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)]);
```

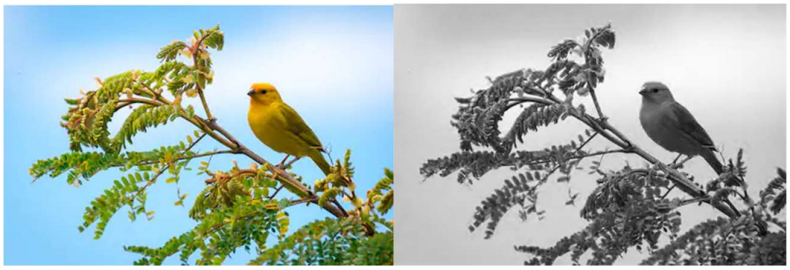
28. Funciones especiales
    a. El lenguaje permite el uso de las siguientes funciones especiales:
        i. void openImg(tag,color arr[][])
            1. Recibe como parámetro la ubicación de un archivo de una imagen y lo guarda en una variable de matriz de colores.

2. 

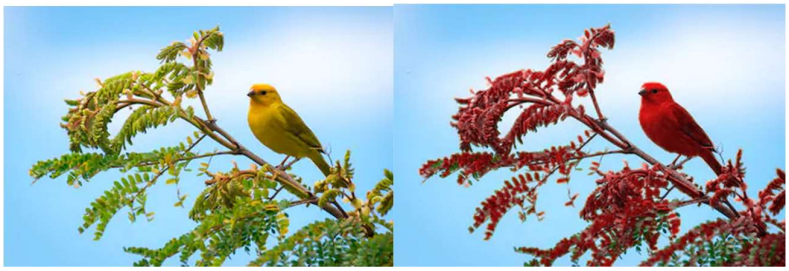
{{#FFFFFF,#FFFFFF,#000000,#000000,#000000,#FFFFFF,#FFFFFF,#FFFFFF,#000000,#000000,#000000,#FFFFFF,#FFFFFF},
{#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#000000,#FFFFFF},
{#000000,#BF1E2E,#FFFFFF,#FFFFFF,#BF1E2E,#BF1E2E,#000000,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#BF1E2E,#000000},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

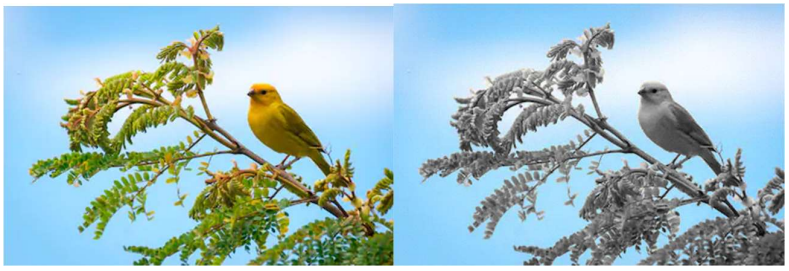
iii. void grayscale(color arr[][])
    1. Recibe una matriz de colores y la convierte la imagen a escala de grises.



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.

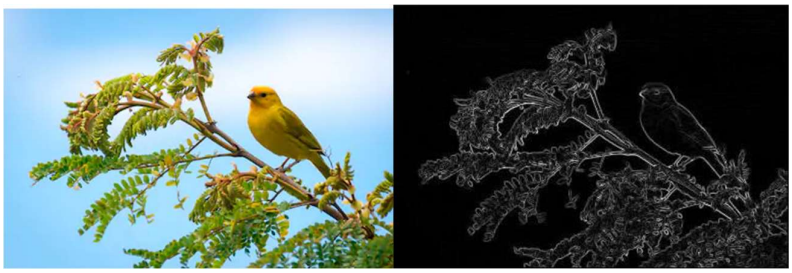


v. void colorFilter(color arr[][],color)
    1. Recibe una matriz de colores y un color. Esta función convierte en escala de gris todos los colores de la matriz que no sean iguales al segundo parámetro.



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.



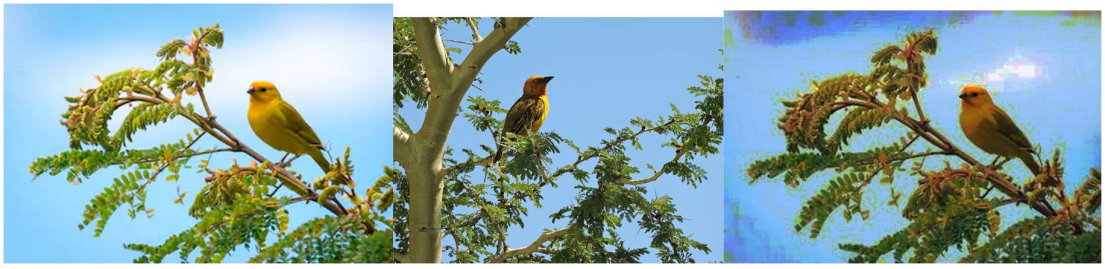
vii. void saveScaleImg(color arr[][],int, int)

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