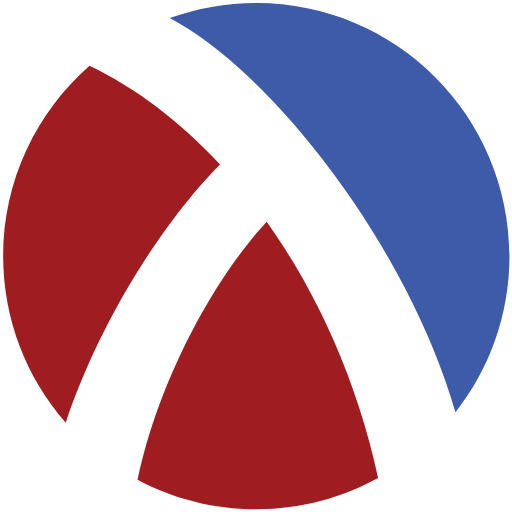
**INFORME DE LABORATORIO 1: SIMULACIÓN DE UN PROGRAMA PARA REALIZAR UN TRATAMIENTO DE IMÁGENES SIMPLIFICADO**

****

Nombre: Aracely Castro V.

Profesor: Roberto Gonzales I.

Asignatura: Paradigmas de Programación (2/2022)

Índice

[1. INTRODUCCIÓN 3](#_Toc114144327)

[1.1 DESCRIPCIÓN DEL PROBLEMA 3](#_Toc114144328)

[1.2 DESCRIPCIÓN DEL PARADIGMA 3](#_Toc114144329)

[2. DESARROLLO 4](#_Toc114144330)

[2.1 ANALISIS DEL PROBLEMA 4](#_Toc114144331)

[2.2 DISEÑO DE LA SOLUCIÓN 5](#_Toc114144332)

[2.3 ASPECTOS DE IMPLEMENTACIÓN 6](#_Toc114144333)

[2.3.1 EJEMPLOS DE USO 6](#_Toc114144334)

[2.3.2 RESULTADOS ESPERADOS 6](#_Toc114144335)

[2.3.3 POSIBLES ERRORES 6](#_Toc114144336)

[2.4 Resultados y autoevaluación 6](#_Toc114144337)

[2.4.1 RESULTADOS 6](#_Toc114144338)

[2.4.2 AUTOEVALUACIÓN 6](#_Toc114144339)

[3. CONCLUSIÓN 7](#_Toc114144340)

[4. BIBLIOGRAFÍA Y REFERENCIAS 7](#_Toc114144341)

[5. ANEXOS 7](#_Toc114144342)

# **INTRODUCCIÓN**

Con el objetivo de aplicar conceptos del paradigma de programación funcional usando el lenguaje de programación Scheme en la resolución de un problema acotado, en las siguientes secciones se dará detalle sobre el proceso de desarrollo de un simulador simplificado de un tratamiento de imágenes mediante el uso de un paradigma funcional a través del compilador Dr. Racket versión 8.6. El informe constará de una breve introducción de cómo surgió el problema, una descripción del paradigma utilizado, el análisis del problema y como fue el diseño soluciones para algunas funciones, sus aspectos de implementación, las instrucciones necesarias para correr el programa junto con ejemplos de uso, los resultados y la autoevaluación. Finalmente, se dará una conclusión respecto a todo lo anterior.

## **DESCRIPCIÓN DEL PROBLEMA**

Se busca hacer una simulación de un programa para el tratamiento de imágenes de forma simplificada como GIMP y Adobe Photoshop. Este simulador permitiría crear imágenes con distintos formatos de pixeles sobre los cuales se pueden aplicar distintas operaciones como, por ejemplo, recortar una imagen, invertir una imagen, obtener histograma de la imagen, editar una imagen, entre otros. Para implementar esto, se deben tener en cuenta que:

Una imagen RGBD o Pixmaps-d es una imagen donde cada uno de sus pixeles contiene información de espacio de colores y profundidad (R)ed, (G)reen, (B)lue y (D)epth. Cada color cubre valores entre 0 y 255 formando en conjunto un color del espectro RGB. La profundidad en cambio ofrece información más detallada de la imagen como espacio tridimensional.

Una imagen Bitmaps-d es una imagen donde, en este caso, cada uno de sus pixeles tiene asociado solo un color, negro (1) o blanco (0) representando si hay tinta o no, cada píxel también tendría una profundidad asociada.

Una imagen Hexmap-d es similar a una imagen Pixmaps-d con la diferencia que los valores RGB son representados en un string hexadecimal.

## **DESCRIPCIÓN DEL PARADIGMA**

Un paradigma funcional es un enfoque para programar en un computador basado únicamente en funciones. Cada función tiene una entrada (dominio) que es transformado hacia una salida (recorrido). En el paradigma funcional no existe el concepto de almacenar o actualizar variables y utiliza notación lambda con operaciones prefijas. Cabe señalar que el lenguaje utilizado para el compilador Dr. Racket no es puramente funcional pues tiene aspectos de programación imperativa.

Al paradigma funcional le importa más el resultado que el desarrollo, es decir, profundiza en el ¿QUÉ? más que en el ¿CÓMO? Lo que implica un mayor nivel de abstracción. Algunos elementos importantes a la hora de construir un código utilizando el paradigma funcional son:

**Composición de funciones**: Es cuando defines una función que tiene como entrada uno o más funciones distintas.

**Recursividad:** Es un método de solución a problemas que pueden depender de soluciones más pequeñas, la función se puede definir en términos de sí mismo y está conformado por una condición de borde y descomposición recursiva. Existen distintos tipos de recursividad como son: recursión natural, recursión de cola y recursión arbolea.

**Funciones Anónimas**: son funciones que se expresan sin nombre utilizando solamente cálculo lambda, quien recibe una entrada y da un resultado solo una vez.

**Currificación:** Es la evaluación de una función de “n” argumentos que tras currificarse es transformada a la evaluación de una secuencia de funciones de un argumento. En la currificación se ocupan funciones anónimas bajo el concepto de funciones de orden superior.

**Funciones de orden superior:** Corresponden a aquellas funciones que reciben como entrada una función o bien su salida es para otra función. Una composición de funciones es una función de orden superior.

# **DESARROLLO**

## **ANALISIS DEL PROBLEMA**

Se identifica que para hacer un tratamiento de imágenes simples en Dr. Racket hay que considerar los siguientes elementos fundamentales:

* **Imagen:** Una lista que contiene dos enteros y una lista de pixeles. (Int X Int X List)
* **Pixrgb-d:** Un píxel del tipo Pixmap, representado como dos enteros que guardan las posiciones del píxel en la imagen, tres enteros que representan el color del píxel y un entero que guarda la profundidad del píxel en la imagen. (Int X int X int X int X int X int)
* **Pixbit-d:** Un píxel del tipo Bitmap, representado como dos enteros que guardan las posiciones del píxel en la imagen, un entero que representa el bit y otro que guarda la profundidad. (Int X Int X Int X Int)
* **Pixhex-d:** Un píxel del tipo Hexmap, representado como dos enteros que guardan las posiciones del píxel en la imagen, un string que representa el color y un entero que guarda la profundidad del píxel en la imagen. (Int X Int X String X Int)

Esperando que a una imagen se le puedan aplicar las siguientes funciones:

* **image:** Crea de imágenes con bitmaps, hexmaps o pixmaps que incluye información de profundidad en cada píxel.
* **bitmap?, pixmap? y hexmap?:** Verifica si la imagen es de tipo bitmaps-d, pixmaps-d o hexmaps-d respectivamente
* **compressed?:** Verifica si la imagen sufrió una compresión
* **flipH y flipV:** Invierte una imagen horizontal y verticalmente respectivamente
* **crop:** Recorta una imagen a partir de un cuadrante definido por dos puntos
* **imgRGB->imgHex:** convierte una imagen Pixmap-d a Hexmap-d
* **histogram:** Retorna un histograma de frecuencias a partir de los colores de una imagen
* **rotate90:** rota la imagen 90° a la derecha.
* **compress:** Comprime una imagen eliminando aquellos pixeles con el color más frecuente.
* **edit:** Permite aplicar funciones especiales a las imágenes.
* **adjustChannel:** Función que permite ajustar cualquier canal de una imagen pixmap, incluido el canal de profundidad d.
* **image->string:** Función que transforma una imagen a una representación string.
* **depthLayers:** Función que permite separar una imagen en capas en base a la profundidad en que se sitúan los pixeles.
* **decompress:** Función que permite descomprimir una imagen comprimida.

Se pide que cada TDA (Tipo de Dato Abstracto) con las implementaciones a estas funciones tenga un archivo único, donde solo se implemente lo necesario para cada TDA y que el archivo principal contenga las funciones obligatorias y opcionales, con el script de pruebas del laboratorio para probar las funciones.

## **DISEÑO DE LA SOLUCIÓN**

Por otra parte, se distinguen algunos casos particulares para soluciones algunos requerimientos funcionales como lo son:

**Histogram:** Para cada TDA se creó su propio histogram para obtener la información relevante. En el caso de bitmap se creo una lista con dos sublistas, donde se guarden la cantidad de veces que se repite el 0 y el 1. Para el caso de pixmap y hexmap, ambos en común comparten que tienen una función que cuenta los elementos iguales a “e” en una lista con recursión de cola y otra función para filtrar los elementos iguales a “e”, esto con el objetivo de recopilar en una lista con sublista cada elemento con la cantidad de veces que se repite en la imagen.

**Compress:** Para poder comprimir se pensaron en distintas estrategias, como la de eliminar el píxel más repetido de la lista o reemplazar esa información con listas vacías en los pixeles. No obstante, aunque se podría identificar que la imagen estaba comprimida (analizando largo del formato de pixeles), no ayudaba a la hora de hacer decompress en la imagen. De modo que se pensó en una forma de cambiar la información del píxel más repetido con otro dato distinto. Para el caso de una imagen bitmap se eligió reemplazar el valor del bit más repetido por -1 para hacer como que se “elimino”, luego para volver a la forma original basta con comprobar si la imagen comprimida le falta el bit 0 o el bit 1, y reemplazar a la forma original en consecuencia. En el caso de una imagen hexmap se eligió reemplazar el píxel repetido por una lista con tres números, cada numero representando en valor color rojo, verde y azul del string, luego para descompress solo hay que leer la lista y con base a ella crear el string original de nuevo para reemplazarlo donde corresponde. Finalmente, para comprimir una imagen pixmap se opto por reemplazar el valor entero del espectro RGB por su equivalente a string hexadecimal, luego para volver a la forma original bastaría con leer el string y transformarlo a numero antes de colocarlo donde se necesite.

**Crop:** La solución para crop fue una recursión natural que filtrara el formato de pixeles cuyas posiciones estén dentro de rango de las coordenadas x1, y1, x2, y2 de modo que se forme un área cuadrada, rectangular o una línea.

## **ASPECTOS DE IMPLEMENTACIÓN**

El compilador utilizado fue Dr. Racket versión 8.6, no se utilizaron bibliotecas externas ni se usaron funciones set!, esto para pensar las soluciones a los requerimientos funcionales usando paradigma funcional y utilizando solo las funciones básicas de Dr. Racket. Mientras se desarrollaban los requerimientos funciones, se crearon los siguientes TDAs cuya estructura respecto el orden de constructor, pertenencia, selectores, modificadores y otras funciones (anexo XX):

**TDA Image:** Corresponde a una imagen. Representación: (int x int x list).

**TDA Pixbit-d:** Corresponde a un píxel bitmap. Representación: (int x int x bit ([0|1]) x int)

**TDA Pixhex-d:** Corresponde a un píxel hexmap. Representación: (int x int x string x int)

**TDA Pixrgb-d:** Corresponde a un píxel pixmap. Representación: (int x int x int x int x int x int)

### **2.3.1 EJEMPLOS DE USO**

Para comenzar, se debe verificar que se tengan todos los archivos TDAs en una misma carpeta, de lo contrario el archivo pruebas\_21090869\_CastroVenegas no se ejecutará al compilar el programa con “Run”. Luego de ejecutado se podrán ver los ejemplos de cada una de las operaciones en el archivo como script de pruebas. Una observación importante es que si se desea crear una nueva imagen para probar las funciones esta no debe tener el mismo nombre que una de las imágenes ya definidas y que las dimensiones y el número de pixeles en la imagen deben de ser correctos, todo esto para que no surja un error al compilar o llamar una función. También que los argumentos de las funciones estén bien ingresados.

Por ejemplo, puedes aplicar la función inverColorRGB a un píxel pixmap, pero si se desea aplicar a una imagen se tiene que utilizar en conjunto con la función edit, como se ve en el anexoXX.

### **2.3.2 RESULTADOS ESPERADOS**

Se espera que cada función no cause errores y compila correctamente.

### **2.3.3 POSIBLES ERRORES**

Puede que algunas funciones que reciben de entrada una imagen no den una salida correcta al ingresar una imagen que sufrió un crop.

## **Resultados y autoevaluación**

### **RESULTADOS**

Las funciones funcionaron correctamente, aunque en medio de las pruebas se notaron algunos detalles como, por ejemplo, que no podías utilizar funciones flipV, flipH y rotate90 en una imagen que sufrió un crop y cuando aplicabas image->string a una imagen crop este no respectaba las coordenadas del píxel cuando se creaba el string. Todo lo anterior se modifico para que funcionarán tales funciones con imágenes al que se aplicaron un crop.

### **2.4.2 AUTOEVALUACIÓN**

La Autoevaluación se realiza de la siguiente forma: 0: No realizado – 0.25: Funciona 25% de las veces – 0.5: Funciona 50% de las veces 0.75: Funciona 75% de las veces – 1: Funciona 100% de las veces. Para ver la tabla de Autoevaluación, ver la Tabla XX del anexo XX

En los requerimientos funcionales no se encontró errores, por lo que se considera de que funcionan el 100% de las veces

# CONCLUSIÓN

# BIBLIOGRAFÍA Y REFERENCIAS

# ANEXOS