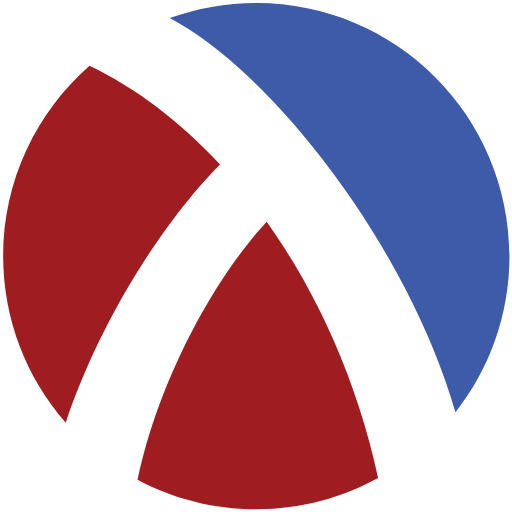
**INFORME DE LABORATORIO 1: SIMULACIÓN DE UN PROGRAMA PARA REALIZAR UN TRATAMIENTO DE IMÁGENES SIMPLIFICADO**

****

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**

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, los objetivos que se abarcaron durante el laboratorio, el análisis del problema y como fue el diseño de su solución, 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 pixel 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 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 dos 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 si mismo y esta conformado por una condición de borde y descomposición recursiva. Existen distintos tipos de recursividad como son: recursión natural, recursión de cola y recursión arbolea, Durante el desarrollo de las implementaciones solo se utilizaron los dos primeros.

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

**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 se tiene que distinguir los siguientes elementos fundamentales. Para comenzar, la imagen, la base central para el tratamiento de imágenes y a quien se le aplica casi todas las funcionalidades definidas para la entrega. Considerando lo anterior se decide representar una imagen como:

* **Imagen:** Una lista que contiene dos enteros y una lista de pixeles. (Int X Int X List)

Luego están los distintos formatos de pixeles en los que se distinguen:

* **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 pixel 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)

Por último, están las operaciones que se pueden realizar a una imagen:

* **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
* **imgRGB->imgHex:** convierte una imagen Pixmaps-d a Hexmaps-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 con pixeles pixrgb-d, 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) tenga un archivo único, donde solo se implemente lo necesario para cada TDA y que el archivo principal contenga las funciones obligatorias y opcionales, donde en este se llamen a las funciones de los TDAs creados

## **DISEÑO DE LA SOLUCIÓN**

Aparte de utilizar los tipos de datos primitivos del lenguaje Scheme, se crearon otros tipos de datos a trasvés del TDA’s. Cada TDA se implemento a través de la siguiente estructura: Representación, Constructores, Pertenencia, Selectores, Modificadores y Otras funciones. Los TDA’s creados son utilizados en conjunto para crear las funciones de los requerimientos funcionales de los que se distinguen:

**TDA Image:** Corresponde a la representación de una imagen. Representación: (int x int x list). Constructor: Se crea una lista dado un ancho y un largo y un conjunto de pixeles. Funciones de pertenencia, selectores, modificadores y otras funciones están en el anexo XX.

**TDA Pixbit-d:** Corresponde a la representación de un píxel tipo bitmap. Representación: (int x int x bit ([0|1]) x int). Constructor: Se crea una lista dado una posición x, posición y, un bit y la profundidad. Funciones de pertenencia, selectores, modificadores y otras funciones están en el anexo XX.

**TDA Pixhex-d:** Corresponde a la representación de un píxel tipo hexmap. Representación: (int x int x string x int). Constructor: Se crea una lista dado una posición x, posición y, un valor hexadecimal y la profundidad. Funciones de pertenencia, selectores, modificadores y otras funciones están en el anexo XX.

**TDA Pixrgb-d:** Corresponde a la representación de un píxel tipo pixmap. Representación: (int x int x int x int x int x int). Constructor: Se crea una lista dado una posición x, posición y, un color rojo, un color verde, un color azul y la profundidad. Funciones de pertenencia, selectores, modificadores y otras funciones están en el anexo XX.

Todos los TDA deben tener una implementación principalmente declarativa – funcional, a través de la programación en Scheme y el compilador Dr. Racket. Los tres últimos TDA´s son utilizados en TDA Image.

[…]

## **ASPECTOS DE IMPLEMENTACIÓN**

### **2.3.1 EJEMPLOS DE USO**

Para comenzar, se debe verificar que se tengan todos los archivos TDA’s en una misma carpeta, de lo contrario el archivo main no se ejecutará. Tras hacer lo anterior se puede compilar el programa con “Run”. Luego de ejecutado se podrán ver los ejemplos de cada una de las operaciones en el archivo main. 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 en el main y 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.

### **2.3.2 RESULTADOS ESPERADOS**

### **2.3.3 POSIBLES ERRORES**

## **Resultados y autoevaluación**

### **2.4.1 RESULTADOS**

### **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

Hasta ahora las únicas funciones no implementadas fueron depthLayers y adjustChannel por lo que se evalúan con un 0. En todas las demás funciones y TDAs no se encontró errores, por lo que se considera de que funcionan el 100% de las veces

# CONCLUSIÓN

# BIBLIOGRAFÍA Y REFERENCIAS

# ANEXOS