UNIVERSIDAD DE SANTIAGO DE CHILE

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

PARADIGMAS DE PROGRAMACIÓN

"Laboratorio N°1: Paradigma Funcional"

Estudiante: *Vicente Mieres S.*

Docente: *Roberto Gonzales I.*

Fecha: *26/09/2022*

INDICE

INTRODUCCIÓN

Aplicaciones como Adobe Photoshop o GIMP tienen su foco principal en la edición de imágenes, las cuales permiten modificar la misma según las necesidades del usuario. De esta forma el presente informe detalla la implementación de un programa similar a los mencionados, utilizando como base el paradigma funcional de programación. En este se podrán manipular imágenes de tres tipos, bitmaps, las cuales están formadas por 0s y 1s; pixmaps, estas equivalen a un conjunto de pixeles del tipo RGB, donde los colores están dictados por los canales R (Red), G (Green) y B (Blue), por último, hexmaps, representaciones de imágenes que poseen pixeles dados por la notación hexadecimal (#FFFFFF). Adicional a lo mencionado anteriormente, para un manejo más sencillo de las imágenes, cada pixel contendrá la información de su posición en horizontal (X) y vertical (Y), además de la profundidad (Depth) en la que se encuentra. Finalmente, algunas de las funcionalidades implementadas comprenden distintos aspectos, tales como rotar, aplicar filtros, invertir, crear imágenes, entre otras.

EL PARADIGMA FUNCIONAL

Un paradigma se comprende como un modo de proceder, pensar, o simplemente, una manera establecida de cómo hacer las cosas. Además, trae consigo, la solución a ciertos problemas que no pueden resolverse con otros paradigmas. Es por esto que modificar la forma en que las personas pensamos o realizamos ciertas acciones no es tarea de todos los días y, un claro ejemplo de esto es el descubrimiento de una nueva tecnología, un método de enseñanza en un establecimiento educacional o una forma de programar.

De esta forma, la principal restricción en este proyecto se establece a la hora de desarrollar la solución. La programación funcional (FP), corresponde a un paradigma declarativo de programación, el cual se enfoca en el “que” de las cosas y no en él “como” y, además, es posible expresar nuestra lógica sin detallar control de flujo. Este paradigma tiene bases en el cálculo lambda, siendo esta una notación formal que permite el cálculo computacional y la definición de funciones mediante la notación prefija o polaca (operadores precede de los operandos).

Al momento de desarrollar software bajo este paradigma, nos centraremos en el uso de funciones como herramienta principal. Estas representan la unidad de abstracción elemental y son las encargadas de transformar un conjunto de elementos pertenecientes a un conjunto de entrada (dominio) a un conjunto de salida (recorrido), donde cada componente del dominio tiene una única imagen en el recorrido. Asimismo, es posible definir dos nuevos términos, en primer lugar las **funciones de orden superior** son aquellas que poseen otras funciones como argumentos de entrada, o incluso, retornan otros procedimientos. En segundo lugar, **currificación** corresponde al paso de la evaluación de una función de n argumentos a la evaluación de funciones de un único argumento. Finalmente, y a modo de ejemplo, alguno de los lenguajes de programación enfocados en este paradigma son Haskell, Ocaml, Scala, Lisp o Scheme.

ANALISIS DEL PROBLEMA

La creación de un editor de imágenes no es sencilla, y el desarrollo de este implica una serie de requerimientos, los cuales se pueden clasificar en no funcionales y funcionales. Tal como se definió anteriormente, se trabajará bajo el paradigma de la programación funcional, es por esto que la solución está definida por veinte requerimientos funcionales. En primer lugar, y previo a cualquier intento de desarrollo de software, es necesario el planteamiento de los Tipos de Datos Abstractos (TDA), estos corresponden a abstracciones del problema principal de los cuales se desprenden tres ejes principales. La **especificación** de un TDA corresponde al listado de operaciones que este podrá realizar (funciones) determinando tanto el dominio como el recorrido de estas, por otro lado, la **representación** será la estructura base sobre la que operarán las funciones, por último, la **implementación** consiste en construir las funciones previamente definidas para luego organizarlas según el tipo de acción que realicen, ya sea modificadoras, selectoras, constructoras, de pertenencia u otras operaciones. Además, para dar formato y apariencia a los distintos TDAs, y considerando las listas como la base del lenguaje Racket, es necesario definir procesos ya existentes bajo el contexto de cada uno, por ejemplo, si se necesita aplicar un filtro a una imagen, la función que realice esta acción debería llamarse "imagen-filter" y no "filter", de esta forma los sinónimos serán la clave para una implementación correcta. En consecuencia, se especifican diecinueve procedimientos a trabajar para dar forma al laboratorio, sin embargo, el desarrollo puede requerir un total de n-funciones extra según sea necesario. Algunos ejemplos, ya sea selectores, modificadores u otros, se define "image" como el constructor de imágenes del proyecto, el cual cuenta con una estructura a base de enlistar tres elementos, ancho, alto y otra lista que contendrá todos los pixeles. También podemos definir las funciones "bitmap?", "pixmap?" y "hexmap?", estas determinan, a través de booleanos el tipo de imagen que se está analizando. El resto de los ejemplos se encuentran el siguiente anexo. (Ejemplos)

Por otro lado, los requisitos no funcionales se sitúan detrás de la programación directa y se basan principalmente en el estudio de un lenguaje específico, la organización de un código y la teoría. Para el desarrollo de este laboratorio se requiere del uso del lenguaje Racket, el cual tiene sus bases en Scheme, lenguaje de programación funcional impuro y un dialecto de Lisp. En cuanto a la organización del código, es necesaria una documentación en particular, siendo esta el nombre de la función, descripción, dominio, recorrido, tipo de recursión y estrategia; este último corresponde a tipos de algoritmos tales como Backtraking, Ramificación y Poda, entre otros. Respecto al código en sí, este deberá tener una organización mediante archivos independientes, donde cada archivo describe un TDA específico implementado y un archivo "main" en el que se detallarán todos los requisitos funcionales del proyecto. Finalmente se usará **GitHub**, portal creado para albergar códigos en la nube, este nos dará la opción de tener a nuestra disposición distintas versiones de nuestro código según sea actualizado con el tiempo, de esta manera siempre tendremos un respaldo al cual acudir en caso de algún problema.

DISEÑO DE LA SOLUCIÓN

A la hora de trabajar en el proyecto, surge la duda de como representar una imagen solo con números y símbolos, es por esto que, tal como se muestra en el siguiente enlace, se escogieron las listas como representación base. De esta forma, el manejo de los datos será sencillo y de fácil acceso. Las funciones image, pixbit-d, pixhex-d y pixrgb-d, serán las encargadas de crear las estructuras básicas tales como imágenes y los distintos tipos de pixeles, correspondiendo a los constructores. En consecuencia, podemos definir funciones tales como getPixeles o getPixel, las cuales describen procesos de obtención de elementos, creando así el apartado de selectores. En particular getPixeles retornará el 3er argumento de una imagen, el cual corresponde a una lista que contiene pixeles homogéneos en su interior, cualquiera sea el tipo. Bajo este concepto el resto de funciones selectoras tendrán un desarrollo similar.

El uso de booleanos será un eje fundamental en el desarrollo funcional de la solución, en primer lugar, las funciones que se encuentran en el apartado de pertenencia retornarán #t (True) en caso de cumplir con los requisitos necesarios para pertenecer a los distintos tipos de representaciones, en caso contrario retornarán #f. Claros ejemplos de esta funcionalidad son “esHex?”, “esBit?” o “esRGB?”; pudiendo verificar si el pixel posee cuatro componentes y que su color pertenezca a la notación hexadecimal, como en el caso de los pixeles construidos a partir de pixhex-d. Por otro lado, los modificadores comprenden una amplia variedad de posibilidades de implementación, siendo los encargados de transformar una imagen, ya sea invirtiéndola, cambiando su color, comprimiéndola, etc. Para este tipo de procesos fue necesaria una mejora en el tipo de estrategia usada, por lo que la recursión, el mapeo y filtrado de imágenes son esenciales. Así pues, recursión hace alusión a funciones que se escriben en términos de sí misma, donde un cierto caso base será el punto de parada. Además, para este laboratorio se abordaron dos tipos. Recursión natural, es aquella en que se generará un proceso de expansión, para luego dar paso a la solución, donde la salida será construida a partir de todos los estados pendientes previamente calculados. Mientras que, la recursión de cola no posee estados pendientes, y la solución forma parte de los argumentos de la función previamente definida. Mapeo y filtrado corresponde a una estrategia en que se extraen los pixeles de una imagen, y por cada pixel que cumpla cierto requisito, se le aplicará una función modificadora a alguno de sus componentes. La función “crop” hace uso de esta estrategia, verificando qué pixeles se encuentran dentro del área descrita en los parámetros, de ser así serán retornados en una lista.

Por último, el apartado de -otras operaciones- estará conformado por aquellas funciones que no son clasificables bajo alguno de los conceptos mencionados previamente. Por ejemplo “n_componentes?”, simplemente cuenta el número de componentes en la estructura de un pixel. El listado completo de funciones se encuentra en el siguiente enlace.

ASPECTOS DE IMPLEMENTACIÓN

El informe en su totalidad cuenta con ochenta y seis funciones, todas organizadas en nueve archivos distintos, de los cuales TDAimage, TDApixbit-d, TDApixrgb-d, TDApixhex-d, TDAPixeles, TDAPixel y TDAHistogram comprenden el apartado de TDAs cada uno organizado individualmente. Bajo el nombre de código fuente, se encuentran los requisitos funcionales del laboratorio, todos en orden. Y por último, en el archivo de pruebas se demuestra el correcto funcionamiento del programa. El uso de bibliotecas externas forma parte de las restricciones, por lo que solamente se hace uso de Racket nativo. Además se hace uso del interprete Dr. Racket para el desarrollo.

INSTRUCCIONES DE USO

Para un perfecto funcionamiento del programa debemos tomar en cuenta que los archivos de TDAs no deben ser modificados bajo ningún concepto. Y tanto el archivo de pruebas como el código fuente, están disponibles para su uso. Se espera un correcto funcionamiento de todas las funciones implementadas, mientras se mantenga un estándar homogéneo a la hora de crear imágenes. Por ejemplo, todos los pixeles deben ser del mismo tipo, en caso contrario se producirán errores a la hora de construir imágenes.

Ahora, el uso de las funciones forma parte de un proceso simple, donde entre paréntesis debe ir el nombre de esta y de los parámetros correspondientes. Se puede definir una imagen previamente o en cada llamado utilizar el constructor “image” de forma manual. Es necesario el uso del editor Dr. Racket, de esta forma podremos acceder a los archivos. Por otro lado en caso de no querer hacer uso de las funciones personalmente, se dispone del archivo de pruebas mencionado con anterioridad, este contiene un listado de los requerimientos funcionales, en el cual se abordan distintos tipos de imágenes.

RESULTADOS Y AUTOEVALUACIÓN

En el apartado ejemplos, se encuentra un listado con los resultados tras hacer uso de las funciones, enumerados del 1 al 18. Si bien el total original corresponde a 20, las funciones “depthLayers” y “decompress”, no han sido implementadas. Primeramente, depthLayers presentó una serie de problemas a la hora de crear un algoritmo que determinara la cantidad de imágenes que deben ser creadas (por capa), y en segundo lugar, “decompress” principalmente no pudo desarrollarse por la estructura creada con “compress”, la cual mantiene la información de las dimensiones de la imagen, colores frecuentes e infrecuentes y la profundidad de los pixeles eliminados, no obstante, las coordenadas de estos son eliminadas, no permitiendo la reconstrucción de la imagen.

Por otro lado, y sin considerar los problemas recién mencionados, se realizaron pruebas con imágenes de distintas dimensiones, colores, tipos de mapas e incluso imágenes de un solo pixel, dando como resultado el correcto funcionamiento de todas estas.

CONCLUSIÓN

El cambio de paradigma no fue sencillo, implicó una gran inversión de tiempo, sin embargo una vez interiorizado, es posible destacar las virtudes de este. El uso de funciones facilita la organización del código, el no depender de variables genera una sensación de seguridad a la hora de realizar nuevas tareas, y además, la iteración sobre listas resulta más llevadera que los típicos ciclos infinitos. Por lo tanto, reconocer que un cambio de paradigma abre un mundo de posibilidades, es todo un acierto.

Los conocimientos adquiridos tras este laboratorio representan tranquilidad a la hora de realizar una actividad similar, el desarrollo será similar, sin embargo se espera una mejora en el estudio de un nuevo lenguaje de programación, este es fundamental cuando una herramienta específica es necesaria.

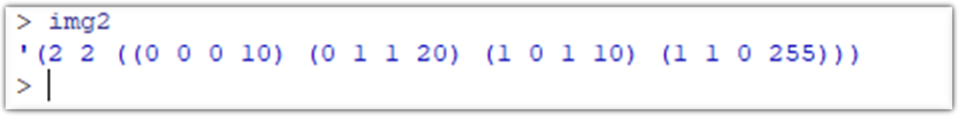
Gracias a lo anterior, es posible concluir que los resultados obtenidos en el proyecto son positivos, todas las funciones implementadas funcionan perfectamente y las limitaciones fueron menores respecto del resultado final.

REFERENCIAS

1. *Definición de función*. (s. f.). Recuperado 25 de septiembre de 2022, de http://www3.uacj.mx/CGTI/CDTE/JPM/Documents/IIT/sterraza/mate2016/funcion/func_def.html
2. Fenollosa, A. (2019, 28 junio). *¿Qué es la programación funcional y por qué es tan especial?* Programador Web Valencia. Recuperado 25 de septiembre de 2022, de https://programadorwebvalencia.com/que-es-la-programacion-funcional-y-por-que-es-tan-especial/
3. Gallardo, P. (s. f.). *Funciones de orden superior* [Diapositivas; Pdf]. http://www.lcc.uma.es/~pepeg/mates/tema4.pdf
4. *Paradigma - ¿Qué es y qué tipos de paradigmas conocemos?* (s. f.). Concepto. Recuperado 25 de septiembre de 2022, de https://concepto.de/que-es-paradigma/
5. *Programación funcional. Qué es y características*. (2020, 19 noviembre). Recuperado 25 de septiembre de 2022, de https://www.incentro.com/es-ES/blog/que-programacion-funcional
6. *¿Qué es la Programación Funcional?* (s. f.). CódigoFacilito. Recuperado 25 de septiembre de 2022, de https://codigofacilito.com/articulos/programacion-funcional
7. Tomé, C. (2019, 13 febrero). *La notación polaca, la de Jan Łukasiewicz*. Cuaderno de Cultura CientÃfica. Recuperado 25 de septiembre de 2022, de https://culturacientifica.com/2019/02/13/la-notacion-polaca-la-de-jan-lukasiewicz/

ANEXOS

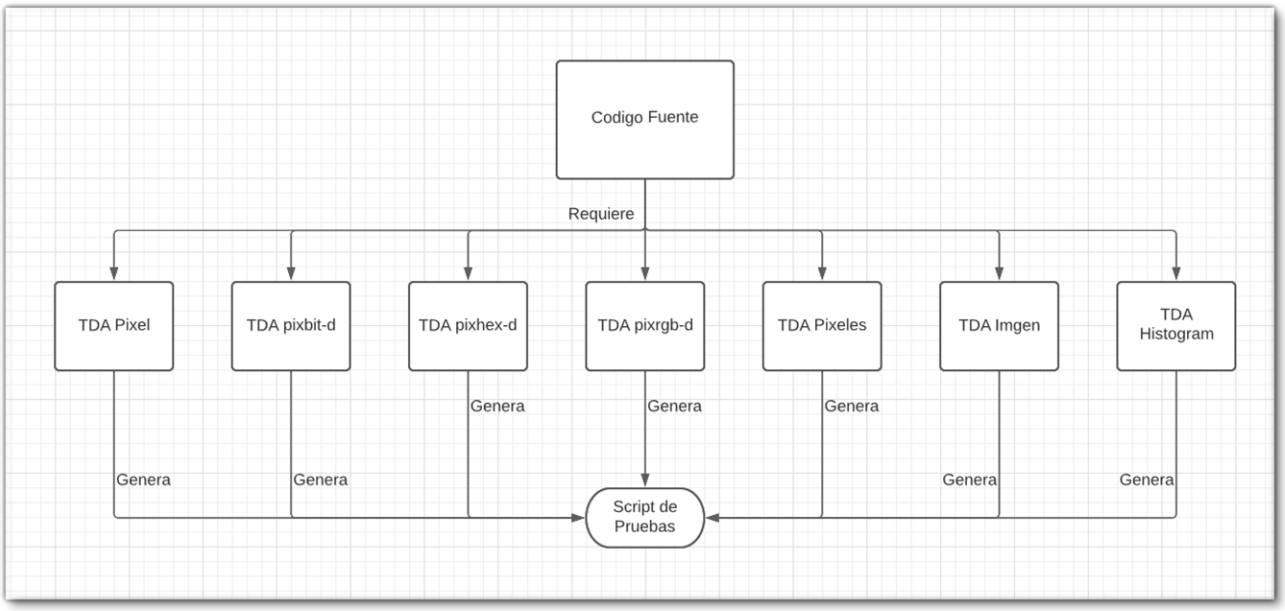
1. Representación de una imagen de 2x2 del tipo bitmap.



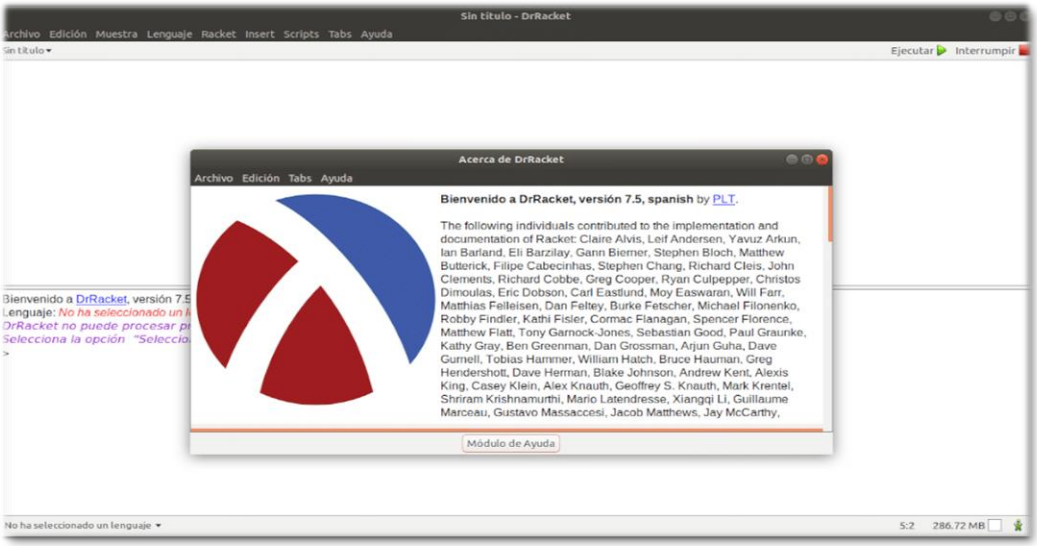
2. Listado de requisitos funcionales.

Nombre	Funcionalidad
image	Constructora de imágenes.
Bitmap?	Permite determinar que si la imagen es de tipo bitmap.
Hexmap?	Permite determinar que si la imagen es de tipo hexmap.
Pixmap?	Permite determinar que si la imagen es de tipo pixmap.
Compressed?	Permite determinar si una imagen esta comprimida.
flipH	Invierte una imagen horizontalmente.
flipV	Invierte una imagen verticalmente.
crop	Recorta una imagen a partir de un área cuadrada.
imgRGB->imgHex	Transforma una imagen de tipo pixmap a una de tipo hexmap.
histogram	Retorna un histograma de frecuencias según los colores.
Rotate90	Rota una imagen 90° a la derecha.
Compress	Comprime una imagen.
Edit	Aplica un filtro a una imagen.
invertColorBit	Invierte el color de una imagen de tipo bitmap. 0 -> 1, 1 -> 0.
invertColorRGB	Invierte los colores del canal RGB de una imagen de tipo pixmap, a su color simétricamente opuesto.
adjustChannel	Modifica cualquier canal de una image de tipo pixmap.
Image->string	Transforma una imagen a string.
depthLayers	Separa una imagen según las profundidades de los pixeles.
decompress	Descomprime una imagen.

3. Estructura de los archivos.



4. Dr. Racket.



5. Ejemplos de uso.

```
; #2. Image
(define img1 (image 2 2
                   (pixrgb-d 0 0 255 0 0 10)
                   (pixrgb-d 0 1 0 255 0 20)
                   (pixrgb-d 1 0 0 0 255 30)
                   (pixrgb-d 1 1 255 255 255 40)
 ))

(define img2 (image 2 2
                   (pixbit-d 0 0 0 10)
                   (pixbit-d 0 1 1 20)
                   (pixbit-d 1 0 1 10)
                   (pixbit-d 1 1 0 255))
 )

; #3. bitmap?
(bitmap? img1)

; #4. pixmap?
(pixmap? img1)

; #5. hexmap?
(hexmap? img1)

; #6. compressed?
(compressed? img1)

; #7. flipH
(flipH img1)

; #8. flipV
(flipV img1)

; #9. crop
(crop img1  0 0 0 1)
```

```
; #9. crop
(crop img1  0 0 0 1)

; #10. imgRGB->imgHex
(imgRGB->imgHex img1)

; #11. histogram
(histogram img1)

; #12. rotate90
(rotate90 img1)

; #13. edit
(edit (adjustChannel getR setR incChR) img1)

; #14. invertColorBit
(edit invertColorBit img2)

; #15. invertColorRGB
(edit invertColorRGB img1)


; #16. adjustChannel
(edit (adjustChannel getR setR incChR) img1)

; #17. image->string
(display(image->string img1 pixrgb->string))
```

¡Aclaración, si bien en el informe se describen 18 funciones, la n°1 corresponde a la definición de TDAs!

6. Resultados.

```
#f
#t
#f
#f
'(2 2 ((0 1 255 0 0 10) (0 0 0 255 0 20) (1 1 0 0 255 30) (1 0 255 255 255 40)))
'(2 2 ((1 0 255 0 0 10) (1 1 0 255 0 20) (0 0 0 0 255 30) (0 1 255 255 255 40)))
'(1 2 ((0 0 255 0 0 10) (0 1 0 255 0 20)))
'(2 2 ((0 0 "#FF0000" 10) (0 1 "#00FF00" 20) (1 0 "#0000FF" 30) (1 1 "#FFFFFF" 40)))
'(((255 0 0) 1) ((0 255 0) 1) ((0 0 255) 1) ((255 255 255) 1))
'(2 2 ((0 1 255 0 0 10) (1 1 0 255 0 20) (0 0 0 0 255 30) (1 0 255 255 255 40)))
'(2 2 ((0 0 0 0 0 10) (0 1 1 255 0 20) (1 0 1 0 255 30) (1 1 0 255 255 40)))
'(2 2 ((0 0 1 10) (0 1 0 20) (1 0 0 10) (1 1 1 255)))
'(2 2 ((0 0 0 255 255 10) (0 1 255 0 255 20) (1 0 255 255 0 30) (1 1 0 0 0 40)))
'(2 2 ((0 0 0 0 0 10) (0 1 1 255 0 20) (1 0 1 0 255 30) (1 1 0 255 255 40)))

#FF0000#00FF00
#0000FF#FFFFFF
>
```

7. Este apartado muestra el grado de alcance de las funciones en la siguiente escala:

    a. 0: No realizado.
    b. 0.25: Implementación con problemas mayores (funciona 25% de las veces o no funciona)
    c. 0.5: Implementación con funcionamiento irregular (funciona 50% de las veces)
    d. 0.75: Implementación con problemas menores (funciona 75% de las veces)
    e. 1: Implementación completa sin problemas (funciona 100% de las veces)

Nombre	Alcance
image	1
bitmap?	1
hexmap?	1
pixmap?	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	0
decompress	0