.1
Nombre	Lectura
Descripción	Es posible leer valores desde la consola con la función predefinida de “read”
Precondición	Variable donde guardar lectura previamente definida
Postcondición	Valor de variable actualizada
Flujo de eventos	1. El usuario escribe la palabra “read” seguida del nombre de la variable a recibir la lectura en paréntesis 2. El usuario ejecuta el código 3. El lenguaje le pide al usuario el valor a leer en la consola 4. El usuario ingresa el valor en la consola 5. El lenguaje actualiza el valor de la variable
Flujo Alternativo	4. El tipo del valor ingresado no corresponde con el tipo de la variable definida dentro del read. El lenguaje despliega un error

ID	8.1
Nombre	Escritura

Descripción	Es posible escribir valores a la consola con la función predefinida de “print”
Precondición	Ninguna
Postcondición	Línea impresa en consola
Flujo de eventos	1. El usuario escribe la palabra “print” seguida de los elementos a imprimir dentro de paréntesis separados entre comas “,” 2. El lenguaje imprime la línea a consola

ID	9.1
Nombre	Operaciones Aritméticas
Descripción	Es posible realizar operaciones aritméticas básicas de tipo suma (+), resta (-), multiplicación (*) y división (/).
Precondición	Ninguna
Postcondición	Resultado calculado de una operación aritmética
Flujo de eventos	El usuario escribe los operandos compatibles separados por un operador El lenguaje calcula el resultado
Flujo Alternativo	1. Los operandos no son compatibles, el lenguaje arroja un error.

ID	10.1
Nombre	Operaciones Lógicas
Descripción	Es posible armar operaciones lógicas con el uso de los siguiente operadores: es igual a (==), mayor que (>), menor que (<), es diferente a (!=), mayor o igual a (>=), menor o igual a (<=), o (||), y (&&)
Precondición	Operación lógica de tipo booleana calculada
Postcondición	Valor bool resultante calculado
Flujo de eventos	1. El usuario escribe los operandos compatibles separados por un operador lógico 2. El lenguaje calcula el resultado

Flujo Alternativo	1. Los operandos no son compatibles, el lenguaje arroja un error.

ID	11.1
Nombre	Estatutos condicionales
Descripción	Uso de estatutos condicionales Si-SI no
Precondición	Ninguna
Postcondición	Estatuto condicional creado
Flujo de eventos	1. Usuario escribe palabra reservada “if” seguida de la condición lógica dentro de paréntesis y el código en medio de llaves 2. Usuario evalúa la condición y ejecuta el código
Variación	1. Usuario agrega inmediatamente la palabra reservada “else” y el código dentro de llaves

ID	12.1
Nombre	Ciclos
Descripción	Uso de Ciclo While
Precondición	Ninguna
Postcondición	Ciclo creado
Flujo de eventos	1. Usuario escribe palabra reservada “while” seguida de la condición lógica dentro de paréntesis y el código en medio de llaves 2. Usuario evalúa la condición y ejecuta el código hasta que se cumpla la condición

ID	13.1
Nombre	Definición de arreglos de varias dimensiones
Descripción	Estructuras de varias dimensiones
Precondición	Ninguna

Postcondición	Arreglo declarado
Flujo de eventos	1. El usuario escribe la palabra define 2. El usuario escribe el tipo de la variable 3. El usuario escribe el id (primer caracter letra minúscula) 4. El usuario escribe un par de corchetes por cada dimensión a declarar conteniendo dentro una constante que representa el tamaño de la dimensión 5. El lenguaje guarda el variable en memoria
Flujo Alternativo	3. El nombre ya existe, el lenguaje despliega el mensaje: “Variable '" + newVar + "' already defined"

ID	13.2
Nombre	Asignación de arreglos de varias dimensiones
Descripción	Se puede asignar un valor al elemento de un arreglo previamente definida invocando su nombre con la posición entre corchetes, seguido del signo de igual “=” y el valor deseado, siempre y cuando coincida con el tipo de la variable.
Precondición	Arreglo previamente definida
Postcondición	Elemento del arreglo actualizado
Flujo de eventos	5. El usuario manda a llamar el nombre del arreglo con su posición a asignar entre corchetes 6. El usuario escribe un signo de igual “=” 7. El usuario escribe un valor constante, de otra variable, de una llamada de función o acceso al elemento de un arreglo con retorno para asignar al elemento 8. El lenguaje actualiza el valor del elemento
Flujo Alternativo	3. El valor asignado no es compatible con el tipo definido del arreglo. El lenguaje despliega el mensaje: "Operand '{asigned}'({asignedType}) is incompatible with operand '{result}'({resultType}) with operator: {op}"

ID	14.1
Nombre	openImg
Descripción	Recibe como parámetro la ubicación de un archivo de una imagen y lo guarda en una variable de matriz de colores

Precondición	Arreglo de colores con tamaño de la imagen previamente definido
Postcondición	Arreglo de colores guarda el color hexadecimal correspondiente al color de cada pixel
Flujo de eventos	1. El usuario llama a la función openImg con la dirección local de la imagen como parámetro (tag) y el arreglo de colores en donde guardar los pixeles 2. El lenguaje rellena los pixeles del arreglo de colores
Flujo Alternativo	1.1 Arreglo de colores con tamaño insuficiente, la imagen es recortada 1.2 Arreglo de colores con tamaño más grande del requerido, el espacio restante se vuelve blanco

ID	14.2
Nombre	saveImg
Descripción	Recibe como parámetro una matriz de colores y lo guarda como imagen según la ruta de archivo especificada.
Precondición	Arreglo de colores llenado con los pixeles de una imagen
Postcondición	Imagen guardada en disco local
Flujo de eventos	1. El usuario llama a la función saveImg con la dirección local y el arreglo de colores como parámetro 2. El lenguaje genera la imagen y la guarda en el disco local

ID	14.3
Nombre	grayscale
Descripción	Recibe una matriz de colores y la convierte la imagen a escala de grises.
Precondición	Arreglo de colores llenado con los pixeles de una imagen
Postcondición	Colores del arreglo de colores ajustados a escala de grises
Flujo de eventos	1. El usuario llama a la función grayscale con el arreglo de colores como parámetro 2. El lenguaje convierte cada pixel a su equivalente en escala de grises y sobreescribe el arreglo

ID	14.4
Nombre	colorReplace
Descripción	Toma una matriz de colores y recibe dos colores como parámetros, el primero será recoloreado por el segundo.
Precondición	Arreglo de colores llenado con los pixeles de una imagen
Postcondición	Tonos similares al primer color del parámetro son reemplazados por el segundo color
Flujo de eventos	1. El usuario llama la función colorReplace con el arreglo de colores como primer parámetro, un color y un segundo color 2. El lenguaje reemplaza los colores de la imagen que coincidan con el primer color por el segundo color

ID	14.5
Nombre	colorFilter
Descripción	Recibe una matriz de colores y un color. Esta función convierte en escala de gris todos los colores de la matriz que no sean iguales al segundo parámetro.
Precondición	Arreglo de colores llenado con los pixeles de una imagen
Postcondición	Colores que no coincidan con el color del parámetro dentro de la imagen se vuelven a escala de grises
Flujo de eventos	1. El usuario llama la función colorFilter con el arreglo de colores como primer parámetro y un color 2. El lenguaje reemplaza los colores de la imagen que no coincidan con el color por su escala de grises

ID	14.6
Nombre	edgeDetection
Descripción	Toma una matriz de colores. Esta función detecta automáticamente los bordes de los principales elementos de una imagen y los delinea.
Precondición	Arreglo de colores llenado con los pixeles de una imagen

Postcondición	Contornos detectados de las formas de una imagen se torna blanco con un fondo negro
Flujo de eventos	1. El usuario llama la función edgeDetection con el arreglo de colores 2. El lenguaje pinta de blanco los bordes de la forma en la imagen y deja el fondo en negro

ID	14.7
Nombre	saveScaleImg
Descripción	Toma como parámetro una matriz de colores y dos números flotantes. La imagen será escalada en x según el valor del segundo parámetro (Ej. 2 = 200%) y en y según el valor del tercer parámetro. Un número negativo escalará la imagen según la magnitud y la invertirá de forma horizontal o vertical según sea el caso.
Precondición	Arreglo de colores llenado con los pixeles de una imagen
Postcondición	Tamaño de imagen ajustada
Flujo de eventos	1. El usuario llama a la función de scaleImg con el arreglo de colores como parámetro seguido de dos valores numéricos 2. El lenguaje ajusta el alto y ancho de la imagen según los valores númericos dados

ID	14.8
Nombre	getColorPalette
Descripción	Función que recibe como parámetro una matriz de colores, un arreglo de una dimensional de colores y un número entero n. Se regresa un arreglo con n colores
Precondición	Arreglo de colores llenado con los pixeles de una imagen
Postcondición	Arreglo de una dimensión poblado con colores predominantes de arreglo de colores
Flujo de eventos	1. El usuario llama a la función getColorPath con el arreglo de colores que representa la imagen, un arreglo de una dimensión tamaño n y un número entero n 2. El lenguaje llena el arreglo de una dimensión con los n colores predominantes de la imagen

ID	14.9
Nombre	colorMatchImage
Descripción	Función que recibe como parámetro dos matrices de colores, siendo la primer matriz la imagen fuente y la segunda la imagen de referencia. Esta función manipula los píxeles de la imagen fuente de modo que el histograma de pixeles concuerde con el histograma de la imagen de referencia.
Precondición	Dos arreglo de colores llenados con los pixeles de una imagen
Postcondición	Primer arreglo de colores ajustado para coincidir con los colores de la segunda imagen
Flujo de eventos	1. El usuario llama a la función colorMatchImage con dos arreglos de colores 2. El lenguaje ajusta los colores del primer arreglo para que coincidían con los del segundo arreglo

ID	14.10
Nombre	encodeSteganography
Descripción	Esta función se basa en la técnica de Esteganografía, la cual oculta un mensaje dentro de una imagen. La función recibe una imagen y un mensaje de texto y retorna la imagen con el código guardado. A simple vista la imagen no parece ser diferente a la original.
Precondición	Arreglo de colores llenado con los pixeles de una imagen
Postcondición	Mensaje encriptado en arreglo de colores
Flujo de eventos	1. Usuario manda a llamar a encodeSteganography con un arreglo de color como parámetro y un tag que sea el mensaje a encriptar 2. El lenguaje encripta el mensaje en la imagen

ID	14.11
Nombre	decodeSteganography

Descripción	Esta función recibe una imagen modificada con esteganografía para obtener el mensaje encriptado que contenga. Recibe una matriz de colores e imprime el mensaje oculto
Precondición	Arreglo de colores llenado con los pixeles de una imagen
Postcondición	Mensaje desencriptado y desplegado en consola
Flujo de eventos	1. Usuario manda a llamar a decodeSteganography con un arreglo de color como parámetro 2. El lenguaje desencripta el mensaje y lo despliega en consola

1.5. Descripción de los principales Test Cases

A continuación se listan los casos de prueba que se utilizarán para evaluar la utilidad del Lenguaje final, ya que se programarán exclusivamente con la herramienta desarrollada:

- Función Fibonacci iterativa
- Función Fibonacci recursiva
- Función Factorial iterativa
- Función Factorial recursiva
- Multiplicación de matrices
- Búsqueda en arreglos
- Ordenamiento Burbuja (Bubble sort)
- Funciones especiales

1.6. Descripción del proceso general

1.6.1. Proceso General

Para el desarrollo de este proyecto se implementó una metodología Ágil debido a que los requerimientos se fueron desarrollando iterativamente, ampliando en cada iteración el alcance del lenguaje. Encontramos que esta metodología se ajustó correctamente a nuestro horario académico impredecible. En la mayoría de los casos, nos enfocamos a trabajar en el proyecto cada fin de semana, excluyendo la última semana de octubre de semana i, correspondiendo igualmente con las fechas de entregas definidas en clase.

1.6.2. Bitácoras generales

1. **Lunes 30 de Septiembre**
    a. Análisis de Léxico y Sintaxis (Scanner y Parser)
2. **Lunes 7 de Octubre**
    a. Semántica Básica de Variables: Directorio de Procedimientos y Tablas de Variables

b. Semántica Básica de Expresiones: Tabla de Consideraciones semánticas (Cubo Semántico)
c. *“Para esta entrega: -Agregamos puntos neurálgicos al momento de definir funciones y variables para así crear el directorio de funciones, variables globales y locales. -Gracias a los directorios ahora se valida que no existan nombres repetidos para variables y funciones según el scope. -También ya creamos el cubo semántico que contempla todas las operaciones entre los tipos de datos del lenguaje. -Hicimos algunas correcciones a las gramática para dar mayor estabilidad. -Al momento estamos al corriente según lo esperado.”*

3. **Lunes 14 de Octubre**
   a. Generación de Código de Expresiones Aritméticas y estatutos secuenciales: Asignación, Lectura, etc.
   b. Generación de Código de Estatutos Condicionales: Decisiones
   c. *“Este avance contiene: -Generación de cuádruplos para expresiones aritméticas (+,-,*,/), comparaciones (>,<,==, ...) y lógicas(and,or). -Generación de cuádruplos para estatutos secuenciales: print, read. -Generación de cuádruplos en: IF -Validación semántica agregada: Se verifica que los nombres usados en las operaciones estén definidos anteriormente y que los tipos de variable correspondan según la operación que se ejecuta. -Manejo de errores: Indica la incompatibilidad de tipos que encuentre primero.”*
4. **Lunes 21 de Octubre**
   a. Generación de Código de Estatutos Condicionales: Ciclos
   b. Generación de Código de Funciones
   c. *“Bitácora de avance: -Generación de código intermedio para ciclos (while). -Generación de código intermedio para funciones y llamadas a función. Incluyendo ERA, GOSUB, parameter... -Verificación semántica en la llamada de funciones. -Se extendió el directorio de funciones: ahora cuenta con la tabla de parámetros y número de cuádruplo.”*
5. **Lunes 4 de Noviembre**
   a. Mapa de Memoria de Ejecución para la Máquina Virtual
   b. Máquina Virtual: Ejecución de Expresiones Aritméticas y Estatutos Secuenciales
   c. *“Bitácora: -Ahora los cuádruplos se generan con las direcciones virtuales de la memoria en vez del nombre de la variable. -Se agregó la memoria virtual y se encarga de asignar los tamaños adecuados según la función. -Se agregó la máquina virtual, es capaz de ejecutar operaciones aritméticas y print.”*
6. **Lunes 11 de Noviembre**
   a. Generación de Código de Arreglos/Tipos estructurados
   b. Máquina Virtual: Ejecución de Estatutos Condicionales
   c. *“Bitácora: -Se agregó la generación de código para funciones con return -Ahora la máquina virtual también reconoce las funciones: eso incluye los*

cuádruplos(gosub,param,return,endproc) -Se agregó la generación de código para variables dimensionadas. -Falta validar que los tamaños y dimensiones accedidas sean las correctas.”

7. **Lunes 18 de Noviembre:**
    a. 1era versión de la Documentación
    b. Generación de Código y Máquina Virtual para una parte de la aplicación particular
    c. “Manejo de arreglos completado: validación de dimensiones e índices funcionando. Funciones especiales agregadas: Abrir imágenes, guardar imagenes, obtener dimensiones en píxeles, conversión de rgb a hexadecimal y viceversa. Correcciones: Validación correcta de uso de funciones que retornan valores y las que no. Generación correcta de cuádruplos al inicializar variables no dimensionadas en su declaración.”
8. **Miércoles 27 de Noviembre:**
    a. Entrega Final

1.6.3. Reflexiones

Este proyecto realmente ha resultado un verdadero reto con respecto a lo que realizado en el transcurso de mi carrera. La realización del compilador conlleva una constante administración del proyecto durante todo un semestre, en mi punto de vista este trabajo permite el desarrollo de dos aspectos muy importantes: el primero es el manejo de entregas periódicas de un producto de software y en segunda, el trabajo constante y en equipo. También aprendí de forma técnica ya que utilicé un lenguaje en el cual no tenía mucha exp

Alejandro González Valles A000818616 _____

Como alumnos de la carrera de ITC, la materia de compiladores es bien conocida por todos, ya sea temida o anticipada, a todos desde el primer día de clases, en la clase de Introducción a la carrera, se nos dijo qué esperar de la materia. Y bajo aviso no hay engaño. “No les voy a volver a dar clase hasta Compiladores”. Desde entonces estuve anticipando el momento de cursar esta materia, sabía que iba a representar un reto intelectual y académico. Y ahora, casi 4 años después, noviembre 27 del 2019, a las 5:53 de la mañana, puedo decir que todo lo que me dijeron y más es cierto.

Lizzie Marielle Guajardo Mozo A000818258 _____

2. Descripción del Lenguaje

2.1. Nombre del Lenguaje

Quetzal

2.2. Descripción genérica de las principales características del lenguaje

- Manejo de tipos (int, float, bool y color)
- Uso de matrices de varias dimensiones como estructura.
- Declaración de variables y funciones
- Uso de variables globales y locales
- Uso de parámetros por referencia
- Uso de funciones con valor de retorno
- Estatuto Condicional: If-Else
- Ciclo: While
- Lectura, escritura y manipulación de imágenes a nivel pixel
- Lectura y Escritura desde consola
- Habilidad de utilizar llamadas de función con tipo de retorno, variables, constantes u operaciones aritméticas como parámetros en funciones y arreglos
- Habilidad de utilizar recursiones
- Función principal main() es ejecutada al principio de cada programa y de carácter obligatorio

2.3. Listado de errores que pueden ocurrir, tanto en compilación como en ejecución

- Errores sintácticos:
    - Al declarar una variable y no colocar tipo o uno no soportado: In line 3, column 7: missing {'bool', 'int', 'float', 'color'} at 'integer'
    - Al no colocar el tipo de función al declararla: In line 5, column 6: no viable alternative at input 'func uno'
    - No declarar correctamente una variable: define int x(1)[2] In line 1, column 14: mismatched input '(' expecting {';', ',', '[', '='}
    - Al faltar el cierre de un par de llaves {}, el programa indicará que falta.
- Errores de compilación:
    - *Cuando una función void se usa dentro de una expresión (f"Cannot use [{name}] function as a expression.")*
    - *Cuando se usa una variable o función no declarada: ("Operand '" + name + "' not defined")*

- *Al detectar que se realiza una operación con tipos incompatibles: (f"Operand '{izqOp}'({izqOpType}) is incompatible with operand '{derOpType}'({derOpType}) with operator: {op}")*
- *Al asignar un tipo que no es compatible: (f"Operand '{asigned}'({asignedType}) is incompatible with operand '{result}'({resultType}) with operator: {op}")*
- *Al exceder el número de parámetros de una función: (f"In function [{fName}]: expected [{parameterNumber}] parameters but more were given")*
- *Cuando se se asigna el tipo esperado al parámetro de una función: (f"In function [{fName}], parameter #{parameterCounter}: Expected type [{paramExpectedType}], but [{paramActualType}] were given")*
- *Se evalúa una condicional con valores no booleanos: ("La expresión dentro del If debe resultar en bool")*
- *Cuando se retorna void en una función que retorna un tipo: (f"Function '{function}' is [{functType}] and returned [void]")*
- *Al retornar un tipo que no es compatible con el tipo que retorna la función: (f"Function '{function}' is [{functType}] and returned [{stackType}]")*
- *Cuando se trata de utilizar una función no definida: ("Function '" + funcName + "' not defined")*
- *Cuando se utiliza una variable de tipo array sin especificar las casillas a acceder: (f"[{varName}] is an Array")*
- *Cuando se trata de acceder a una variable no definida previamente: (f"Variable [{varName}] is not defined")*
- *Cuando se intenta indexar una variable que no es dimensionada: (f"Variable [{varName}] is not an array")*
- *Cuando se accede al elemento de un arreglo pero no especifican todas las dimensiones que tiene: (f"Variable [{varName}]: [{self.Dims[-1][1] }] dimentions accessed but [{dimCount}] were expected.")*
- *Cuando se accede a más de las dimensiones definidas en la variable: ("Variable '" + frontName + "' with "+ str(frontDim) + " Dimensions not defined, " + str(len(listaDim)) + " where expected")*
- *Cuando la expresión dentro de un acceso a una dimensión no es tipo int: ("Index must be int"), (f"Index must be int, given [{indexType}]")*
- *Validación de variables de dos dimensiones en funciones especiales: (f"This function needs [2] dimentions in variable [{varName}] but it has [{len(varDims)}]")*

- Errores captados en Ejecución:
  - *De sobrecupo de memoria*
    - *"Stack Overflow"*
    - *"index out of range"*
  - *De tipo incorrecto en Input*
    - *"Input Value invalid, expected int"*

- *"Input Value invalid, expected float"*
- *"Input Value invalid, expected color (#ffffff)"*
- *"Input Value invalid, expected bool"*

- *De funciones especiales*
  - *"In function [getColorPalette], the number of colors to extract must be grater than 0"*
  - *"In function [getColorPalette], the number of colors to extract cannot exceed palette array size"*

3. Descripción Del Compilador

3.1. Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto.

- Laptop Asus, Processador i7, 16 GB ram, Windows 10
- Laptop Asus, Procesador i7, 8 GB ram, Windows 10
- Visual Studio Live Share
- GitHub Desktop
- ANTLR4
- Python3
- Libraries
  - scikit image (https://scikit-image.org/)
  - cologram (https://github.com/obskyr/colorgram.py)
  - Numpy (https://numpy.org/)

3.2. Descripción del Análisis de Léxico.

3.2.1. Patrones de Construcción de los elementos principales

```
/* TYPES */
TYPE_FLOAT : FRAG_DIGIT+ ('.' FRAG_DIGIT+);
TYPE_INT : FRAG_DIGIT+;
TYPE_ID : FRAG_LOWER_CASE FRAG_FOLLOW*;
TYPE_COLOR : CTE_COLOR | FRAG_HEX_COLOR;
CTE_TAG: '"' (FRAG_FOLLOW|' ' | '/')* '"';
TYPE_BOOL: TK_TRUE | TK_FALSE;

/* FRAGMENTS */
```

```
fragment FRAG_LETTER: FRAG_UPPER_CASE | FRAG_LOWER_CASE;
fragment FRAG_FOLLOW: FRAG_LETTER|FRAG_DIGIT|SYM_UNDER_SCORE|':'|'.'|'\\';
fragment FRAG_DIGIT : [0-9];
fragment FRAG_UPPER_CASE: [A-Z];
fragment FRAG_LOWER_CASE: [a-z];
fragment FRAG_HEX_VAL: ('A'|'B'|'C'|'D'|'E'|'F') | ('a'|'b'|'c'|'d'|'e'|'f') | FRAG_DIGIT;
fragment FRAG_HEX_COLOR: '#' FRAG_HEX_VAL FRAG_HEX_VAL FRAG_HEX_VAL
FRAG_HEX_VAL FRAG_HEX_VAL FRAG_HEX_VAL;
/* NEW LINES AND WHITESPACE
   Telling the parser to skip new lines and white spaces (works in windows and linux)*/
NEWLINE : '\r'? '\n' -> skip;
WS: (' ' | '\t')+ -> skip;

/* COMMENTS */
BLOCK_COMMENT : SYM_DIV SYM_MULT .*? SYM_MULT SYM_DIV -> channel(HIDDEN); /*
ALLOW FOR BLOCK COMMENTS WITH THE FORMAT OF '/* this is a block comment */
LINE_COMMENT : SYM_DIV  SYM_DIV  ~[\r\n]* -> channel(HIDDEN); /* ALLOW FOR LINE
COMMENTS WITH THE FORMAT OF //this is a line comment */
```

3.2.2. Enumeración de los "tokens" del lenguaje y su código asociado

```
/* TOKENS */
TK_PROGRAM: 'program';
TK_FUNC: 'func';
TK_DEFINE: 'define';
TK_RETURN: 'return';
TK_IF: 'if';
TK_ELSE: 'else';
TK_WHILE: 'while';
TK_PRINT: 'print';
TK_READ: 'read';
fragment TK_TRUE: 'True';
fragment TK_FALSE: 'False';
TK_MAIN: 'main';
TK_VOID: 'void';

//Special Functions
TK_OPENIMG: 'openImg';
TK_SAVEIMG: 'saveImg';
TK_GRAYSCALE: 'grayscale';
TK_COLOR_REPLACE: 'colorReplace';
```

```
TK_COLOR_FILTER: 'colorFilter';
TK_EDGE_DETECTION: 'edgeDetection';
TK_SCALE_IMAGE: 'saveScaleImg';
TK_GET_COLOR_PALETTE: 'getColorPalette';
TK_COLOR_MATCH_IMAGE: 'colorMatchImage';
TK_ENCODE_STEGANOGRAPHY: 'encodeSteganography';
TK_DECODE_STEGANOGRAPHY: 'decodeSteganography';

//Types Tokens
TK_BOOL: 'bool';
TK_INT: 'int';
TK_FLOAT: 'float';
TK_COLOR: 'color';

/* SYMBOL */
SYM_SEMI_COL : ';';
SYM_DOUB_COL : ':';
SYM_COMMA : ',';
SYM_UNDER_SCORE: '_';
SYM_CURLY_BRACK_OPEN: '{';
SYM_CURLY_BRACK_CLOSE: '}';
SYM_SQUARE_BRACK_OPEN: '[';
SYM_SQUARE_BRACK_CLOSE: ']';
SYM_PAREN_OPEN: '(';
SYM_PAREN_CLOSE: ')';
SYM_ASSIGN : '=';
SYM_EQUAL : '==';
SYM_GRE_THAN : '>';
SYM_LOW_THAN : '<';
SYM_NOT_EQUAL : '!=';
SYM_GRE_EQ: '>=';
SYM_LOW_EQ: '<=';
SYM_OR: '||';
SYM_AND: '&&';
SYM_QUOT : '"';
SYM_PLUS: '+';
SYM_MINUS: '-';
SYM_MULT: '*';
SYM_DIV: '/';
```

3.3. Descripción del Análisis de Sintaxis

3.3.1. Gramática Formal empleada para representar las estructuras sintácticas

```
program: variables* function* main;
main:TK_FUNCTK_MAINSYM_PAREN_OPEN SYM_PAREN_CLOSE SYM_CURLY_BRACK_OPEN
variables* statute* SYM_CURLY_BRACK_CLOSE;
variables:
   TK_DEFINE types TYPE_ID
      ((SYM_SQUARE_BRACK_OPEN TYPE_INT
       SYM_SQUARE_BRACK_CLOSE)+
      |(SYM_ASSIGNexpression)
      |)?
   (SYM_COMMA TYPE_ID
      ((SYM_SQUARE_BRACK_OPEN TYPE_INT
      SYM_SQUARE_BRACK_CLOSE)+
      |(SYM_ASSIGNexpression )|)? )* SYM_SEMI_COL;
function: TK_FUNC
      (types TYPE_ID | TK_VOID TYPE_ID
      SYM_PAREN_OPEN
      (types TYPE_ID
      (SYM_COMMA types TYPE_ID)*)?
      SYM_PAREN_CLOSE
      SYM_CURLY_BRACK_OPEN variables* statute*
      SYM_CURLY_BRACK_CLOSE};
block: SYM_CURLY_BRACK_OPEN variables* statute* SYM_CURLY_BRACK_CLOSE;
types: (TK_INT | TK_FLOAT | TK_COLOR | TK_BOOL);
constants:
   TYPE_INT
   | TYPE_FLOAT
   | CTE_TAG
   | TYPE_BOOL
   | TYPE_COLOR;

prints: TK_PRINT SYM_PAREN_OPEN expression (SYM_COMMA expression)*
SYM_PAREN_CLOSE SYM_SEMI_COL;
read: TK_READ SYM_PAREN_OPEN var_ SYM_PAREN_CLOSESYM_SEMI_COL;

statute: returning|condition|loop|prints|read|(callfunc SYM_SEMI_COL)|(specfunct
SYM_SEMI_COL)|assignation;
assignation:
```

```
   TYPE_ID  SYM_ASSIGN(specfunct|(expression)) SYM_SEMI_COL
   | TYPE_ID (SYM_SQUARE_BRACK_OPEN
expressionSYM_SQUARE_BRACK_CLOSE)+SYM_ASSIGN (specfunct|expression)
SYM_SEMI_COL
condition:
   TK_IF SYM_PAREN_OPEN
   expression
   SYM_PAREN_CLOSE
   block (TK_ELSEblock)? ;

var_cte: constants
   | callfunc
   | var_;

var_:
   TYPE_ID
   | TYPE_IDSYM_SQUARE_BRACK_OPEN expressionSYM_SQUARE_BRACK_CLOSE)+;


//EXPRESIONES->EXP->TERM->FACTOR
expression:  expLogic((SYM_OR | SYM_AND) expLogic})*;
expLogic: exp (logic_opexp)?;
exp:  term  ((SYM_PLUS | SYM_MINUS) term)*;
term:  factor (( SYM_MULT|  SYM_DIV ) factor)*;
factor:  (SYM_PAREN_OPEN expression SYM_PAREN_CLOSE ) | ( (SYM_PLUS|SYM_MINUS)?
var_cte );

logic_op: SYM_EQUAL| SYM_GRE_THAN | SYM_LOW_THAN| SYM_NOT_EQUAL |
SYM_GRE_EQ | SYM_LOW_EQ;


returning: TK_RETURN expression SYM_SEMI_COL
   |TK_RETURN SYM_SEMI_COL;

callfunc: TYPE_ID
      SYM_PAREN_OPEN
      (expression (SYM_COMMA expression)*)?
      SYM_PAREN_CLOSE;

loop: TK_WHILE SYM_PAREN_OPEN expression SYM_PAREN_CLOSE block;

//SPECIAL FUNCTIONS
```

openimg: TK_OPENIMG SYM_PAREN_OPEN CTE_TAG SYM_COMMA TYPE_IDSYM_PAREN_CLOSE ;
saveimg: TK_SAVEIMG SYM_PAREN_OPEN CTE_TAGSYM_COMMA TYPE_IDSYM_PAREN_CLOSE;
color_replace: TK_COLOR_REPLACE SYM_PAREN_OPEN TYPE_ID SYM_COMMA expression SYM_COMMA expression SYM_PAREN_CLOSE;
grayscale: TK_GRAYSCALE SYM_PAREN_OPEN TYPE_IDSYM_PAREN_CLOSE;
color_filter: TK_COLOR_FILTER SYM_PAREN_OPEN TYPE_ID SYM_COMMA expression SYM_PAREN_CLOSE;
edgeDetection: TK_EDGE_DETECTION SYM_PAREN_OPEN TYPE_ID SYM_PAREN_CLOSE;
scaleImg: TK_SCALE_IMAGE SYM_PAREN_OPEN CTE_TAG SYM_COMMA TYPE_ID SYM_COMMA expression SYM_COMMA expression SYM_PAREN_CLOSE;
getColorPalette: TK_GET_COLOR_PALETTE SYM_PAREN_OPEN TYPE_IDSYM_COMMA TYPE_ID SYM_COMMA expression SYM_PAREN_CLOSE ;
colorMatchImage: TK_COLOR_MATCH_IMAGE SYM_PAREN_OPEN TYPE_ID SYM_COMMA TYPE_ID SYM_PAREN_CLOSE;
encodeSteganography: TK_ENCODE_STEGANOGRAPHY SYM_PAREN_OPEN TYPE_ID SYM_COMMA CTE_TAGSYM_PAREN_CLOSE;
decodeSteganography: TK_DECODE_STEGANOGRAPHY SYM_PAREN_OPEN TYPE_ID SYM_PAREN_CLOSE;
specfunct:openimg | saveimg | grayscale | color_replace | color_filter | edgeDetection | scaleImg | getColorPalette | colorMatchImage | encodeSteganography | decodeSteganography;

3.4. Descripción de Generación de Código Intermedio y Análisis Semántico

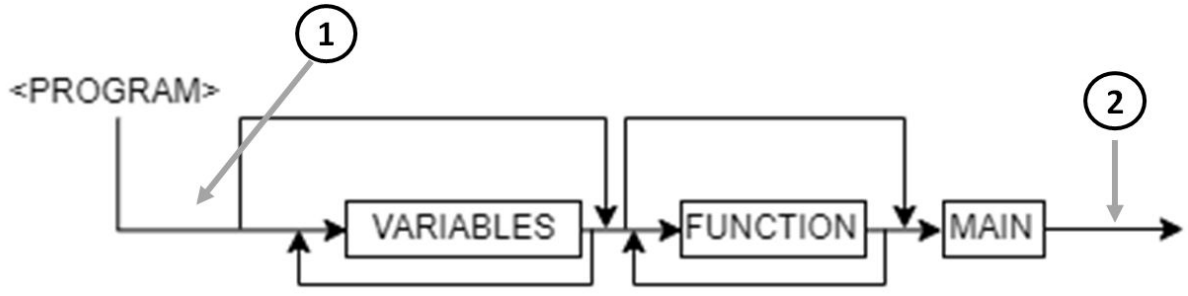
3.4.1. Código de operación y direcciones virtuales asociadas a los elementos del código

Se asignaron segmentos de memoria para las variables globales, locales, temporales y constantes. Cada segmento se subdivide según los tipos de variables (int,float,bool,color) y un subsegmento de temporales para el bloque de memorias locales. Además de un bloque adicional de de constantes para los tipos “tag”.

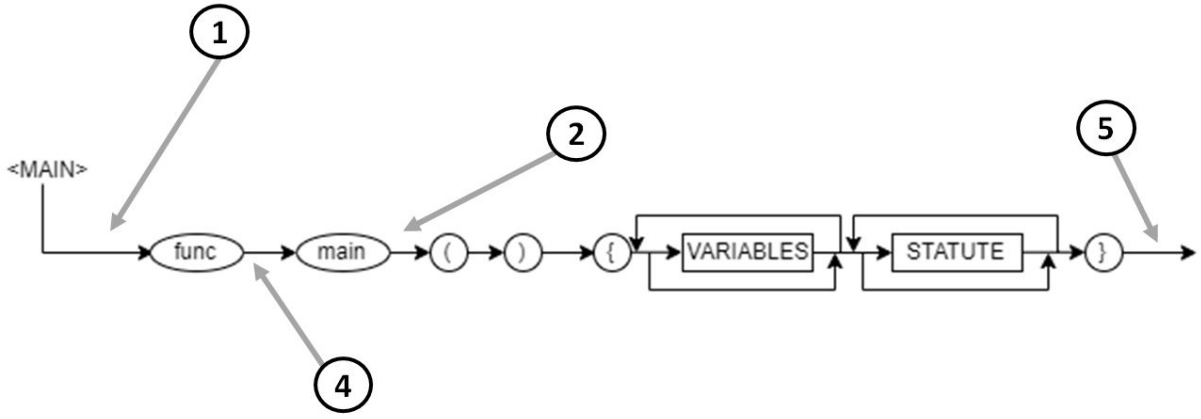
Bloque	**Rango**
Globales enteras	1000000-2000000
Globales flotantes	2000000-3000000

Globales bool	3000000-4000000
Globales color	5000000-6000000
Globales tag	7000000-8000000
Locales enteras	9000000-10000000
Locales flotantes	11000000-12000000
Locales bool	13000000-14000000
Locales color	15000000-16000000
Locales tag	17000000-18000000
Temporales enteras	19000000-20000000
Temporales flotantes	21000000-22000000
Temporales bool	23000000-24000000
Temporales color	25000000-26000000
Temporales tag	27000000-28000000
Constantes enteras	29000000-30000000
Constantes flotantes	31000000-32000000
Constantes bool	33000000-34000000
Constantes color	35000000-36000000
Constantes tag	37000000-38000000

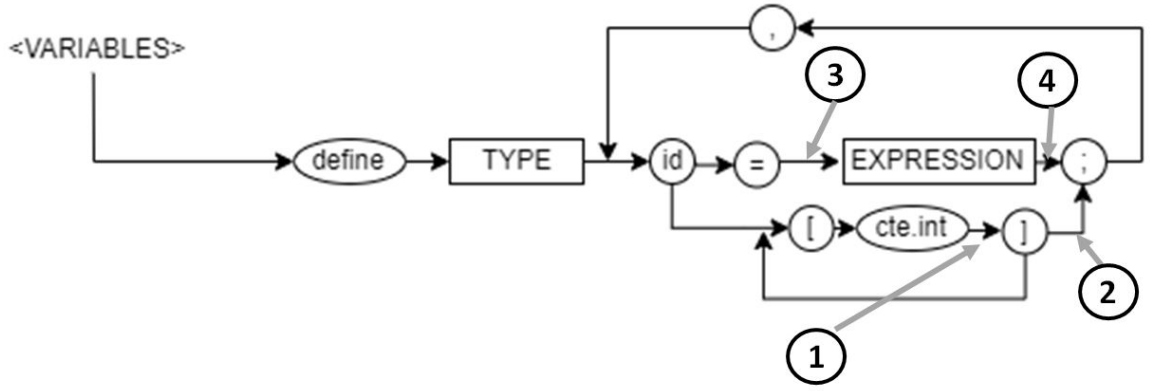
3.4.2. Diagramas de Sintaxis con las acciones correspondientes



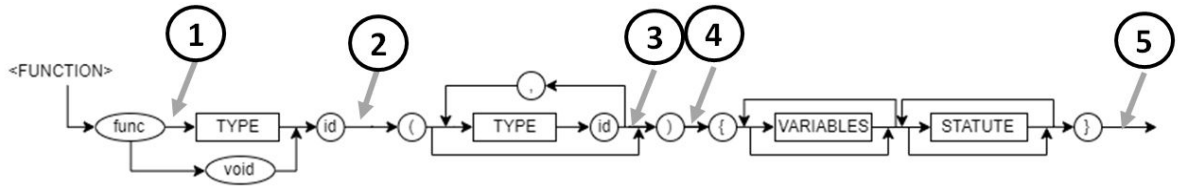
1. Inicializar tabla constantes y locales. Generar Goto para el main y meter el contador-1 a la pila de saltos.
2.Terminar ejecución del compilador, crear instancia de la máquina virtual enviándole los cuádruplos y las inicializaciones de los bloques de memoria. Iniciar ejecución de la máquina virtual.



1-Hacer pop a la pila de saltos y rellenar el GoTo inicial.
2- Borrar tabla de variables anterior y generar una nueva.
4- Agregar a main a la tabla de funciones
5-Generar cuádruplo ENDPROGRAM, contabilizar y registrar cantidades de variables y temporales usadas.

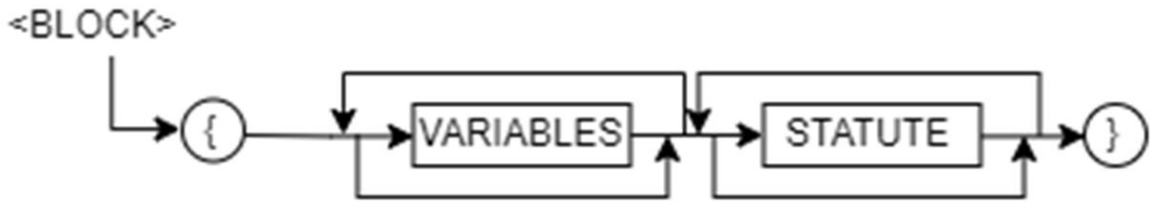


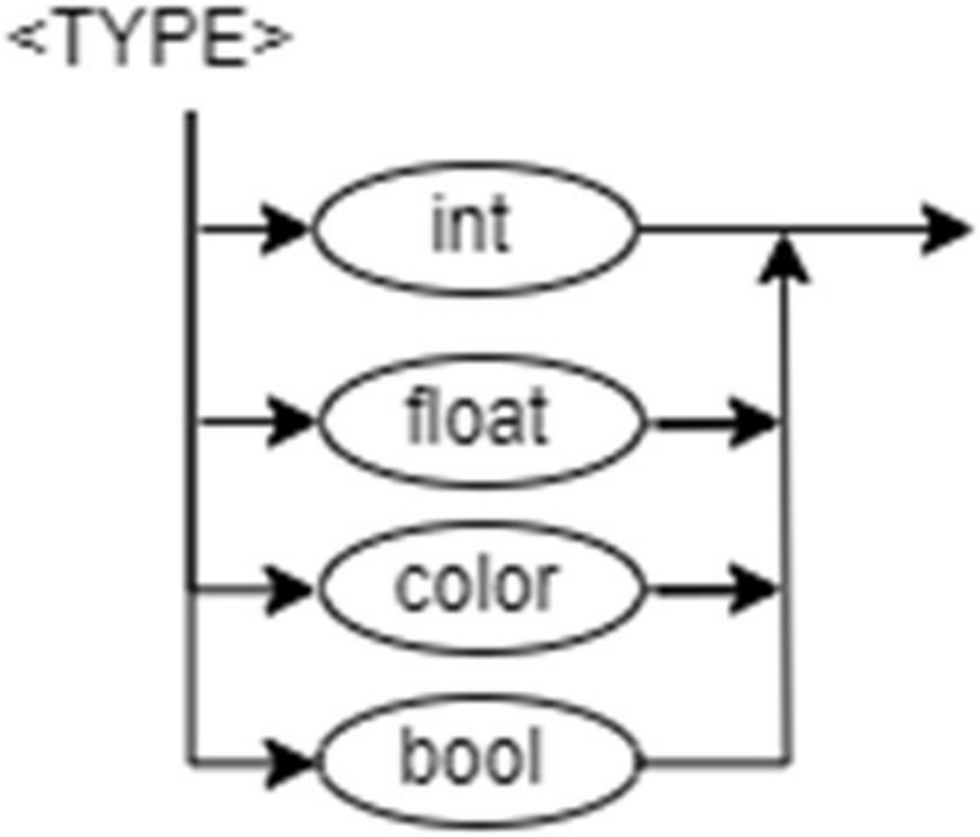
1-Agregar dimensión a la Variable, Acumular R.
2-Volver a recorrer arreglo de dimensiones, calcular -K y Mn. Registrar la variable en la tabla de variables de su Scope.
3-Crear nueva variable y meterla a la pila de operandos, Meter ‘=’ a la pila de operadores.
4-Sacar resultado de la pila de operadores y generar Cuádruplo con (= epx _ newVar)

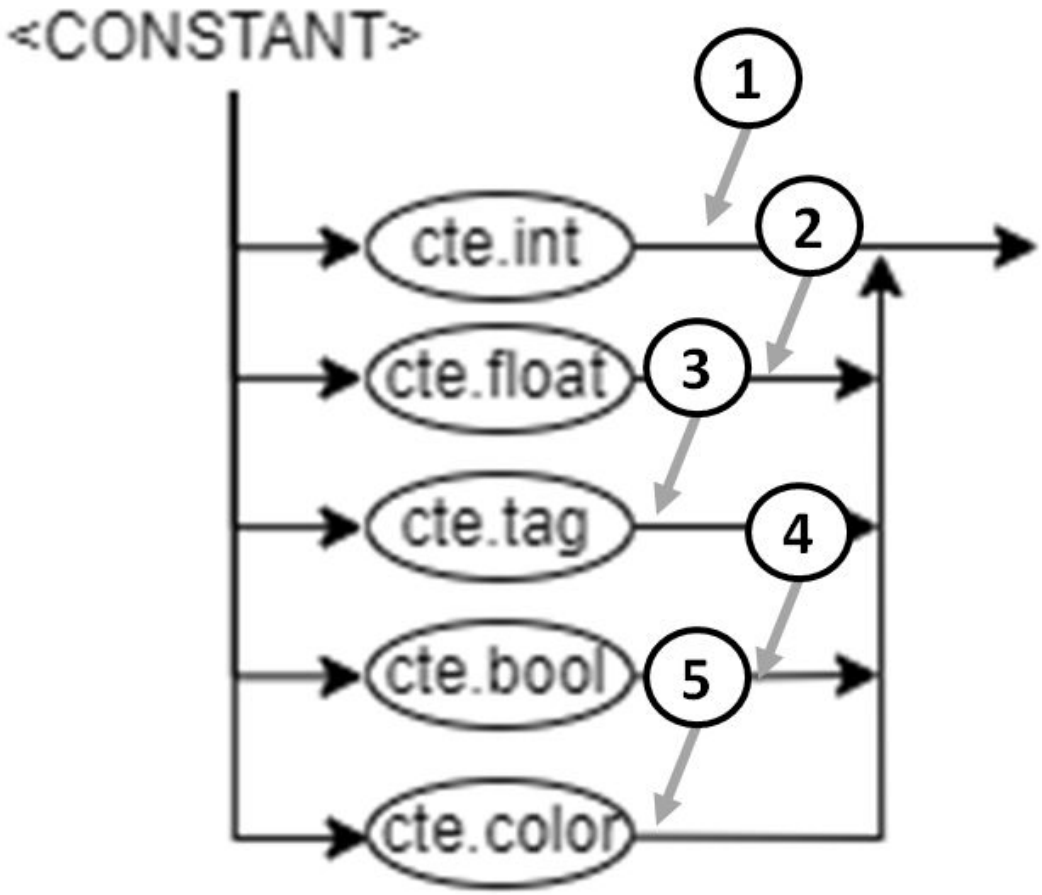


1-Inicializar tabla de variablesLocales

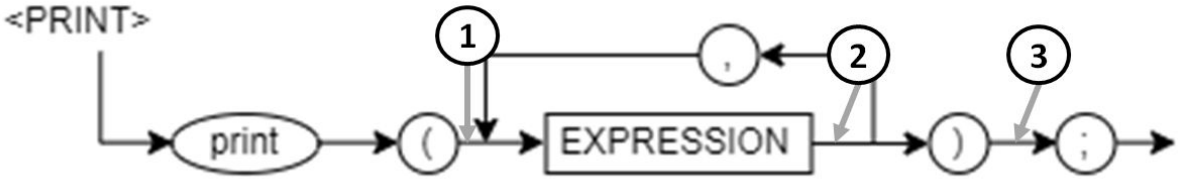
2-Verificar que el nombre no haya sido declarado, si nó, agregar la función a la tabla de funciones junto con el tipo de la función.
3-Agregar parámetro a la lista de parámetros
4-Registrar parámetros en la tabla de variables locales y asignarlas a su registro de la tabla de funciones correspondiente.
5-Contabilizar variables y temporales declaradas y registrarlas el fila de la tabla de funciones correspondientes.



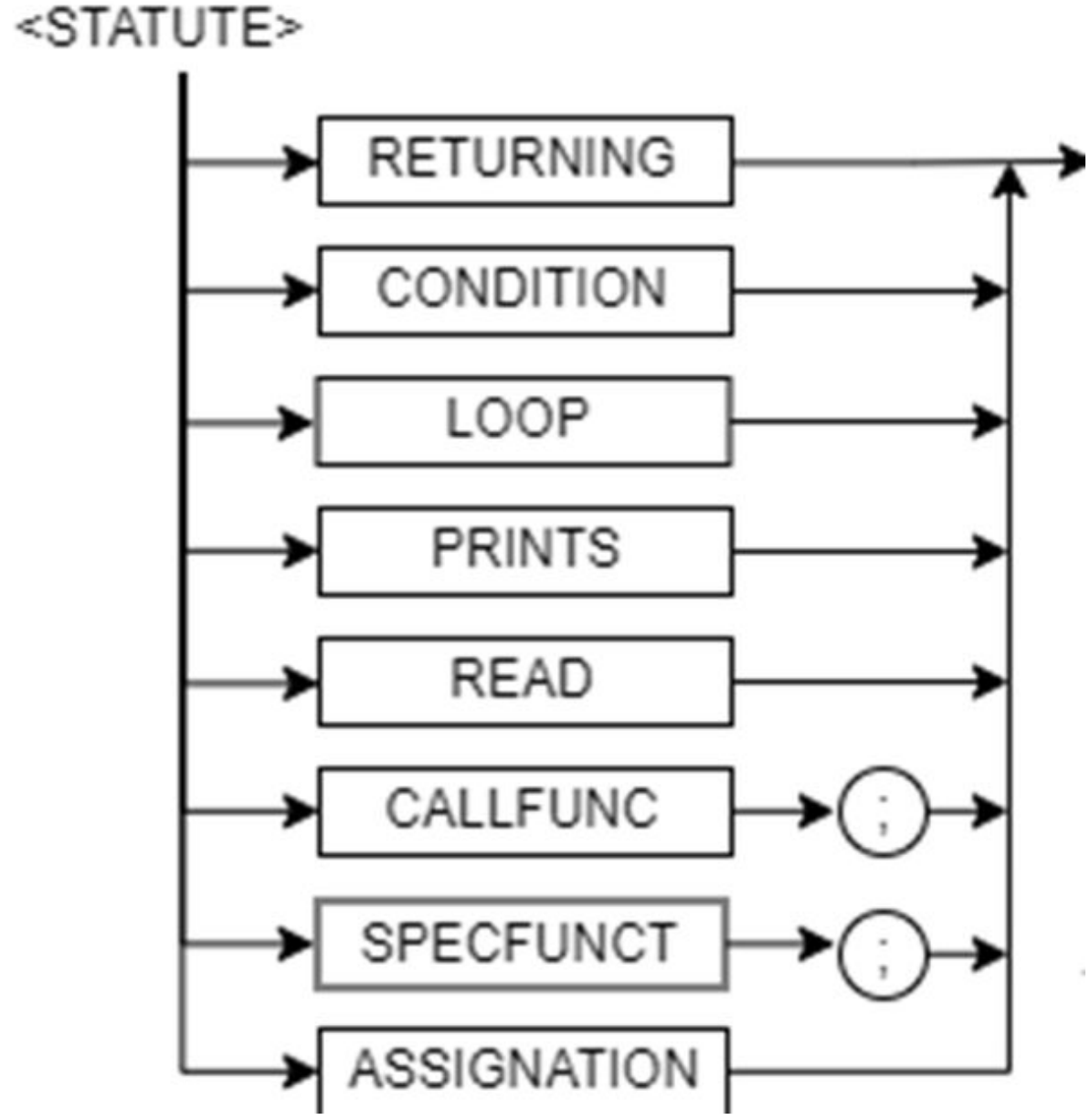

<TYPE>
int
float
color
bool

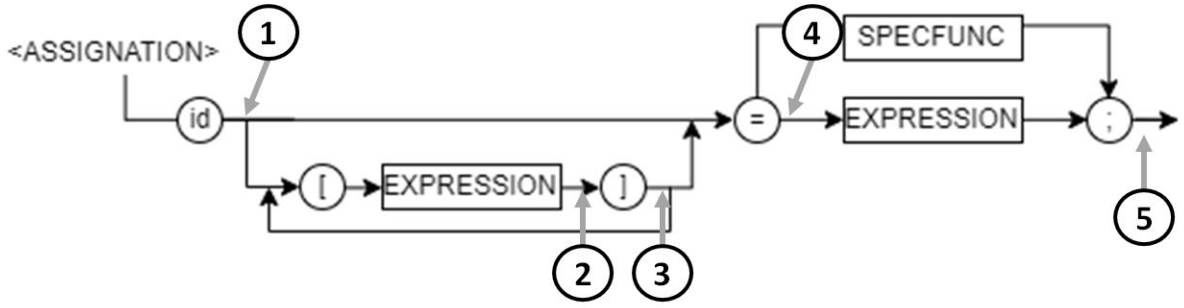


1- Agregar “int” a la pila de tipos
2- Agregar “float” a la pila de tipos
3- Agregar “tag” a la pila de tipos
4-Agregar “bool” a la pila de tipos
5-Agregar “color” a la pila de tipos



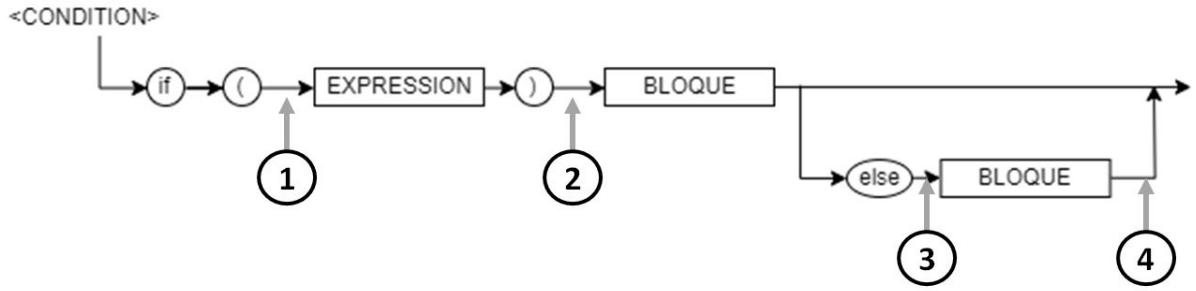
1. Agregar print a la pila de operadores,meter tope a la pila de operadores.
2. Sacar el resultado de la expresión de la pila de operandos y hacer pop a pila de tipos.
    a. Generar cuádruplo “Print _ _ exp”
3. Sacar el tope de la pila de operadores.



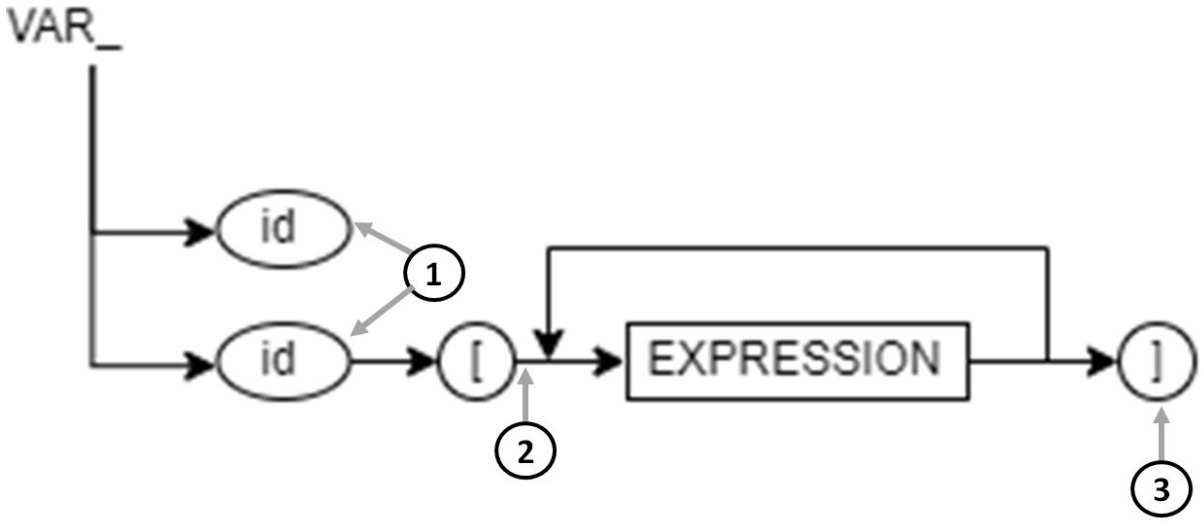


1- Meter id a la pila de operadores, verificando antes que exista.
2-Validar que la expresión resulte con valor de tipo entero. Meter dimensión a la pila de dimensiones, calcular sus cuádruplos correspondientes con base al acceso de variables dimensionadas.

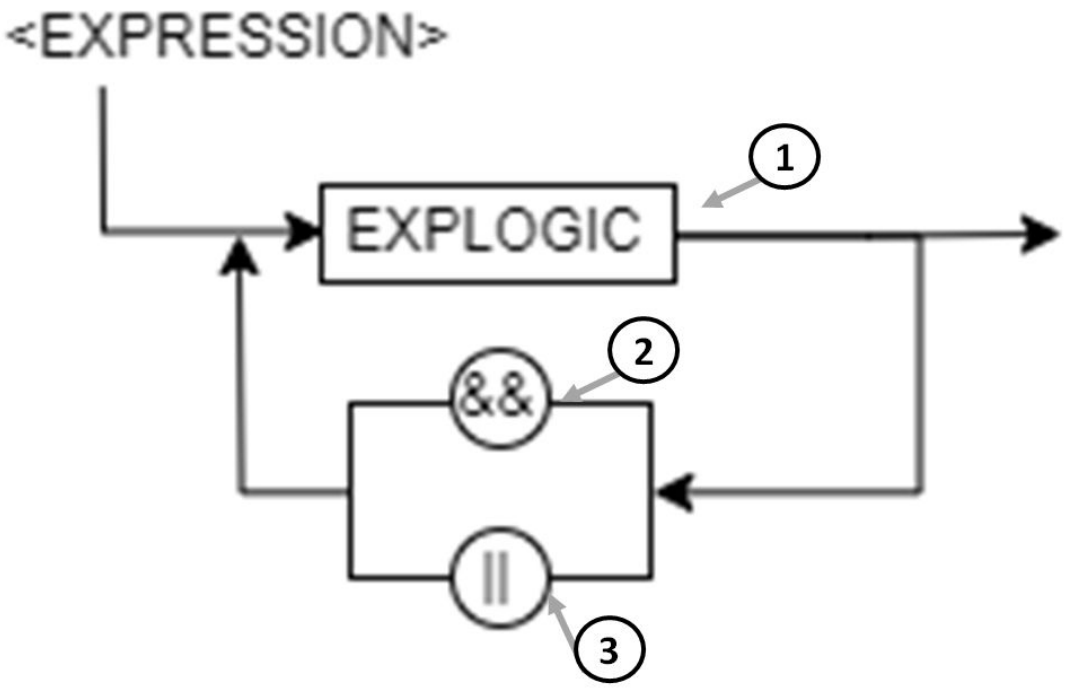
3- Remover las dimensiones de la pila de dimensiones.
4- Agregar “=” a la pila de operadores.
5-Sacar resultado de la expresión de la pila de operandos y su tipo de la pila de tipos.
6-Verificar que los tipos coincidan. Generar cuádruplo de asignación



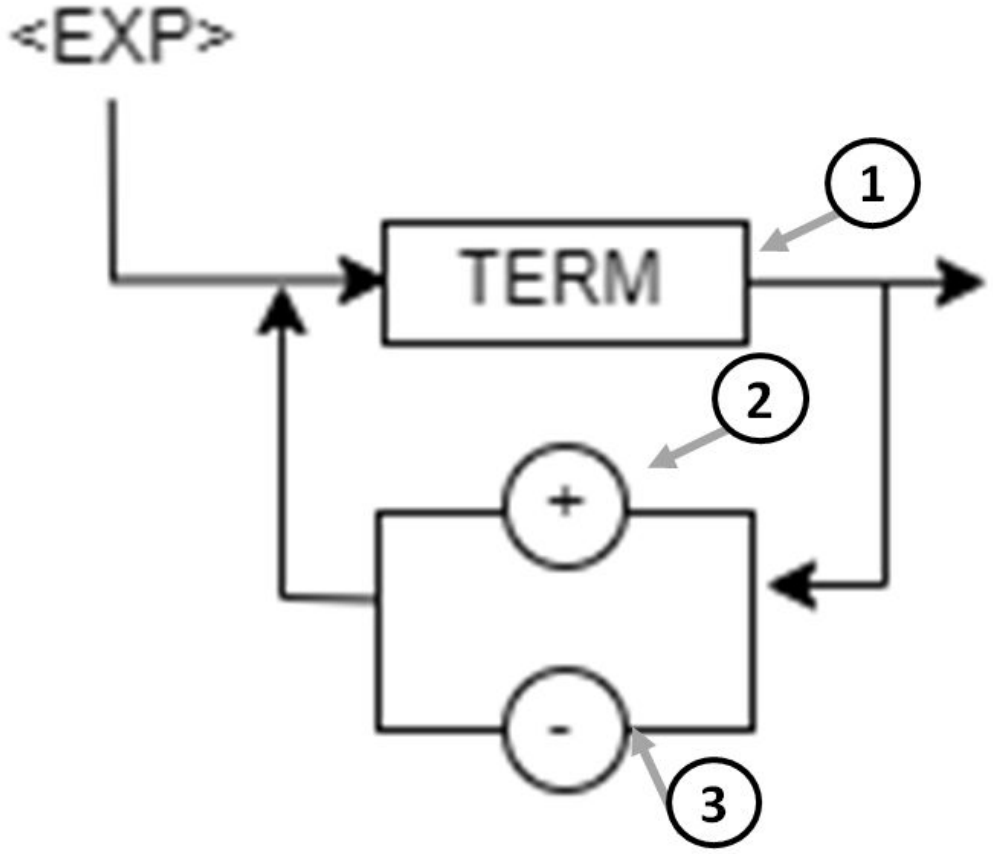
1- Agregar tope a la pila de operadores.
2- Remover tope de la pila de operadores. Expresion = pop.pilaOperadores, tipo=pop.pilaTypos. Verificar que el tipo sea booleano. Agregar contador a pila de saltos y generar GoToF
3-salto=pop.Saltos, agregar contador a pila de saltos, hacer fill(salto)
4-salto=pop.Saltos, fill(salto,contador)



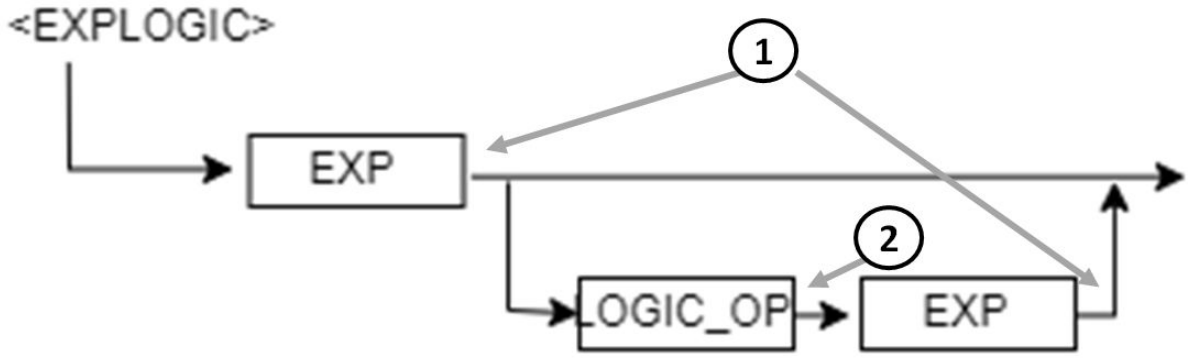
1-Agregar variable a la pila operadores. Verificar que su uso sea adecuado con base a su cantidad de dimensiones .
2-Agregar dimensión a la pila de dimensiones, verificar cantidad de dimensiones y generar cuádruplos de acceso a dimensión.
3-Generar cuádruplo final de acceso a dimensión, verificar dimensiones y hacer pop a la pila de dimensiones hasta que se remueva la variable.



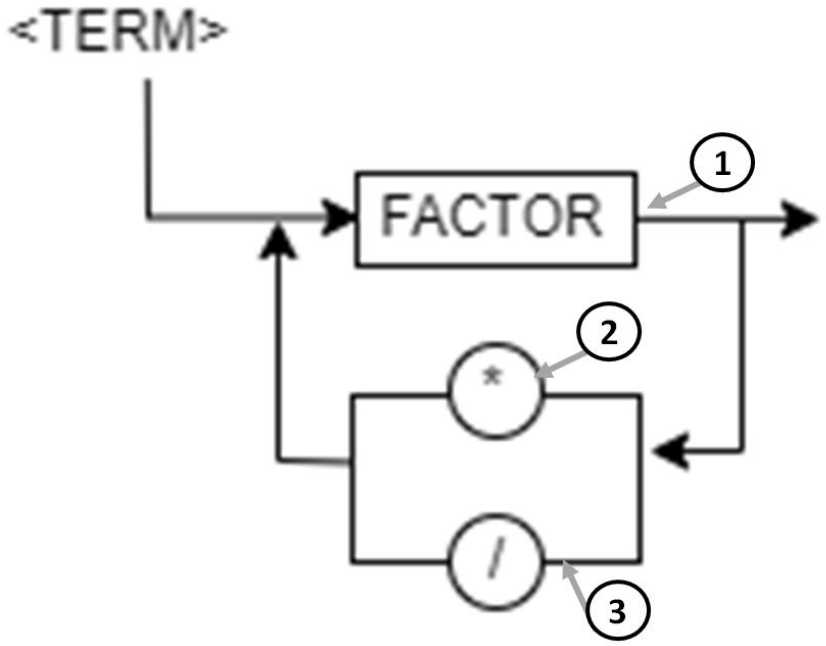
1. Verifica si hay operador disponible en la pila de operadores, si es así: hacer pop a la pila de operadores, hacer pop a pila de operandos, generar cuádruplo binario ej:(&& exp exp temp)
2. Agregar && a la pila de operadores
3. Agregar || a la pila de operadores



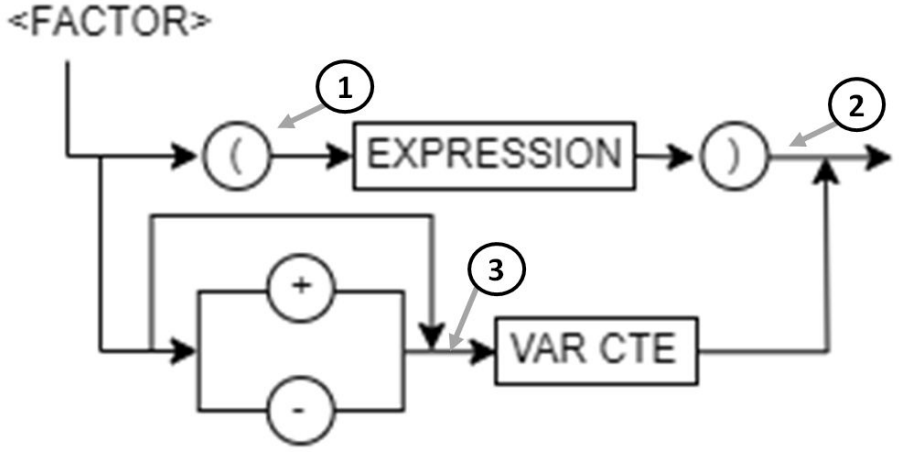
1- Verificar que haya operadores disponibles, si es así: hacer pop a la pila de operadores, hacer pop a pila de operandos, generar cuádruplo binario ej:(+ exp exp temp)
2- Agregar ‘+’ a la pila de operadores
3- Agregar ‘-’ a la pila de operadores



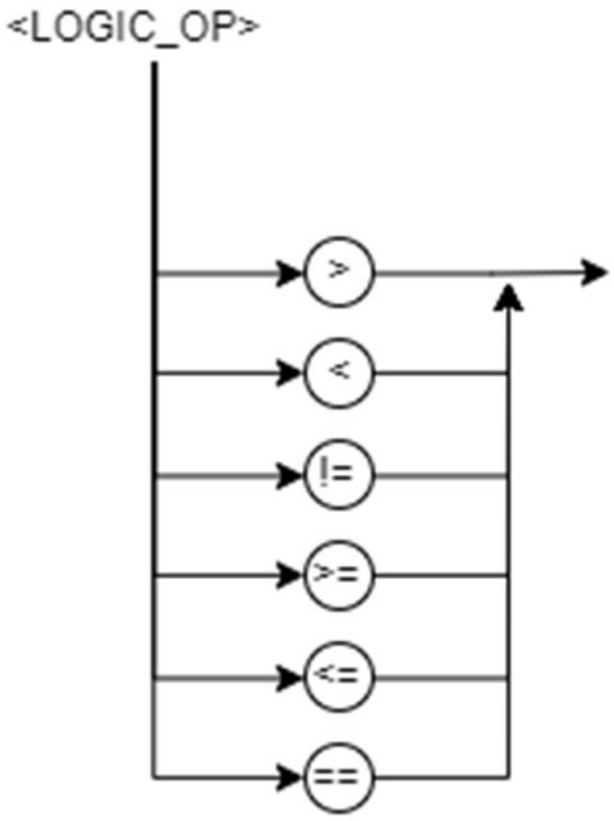
1- Si hay operador disponible, generar cuádruplo binario.
2- Agegar el operador lógico a la pila de operadores

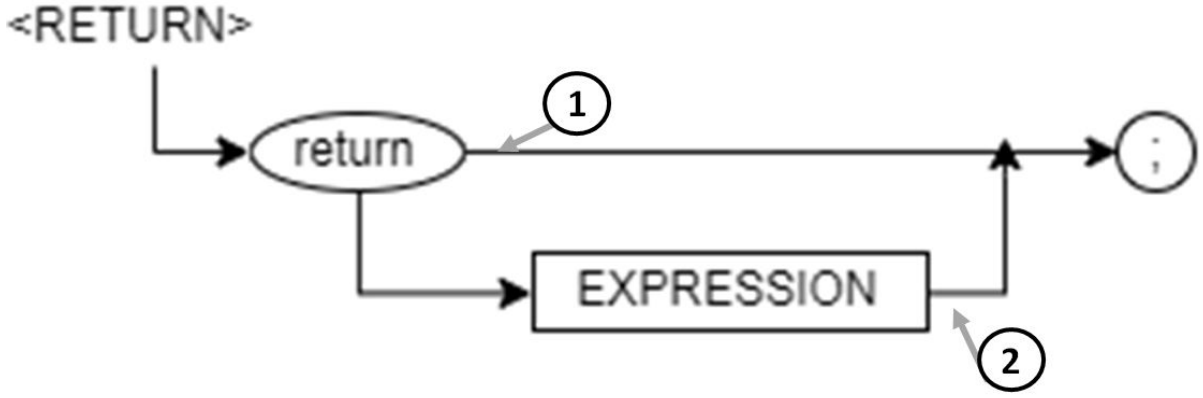


1- Mientras exista operador * o / disponible, hacer pop a la pila de operadores, hacer pop a pila de operandos y generar cuádruplo binario
2-Agregar ‘*’ a la pila de operadores
3- Agregar ‘/’ a la pila de operadores

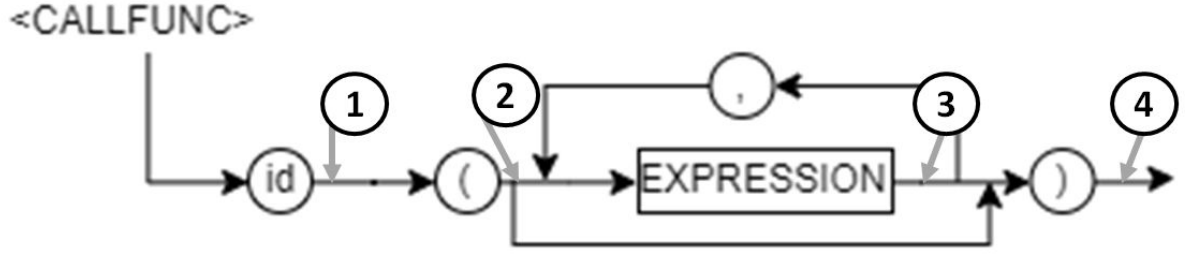


1-Agregar tope ‘(‘ a la pila de operadores
2-Eliminar tope de la pila de operadores
3-Leer signo asignado a VAR_CTE en caso de ser negativo generar cuádruplo que multiplica el resultado por -1





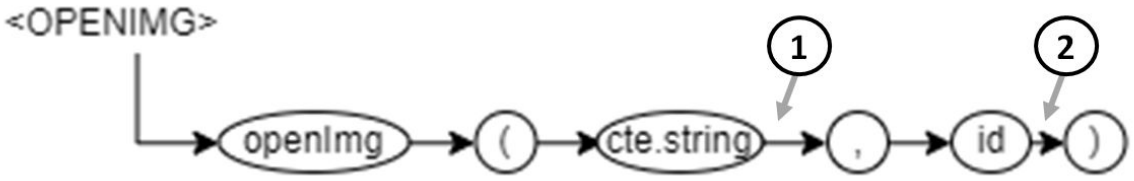
1- Verificar que la función sea de tipo void, si lo es, generar Return, si nó marcar error.
2- Verificar que la función no sea de tipo void, lo es marcar error, si no lo es entonces verificar que el tipo de la expresión corresponda con el tipo de retorno de la función



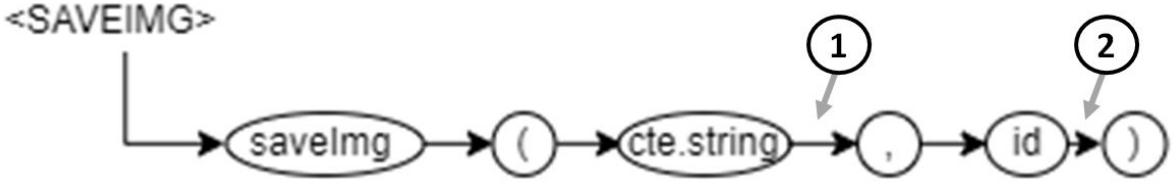
1-Verificar que identificador dado exista y que sea una función. Generar cuádruplo ERA.

2-Agregar Tope a la pila de operadores
3- Obtener resultado de la aplicación haciendo pop a las pilas de operadores y tipos, verificar que la cantidad de expresiones dadas y los tipos correspondan. Generar cuádruplo param
4-Gener cuádruplo GoSub, en caso de se return generar los cuádruplos correspondientes. Elimina el tope de la pila de la pila de operadores.

1- Hacer pop a la pila de tipos, validar que la expresión sea bool, hacer pop a la pila de operando, generar cuádruplo de la evaluación bool, agregar contador-1 a la pila de saltos, generar gotoF.
2-hacer pop pila de saltos = salto, generarGoto => salto, hacer fill(salto,contador)

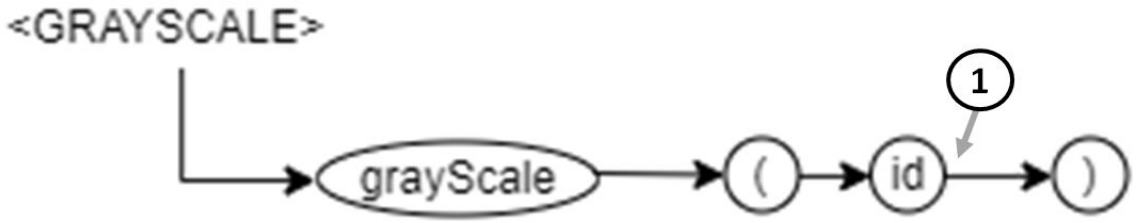


1- Verificar que la expresión dada sea de tipo tag constante.
2- Verificar que la variable dada exista y que sea una variable dimensionada de dos dimensiones.

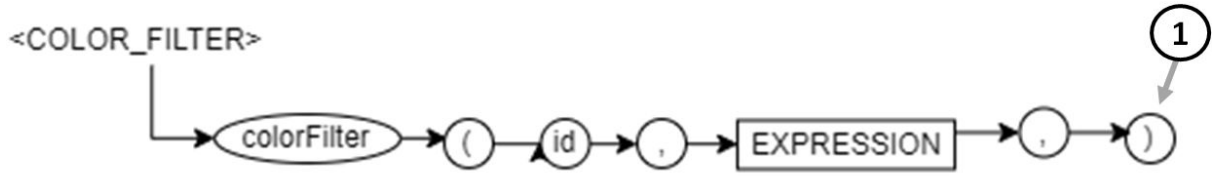


1- Verificar que la expresión dada sea de tipo tag constante.
2- Verificar que la variable dada exista y que sea una variable dimensionada de dos dimensiones de tipo color.

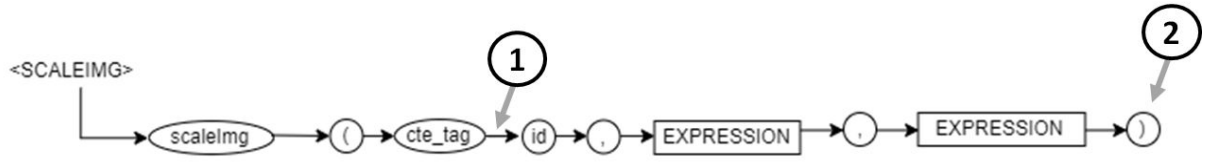
1-Hacer pop a la pila de operadores, validar que ambas expresiones sean de tipo color, y también validar que el identificador exista y sea variable de 2 dimensiones de tipo color.



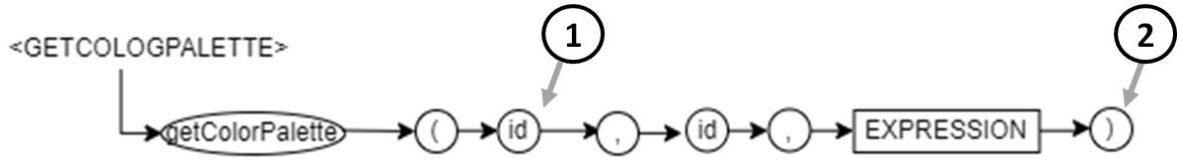
1- Validar que el id exista, sea una variable dimensionada de 2 dimensiones y que sea de tipo color, de no ser así marca error.



1-Hacer pop a la pila de operadores, validar que la expresión sea de tipo color. Validar que el id sea una variable de tipo color de dos dimensiones.



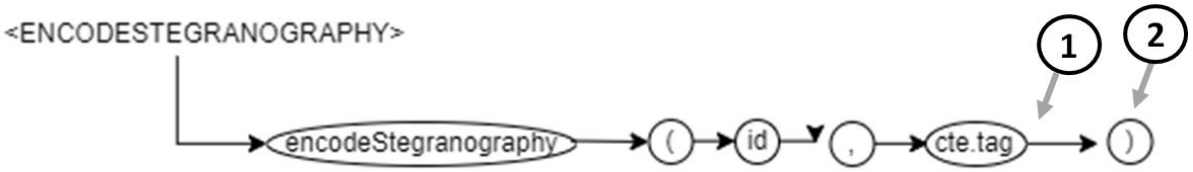
1- Agregar la constante a la pila de operadores, agregar tipo a la pila de tipos.
2- Hacer pops a la pila de tipos y operadores, validar que el se tenga un tag y una variable color de dos dimensiones como parámetros según el orden establecido en la gramática.



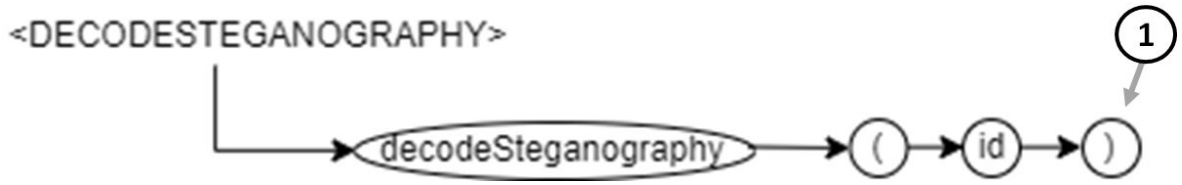
1. Meter el operando a la pila de operandos y su tipo a la pila de tipos.
2. Sacar de las pilas los tipos y direcciones de los parámetros, verificar que el primer parámetro sea de tipo color de dos dimensiones, verificar que el segundo parámetro sea de tipo color y de una dimensión, verificar que el tercer parámetro sea de tipo entero o flotantes de 0 dimensiones.

1. Meter el operando a la pila de operandos y su tipo a la pila de tipos.

2. Sacar de las pilas los tipos y direcciones de los parámetros, verificar que el primer parámetro sea de tipo color de dos dimensiones, verificar que el segundo parámetro sea de tipo color y de dos dimensiones,



1-Meter la constante tag a la pila de operadores y meter tag a la pila de tipos.
2-Verificar que el primer parámetro sea de tipo color y tenga dos dimensiones.



1. Verificar que la variable exista, sea de tipo color y de 2 dimensiones.

3.4.3. Tabla de consideraciones semánticas

Regla semántica	Descripción
Validación de tipos en funciones	Validación semántica en los tipos de los parámetros que recibe la función y en los tipos de los valores al retornar un valor.
Validación de tipos en operaciones aritméticas, comparativas y lógicas.	Con base a un cubo semántico, el compilador determina si es válido realizar la operación según los tipos de los operadores y el operando. También incluyen los valores de retorno de las funciones y las funciones void.
Repetición de nombres de variables y funciones.	El compilador detecta cuando el nombre de una variable o función se está volviendo a utilizar para otra variable o función. Es importante evitar duplicidad de nombres entre variables y funciones debido al manejo de returns mediante la creación de variables globales.
Manejo de variables dimensionadas	El compilador verifica que la cantidad de dimensiones accedidas correspondan con la que fueron declaradas. También comprueba que las variables no dimensionadas no se usen como dimensionadas.

Validación especial: “Tag”	Los tags son un tipo de dato especial que el compilador solamente permite utilizarse en los prints y en algunas funciones especiales para marcar rutas de archivos.

3.5. Descripción detallada del proceso de Administración de Memoria usado en la compilación.

3.5.1. Especificación gráfica de CADA estructura de datos usada (Dir.Funciones, Tablas de Var's, Cuádruplos, etc...)

- Directorio de funciones:
  - El índice del diccionario contiene el nombre de la función.
  - Type: Guarda el tipo de la función, eso incluye el tipo void
  - Position: indica la posición en el cuádruplo en donde se encuentra
  - Parameters: Un arreglo de tuplas las cuales están compuestas de 3 elementos cada una (nombreParámetro,tipo del parámetro, dirección local de memoria)
  - Vars: diccionario que indica el cada uno de los tipos de variables seguido de solamente la cantidad de variables utilizadas en la función.
  - Temps: diccionario que indica el cada uno de los tipos de variables seguido de solamente la cantidad de temporales utilizadas en la función.
- Tablas de variables: Este concepto se aplica tanto a las tablas de variables de cada función, globales y constantes
  - El índice del diccionario pertenece al nombre de la variable/constante
  - type: indica el tipo de la variable/constante
  - Dir: indica la dirección de memoria relativa que le pertenece.
  - Dim: si la variable es dimensionada contará con una lista del tamaño de la cantidad de dimensiones que indica:
    - ls: límite superior de la dimensión
    - m: valor Mi de la dimensión, en caso de ser el último, el valor de -k
- Cuádruplos:
  - Está compuesto de 4 valores, en orden de izquierda a derecha.
    - Operador: Indica la acción que debe de realizar la máquina virtual
    - OperandoIzq: Generalmente indica el operador izquierdo de para el operador, en otros casos mantiene direcciones de apoyo.
    - OperandoDer: indica el operador derecho.
    - Resultado: Generalmente indica el resultado de la operación.
- Memory Manager: Utilizado por el compilador, ayuda a contar a cantidad de variables que se van declarando ya sea globales, constantes y temporales.
  - Bloque: Llave que indica el Scope (Local,Global,Constantes,Temporales)

- El bloque contiene otro diccionario donde contiene cada uno de los tipos de variable:
  - Cada variable tiene su valor de memoria
  - Declared: Cantidad de variables declaradas al momento, este valor se reinicia en la variables locales y temporales al compilar una nueva función.

4. Descripción de la Máquina Virtual

4.1. Equipo de cómputo, lenguaje y utilerías especiales usadas (en caso de ser diferente que el compilador).

El equipo de cómputo y lenguaje es el mísmo que el usado en el compilador. Las librerías agregadas se usaron solamente para las funciones especiales:

- scikit image (https://scikit-image.org/)
- cologram (https://github.com/obskyr/colorgram.py)
- Numpy (https://numpy.org/)
- Pillow (Manejo de imágenes/archivos)

4.2. Descripción detallada del proceso de Administración de Memoria en ejecución (Arquitectura)

4.2.1. Especificación gráfica de CADA estructura de datos usada para manejo de scopes (Memoria Local, global, etc..)

- Tabla de Referencia:
  - La construye la máquina virtual con base al tamaño de la memoria que le advirtió el compilador.
  - Genera un diccionario y coloca como llave la posición inicial de cada bloque de memoria.
  - Cada valor indica el Scope al que pertenece esa dirección y el tipo de variable.
  - Por ejemplo: en el diccionario la llave “1000000” regresa como Scope: Global y como tipo “int”.
- Virtual Memory: Tabla principal de la máquina virtual.
  - Se divide en 3 bloques principales; Globales, Constantes y Stack.
  - Cada bloque cuenta con su propio grupo de variables separadas por tipo.
    - Globals: { int:[],float:[],color:[],bool:[]}, Constants{...}, Stack[{},{}]
  - El bloque Stack es una pila que contiene cada bloque de memoria que se genera con cada ERA y se va llenando como una pila con un bloque nuevo por cada llamada a una función incluyendo las llamadas recursivas. Se libera al terminar la ejecución de esa llamada de la función.

- Cada entrada de los tipos está compuesto de un arreglo de valores correspondiente a sus tipos, el tamaño de los arreglos se define con base a los tamaños generados en compilación y solamente crea el espacio en memoria necesario.

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

Cuando la máquina virtual lee una dirección primero manda a llamar a la Tabla de Referencias para saber en qué Scope se encuentra esa dirección y a qué tipo pertenece. Si el Scope de la dirección es Global o Constante, la máquina virtual accede directamente a esos bloques. Para saber cómo indexar al arreglo del tipo encontrado se saca el residuo de la dirección accedida entre la dirección inicial del bloque del tipo. Luego se usa el residuo para iterar sobre el arreglo de la memoria y de esa manera se obtiene o modifica el valor.
Para los bloques temporales y locales: Gracias a la tabla reference es posible saber qué direcciones de memoria pertenecen a esta sección de la memoria. Una vez identificada la ubicación la máquina virtual accede al stack de memoria y seleccionará al tope de la pila como memoria activa. Esa memoria activa contiene las direcciones encontradas en el cuádruplo, se realiza el cálculo del residuo para acceder a la posición del arreglo en donde se encuentra el valor de la dirección.

4.3. Pruebas del Funcionamiento del Lenguaje

4.3.1. Incluir pruebas que "comprueben” el funcionamiento del proyecto

4.3.1.1. Codificación de la prueba (en su lenguaje)

```
//Fibonacci iterativo
func main(){
   define int n1=0,n2=1,num;
   define int i=0;
   print("Intrucir numero de elementos de la secuencia:");
   read(num);
   num = num-2;
   print(n1,n2);

   while(i<num){
      n2 = n1 + n2;
      n1 = n2 - n1;
      print(n2);
      i = i+1;
   }
}
```

```
//Fibonacci recursivo
```

```
func int fibo(int n){
   if(n==0 || n == 1){
      return n;
   }else{
      return fibo(n-2) + fibo(n-1);
   }
}

func main(){
   define int i=0, num;
   num = -1;

   //FIBONACCI RECURSIVO
   while(num<0){
      print("Introducir numero de digitos de la serie a mostrar:");
      read(num);
   }

   while(i<num){
      print(fibo( i));
      i = i+1;
   }
}
```

```
//Multiplicación de Matrices
func main(){
   define int i=0,j=0,k=0;
   define int mat1[3][3],mat2[3][3],res[3][3];

   mat1[0][0] = 1;
   mat1[0][1] = 2;
   mat1[0][2] = 3;
   mat1[1][0] = 4;
   mat1[1][1] = 5;
   mat1[1][2] = 6;
   mat1[2][0] = 7;
   mat1[2][1] = 8;
   mat1[2][2] = 9;

   mat2[0][0] = 1;
   mat2[0][1] = 2;
   mat2[0][2] = 3;
   mat2[1][0] = 4;
   mat2[1][1] = 5;
   mat2[1][2] = 6;
   mat2[2][0] = 7;
   mat2[2][1] = 8;
   mat2[2][2] = 9;
```

```
   while(i<3){
      j=0;
      while(j<3){
         res[i][j] = 0;
         k=0;
         while(k<3){
            res[i][j] = res[i][j] + mat1[i][k]*mat2[k][j];
            k=k+1;
         }
         j=j+1;
      }
      i = i+1;
   }
   i=0;
   j=0;
   while(i<3){
      j=0;
      while(j<3){
         print(res[i][j]);
         j=j+1;
      }
      i=i+1;
   }
}
```

```
//Bubble Sort
func main(){
   define int x[10], i=0,j=0;
   define int temp;
   print("Introducir valores del arreglo");
   while(i<10){
      read(x[i]);
      i = i+1;
   }
   i=0;
   while(i<9){
      while(j<9-i){
         if(x[j] > x[j+1]){
            temp = x[j];
            x[j] = x[j+1];
            x[j+1] = temp;
         }
         j =j+1;
      }
      i=i+1;
      j=0;
```

```
   }
   i = 0;
   print("Arreglo ordenado");
   while(i<10){
      print(x[i]);
      i = i+1;
   }
}
```

```
//Búsqueda en arreglos
func main(){
   define int x[5], y,i=0;
   define bool encontrado = False;
   x[0] = 1;
   x[1] = 2;
   x[2] = 3;
   x[3] = 4;
   x[4] = 5;
   print("Introducir el valor a buscar");
   read(y);
   while(i<5){
      if(y==x[i]){
         print("Encontrado en la posicion: ",i);
         encontrado = True;
      }
      i=i+1;
   }
   if(encontrado == False){
      print("Valor no encontrado");
   }
}
```

```
//Factoriaes
```

```
func int factorial(int n)
{
  if (n == 0)
  {
    return 1;
  }
  else
  {
    return(n * factorial(n-1));
  }
}

func int iterativeFactorial(int n)
{
```

```
   define int fact = 1;
   define int num, i;
   num = n;
   i = num;
   while(i>=1)
   {
     fact=fact*i;
     i = i -1;
   }
   return fact;
}

func main(){
   print(factorial(5));
   print(iterativeFactorial(5));
}
```

```
define color c[200][200];
define color match[100][100];
define color cPal[5];

func main(){
define int x=0;
   openImg("media/image1.png",c);
   grayscale(c);
   saveImg("grayscale.png",c);

   openImg("media/image1.png",c);
   colorReplace(c,#000000,#ffffff);
   saveImg("replaced.png",c);

   openImg("media/image1.png",c);
   colorFilter(c,#1b5cc8);
   saveImg("filtered.png",c);

   openImg("media/image1.png",c);
   edgeDetection(c);
   saveImg("edged.png",c);

   openImg("media/image1.png",c);
   saveScaleImg("scaled.png",c,-1,-1);

   openImg("media/image1.png",c);
   getColorPalette(c,cPal,5);
   print(cPal[0],cPal[1],cPal[2],cPal[3],cPal[4]);

   openImg("media/image1.png",c);
   openImg("media/image4.png",match);
```

```
   colorMatchImage(c,match);
   saveImg("matched.png",c);

   openImg("media/image1.png",c);
   encodeSteganography(c,"Este es un mensaje encriptado");
   saveImg("encriptado.png",c);

   openImg("encriptado.png",c);
   decodeSteganography(c);
}
```

4.3.1.2. Resultados arrojados por la generación de código intermedio y por la ejecución

```
Factorial Recursivo e Iterativo
1 GOTO None None 26
2 == 60000000 160000000 130000000
3 GOTOF 130000000 None 6
4 RETURN None None 160000001
5 GOTO None None 13
6 ERA factorial None None
7 - 60000000 160000001 110000000
8 param 110000000 None param1
9 gosub factorial None None
10 = 10000000 None 110000001
11 * 60000000 110000001 110000002
12 RETURN None None 110000002
13 ENDPROC None None None
14 = 160000001 None 60000001
15 = 60000000 None 60000002
16 = 60000002 None 60000003
17 >= 60000003 160000001 130000000
18 GOTOF 130000000 None 24
19 * 60000001 60000003 110000000
20 = 110000000 None 60000001
21 - 60000003 160000001 110000001
22 = 110000001 None 60000003
23 GOTO None None 17
24 RETURN None None 60000001
25 ENDPROC None None None
26 ERA factorial None None
27 param 160000002 None param1
28 gosub factorial None None
29 = 10000000 None 110000000
30 print None None 110000000
```

```
31 ERA iterativeFactorial None None
32 param 160000002 None param1
33 gosub iterativeFactorial None None
34 = 10000001 None 110000001
35 print None None 110000001
36 end None None None
Factorial 5: 120
Factorial 5: 120
```

```
Fibonacci Recursivo
1 GOTO None None 21
2 == 60000000 160000000 130000000
3 == 60000000 160000001 130000001
4 || 130000000 130000001 130000002
5 GOTOF 130000002 None 8
6 RETURN None None 60000000
7 GOTO None None 20
8 ERA fibo None None
9 - 60000000 160000002 110000000
10 param 110000000 None param1
11 gosub fibo None None
12 = 10000000 None 110000001
13 ERA fibo None None
14 - 60000000 160000001 110000002
15 param 110000002 None param1
16 gosub fibo None None
17 = 10000000 None 110000003
18 + 110000001 110000003 110000004
19 RETURN None None 110000004
20 ENDPROC None None None
21 = 160000000 None 60000000
22 * 160000003 160000001 110000000
23 = 110000000 None 60000001
24 < 60000001 160000000 130000000
25 GOTOF 130000000 None 29
26 print None None 200000000
27 read None None 60000001
28 GOTO None None 24
29 < 60000000 60000001 130000001
30 GOTOF 130000001 None 39
31 ERA fibo None None
32 param 60000000 None param1
33 gosub fibo None None
34 = 10000000 None 110000001
35 print None None 110000001
36 + 60000000 160000001 110000002
37 = 110000002 None 60000000
```

```
38 GOTO None None 29
39 end None None None
Introducir numero de digitos de la serie a mostrar:
6
0
1
1
2
3
5
```

```
Fibonacci Iterativo
1 GOTO None None 2
2 = 160000000 None 60000000
3 = 160000001 None 60000001
4 = 160000000 None 60000003
5 print None None 200000000
6 read None None 60000002
7 - 60000002 160000002 110000000
8 = 110000000 None 60000002
9 print None None 60000000
10 print None None 60000001
11 < 60000003 60000002 130000000
12 GOTOF 130000000 None 21
13 + 60000000 60000001 110000001
14 = 110000001 None 60000001
15 - 60000001 60000000 110000002
16 = 110000002 None 60000000
17 print None None 60000001
18 + 60000003 160000001 110000003
19 = 110000003 None 60000003
20 GOTO None None 11
21 end None None None
Intrucir numero de elementos de la secuencia:
10
0
1
1
2
3
5
8
13
21
34
```

Multiplicación de Matrices

```
1 GOTO None None 2
2 = 160000000 None 60000000
3 = 160000000 None 60000001
4 = 160000000 None 60000002
5 VER 160000000 0 2
6 * 160000000 160000001 110000000
7 VER 160000000 0 2
8 + 110000000 160000000 110000001
9 + 110000001 160000002 110000002
10 = 160000003 None (110000002)
11 VER 160000000 0 2
12 * 160000000 160000001 110000003
13 VER 160000003 0 2
14 + 110000003 160000003 110000004
15 + 110000004 160000002 110000005
16 = 160000004 None (110000005)
17 VER 160000000 0 2
18 * 160000000 160000001 110000006
19 VER 160000004 0 2
20 + 110000006 160000004 110000007
21 + 110000007 160000002 110000008
22 = 160000001 None (110000008)
23 VER 160000003 0 2
24 * 160000003 160000001 110000009
25 VER 160000000 0 2
26 + 110000009 160000000 110000010
27 + 110000010 160000002 110000011
28 = 160000005 None (110000011)
29 VER 160000003 0 2
30 * 160000003 160000001 110000012
31 VER 160000003 0 2
32 + 110000012 160000003 110000013
33 + 110000013 160000002 110000014
34 = 160000006 None (110000014)
35 VER 160000003 0 2
36 * 160000003 160000001 110000015
37 VER 160000004 0 2
38 + 110000015 160000004 110000016
39 + 110000016 160000002 110000017
40 = 160000007 None (110000017)
41 VER 160000004 0 2
42 * 160000004 160000001 110000018
43 VER 160000000 0 2
44 + 110000018 160000000 110000019
45 + 110000019 160000002 110000020
46 = 160000008 None (110000020)
47 VER 160000004 0 2
48 * 160000004 160000001 110000021
```

```
49 VER 160000003 0 2
50 + 110000021 160000003 110000022
51 + 110000022 160000002 110000023
52 = 160000009 None (110000023)
53 VER 160000004 0 2
54 * 160000004 160000001 110000024
55 VER 160000004 0 2
56 + 110000024 160000004 110000025
57 + 110000025 160000002 110000026
58 = 160000010 None (110000026)
59 VER 160000000 0 2
60 * 160000000 160000001 110000027
61 VER 160000000 0 2
62 + 110000027 160000000 110000028
63 + 110000028 160000011 110000029
64 = 160000003 None (110000029)
65 VER 160000000 0 2
66 * 160000000 160000001 110000030
67 VER 160000003 0 2
68 + 110000030 160000003 110000031
69 + 110000031 160000011 110000032
70 = 160000004 None (110000032)
71 VER 160000000 0 2
72 * 160000000 160000001 110000033
73 VER 160000004 0 2
74 + 110000033 160000004 110000034
75 + 110000034 160000011 110000035
76 = 160000001 None (110000035)
77 VER 160000003 0 2
78 * 160000003 160000001 110000036
79 VER 160000000 0 2
80 + 110000036 160000000 110000037
81 + 110000037 160000011 110000038
82 = 160000005 None (110000038)
83 VER 160000003 0 2
84 * 160000003 160000001 110000039
85 VER 160000003 0 2
86 + 110000039 160000003 110000040
87 + 110000040 160000011 110000041
88 = 160000006 None (110000041)
89 VER 160000003 0 2
90 * 160000003 160000001 110000042
91 VER 160000004 0 2
92 + 110000042 160000004 110000043
93 + 110000043 160000011 110000044
94 = 160000007 None (110000044)
95 VER 160000004 0 2
96 * 160000004 160000001 110000045
```

```
97 VER 160000000 0 2
98 + 110000045 160000000 110000046
99 + 110000046 160000011 110000047
100 = 160000008 None (110000047)
101 VER 160000004 0 2
102 * 160000004 160000001 110000048
103 VER 160000003 0 2
104 + 110000048 160000003 110000049
105 + 110000049 160000011 110000050
106 = 160000009 None (110000050)
107 VER 160000004 0 2
108 * 160000004 160000001 110000051
109 VER 160000004 0 2
110 + 110000051 160000004 110000052
111 + 110000052 160000011 110000053
112 = 160000010 None (110000053)
113 < 60000000 160000001 130000000
114 GOTOF 130000000 None 159
115 = 160000000 None 60000001
116 < 60000001 160000001 130000001
117 GOTOF 130000001 None 156
118 VER 60000000 0 2
119 * 60000000 160000001 110000054
120 VER 60000001 0 2
121 + 110000054 60000001 110000055
122 + 110000055 160000012 110000056
123 = 160000000 None (110000056)
124 = 160000000 None 60000002
125 < 60000002 160000001 130000002
126 GOTOF 130000002 None 153
127 VER 60000000 0 2
128 * 60000000 160000001 110000057
129 VER 60000001 0 2
130 + 110000057 60000001 110000058
131 + 110000058 160000012 110000059
132 VER 60000000 0 2
133 * 60000000 160000001 110000060
134 VER 60000001 0 2
135 + 110000060 60000001 110000061
136 + 110000061 160000012 110000062
137 VER 60000000 0 2
138 * 60000000 160000001 110000063
139 VER 60000002 0 2
140 + 110000063 60000002 110000064
141 + 110000064 160000002 110000065
142 VER 60000002 0 2
143 * 60000002 160000001 110000066
144 VER 60000001 0 2
```

```
145 + 110000066 60000001 110000067
146 + 110000067 160000011 110000068
147 * (110000065) (110000068) 110000069
148 + (110000062) 110000069 110000070
149 = 110000070 None (110000059)
150 + 60000002 160000003 110000071
151 = 110000071 None 60000002
152 GOTO None None 125
153 + 60000001 160000003 110000072
154 = 110000072 None 60000001
155 GOTO None None 116
156 + 60000000 160000003 110000073
157 = 110000073 None 60000000
158 GOTO None None 113
159 = 160000000 None 60000000
160 = 160000000 None 60000001
161 < 60000000 160000001 130000003
162 GOTOF 130000003 None 178
163 = 160000000 None 60000001
164 < 60000001 160000001 130000004
165 GOTOF 130000004 None 175
166 VER 60000000 0 2
167 * 60000000 160000001 110000074
168 VER 60000001 0 2
169 + 110000074 60000001 110000075
170 + 110000075 160000012 110000076
171 print None None (110000076)
172 + 60000001 160000003 110000077
173 = 110000077 None 60000001
174 GOTO None None 164
175 + 60000000 160000003 110000078
176 = 110000078 None 60000000
177 GOTO None None 161
178 end None None None
30
36
42
66
81
96
102
126
150
```

```
Búsqueda de arreglos
1 GOTO None None 2
2 = 160000000 None 60000006
3 = 180000000 None 80000000
```

```
4 VER 160000000 0 4
5 + 160000000 160000001 110000000
6 = 160000002 None (110000000)
7 VER 160000002 0 4
8 + 160000002 160000001 110000001
9 = 160000003 None (110000001)
10 VER 160000003 0 4
11 + 160000003 160000001 110000002
12 = 160000004 None (110000002)
13 VER 160000004 0 4
14 + 160000004 160000001 110000003
15 = 160000005 None (110000003)
16 VER 160000005 0 4
17 + 160000005 160000001 110000004
18 = 160000006 None (110000004)
19 print None None 200000000
20 read None None 60000005
21 < 60000006 160000006 130000000
22 GOTOF 130000000 None 33
23 VER 60000006 0 4
24 + 60000006 160000001 110000005
25 == 60000005 (110000005) 130000001
26 GOTOF 130000001 None 30
27 print None None 200000001
28 print None None 60000006
29 = 180000001 None 80000000
30 + 60000006 160000002 110000006
31 = 110000006 None 60000006
32 GOTO None None 21
33 == 80000000 180000000 130000002
34 GOTOF 130000002 None 36
35 print None None 200000002
36 end None None None
Introducir el valor a buscar
3
Encontrado en la posicion:
2
```

```
Bubble Sort
1 GOTO None None 2
2 = 160000000 None 60000010
3 = 160000000 None 60000011
4 print None None 200000000
5 < 60000010 160000001 130000000
6 GOTOF 130000000 None 13
7 VER 60000010 0 9
8 + 60000010 160000002 110000000
9 read None None (110000000)
```

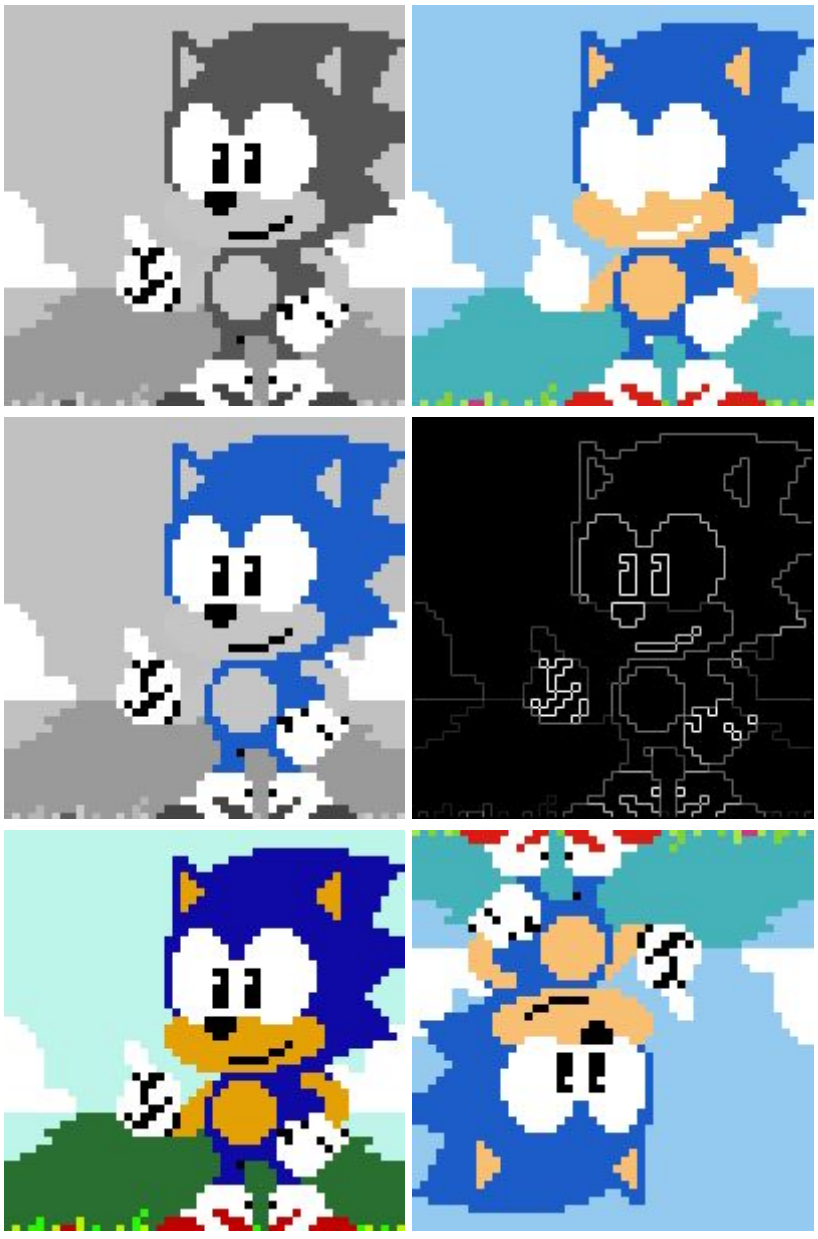
```
10 + 60000010 160000003 110000001
11 = 110000001 None 60000010
12 GOTO None None 5
13 = 160000000 None 60000010
14 < 60000010 160000004 130000001
15 GOTOF 130000001 None 46
16 - 160000004 60000010 110000002
17 < 60000011 110000002 130000002
18 GOTOF 130000002 None 42
19 VER 60000011 0 9
20 + 60000011 160000002 110000003
21 + 60000011 160000003 110000004
22 VER 110000004 0 9
23 + 110000004 160000002 110000005
24 > (110000003) (110000005) 130000003
25 GOTOF 130000003 None 39
26 VER 60000011 0 9
27 + 60000011 160000002 110000006
28 = (110000006) None 60000012
29 VER 60000011 0 9
30 + 60000011 160000002 110000007
31 + 60000011 160000003 110000008
32 VER 110000008 0 9
33 + 110000008 160000002 110000009
34 = (110000009) None (110000007)
35 + 60000011 160000003 110000010
36 VER 110000010 0 9
37 + 110000010 160000002 110000011
38 = 60000012 None (110000011)
39 + 60000011 160000003 110000012
40 = 110000012 None 60000011
41 GOTO None None 17
42 + 60000010 160000003 110000013
43 = 110000013 None 60000010
44 = 160000000 None 60000011
45 GOTO None None 14
46 = 160000000 None 60000010
47 print None None 200000001
48 < 60000010 160000001 130000004
49 GOTOF 130000004 None 56
50 VER 60000010 0 9
51 + 60000010 160000002 110000014
52 print None None (110000014)
53 + 60000010 160000003 110000015
54 = 110000015 None 60000010
55 GOTO None None 48
56 end None None None
Introducir valores del arreglo
```

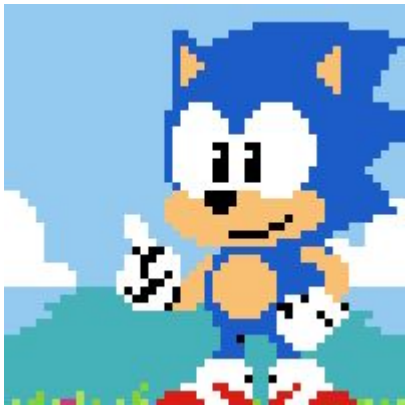
```
8
2
64
72
17
73
2
8
4
1
Arreglo ordenado
1
2
2
4
8
8
17
64
72
73
```

```
1 GOTO None None 2
2 = 160000000 None 60000000
3 openimg 200000000 None 40000000
4 openimgData 200 200 None
5 grayscale 200 200 40000000
6 saveimg 200000001 None 40000000
7 saveimgData 200 200 None
8 openimg 200000000 None 40000000
9 openimgData 200 200 None
10 colorReplace 190000000 190000001 40000000
11 colorReplaceData 200 200 None
12 saveimg 200000002 None 40000000
13 saveimgData 200 200 None
14 openimg 200000000 None 40000000
15 openimgData 200 200 None
16 colorFilter 190000002 None 40000000
17 colorFilterData 200 200 None
18 saveimg 200000003 None 40000000
19 saveimgData 200 200 None
20 openimg 200000000 None 40000000
21 openimgData 200 200 None
22 edgeDetection 200 200 40000000
23 saveimg 200000004 None 40000000
24 saveimgData 200 200 None
25 openimg 200000000 None 40000000
26 openimgData 200 200 None
```

```
27 * 160000001 160000002 110000000
28 * 160000001 160000002 110000001
29 scaleImg 110000000 110000001 40000000
30 scaleImgData 200 200 200000005
31 openimg 200000000 None 40000000
32 openimgData 200 200 None
33 getColorPalette 160000003 5 40050000
34 getColorPaletteData 200 200 40000000
35 VER 160000000 0 4
36 + 160000000 160000004 110000002
37 print None None (110000002)
38 VER 160000002 0 4
39 + 160000002 160000004 110000003
40 print None None (110000003)
41 VER 160000005 0 4
42 + 160000005 160000004 110000004
43 print None None (110000004)
44 VER 160000006 0 4
45 + 160000006 160000004 110000005
46 print None None (110000005)
47 VER 160000007 0 4
48 + 160000007 160000004 110000006
49 print None None (110000006)
50 openimg 200000000 None 40000000
51 openimgData 200 200 None
52 openimg 200000006 None 40040000
53 openimgData 100 100 None
54 colorMatchImage 200 200 40000000
55 colorMatchImageData 100 100 40040000
56 saveimg 200000007 None 40000000
57 saveimgData 200 200 None
58 openimg 200000000 None 40000000
59 openimgData 200 200 None
60 encodeSteganography 200000008 None 40000000
61 encodeSteganographyData 200 200 None
62 saveimg 200000009 None 40000000
63 saveimgData 200 200 None
64 openimg 200000009 None 40000000
65 openimgData 200 200 None
66 decodeStenography 200 200 40000000
67 end None None None
Image saved as : grayscale.png
Image saved as : replaced.png
Image saved as : filtered.png
Image saved as : edged.png
Image saved as : scaled.png
#98ccf0
#1f5bc0
```

```
#4fb1bf
#fbf6ef
#f3c178
Image saved as : matched.png
Image saved as : encriptado.png
The encryped message is:  Este es un mensaje encriptado
```





4.3.2. Listados Perfectamente Documentados del Proyecto

4.3.2.1. Incluir comentarios de Documentación, es decir: para cada módulo, una pequeña explicación de qué hace, qué parámetros recibe, qué genera como salida y cuáles son los módulos más importantes que hacen uso de él.

4.3.2.2. Dentro de los módulos principales se esperan comentarios de Implementación, es decir: pequeña descripción de cuál es la función de algún estatuto que sea importante de ese módulo.

```
Código para encriptar y des encriptar una imagen con base a sus
pixeles
def encode(data,message):
    messageCounter = 0
    data = np.array(data)
    char = tobits(message[0])
    bitCounter = 0;
    for row in range(data.shape[0]):
        for cell in range(data.shape[1]):
            for color in range(3):
                if(bitCounter==8):
                    bitCounter = -1
                    messageCounter+=1
                    if(messageCounter>=len(message)):
                        if(data[row][cell][color]%2==0):
                            data[row][cell][color] -= 1
                            return(data)
                    else:
                        if(data[row][cell][color]%2==1):
                            data[row][cell][color] -=1
                        char = tobits(message[messageCounter])
                elif(data[row][cell][color]%2 !=
char[bitCounter]):
```

```
                    data[row][cell][color] += -1
                bitCounter+=1
    print(data)

def decode(data):
    message = ""
    data = np.array(data)
    bitStack = []
    for row in range(data.shape[0]):
        for cell in range(data.shape[1]):
            for color in range(3):
                bitStack.append(data[row][cell][color]%2)
                if(len(bitStack)==8):
                    message += frombits(bitStack)
                elif(len(bitStack)==9):
                    bitStack = []
                    if(data[row][cell][color]%2 == 1):
                        return message
```

```
 def getValue(self,address):
        #En caso de que se acceda a un apuntador ejemplo: (5000)
        if(str(address)[0]=="("):
            address = address[1:-1]#remover los paréntesis
            address = self.getValue(int(address)) #La dirección
ahora será el valor del apuntador
        index = int(address)%self.MemSize
        location = self.getLocation(address)
        scope = location[0]
        type = location[1]
        if(scope == "Global" or scope == "Constant"):
            valor=self.vMemory[scope][type][index]
        else:
            valor=self.vMemory["Stack"][-1][scope][type][index]
#El -1 permite acceder al último elemento del stack
        if(valor==None):
            raise Exception("Accessed to a variable or index
without defining a value before")
        if(type == "int"):
            return int(valor)
        elif(type == "float"):
            return float(valor)
        elif(type == "bool"):
            if(valor=="True" or valor==True):
                return True
            elif(valor=="False" or valor==False):
                return False
            else:
                return None
```

```
        elif(type == "memory"):
            return self.getValue(valor)
        elif(type == 'tag'): #Si es tag, extra el caracter de
comillas " que se encuentran en la primer y última posición
            return valor[1:-1]
        else:
            return valor

    def setValue(self,value,address):

        #En caso de que se acceda a un apuntador ejemplo: (5000)
        if(str(address)[0]=="("):
            address = address[1:-1]#remover los paréntesis
            address = self.getValue(int(address)) #La dirección
ahora será el valor del apuntador
        else:
            address = int(address)
        index = address%self.MemSize
        location = self.getLocation(address)
        scope = location[0]
        type = location[1]
        if(scope == "Global" or scope == "Constant"): ##SI LA
DIRECCION ES PARTE DEL BLOQUE DE VARIABLES GLOBALES O CONSTANTES
            self.vMemory[scope][type][index] = value
        else:
            self.vMemory["Stack"][-1][scope][type][index] = value
#El -1 permite acceder al último elemento del stack de llamadas
```

```
##Al detectar el acceso a un índice
    #Genera los cuádruplos requeridos para obtener el desfase de
la memoria con base a los índices
    def dimEnter(self,varName):
        #Agrega la dimensión a la pila de dimesiones
        if(len(self.Dims)>0):
            front = self.Dims[-1] #Front guarda la dimensión
agregada más recientemente ##PENDIENTE: CHECAR SI SE REPITE EL
NOMBRE ARREGLO DENTRO DE SÍ MISMO X[X[]][]
            if(front[0] == varName): #x[1][2]
                self.Dims.append([varName,front[1]+1])
            else: #Ej que cumple la condición x[1][a[1]]
                self.Dims.append([varName,1])
        else:
            self.Dims.append([varName,1]) #Agrega el nombre de la
dimension y 1 (X[5] = append("X",1))


        ##Obtener el nombre de la variable y el número de la
dimensión actual de la pila de Dimensiones
```

```
        frontName = self.Dims[-1][0]
        frontDim = self.Dims[-1][1] #Número de la dimensión
actual

        if(varName in namesTable.actualT):
            listaDim = namesTable.actualT[varName]["dim"] #lista
de dimensiones de la variable
        elif(varName in namesTable.globalsT):
            listaDim = namesTable.globalsT[varName]["dim"] #lista
de dimensiones de la variable
        else:
            raise Exception(f"Variable [{varName}] is not
defined")

        if(frontDim > len(listaDim)): #Checa que el número de
dimensiones no exceda al número de dimensiones dadas en la
definición de la variable
            raise Exception("Variable '" + frontName + "' with "+
str(frontDim) + " Dimensions not defined, " + str(len(listaDim))
+ " where expected") #display exception

        aux = self.Opd[-1]

        #print("VER", aux, 0 , listaDim[frontDim-1]["ls"])
        self.generateVer(aux,0,listaDim[frontDim-1]["ls"])
        if(frontDim<len(listaDim)):
            aux=self.Opd.pop() #Obtiene el index dentro de
casilla
            indexType = self.Types.pop() #obtener el típo del
índice
            if(indexType != "int"): raise Exception("Index must
be int")
            T = Memory.assignMemory("Temp","int",1)
            self.tempCounter+=1
            #print("*",aux,listaDim[frontDim-1]["m"],T)
            #obtener valor constante de M
            #M = generateConstant(...)

            ##Generar u obtener dirección de la constante de la M
de la dimensión
            mValue = int(listaDim[frontDim-1]["m"])
            self.Types.append("int")
            self.addConstant(str(mValue))
            self.Types.pop()
            mAdd = self.Opd.pop()

            self.Quads.append(Quad("*",aux,mAdd,T))
            self.QuadCounter+=1

            self.Opd.append(T);
```

```
            #self.Types.append("int")

        if(frontDim>1):
            aux2 = self.Opd.pop() #SN*MN
            aux1 = self.Opd.pop() #SN+1*MN+1

            T = Memory.assignMemory("Temp","int",1)
            self.Opd.append(T)
            #print("+",aux1,aux2,T)
            self.Quads.append(Quad("+",aux1,aux2,T))
            self.QuadCounter+=1

        if(frontDim==len(listaDim)):
            aux = self.Opd.pop() #Valor del índice
            indexType = self.Types.pop() #obtener el típo del
índice
            if(indexType != "int"): raise Exception(f"Index must
be int, given [{indexType}]")
            T = Memory.assignMemory("Temp","int",1)

            ##Generar u obtener dirección de la constante de la
dirección inicial del arreglo
            if(frontName in namesTable.actualT):
                addressValue =
namesTable.actualT[frontName]["dir"] #lista de dimensiones de la
variable
            elif(frontName in namesTable.globalsT):
                addressValue =
namesTable.globalsT[frontName]["dir"] #lista de dimensiones de la
variable
            else:
                raise Exception(f"Variable [{varName}] is not
defined")

            self.Types.append("int")
            self.addConstant(addressValue)
            self.Types.pop()
            addressLocation = self.Opd.pop()


            self.Quads.append(Quad("+",aux,addressLocation,T))
            self.QuadCounter+=1
            self.Opd.append("(" +str(T)+")")
```

5. Manual de Usuario

Video explicativo:
https://drive.google.com/open?id=194EQo6MYMK3JcPr2hZc9cJsnOC2zeT70

Gracias por tu interés en el Lenguaje de Programación Quetzal!
Este lenguaje está basado en la manipulación de imágenes y de colores.
A continuación se hará un listado de sus funciones y declaraciones principales que te servirán como referencia.

El lenguaje de Programación Quetzal cumple con los siguientes requerimientos

15. Definición y asignación de variables
    a. La definición de una variable debe cumplir con el formato: la palabra reservada “define” seguido del tipo de la variable y el nombre (id)
    b. Es posible asignar un valor a una variable directamente desde su asignación
    c. El nombre de una variable deberá de empezar con un letra minúscula seguida de caracteres en minúscula o mayúscula
    d. No se puede cambiar el tipo de una variable ya definida
    e. Una variable definida sin asignársele un valor toma un valor por defecto según su tipo.
    f. La definición de variables puede ser a nivel global (fuera de una función*)* o a nivel local (dentro de una función)
    g. Se pueden declarar varias variables del mismo tipo al mismo tiempo si estas son del mismo tipo, los ids se separan por comas “,”.
    h. No se pueden acceder variables locales fuera de la instancia en la que fue definida
    i. Se puede asignar un valor a una variable previamente definida invocando su nombre seguido del signo de igual “=” y el valor deseado, siempre y cuando coincida con el tipo de la variable.

```
define int iMyInt;
define color cRed, cWhite, cBlack;
define float fMyFloat=12.5;
```

16. Líneas de código delimitadas por el símbolo punto y coma “;”
17. Tipos de variables
    a. int: número entero positivo o negativo.
        i. Valor por defecto: 0
    b. float: número decimal positivo o negativo.
        i. Valor por defecto: 0.0
    c. bool: True o False.

i. Valor por defecto: False

d. color: un número hexadecimal de 6 dígitos con el formato “#FFFFFF”.
   i. Valor por defecto: #000000
e. tag: secuencia de caracteres.
   i. No puede ser declarado ni acepta que se le hagan operaciones.
   ii. Utilizable únicamente en prints y para funciones especiales.

18. Definición de funciones con retorno de variable o sin retorno (void)
    a. El encabezado de la función sigue el formato: palabra reservada *“func”* seguido del tipo de retorno, el nombre de la función y un par de paréntesis con los parámetros que se recibe.
    b. El uso de parámetros dentro de una función se hace dentro de paréntesis “(...)” siendo los parámetros definidos como “*tipo* nombre” y una coma “,” separándolos.
    c. Funciones que no reciben parámetros deberán incluir en el encabezado los paréntesis vacíos “()”.
    d. El retorno de la función se construye con la palabra reservada *“return”* seguido de la variable o el valor de retorno correspondiente al tipo de la función.
    e. Funciones de tipo void deberán incluir también la palabra definida “*return”* sin ninguna variable o valor de retorno.
    f. El contenido de la función deberá de estar contenido dentro de un par de llaves “{...}” después del encabezado de la función

```
func void hola(tag nombre, int edad)
{
    print(“Hola”,nombre,” de “,edad,“años” );
}
func int sum()
{
    return 1 + 1;
}
```

19. Llamada de funciones
    a. El llamado de la función se hace escribiendo el nombre de la función seguido de los parámetros, sin la necesidad de volver a especificar los tipos.
    b. Las llamadas de función con un tipo de retorno distinto a *void* deben de ser precedidos por una variable (id) que guarde el valor de retorno de la función (el tipo del id debe de corresponder con el tipo de retorno) seguido de un signo de igual “=” y el nombre de la función

c. Como parámetro dentro de una llamada de función se puede usar una constante, una variable, una operación aritmética, otra llamada a una función o el acceso a una casilla de arreglo.
d. No se permite la definición de variables o funciones como parámetro de una llamada de función.
e. Para poder realizar una llamada de función, la función debe de estar previamente definida
f. El número de parámetros así como el tipo de los parámetros usados deben de coincidir con la definición de los parámetros en la definición de la función.

```
x = nombreDeFuncionConRetorno(y,z);
nombreDeFuncionSinRetornoYSinParametros();
miFunción(1+1,a[2]);
```

20. Comentarios por bloque o por línea
    a. Comentarios por bloque se especifican con el carácter de barra oblicua seguido de un asterisco para abrir el comentario “/*” y para cerrarlos otro asterisco y una barra oblicua “*/”.

```
/* Esto es un comentario
de tipo bloque */
```

    b. Comentarios por línea se especifican con dos caracteres de barra oblicua continuos “//”

```
//Esto es un comentario de línea
```

21. Lectura
    a. Es posible leer valores desde la consola con la función predefinida de “read” la cual sigue el formato: palabra reservada “read” que recibe entre paréntesis “(...)” como parámetro la variable (id) en donde se guardará el valor leído.
    b. Sólo se puede leer una variable por cada “read”
    c. La variable ingresada como parámetro debe de estar previamente definida.
    d. El valor recibido debe de coincidir con el tipo de la variable del parámetro.

```
read(x);
```

22. Escritura
    a. Es posible escribir valores a la consola con la función predefinida de “print” la cual sigue el formato: palabra reservada “read” que recibe entre paréntesis “(...)” como parámetros las variables, constantes,

tags, accesos a índices de arreglos o llamadas a funciones que regresen valores, separadas por comas “,”.

b. Por defecto, el print hace un salto de línea al final de la impresión.

```
print(“Mi nombre es: ”,nombre, “tengo ”,edad[i], “años”)
“Mi nombre es Lizzie tengo 22 años”
```

c.

23. Operaciones aritméticas
    a. Es posible realizar operaciones aritméticas básicas de tipo suma (+), resta (-), multiplicación (*) y división (/).
    b. Los operandos de las operaciones aritméticas deben ser de tipos compatibles según se define en el cubo semántico.
    c. Los operandos deben de estar separados por un operador (Ej. 5 + 5)
    d. Tales operandos pueden ser constantes, variables, llamadas de funciones que retornan valores o accesos de índices a arreglos.

```
5 + 5
x - suma(4,3)
arr[5] * x
```

24. Operaciones Lógicas
    a. Es posible armar operaciones lógicas con el uso de los siguiente operadores: es igual a (==), mayor que (>), menor que (<), es diferente a (!=), mayor o igual a (>=), menor o igual a (<=), o (||), y (&&).
    b. Los operandos de las operaciones lógicas deben ser de tipos compatibles según se define en el cubo semántico.
    c. Los operandos deben de estar separados por un operador (Ej. 5 == 5)
    d. Tales operandos pueden ser constantes, variables, llamadas de funciones que retornan valores o accesos de índices a arreglos.

```
5 > 4 && x < 10
x != suma(4,3)
arr[5] <= x
```

25. Estatutos condicionales
    a. Los estatutos condiciones “Si” deben de seguir el siguiente formato: la palabra reservada “*if*” seguida de un par de paréntesis con una expresión lógica dentro “(Expresión Lógica)”, seguida de un par de llaves que contengan el código deseado “{ ... }”
    b. Opcionalmente, el estatuto “Si no” deberá de proseguir el estatuto “if” siguiendo el formato de: palabra reservada “*else*” seguida de un par de llaves que contengan el código deseado “{...}”

```
if(condición)
{
   Código
}
```

```
else
{
   Código
}
```

26. Ciclos
    a. El ciclo “mientras” se formulan con la palabra reservada “while” seguida de un par de paréntesis con una expresión lógica dentro “(Expresión Lógica)”, seguida de un par de llaves que contengan el código deseado “{...}”

```
while(condición)
{
   Código
}
```

27. Elementos estructurados: Arreglos de varias dimensiones
    a. Se definen arreglos siguiendo la estructura: palabra reservada “*define*” seguida del *tipo* del arreglo, el *nombre* del arreglo y un par de corchetes “[XX]” que contienen dentro un número entero constante que representa el tamaño de la dimensión.
    b. El número de corchetes representa el número de dimensiones
    c. La asignación del tamaño de las dimensiones es estático (No se pueden usar variables).
    d. No se pueden inicializar arreglos
    e. No se pueden hacer operaciones directas con todo un arreglo.
    f. Se accede al elemento de un arreglo, con el fin de asignar o realizar operaciones, escribiendo el nombre del arreglo ya definido, seguido de pares de corchetes “[X]” correspondientes al número de dimensiones del arreglo. Cada corchete debe contener una constante, una variable, una llamada de función con retorno o un acceso a otro arreglo que representa la posición a acceder del arreglo, siempre y cuando no exceda el tamaño dado en la definición del arreglo.

```
define int arr[5][10];
print(arr[i][suma(1+2)]);
```

28. Funciones especiales
    a. El lenguaje permite el uso de las siguientes funciones especiales:
        i. void openImg(tag,color arr[][])
            1. Recibe como parámetro la ubicación de un archivo de una imagen y lo guarda en una variable de matriz de colores.
            2. 

{{#FFFFFF,#FFFFFF,#000000,#000000,#000000,#FFFFFF,#FFFFFF,#FFFFFF,#000000,#000000,#000000,#FFFFFF,#FFFFFF},
{#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF},
{#000000,#BF1E2E,#FFFFFF,#FFFFFF,#BF1E2E,#BF1E2E,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000},
{#000000,#BF1E2E,#FFFFFF,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000},
{#000000,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000},
{#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF},
{#FFFFFF,#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF,#FFFFFF},
{#FFFFFF,#FFFFFF,#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF,#FFFFFF,#FFFFFF},
{#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF},
{#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#000000,#BF1E2E,#000000,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF},
{#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#000000,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF,#FFFFFF}}

ii. void saveImg(tag,color arr[][])
    1. Recibe como parámetro una matriz de colores y lo guarda como imagen según la ruta de archivo especificada.

iii. void grayscale(color arr[][])
    1. Recibe una matriz de colores y la convierte la imagen a escala de grises.

iv. void colorReplace(color arr[][],color,color)
    1. Toma una matriz de colores y recibe dos colores como parámetros, el primero será recoloreado por el segundo.

v. void colorFilter(color arr[][],color)
    1. Recibe una matriz de colores y un color. Esta función convierte en escala de gris todos los colores de la matriz que no sean iguales al segundo parámetro.

vi. void edgeDetection(color arr[][])

1. Toma una matriz de colores. Esta función detecta automáticamente los bordes de los principales elementos de una imagen y los delinea.


vii. void saveScaleImg(color arr[][],int, int)

1. Toma como parámetro una matriz de colores y dos números flotantes. La imagen será escalada en x según el valor del segundo parámetro (Ej. 2 = 200%) y en y según el valor del tercer parámetro. Un número negativo escalará la imagen según la magnitud y la invertirá de forma horizontal o vertical según sea el caso.


viii. void getColorPalette(color arr[][],color arr[],int)

1. Función que recibe como parámetro una matriz de colores, un arreglo de una dimensional de colores y un número entero n. Se regresa un arreglo con n colores


ix. void colorMatchImage(color arr[][], color arr[][])

1. Función que recibe como parámetro dos matrices de colores, siendo la primer matriz la imagen fuente y la segunda la imagen de referencia. Esta función manipula los píxeles de la imagen fuente de modo que

el histograma de pixeles concuerde con el histograma de la imagen de referencia.



x. void encodeSteganography(color arr[][], tag)
    1. Esta función se basa en la técnica de Esteganografía, la cual oculta un mensaje dentro de una imagen. La función recibe una imagen y un mensaje de texto y retorna la imagen con el código guardado. A simple vista la imagen no parece ser diferente a la original.

xi. void decodeSteganography(color arr[][])
    1. Esta función recibe una imagen modificada con esteganografía para obtener el mensaje encriptado que contenga. Recibe una matriz de colores e imprime el mensaje oculto