UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Informe De Laboratorio: Creación De Una Aplicación Estilo Photoshop En Scheme

Integrantes: Joaquín Saldivia
Profesor: Roberto Gonzales
Asignatura: Paradigmas de programación

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Tabla de contenido

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

1.- Introducción

En este informe se expondrá una solución al problema de laboratorio utilizando el paradigma funcional, donde se utilizará el lenguaje racket para la implementación de este.

1.1.- Descripción del problema

Se desea construir un programa que permita crear y editar imágenes al estilo de Photoshop o GIMP, agregando también la cualidad de profundidad a las imágenes. Se debe tener en cuenta que se tienen las siguientes estructuras como base:

Imagen: Siendo la estructura principal del programa este se compone de un ancho, largo y pixeles, pudiendo aplicarles distintos tipos de edición.

Pixeles: Son los que representan la imagen como tal teniendo una estructura de posición en el ancho y largo, color del pixel y su profundidad, este tendrá 3 tipos de formato y son pixbit-d, pixrgb-d y pixhex-d.

1.2.- Descripción del paradigma

El paradigma funcional, se enfoca en la resolución de problemas a través de funciones, respondiendo al ¿Qué? Y no al ¿Cómo?

Donde una función se puede describir como algoritmo que ante una entrada concreta, entrega una salida siendo estos su dominio y recorrido respectivamente, siempre teniendo en cuenta que la misma entrada siempre tendrá una misma salida. dado que se trabaja mediante funciones no se conoce el concepto de variable al momento de idear soluciones, además, las herramientas que proporciona este paradigma son:

Funciones anónimas: Se le llama función anónima a aquella que no tiene un nombre y por lo tanto no se guarda en memoria.

Recursividad: siendo una propiedad de las funciones, aunque no aplicable a las anónimas y se reconoce cuando una función se llama a si misma, siendo la herramienta principal en este paradigma y existen 3 tipos que son: de cola, natural y arbórea.

Funciones de orden superior: la principal característica es que reciben funciones como parámetro.

Currificación: una función currificada es aquella recibe los argumentos de manera secuencial, en racket dejando un procedimiento en espera de aquellos que faltan.

1.3.- Objetivos

Como objetivo se tiene desarrollar el entendimiento del paradigma funcional a través de racket con la idea de realizar por completo la solución al problema planteado.

2.- Desarrollo

2.1.- Análisis de problema

Se tienen 2 entidades principales las que son la imagen y los pixeles que formaran parte de esta donde estos pueden ser de 3 tipos pixbit, pixhex y pixrgb, además la imagen tiene que aceptar distintos tipos de cambios también los pixeles deben aceptar cambios para operar con éxito las funciones en las imágenes y las funciones mencionadas que debe aceptar son:

pixmap?, hexmap?, bitmap?, compressed?, flipH, flipV, crop, imgRGB->imgHex, histogram, rotate90, compress, edit, invertColorBit, invertColorRGB, adjustChannel, image->string, depthLayers, decompress.

También se solicita que cada TDA utilizado este en un archivo propio.

2.2.- Diseño de solución

Al diseñar la solución se ocuparon los tipos de datos nativos del lenguaje y se crean 2 TDA que serán utilizados para simplificar el problema loque son:

TDA Image:

- Representación: una lista con la composición 2 int representando ancho y largo, una lista de pixeles y un color comprimido

- Constructor: se entregan solo los primeros 3 de los antes mencionados, es decir: 2 int y una lista de pixeles que puede ser vacía para crear la lista imagen

- Funciones: Anexo 1

TDA Pixel:

- Representación: una lista que contiene 2 int indicando las posiciones x e y, un color que puede ser una lista de 3 int, un string o un int y la profundidad dependiendo del tipo de color que posea se le agregara un apellido al píxel.

- Constructor: al entregar 2 int un color y la profundidad se crea una lista que será conocido como pixbit-d, pixrgb-d o pixhex-d por lo antes mencionado.

- Funciones: Anexo 2

Luego la relación entre estos TDA se podrá ver en Anexo 3, además de mencionar que, dada la similitud de varias funciones estas se descomponen en funciones más pequeñas para lograr avances en más de una, utilizando principalmente recursión del tipo natural y de cola por la facilidad que otorgan al momento de recorrer una lista o crearla.

2.3.- Aspectos de implementación

El compilador utilizado es DrRacket en versiones 6.11 o superior, se trabaja principalmente con listas, pero se hacen funciones para acceder a estas.

El código se estructura por TDA, para facilitar la edición a la hora de agregar contenido a alguno de estos, utilizando require / provide para importar y exportar información de estos a otros archivos.

Para este laboratorio no se usan bibliotecas externas por lo que solo se define que se utiliza el lenguaje racket siendo escogido por sobre scheme por simplicidad a la hora de acceder a la documentación.

2.4.- Ejemplos de uso

Se tiene un archivo main que contiene varios ejemplos y es requisito el tener los TDA Image y Pixel para su correcta ejecución.

Ahora para realizar operaciones fuera de estos ejemplos se debe tener en consideración que una imagen debe tener el mismo tipo de pixel y el anexo 4 se puede observar una manera de crear una imagen y la forma de aplicarle funciones.

También se espera poder aplicar a las imágenes las funciones de edición sin ningún problema, donde ante entradas erróneas no ocurran errores. Refiriéndose a los errores no se espera ningún error en concreto.

2.5.- Resultados y autoevaluación

Los resultados son los esperados, logrando hacer un programa completamente funcional, donde se intentó colocando entradas erróneas aplicando mas de una vez una función a una imagen e intentando imprimir imágenes comprimidas.

Se logran completar las 20 funciones propuestas para el proyecto

La autoevaluación radica entre valores de 0 (no implementado) hasta 1 (implementado y sin errores) aumentando en una escala de 0.25 y se puede observar en el anexo 5. Se les asigna a todas las funciones 1 dado que al momento de realizar diversas pruebas estas trabajan correctamente.

3.- Conclusión

Una vez terminado el trabajo se logra cumplir los objetivos, mejorando el entendimiento del paradigma funcional, así como también lograr una aplicación en racket para editar y crear imágenes.

La mayor limitación de este paradigma y lo que causo mas dificultad al empezar este proyecto fue la no utilización de variables, ya que la falta de estas complicaba el pensamiento respecto a como recorrer listas y modificar algún elemento de esta sin modificar lo demás, luego de encontrar una solución había problemas sobre cómo usar la recursión.

4.- Bibliografía

1.- Gonzales, R. (2022). “Proyecto semestral de laboratorio”. https://docs.google.com/document/d/1hUAooKwBj3TWv-yuzBZtNbuaC8iNkzOZdbLpD8P9B8c/edit#

2.- Flatt, M. y Bruce, R. (2021). “The Racket Guide” https://docs.racket-lang.org/guide/

5.- Anexos

1.- Funciones del TDA image:

Tipo de función	Nombre	Descripción
pertenencia	pixmap?	verifica que la imagen tenga pixeles del tipo pixrgb-d
pertenencia	bitmap?	verifica que la imagen tenga pixeles del tipo pixbit-d
pertenencia	hexmap?	verifica que la imagen tenga pixeles del tipo pixhex-d
pertenencia	image?	verifica si la imagen es de alguno de los 3 anteriores
pertenencia	compressed?	verifica si la imagen esta comprimida
selector	getLenX	entrega el ancho de la imagen
selector	getLenY	entrega el largo de la imagen
selector	getPixeles	entre los pixeles que componen la imagen
selector	getCompressedV	si la imagen esta comprimida entrega el color eliminado
selector	firstPix	entrega el primer elemento de la lista de pixeles
selector	nextPix	entrega el resto de elementos de la lista de pixeles
modificador	setPixeles	cambia la lista de pixeles por otra ingresada
modificador	setLenX	cambia el ancho de la imagen
modificador	setLenY	cambia el largo de la imagen
modificador	setCompressedV	agrega a la imagen el color que fue eliminado
otras funciones	flipH	voltea la imagen horizontalmente
otras funciones	flipV	voltea la imagen verticalmente
otras funciones	crop	recorta la imagen desde un (x0, y0) hasta un (x1, y1)
otras funciones	imgRGB->imgHex	transforma una imagen rgb a una imagen hexagesimal
otras funciones	histogram	entrega una lista de colores que aparecen en la imagen junto con la cantidad de veces que se repiten
otras funciones	rotate90	gira la imagen 90 grados en sentido horario
otras funciones	compress	comprime la imagen eliminando el color mas repetido
otras funciones	edit	aplica una funcion que recibe por parametro a una imagen
otras funciones	invertColorBit	invierte el color de un bit

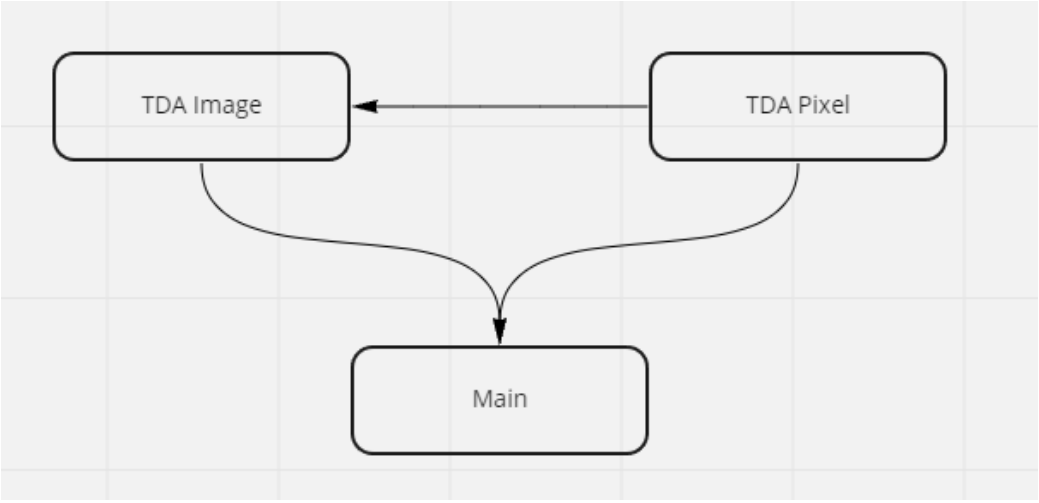
otras funciones	invertColorRGB	invierte el color rgb
otras funciones	adjustChannel	modifica un pixel rgb recibiendo el canal y la operación a realizar
otras funciones	image->string	transforma una imagen en string
otras funciones	depthLayers	crea una lista de imágenes, donde cada una de estas posee una profundidad diferente y los espacios vacios se rellenan con blanco
otras funciones	decompress	descomprime una imagen perdiendo la información de profundidad
otras funciones	getMayor	entrega el color mas repetido en la lista retornada de histograma
otras funciones	estaC?	verifica si esta un color en la lista del histograma
otras funciones	agregar	agrega un color a la lista del histograma y si ya esta llama a agregar 1
otras funciones	agregar1	agrega 1 en el contador de veces que se repite en un color
otras funciones	agregarDepto	agrega el pixel a la lista depto en caso de no estar y si esta se llama a la funcion agregarD
otras funciones	agregarD	agrega el pixel a la lista de pixeles que posea la misma profundidad
otras funciones	estaDepto?	verifica si la profundidad del pixel ingresado ya existe en alguna de las listas almacenadas en la lista depto
otras funciones	entregaP	entrega un pixel de la imagen en una posición especifica
otras funciones	formarString	entrega la imagen en formato string
otras funciones	rellenarPix	completa una lista de pixeles rellenando los faltantes con un color ingresado a una profundidad ingresada

2.- Funciones del TDA pixel:

Tipo de función	Nombre	Descripción
pertenencia	pixrgb-d?	verifica que la imagen tenga pixeles del tipo pixrgb-d
pertenencia	pixbit-d?	verifica que la imagen tenga pixeles del tipo pixbit-d
pertenencia	pixhex-d?	verifica que la imagen tenga pixeles del tipo pixhex-d
selector	getPosX	entrega la posicion en x del pixel
selector	getPosY	entrega la posicion en y del pixel
selector	getColor	entrega el color del pixel
selector	getDepth	entrega la profundidad del pixel

selector	getR	entrega el color rojo del pixel rgb
selector	getG	entrega el color rojo del pixel rgb
selector	getB	entrega el color rojo del pixel rgb
modificador	setPosX	cambia la posición en X del pixel
modificador	setPosY	cambia la posición en Y del pixel
modificador	setBit	cambia el color de un pixel bit
modificador	setR	cambia el color rojo de un pixel rgb
modificador	setG	cambia el color verde de un pixel rgb
modificador	setB	cambia el color azul de un pixel rgb
modificador	setHex	cambia el color de un pixel hex
otras funciones	pixrgb->pixhex	transforma una pixrgb en un pixhex
otras funciones	rgb->stringHex	tranforma la lista string que retorna rgb->hex en un string
otras funciones	rgb->hex	transforma cada color en la lista de un pixrgb en hex
otras funciones	pixbit->string	transforma el color en string
otras funciones	pixrgb->string	transforma el color en string
otras funciones	pixhex->string	transforma el color en string

3.- Relación entre TDA y main:



4.- ejemplos de uso:

```
> (define img1 (image 2 2 (pixbit-d 0 0 1 15) (pixbit-d 1 0 0 20) (pixbit-d 0 1 1 1) (pixbit-d 1 1 0 10)))
> img1
'(2 2 ((0 0 1 15) (1 0 0 20) (0 1 1 1) (1 1 0 10)))
```

```
> (flipH img1)
'(2 2 ((0 1 1 15) (1 1 0 20) (0 0 1 1) (1 0 0 10)))
```

```
> (compress img1)
'(2 2 ((1 0 0 20) (1 1 0 10)) 1)
```

```
> (depthLayers img1)
'((2 2 ((0 0 1 15) (1 0 1 15) (0 1 1 15) (1 1 1 15)))
  (2 2 ((0 0 1 20) (1 0 0 20) (0 1 1 20) (1 1 1 20)))
  (2 2 ((0 0 1 1) (1 0 1 1) (0 1 1 1) (1 1 1 1)))
  (2 2 ((0 0 1 10) (1 0 1 10) (0 1 1 10) (1 1 0 10))))
```

```
> (histogram img1)
'((1 2) (0 2))
```

```
> (image->string img1 pixbit->string)
"1 0 \n1 0 \n\n"
```

5.- Autoevaluación:

Autoevaluacion	Puntaje
TDA	1
image	1
pixmap?	1
hexmap?	1
bitmap?	1
compressed?	1
flipH	1
flipV	1
crop	1
imgRGB->imgHex	1
histogram	1
rotate90	1
compress	1
edit	1
invertColorBit	1
invertColorRGB	1
adjustChannel	1
image->string	1
depthLayers	1
decompress	1