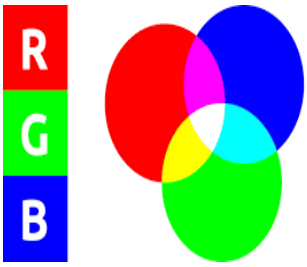
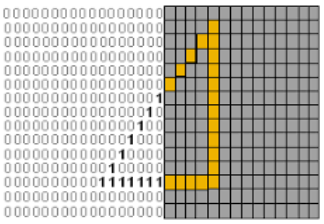
UNIVERSIDAD DE SANTIAGO DE CHILE

**UNIVERSIDAD DE SANTIAGO DE CHILE**
**FACULTAD DE INGENIERÍA**
**DEPARTAMENTO DE INGENIERÍA INFORMATICA**

# INFORME LABORATORIO 2:

# Editor de imagen en Prolog




Nombre: Jean Paul Rojas Ramos de Rosas

Fecha: 03-11-2022

Profesor: Miguel Truffa

# índice:

## 1. Introducción:

El mundo siempre encuentra nuevos problemas o necesidades, con el avance tecnológico de hoy en día es fácil lograr soluciones rápidas a problemas que a nivel manual o para un humano puede volverse tedioso o algo repetitivo, incluso cansador y que genere mucho desgaste, gracias a la programación y las diferentes formas de implementar esta, podemos solucionar dichos problemas con los diferentes paradigmas de programación que existen y llegar a las mismas soluciones que se necesitan lograr aunque estas soluciones se basen en otro paradigma, anteriormente fue vista la solución de este problema de editor de fotos, basado en el paradigma funcional implementado en el lenguaje Scheme, para esta oportunidad la solución del problema se enfocara en el paradigma lógico, implementado en el lenguaje SWI-Prolog, buscando lograr los mismo resultados que la implementación hecha en el paradigma funcional.

## 2. Descripción del problema:

La necesidad de editar imágenes independiente del tipo de pixeles que contenga es lo que se busca para satisfacer la problemática, iterar imágenes, su contenido, edición de coordenadas de cada pixel en un mapa de estos, existiendo 3 tipos de pixeles que conforman una imagen la cual no puede ser, homogénea o por decir contener pixeles de distintos tipos, una imagen tiene dimensiones, que son la cantidad de pixeles que tiene cada imagen en formato ancho y alto, la multiplicación de estos entrega exactamente la cantidad de pixeles que contendría dicha imagen, por lo que las bases de esta resolución son las siguientes:

**Imagen:** una lista que contiene como datos, ancho-alto-pixeles, obteniendo así una lista general que contiene dichos datos, cumpliendo la heterogeneidad de los pixeles esto quiere decir que los pixeles que ingresan deben ser solo de un tipo, y no pueden ingresar pixeles mezclados, esta imagen será la encargada de analizarse y editarse para lo que se le solicite realizar.

**Pixbit:** una lista que contiene como datos, x-y-bit-depth, obteniendo así una lista que cumple los requisitos, x e y posiciones en las que se ubican en el mapa de pixeles que viene siendo la imagen, bit que solo puede variar entre 0 y 1, y depth que es un entero con información para la profundidad, este pixbit será una parte de imagen que en conjunto creará una imagen.

**Pixrgb:** una lista que contiene como datos, x-y-r-g-b-depth, obteniendo así una lista que cumple los requisitos, x e y posiciones en las que se ubican en el mapa de pixeles que viene siendo la imagen, r-g-b que solo pueden variar entre 0 y 255, y depth que es un entero con información para la profundidad, este pixbit será una parte de imagen que en conjunto creará una imagen.

**Pixhex:** una lista que contiene como datos, x-y-hex-depth, obteniendo así una lista que cumple los requisitos, x e y posiciones en las que se ubican en el mapa de pixeles que viene siendo la imagen, hex que es una palabra con datos hexadecimales con respecto a los colores, y depth que es un entero con información para la profundidad, este pixbit será una parte de imagen que en conjunto creará una imagen.

**Funciones**: estas se piden implementar para poder aplicar cambios a las imágenes creadas. El programa debe ser implementado en Swi-Prolog utilizando el paradigma lógico.

## 3. Descripción del paradigma:

Al igual que el paradigma usado en la entrega anterior, el paradigma funcional es parte de la programación declarativa. El paradigma lógico está fundamentado en una base de conocimientos o datos, en conjunto de hechos reales y reglas, para posteriormente poder consultar con respecto a lo planteado en los hechos y las reglas, en este desarrollo Swi-Prolog que es el encargado de enfrentar esta problemática, los mecanismos básicos de este se basa en la unificación, backtracking automático y estructuras de datos basadas en árboles, este enfrenta las problemáticas en base a funciones conocidas como predicados, para poder suplir las demás necesidades de estos predicados, solo es necesaria la edición de estos , los llamados predicados están repartidos entre antecedentes y consecuentes donde los antecedentes son las necesidades que debe cumplir el predicado para poder editar las variables que entran a dichos predicados, y el consecuente es lo que debería realizar luego de cumplidos todos estos antecedentes así cuando cumplan los antecedentes de esta pueda cumplir el consecuente y así entregar lo solicitado en estos. Al igual que en la anterior entrega en el lenguaje Swi-Prolog como en Scheme no existen los ciclos, por lo que queda utilizar la conocida recursión que ha sido bien vista hasta este punto del desarrollo a este problema, por lo tanto, la estructura de código de esta entrega volverá a basarse en recursiones para poder suplir lo requerido en las funciones solicitadas a implementar.

## 4. Análisis del problema:

La creación de una imagen no es del todo fácil, hay muchas restricciones para llevar a cabo en este caso, en el caso de que una imagen sea un bitmap asegurarse que sus bit solo se basen entre 0’s y 1’s, en el caso de que sea pixmap asegurarse que sus colores r g b tengan valores entre 0 y 255 para poder simular un color y en el caso de ser hexmap este tenga su dato de color expresada de forma hexadecimal para poder interpretarlo y darle color con los datos que se crearon, por lo mismo las restricciones para crear una imagen de cierto tipo tiene que tener ciertas condiciones para que sea posible crearla, además de cumplir con la heterogeneidad de la lista de pixeles ingresados, en el caso de crear una imagen comprimida esta podría crearse pero no se podría aplicar las funciones que se le podrían aplicar a imágenes descomprimidas puesto que la falta de pixeles puede tener unos factores en contra. Esto podría llevar a la falla de las funciones sin antes descomprimirlas puesto que la mayoría suele estructurar sus imágenes de una manera descomprimida para poder aplicarle cambios con las funciones creadas.

## 5. Diseño de la solución:

La forma en la que fue creada la solución fue respetando los requerimientos planteados, para crear una imagen se comprueba que ambos parámetros ingresado como ancho y alto sean enteros de esta forma me aceptaría los datos, y que respetando las restricciones antes entregadas acerca de pixbit , pixrgb y pixhex se entregue una lista con estos parámetros de manera homogénea, al mismo momento de crear la imagen esta lista es ordenada automáticamente ordenando por valores de sus x e y para poder aplicar las funciones que necesitan las coordenadas ordenadas, la mayoría de las funciones creadas para este laboratorio requerían que los pixeles de la imagen estuvieran ordenados por lo tanto esto genero la idea de ordenar todos los datos de los pixeles, para ordenar se ocupa la función sort que entregándole la lista de pixeles este ordena por los primeros dos datos de cada pixel con respecto a x e y así puede dejar de manera ordenada los pixeles respetando las posiciones que ocuparían dentro de la imagen. Se entregará un anexo explicando la descripción de cada función creada en este laboratorio.

## 6. Aspectos de implementación:

**Compilador:** SWI-Prolog desde la versión 8.4 o superior.

**Estructura código:** El código se estructura con funciones para filtrar TDAimage, en conjunto con las funciones para pixbit, pixhex, pixrgb y por consiguiente las funciones solicitadas a implementar.

## 7. Instrucciones de uso:

Para poder usar el código se deben tener todos los archivos ".pl" en una misma carpeta puesto que usan las funciones para importar para que las funciones sean aceptadas en otros archivos como el main. Se entregará un archivo .pl comentado en el cual se entregarán los llamados a funciones, para esto es necesario compilar el archivo en el cual están todas las funciones generadas y consultar los llamados a los predicados comentados en el consultor de prolog.

## 8. Resultados:

No se logro implementar las funciones obligatorias, si no hasta casi la mitad, la mayoría funciona de manera correcta, se hará entrega de un anexo evaluando cada función. La autoevaluación es 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.

## 9. Conclusión:

En comparación a la entrega anterior esta vez no se pudo completar los requerimientos básicos funcionales para este laboratorio, si bien se logro ampliar el conocimiento acerca del paradigma funcional, no se logro implementar correctamente con lo solicitado, en comparación al conocimiento aprendido con respecto al paradigma funcional se logra encontrar las diferencias, y como enfrentar problemas con este nuevo paradigma empleado, si bien el lenguaje Prolog tanto como Scheme son parecidos, no funciona de la misma manera, en tanto Prolog o el paradigma funcional viven en un ámbito más matemático, cumpliendo así con los antecedentes y consecuentes para luego ver si logra la funcionalidad de esta, en cambio el paradigma funcional tenia otro tipo de validaciones más programadas al ámbito de desarrollo propio, con respecto a que cada uno debía poner los requerimientos de las funciones y no cumplirlas en un ámbito algo matemático o del mundo de los universos y conjuntos.

## 9.1. Referencias:

*1). 5.3 Tipos de Datos en Prolog*. (n.d.). Retrieved from https://www.inf.utfsm.cl/~mcloud/iwi-253/apuntes/apunte05-03-2x.pdf

*2).* P-99: Ninety-Nine Prolog Problems. (2022). Retrieved November 4, 2022, from Unicamp.br website: https://www.ic.unicamp.br/~meidanis/courses/mc336/2009s2/prolog/problemas/

*3). Programación Elemental con Bases de Datos y Programación*. (n.d.). Retrieved from http://www.lcc.uma.es/~pacog/apuntes/pd/TemaII.pdf

# Anexo:

## Funciones del TDA bit

| Tipo función | Nombre | Descripción |
|---|---|---|
| Predicado | pixbit | Crea un píxel tipo bit |
| Clausula | imageIsBitmap | Comprueba si la imagen es un bitmap |

## Funciones del TDA rgb

| Tipo función | Nombre | Descripción |
|---|---|---|
| Predicado | pixrgb | Crea un píxel tipo rgb |
| Clausula | imageIsPixmap | Comprueba si la imagen es un pixmap |

## Funciones del TDA hex

| Tipo función | Nombre | Descripción |
|---|---|---|
| Predicado | pixhex | Crea píxel tipo hex |
| Clausula | imagenIsHexmap | Comprueba si la imagen es un hexmap |

# Funciones del TDA image

| Tipo función | Nombre | Descripción |
|---|---|---|
| Predicado | image | Crea una imagen |
| Clausula | imagenIsCompressed | Comprueba si la imagen esta comprimida |
| Clausula | pegarLista | Pega una lista invertida en una lista general |
| Clausula | agrupar | Agrupa una lista invertida con los datos ingresado |
| Clausula | imageFlipH | Gira la imagen horizontalmente |