

**The report HSLA Image color space**

## Objective : Applying inheritance knowledge to manipulate HSLa images and to create three classes: grayscale ,illini and spothlight which inherit from the image class

## Understanding the HSL Color Space :

**HSL** (for **hue, saturation, lightness** representations of the [RGB color model](https://en.wikipedia.org/wiki/RGB_color_model),The HSL representation models the way different paints mix together to create colour in the real world, with the *lightness* dimension resembling the varying amounts of black or white paint in the mixture (e.g. to create "light red", a red pigment can be mixed with white paint; this white paint corresponds to a high "lightness" value in the HSL representation). Fully saturated colors are placed around a circle at a lightness value of ½, with a lightness value of 0 or 1 corresponding to fully black or white, respectively.

[**The PNG class**](https://anassbelcaid.github.io/CS311/hslaimages/#the-png-class)

[**Inhertance diagram**](https://anassbelcaid.github.io/CS311/hslaimages/#inhertance-diagram) :

the goal is to write additional classes that **inherit** from this class and implement addtional functionalities.

**classed programming :**

**Image :**

**Image.h**

**#ifndef IMAGE\_H**

**#define IMAGE\_H**

**#include "PNG.h"**

***class* Image:*public* PNG**

**{**

***public*:**

***using* PNG::PNG;**

**Image(string);**

**void saturate(double amount=0.1);**

**void lighten(double amount=0.1);**

**void rotateColor(double angle);**

**};**

**#endif *//* *IMAGE\_H***

**Image c++**

**#include "image.h"**

**Image::Image(string filemane):PNG()**

**{**

**readFromFile(filemane);**

**}**

**void Image ::lighten(double amount){**

***//parcourir* *tous* *mes* *pixles***

***for*(unsigned x=0;x<width();x++)**

***for*(unsigned y=0;y<height();y++){**

**HSLAPixel &P=getPixel(x,y);**

***//augumenter* *la* *luminence***

**P.l+=amount;**

***//ne* *pas* *depasser* *1***

**P.l=(P.l<1)? P.l :1;**

***//ne* *pas* *depasser* *0***

**P.l=(P.l<0)? 0: P.l;**

**}**

**}**

**void Image::saturate(double amount){**

***//parcourir* *tous* *mes* *pixles***

***//parcourir* *tous* *mes* *pixles***

***for*(unsigned x=0;x<width();x++)**

***for*(unsigned y=0;y<height();y++){**

**HSLAPixel &P=getPixel(x,y);**

***//augumenter* *la* *luminence***

**P.s+=amount;**

***//ne* *pas* *depasser* *1***

**P.s=(P.s<1)? P.s :1;**

***//ne* *pas* *depasser* *0***

**P.s=(P.s<0)? 0: P.s;**

**}**

**}**

**void Image ::rotateColor(double angle){**

***for*(unsigned x=0;x<width();x++)**

***for*(unsigned y=0;y<height();y++){**

**HSLAPixel &P=getPixel(x,y);**

**P.h+=angle;**

***//s'asurer* *que* *P.h<360***

***while*(P.h<360)**

**P.h+=360;**

***//s'asurer* *que* *P.h>360***

***while*(P.h>360)**

**P.h-=360;**

**}**

**}**

**Grayscale class :**

**Grayscale.h**

**#ifndef GRAYSCALE\_H**

**#define GRAYSCALE\_H**

**#include"image.h"**

***class* grayscale:*public* Image**

**{**

***public*:**

**grayscale();**

***using* Image::Image;**

**void reducesaturation();**

**};**

**#endif *//* *GRAYSCALE\_H***

**Grayscale.c++**

**#include "grayscale.h"**

**grayscale::grayscale()**

**{**

**}**

**void grayscale::reducesaturation() {**

***for*(unsigned x=0;x<width();x++)**

***for*(unsigned y=0;y<height();y++){**

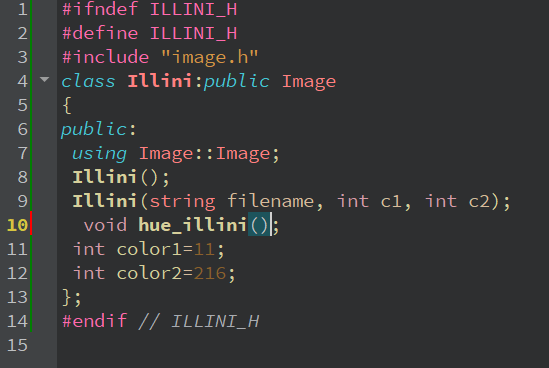
**HSLAPixel &P=getPixel(x,y);**

**P.s=0;**

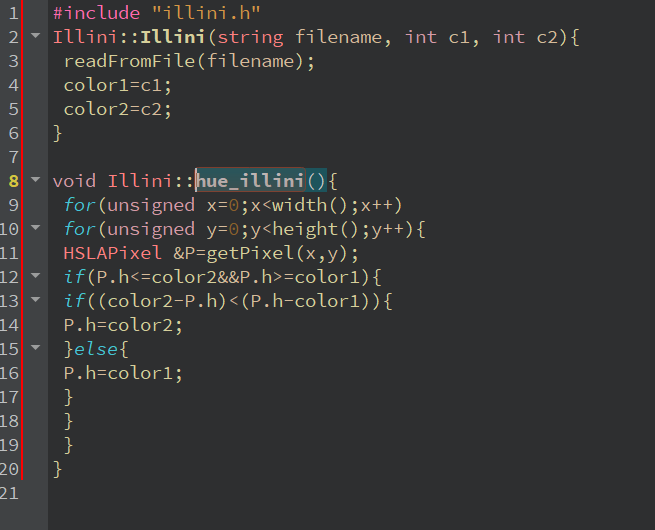
**}}**

**Illini class : the first case**

**Illini.h**



**Illini.c++**



**Illini the second case**

**Illini.h**

**#ifndef ILLINI\_H**

**#define ILLINI\_H**

**#include "image.h"**

***class* Illini:*public* Image**

**{**

***public*:**

***using* Image::Image;**

**Illini();**

**Illini(string filename, int c1, int c2);**

**void hue\_illini(int c1 ,int c2);**

**int color1;**

**int color2;**

**};**

**#endif *//* *ILLINI\_H***

**Illini.c++**

**#include "illini.h"**

**Illini::Illini(string filename, int c1, int c2){**

**readFromFile(filename);**

**color1=c1;**

**color2=c2;**

**}**

**void Illini::hue\_illini(int c1 , int c2 ){**

***for*(unsigned x=0;x<width();x++)**

***for*(unsigned y=0;y<height();y++){**

**HSLAPixel &P=getPixel(x,y);**

***if*(P.h<=c2&&P.h>=c1){**

***if*((c2-P.h)<(P.h-c1)){**

**P.h=c2;**

**}*else*{**

**P.h=color1;**

**}**

**}**

**}**

**}**

**Spothlight class :**

**Spothlight.h**

**#ifndef SPOTLIGHT\_H**

**#define SPOTLIGHT\_H**

**#include "image.h"**

***class* Spotlight : *public* Image**

**{**

***public*:**

***using* Image::Image;**

**int Cx=200;**

**int Cy=400;**

**Spotlight();**

**Spotlight(string filename, int cx, int cy);**

**void spotlighting();**

**};**

**#endif *//* *SPOTLIGHT\_H***

**Spothlight.c++**

**#include "spotlight.h"**

**#include "math.h"**

**Spotlight::Spotlight(){**

**}**

**Spotlight::Spotlight(string filename, int cx, int cy){**

**readFromFile(filename);**

**Cx=cx;**

**Cy=cy;**

**}**

**void Spotlight::spotlighting()**

**{**

***for*(unsigned x=0;x<width();x++)**

***for*(unsigned y=0;y<height();y++){**

**HSLAPixel &P=getPixel(x,y);**

***auto* distance=sqrt((Cx-x)\*( Cx-x)+( Cy-y)\*(Cy-y));**

***if*(distance>160){**

**P.l=P.l-0.8\*P.l;**

**}*else*{**

**P.l=P.l-(P.l\*(0.5\*distance)/100);}**

**}}**

**Made by Ghita Chraibi**