

## Assignment #6 [9 Marks]

Date / Time	6 June 2022 – 19 June 2022 23:59
Course	<b>[M1522.00600] Computer Programming</b>
Instructor	Jae W. Lee

- You are allowed to discuss with fellow students to understand the questions better, but your code must be **SOLELY YOURS**. You **MUST NOT** show your code to anyone else, or vice versa.
- We will use the automated copy detector to check the possible plagiarism of the code between the students. The copy checker is reliable so that it is highly likely to mark a pair of code as the copy even though two students quickly discuss the idea without looking at each other's code. Of course, we will evaluate the similarity of a pair compared to the overall similarity for the entire class.
- We will do the manual inspection of the code. In case we doubt that the code may be written by someone else (outside of the class like github), we reserve the right to request an explanation about the code. We will ask detailed questions that cannot be answered if the code is not written by yourself.
- If any of the above cases happens, you will **get 0 mark** for the assignment and may get **a further penalty**.
- Download and unzip "HW6.zip" file from the NeweTL. "HW6.zip" file contains skeleton codes for Question 1 and 2 (in "problem1" and "problem2" directory, respectively).
- Do not modify the overall directory structure after unzipping the file, and fill in the code in appropriate files. It is okay to add new directories or files if needed.
- When you submit, compress the "HW6" directory which contains "problem1" and "problem2" directories in a single zip file named "**{studentID}.zip**" (e.g. 2022-12345.zip) and upload it to NeweTL. Contact the TA if you are not sure how to submit. Double-check if your final zip file is properly submitted. You will **get 0 mark** for the wrong submission format.
- CLion sometimes successfully completes the compilation even without including some header files. Nevertheless, you **MUST include all the necessary header files in the code** to make it successfully compile on our server. Otherwise, you may not be able to get the points.
- C++ Standard Library (including Standard Template Library) is allowed.
- Do not use any external libraries. If you are not sure, contact TAs to clarify.
- Trailing whitespaces (space, line break) at the end of the lines are allowed.

# Contents

## Submission Guidelines

### Problem 1. Shopping Application [4 Marks]

- 1-1. Interface for Admins [0.5]
- 1-2. Interface for Users [1.5]
- 1-3. Product Recommendation [2]

### Problem 2: Simple Digital Image Processing [5 Marks]

- 2-1. Implement Scene & Layer Classes [1]
- 2-2. Save the Scene [1]
- 2-3. Layer Manipulation Functions I [1]
- 2-4. Layer Manipulation Functions II & Stamp [1.5]
- 2-5. Have some fun [0.5]

### References: Standard Template Library (STL) for Assignment 6

- 1. vector (for Assignment 6-1 and 6-2)
- 2. unordered\_set (for Assignment 6-1)

## Submission Guidelines

1. You should submit your code on NewETL.
2. After you extract the zip file, you must have an “HW6” directory. The submission directory structure should be as shown in the table below.
3. You can create additional directories or files in each problem directory. Make sure to update CMakeLists.txt to reflect your code structure changes.
4. You can create additional methods or classes, but do not remove or change signatures of existing methods.
5. Compress the “HW6” directory and name the file “20XX-XXXXX.zip” (your student ID).

### Submission Directory Structure (Directories or Files can be added)

- Inside the “HW6” directory, there should be “problem1” and “problem2” directory.

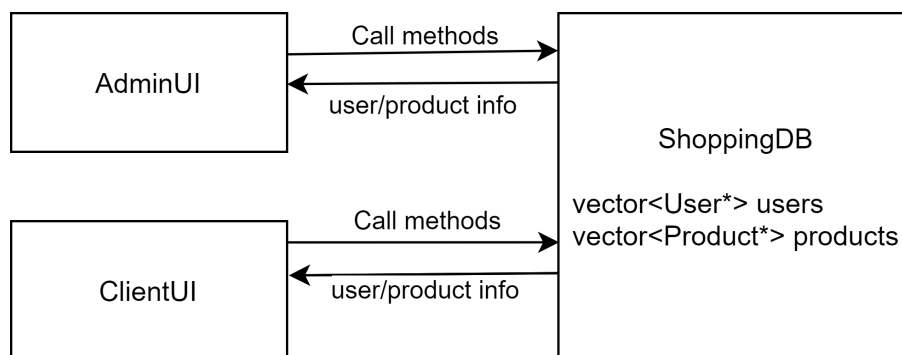
Directory Structure of Problem 1	Directory Structure of Problem 2
<pre> problem1/ ├── test/ ├── CMakeLists.txt ├── admin_ui.cpp ├── admin_ui.h ├── client_ui.cpp ├── client_ui.h ├── shopping_db.cpp ├── shopping_db.h ├── ui.h ├── ui.cpp ├── user.h ├── user.cpp ├── product.h ├── product.cpp ├── test.cpp └── (You can add more files) </pre>	<pre> problem2/ ├── data/ ├── tools/ │   ├── LayerManipulator.cpp │   ├── LayerManipulator.h │   ├── Stamp.cpp │   └── Stamp.h ├── CMakeLists.txt ├── compareImage.cpp ├── Layer.cpp ├── Layer.h ├── Scene.cpp ├── Scene.h ├── main.cpp ├── stb_image.h ├── stb_image_resize.h ├── stb_image_write.h ├── utils.h ├── yourimage.cpp ├── yourimage.h └── (You can add more files) </pre>

## Problem 1: Shopping Service [4 Marks]

**Objective:** Implement a simple shopping service with client/admin interfaces and a database to store the product and user information.

**Descriptions:** You are a senior developer of an online shopping mall. Your mission is to implement an online shopping service to manage the product and user information. In addition, you are requested to implement a customized product recommendation functionality.

Before you jump into the implementations, you may want to understand the overall structure of this service. The below figure shows the overall structure of the shopping system.

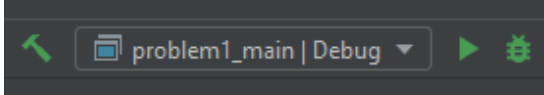
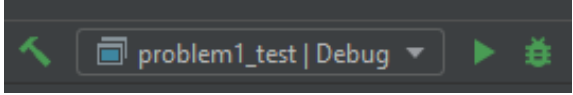
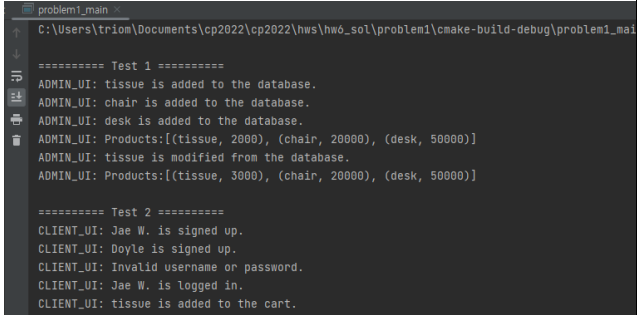
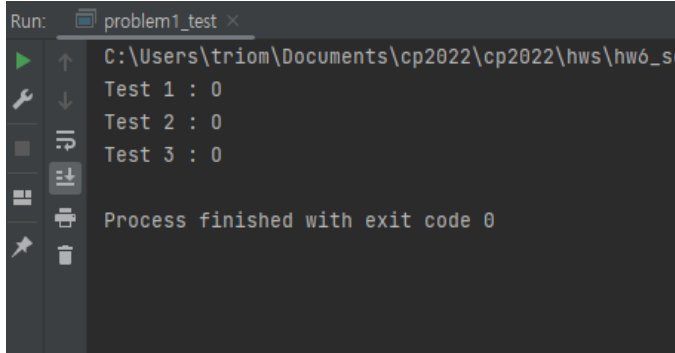


The system mainly consists of 3 classes: `AdminUI`, `ClientUI`, and `ShoppingDB` (along with a few additional classes supporting these key classes.)

- The `AdminUI` class provides interfaces for admins.
- The `ClientUI` class provides interfaces for clients.
- The `ShoppingDB` object receives the queries from the `AdminUI` object and `ClientUI` object, and returns the corresponding data.

How to develop and test assignment 6-1:

- For convenience, we provide two different modes to run your code: "MAIN" and "TEST". In the MAIN mode, "test.cpp" prints the output to the console. In the TEST mode, it compares your results and the expected results in "test/{1,2,3}.out". You can choose the desired mode in the CLion toolbar at the top.
- "test.cpp" includes the example test cases specified in this document. However, we will evaluate your code with a richer set of test cases. We strongly encourage you to add more test cases to make sure your application works as expected.
- If you want to add more test cases to "test/{1,2,3}.out", press **"File" → "Reload CMake Project"** in the CLion before you run "test.cpp". Otherwise, the additional test cases are not copied to the build directory (cmake-build-debug).

MAIN mode	TEST mode
	
	

A few important notes:

- Do **\*NOT\*** modify the method signatures of `AdminUI` and `ClientUI` declared in "admin\_ui.h" and "client\_ui.h", respectively. For evaluation, we will use the functions as declared in these header files. Also, do **\*NOT\*** modify "ui.h" and "ui.cpp".
- Do **\*NOT\*** remove `add_executable(problem1_test ...)`, and `target_compile_definitions(problem1_test PRIVATE TEST)` from "CMakeLists.txt". We will use them for evaluation.
- Feel free to add new functions to `AdminUI` and `ClientUI`. You can also add new functions or modify existing functions in other classes.
- You can add new ".h" and ".cpp" files, but you **\*MUST\*** add them to "CMakeLists.txt". We will use your "CMakeLists.txt" to evaluate your code.
- There are dependencies between subquestions. **This means that the wrong implementation of a method may affect other methods**, and you may lose scores for multiple problems.
- You do not have to store the product and user data in files. It is sufficient to store the data in memory with proper data structures.
- You **\*MUST\*** use `std::ostream` os of the `AdminUI` and `ClientUI` class to print outputs to the console or for the final evaluation.

Great! You are now ready to implement the cool shopping system.

## Problem 1-1: Interface for Admins [0.5 Mark]

**Objective:** Implement three methods of `AdminUI` to manage product data.

**Descriptions:** For the first step, implement an interface for admins to manage product data. In particular, implement the following methods of `AdminUI`. Feel free to change the `ShoppingDB` class; for instance, you can add functions or modify given functions and parameters. However, do **\*NOT\*** change the function signatures in the `AdminUI` class.

### Function 1. *Add Product*

- Implement `void ShoppingDB::add_product(std::string name, int price)`
  - This function adds a product entry with the given name and price in the database.
- Implement `void AdminUI::add_product(std::string name, int price)`
  - This function adds a product entry in the database using the `ShoppingDB::add_product` function.
  - If the given price is positive, print "ADMIN\_UI: <PRODUCT\_NAME> is added to the database.". ("<PRODUCT\_NAME>" should be replaced with the name of the product; do **not** print "<" and ">".)
  - If the given price is equal to or less than zero, print "ADMIN\_UI: Invalid price.".
  - Assume that no duplicated names of products or users will be given.

### Function 2. *Edit Product*

- Implement `bool ShoppingDB::edit_product(std::string name, int price)`
  - This function modifies the prices of the product of a given name in the database.
  - Modify the price only when the price is positive.
  - Return true when the modification is successful.
- Implement `void AdminUI::edit_product(std::string name, int price)`
  - This function modifies the price of the product of a given name using the `ShoppingDB::edit_product` function.
  - If there is a product with the given name in the database, print "ADMIN\_UI: <PRODUCT\_NAME> is modified from the database.". ("<PRODUCT\_NAME>" should be replaced with the name of the product; do **not** print "<" and ">".)
  - If the given price is equal to or less than zero, print "ADMIN\_UI: Invalid price." without modifying the info of the product.
  - If there is no product with the given name in the database, print "ADMIN\_UI: Invalid product name.". If this is the case, do not check if the given price is valid.

### Function 3. *List Products*

- Implement the `get` method of `products` in the `ShoppingDB` class.
- Implement `void AdminUI::list_products()`.
  - This function prints the list of entire products with the following format:  
"ADMIN\_UI: Products: [(<PRODUCT\_NAME1>, <PRODUCT\_PRICE1>), (<PRODUCT\_NAME2>, <PRODUCT\_PRICE2>), ..., (<PRODUCT\_NAME<N>, <PRODUCT\_PRICE<N>)]". (<PRODUCT\_NAME#> should be replaced with the name of the product, <PRODUCT\_PRICE#> should be replaced with the price of the product; do **not** print "<" and ">". Note that there is a space behind the comma.)
  - Print the products in the order of the sequence the products are added to the database with the `add_product` method.
  - If there is **no** product in the database, print "ADMIN\_UI: Products: []".

### Expected result for the skeleton test case:

```
===== Test 1 =====
ADMIN_UI: tissue is added to the database.
ADMIN_UI: chair is added to the database.
ADMIN_UI: desk is added to the database.
ADMIN_UI: Products: [(tissue, 2000), (chair, 20000), (desk, 50000)]
ADMIN_UI: tissue is modified from the database.
ADMIN_UI: Products: [(tissue, 3000), (chair, 20000), (desk, 50000)]
```

## Problem 1-2: Interface for Users [1.5 Mark]

**Objective:** Implement methods of `ClientUI` to support the users.

**Descriptions:** `ClientUI` provides an interface for clients to login, logout, search for products. Also, it allows a user to add products to his/her shopping cart and purchase them. There are two types of users, "normal user" and "premium user". Premium users get discounts off the normal price while normal users do not have any discount. (HINT: Refer to `user.h` to set the discount rate for each user type.) Feel free to change the `ShoppingDB` class; for instance, you can add functions or modify given functions and parameters. However, do **\*NOT\*** change the parameters of the functions in the `ClientUI` class.

**Part 1. Sign-up, Login, and Logout:** Implement a sign-up, login, and logout functions of `ClientUI`. You also need to modify/add functions in the `ShoppingDB` class to store user data in the `ShoppingDB` object.

## Function 1. *Sign-up*

- Implement `void ShoppingDB::add_user(std::string username, std::string password, bool premium)`.
  - This function creates an account with the given username and password. Then, it adds the account to the database.
  - Assume that **no** duplicated username is given.
  - The premium parameter specifies whether the type of the user is a "premium user" or a "normal user". (if `premium` is true, it indicates that user is a premium user)
- Implement `void ClientUI::signup(std::string username, std::string password, bool premium)`.
  - This function creates an account with the given username and password using the `ShoppingDB::add_user` function.
  - Print "`CLIENT_UI: <USERNAME> is signed up.`" when a user signed up. ("`<USERNAME>`" should be replaced with the name of the user; do **not** print "`<`" and "`>`".)

## Function 2. *Login*

- Implement `void ClientUI::login(std::string username, std::string password)`.
  - This allows a user to log in to the service. This means that you need to keep the log-in state of a user in an appropriate place.
    - HINT: The private variable `current_user` can be used to indicate and check whether the user is currently logged in.
  - If the login is successful, print "`CLIENT_UI: <USERNAME> is logged in.`". ("`<USERNAME>`" should be replaced with the name of the user; do **not** print "`<`" and "`>`".)
  - If there is no user with the given name or the password does not match, print "`CLIENT_UI: Invalid username or password.`".
  - If there is a user currently logged-in, print "`CLIENT_UI: Please logout first.`". If this is the case, do not check if the given username and password are valid.

## Function 3. *Logout*

- Implement `void ClientUI::logout()`.
  - This allows a user to log out from the service.
  - If there is a currently logged-in user, print "`CLIENT_UI: <USERNAME> is logged out.`".
  - Otherwise, print "`CLIENT_UI: There is no logged-in user.`".



**Part 2. Add to cart and Purchase:** Implement “add-to-cart” and “purchase” functions. In particular, implement the following methods of the `ClientUI`. You also need to modify/add functions in the `ShoppingDB` class to store user data in the `ShoppingDB` object.

- For all the functions (i.e., `buy`, `add_to_cart`, `list_cart_products`, and `buy_all_in_cart` functions), check if a user is logged in.
  - If there is no logged-in user, print “CLIENT\_UI: Please login first.”.
  - If a user is logged in, perform the requested task.

### Function 1. *Buy*

- Implement `void User::add_purchase_history(Product* product)`.
  - Before buying the product, store the purchase information in the database. The `ClientUI::buy_all_in_cart` function and Problem 1-3 will use this purchase information of users.
- Implement `void ClientUI::buy(std::string product_name)`
  - This function prints the price of a product with the given name (format: “CLIENT\_UI: Purchase completed. Price: <PRICE>.”).
  - A “premium user” gets a 10% discount (round off to the nearest integer, round up if tenths digit is greater than or equal to 5). Print the discounted price for a premium user, and the original price for a normal user.
  - If the given product name is invalid, print “CLIENT\_UI: Invalid product name.”.
  - Note that this method has nothing to do with the products in the cart.

### Function 2. *Add to the cart*

- Implement `void ClientUI::add_to_cart(std::string product_name)`.
  - This function adds a product to the cart of the logged-in user.
  - The user can add the same product to the cart multiple times.
  - Print “CLIENT\_UI: <PRODUCT\_NAME> is added to the cart.” after adding the product to the cart.
  - If there is no product with the given name, print “CLIENT\_UI: Invalid product name.”.

### Function 3. *List all products in the cart*

- Implement `void ClientUI::list_cart_products()`.
  - This function prints all products in the cart of the logged-in user in ascending order of time (oldest first) that the products are added to the cart.
  - The output format is “CLIENT\_UI: Cart: [(<PRODUCT\_NAME1>, <PRODUCT\_PRICE1>), (<PRODUCT\_NAME2>, <PRODUCT\_PRICE2>), ...]”. For example, if the user puts an apple, a banana, and an apple to the cart in

a sequence, then print "CLIENT\_UI: Cart: [(Apple, 1000), (Banana, 1500), (Apple, 1000)]".

- If there is no product in the cart, print "CLIENT\_UI: Cart: []".
- You can assume that products in the database are not deleted when listing the product in the cart.
- Again, the discounted prices (i.e., 10% discount, round off to the nearest integer, round up if the first digit of the decimal point is 5) should be displayed for a premium user and the original prices for a normal user.

#### Function 4. *Buy all products in the cart*

- Implement void ClientUI::buy\_all\_in\_cart().
  - Before buying all products in the cart, store the purchase information in the database, as you did in ClientUI::buy function.
  - Print the following message: "CLIENT\_UI: Cart purchase completed. Total price: <TOTAL\_PRICE>." After that, clear the cart.
  - If the cart is empty, print a message in the same format with zero total price.
  - After purchase, make the cart empty

#### Expected result for the skeleton test case:

```
===== Test 2 =====
CLIENT_UI: Jae W. is signed up.
CLIENT_UI: Doyle is signed up.
CLIENT_UI: Invalid username or password.
CLIENT_UI: Jae W. is logged in.
CLIENT_UI: tissue is added to the cart.
CLIENT_UI: chair is added to the cart.
CLIENT_UI: desk is added to the cart.
CLIENT_UI: Cart: [(tissue, 2700), (chair, 18000), (desk, 45000)]
CLIENT_UI: Cart purchase completed. Total price: 65700.
CLIENT_UI: Please logout first.
CLIENT_UI: Jae W. is logged out.
CLIENT_UI: Please login first.
CLIENT_UI: Doyle is logged in.
CLIENT_UI: Purchase completed. Price: 20000.
CLIENT_UI: Doyle is logged out.
CLIENT_UI: There is no logged-in user.
```

## Problem 1-3: Product Recommendation [2 Marks]

**Objective:** Implement `void ClientUI::recommend_products()`.

**Descriptions:** Now, you are asked to implement a product recommendation method based on the user's purchase pattern. The recommendation for premium users will be different from that of normal users. In particular, implement `void ClientUI::recommend_products()` which prints the recommended products for the logged-in user. The following are requirements and guidelines with details.

- The printing format of `void ClientUI::recommend_products()` is "CLIENT\_UI: Recommended products: [(<PRODUCT\_NAME1>, <PRODUCT\_PRICE1>), (<PRODUCT\_NAME2>, <PRODUCT\_PRICE2>), ...]".
- If there is no product to recommend, print "CLIENT\_UI: Recommended products: []".
- The discounted prices (i.e., 10% discount, round off to the nearest integer, round up if the first digit of the decimal point is 5) should be displayed for a premium user and the original prices for a normal user.
- The print messages should reflect the current prices. Remember that the prices of the products can be modified with the `void AdminUI::edit_product(std::string name, int price)` method.
- For the method `recommend_products` check if a user is logged in.
  - If there is no logged-in user, print "CLIENT\_UI: Please login first.", instead of printing the recommended products.
  - If a user is logged in, perform the requested task.

The following explains more details about the recommendation policies:

- Recommendation for a "normal user":
  - Recommend three most recently purchased items of the logged-in user (which will be the current user). Recommend three *unique*, most bought items on the list. If the number of purchases is equal between items, recommend the most recent one. The recommendation should stay the same if the purchase history was not altered.
  - For instance, if a user purchased products "a(oldest)-d-c-b-c-c(newest)" in sequence, the recommended items should be "c-b-d".
  - If the user purchased multiple products at the same time with a `void ClientUI::buy_all_in_cart()` call, assume that the product added to the cart later is purchased later. For example, if 1) a user adds a "tissue", and then a "chair" to the cart, 2) calls `void ClientUI::buy_all_in_cart()`, and 3) buys a "wallet" with `void ClientUI::buy()`, the recommendation list should be "wallet", "chair", and "tissue" in order.
  - The number of items can be less than three if the purchase history is short.

- Recommendation for a "premium user":
  - Recommend **up to three** recently purchased items of other users with the "similar" purchase history. The detailed logic is described below.
  - First, sort all users (except for the currently logged-in user) based on purchase history similarities. The similarity between two users is defined as the number of products purchased by both users. In case two users have the same similarities with the logged-in user, the user registered earlier goes first.
  - Then, recommend the **most recently purchased products** of the three users with the highest similarities. Recommend products in descending order of the similarity score.
  - The three recommended items should be unique. If there are duplicate items, consider it only once and include the most recently purchased product of the next similar user. The number of recommended items can be less than three if there aren't enough unique most recently purchased items.
  - If the user purchased multiple products at the same time with a `void ClientUI::buy_all_in_cart()` call, assume that the product added to the cart later is purchased later.
  - For example, let's see how the recommended list can be decided for a premium user, Amy. The table below shows the purchase history of 5 different users (including Amy) and the purchase history similarities with Amy.

The most recently purchased product of a user is marked in **blue**.

Name (Oldest ↓ Newest)	Purchase History (Oldest →Newest)	Commonly purchased items	Purchase history similarity
<b>Amy</b>	A-B-C-D- <b>C</b>		-
Bobby	A-C- <b>A</b>	Two As, One C	3
Carry	B-B-C-D-A- <b>E</b>	One A, Two Bs, One C, One D	5
Dally	A-C-E- <b>B</b>	One A, One B, One C	3
Emily	C-C- <b>A</b>	One A, Two Cs	3
Freddy	A-A-A-A-A- <b>A</b>	Six As	6

The top three users with the highest similarities are Freddy, Carry, and Bobby; here the similarities of Bobby, Dally and Emily are the same, but Bobby was considered since he was registered before Dally and Emily. The most recently purchased products of the three users with the highest similarities is A-E-A in order. However, A is duplicated. Thus, instead of recommending two As,

recommend B as the third item, which is the most recently purchased product of the next similar user, Dally. So the final recommendation is A-E-B.

**Expected result for the skeleton test case:**

```
===== Test 3 =====
CLIENT_UI: SeungYul is signed up.
CLIENT_UI: Soosung is signed up.
CLIENT_UI: Rihae is signed up.
CLIENT_UI: SeungYul is logged in.
CLIENT_UI: Purchase completed. Price: 20000.
CLIENT_UI: Purchase completed. Price: 20000.
CLIENT_UI: Purchase completed. Price: 20000.
CLIENT_UI: Purchase completed. Price: 50000.
CLIENT_UI: SeungYul is logged out.
CLIENT_UI: Rihae is logged in.
CLIENT_UI: Purchase completed. Price: 20000.
CLIENT_UI: Purchase completed. Price: 3000.
CLIENT_UI: Recommended products: [(tissue, 3000), (chair, 20000)]
CLIENT_UI: Rihae is logged out.
CLIENT_UI: Jae W. is logged in.
CLIENT_UI: Recommended products: [(desk, 45000), (tissue, 2700), (chair, 18000)]
CLIENT_UI: Jae W. is logged out.
```

## Problem 2: Mini-PhotoShop! [5 Marks]

**Objective:** Write a simple digital image processing application in C++. It can adjust color values of an image, blur the image, mask and merge multiple images to construct a new image.

**Descriptions:** You will write a simple digital image processing application. Please note:

- Default test cases specified in this document are provided as functions in the "main.cpp". You can test your code with "main.cpp" by comparing your result with the expected outputs. For grading, we will apply a richer set of test cases. We strongly encourage you to add more diverse test cases to make sure your application works as expected.
- Exact value of output can be compared with the execution file generated from compareImage in cmake. It takes the file path of two images as arguments and if the size is correct it checks the per channel root mean square error (RMSE) between each pixel value of two images. **To be correct, all RMSE of each channel should be smaller than 2.** Arguments can be given in the command line in terminal or as "Program arguments" option in Run/Debug configuration.
- We provide one target executable code: "main.cpp". Use the test function in "main.cpp" in main() to run specific test cases.
- Do **\*NOT\*** modify any predefined function signatures, and do **\*NOT\*** remove add\_executable(problem2 ...) from "CMakeLists.txt" since we will use them for grading.
- You can add new ".h" and ".cpp" files, but do **\*NOT\*** forget to add them to "CMakeLists.txt". We will use your "CMakeLists.txt" to test your codes.
- If you want to add or modify the test cases in the "test" directory, press **"File → Reload CMake Project"** in the CLion before you run "test.cpp" or "main.cpp". Otherwise, the added/modified test cases are not copied to the build directory (cmake-build-debug).
- There are 5 subproblems that are meant to be solved in order. If you proceed without solving the earlier problems, it may influence the later problems.
- When rounding floating point to integer data type (int, unsigned char, etc) use std::round() in <cmath>.
- **You MUST use a relative path to the image file, that is ./problem2 as root.** For example, to access image0.jpg in the data directory, use file path "./data/image0.jpg". To use relative paths when using CLion, change the "Working directory" option in Run/Debug configuration.

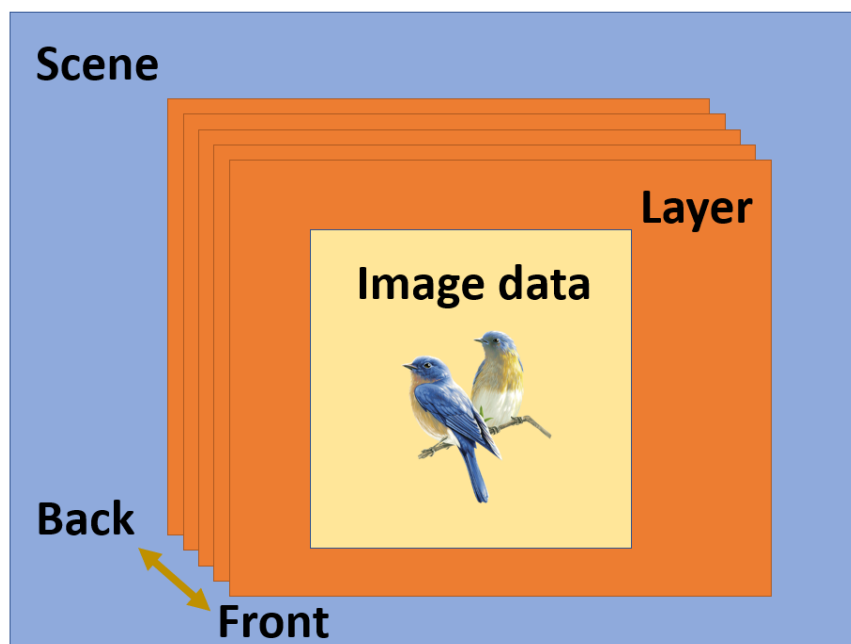
Great! You are now ready for the last problem of this course! You are almost there!

## Problem 2-1: Implement Scene & Layer Classes [1 Mark]

**Objectives:** Implement functions in Scene (Scene.cpp, Scene.h) and Layer classes (Layer.cpp, Layer.h) by following the specifications. Feel free to add any new functions or parameters to classes. Just make sure function signatures of predefined functions are not modified.

**Descriptions:** Figure 1 is a diagram of how Scene and Layer objects are organized. A Scene instance is composed of multiple Layer instances. There are orders between Layers in the Scene. The Layer in the front will appear on top of other Layers. So, a Scene should have a data structure of your choice (e.g. array, vector, list, map) to organize and store the layers. In this data structure each Layer should be stored as Layer\*. When saving an image from a Scene (Problem 2-2) Scene merges Layers that are in a visible region of the scene (a.k.a. Canvas) defined by width and height of the Scene.

Each Layer instance has a pointer to its own image data which is an array of unsigned int8 (== unsigned char).



**Figure 1.** Diagram on how Scene and Layers are organized.

### Descriptions of Functions to Implement:

The table below are specifications of the function that you have to implement.

Member functions to Implement in class Scene	
Explanation for given variables of Scene	Width: Width of image to save Height: Height of image to save Assume that all Width and Height used for constructing new Scene class are non-zero positive integers.

<pre>int addLayerFromFile(string filename, string layername)</pre>	<p><b>Description</b> Create a Layer instance with a given name and image data loaded from the given filepath. Then add it to the current Scene instance.</p> <p><b>Inputs variables</b> filename: file path to target image layername: name of Layer being created</p> <p><b>Return value</b> Return 0 if success, else return 1 (No file with given filename, etc.)</p> <p><b>Detailed Specification</b> Created Layer instances should have image data corresponding to image at filename and layername. It should be added at the frontmost position of this Scene instance. Data type of the loaded image should be an unsigned int8(==char) array with 4 channels for a pixel. (i.e. [R0, G0, B0, A0, R1, G1, B1, A1 ... ] where R stands for Red and G for Green and B for blue value and A for alpha value.) Use the stbi_load() function from “stb_image.h”.</p>
<pre>int addLayerfromLayer(string layername, string baselayername)</pre>	<p><b>Description</b> Make a copy of existing Layer and add it to current Scene instance</p> <p><b>Inputs variables</b> baselayername: name of existing Layer to copy from layername: name of newly created Layer</p> <p><b>Return value</b> Return 0 if success else return 1 (No Layer with given baselayername, etc.)</p> <p><b>Detailed Specification</b> <u>Assume no duplicate names are given when creating a Layer.</u> Copied Layer should be independent from changes made to the original Layer(Deep copy). New Layer should be added at the frontmost position of this Scene instance.</p>
<pre>int removeLayer(string layername)</pre>	<p><b>Description</b> Remove a single Layer with a given layername from the Scene and delete it.</p> <p><b>Inputs variables</b> layername: the name of Layer to remove and delete from this Scene instance</p> <p><b>Return value</b> Return 0 if success else return 1 (No Layer with given layername, etc.)</p> <p><b>Detailed Specification</b> <u>Assume no duplicate names are given when creating a Layer.</u></p>
<pre>Layer* selectLayer(string layername)</pre>	<p><b>Description</b> Search this Scene instance for Layer with matching layername and return the pointer to that Layer</p> <p><b>Inputs variables</b> layername: the name of Layer to find and return Layer* .</p> <p><b>Return value</b> Layer* of the matching Layer. If there is no such Layer return NULL.</p> <p><b>Detailed Specification</b> <u>Assume no duplicate names are given when creating a Layer.</u></p>
<pre>int changeLayerOrder(string layername, int order)</pre>	<p><b>Description</b> Change the order of Layer that matches with layername to given order.</p>



	<p><b>Inputs variables</b>  layername: the name of the Layer to change the order  order: integer value that specifies the order of the Layer.</p> <p><b>Return value</b>  Return 0 if success else return 1 (No Layer with given layername, order is not valid, etc.)</p> <p><b>Detailed Specification</b>  In a Scene backmost Layer has order of 0 and frontmost Layer has order total_number_layer - 1.  order should range from -1 to total_number_layer - 1, where -1 is an alias for frontmost position (== total_number_layer - 1). This is for convenience.</p>
int changeCanvasSize(int Width, int Height)	<p><b>Description</b>  Change the Width and Height of this Scene</p> <p><b>Inputs variables</b>  Width: value to change current Width to.  Height: value to change current Height to.</p> <p><b>Return value</b>  Return 0 if success else return 1 (Width or Height&lt;= 0, etc.)</p> <p><b>Detailed Specification</b>  Make sure Width and Height are positive integers.</p>

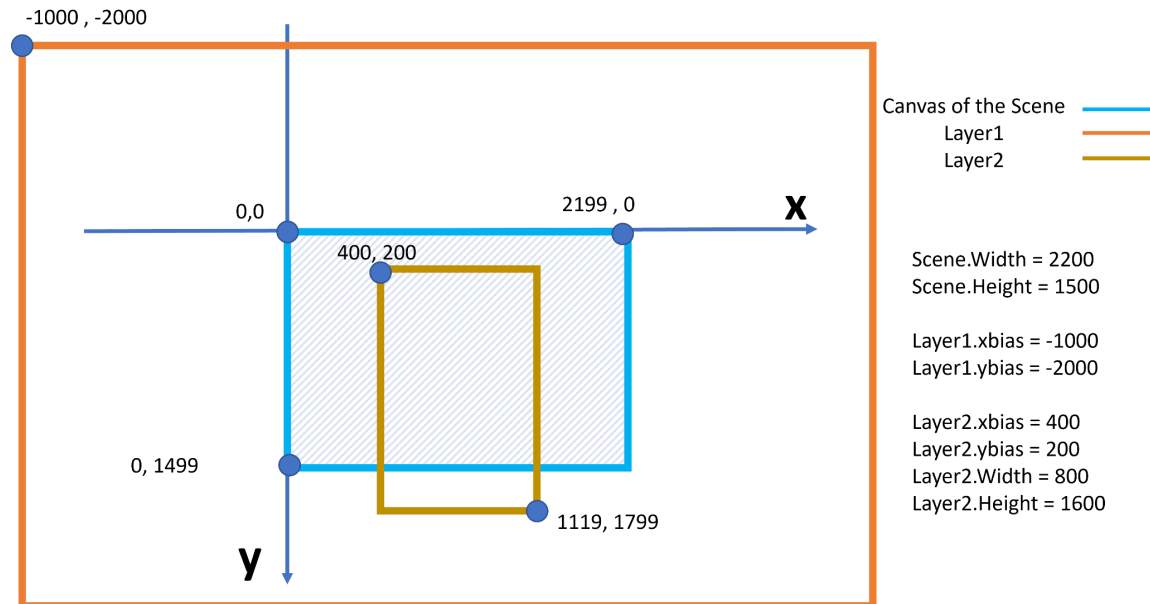
Member functions to Implement in class Layer	
Explanation for given variables of Layer	<p><b>int Width, Height</b> : Size of the Image (image is rectangle. Top left is (0,0) bottom right is (Width-1, Height-1) ) Assume that all Width and Height used for constructing new Layer class are non-zero positive integers.</p> <p><b>int x_bias, y_bias</b> : Position of image respective to the Canvas of the Scene (for further explanation please read the description at Problem 2.2)</p> <p><b>std::string layername</b> : name of this layer</p> <p><b>uint8_t* RGBA</b> : the data array that stores the image (data structure is explained above)</p> <p><b>bool visible</b> : boolean that represent whether to consider this layer when generating the output image of the Scene</p>
Layer(string layername, const Layer* layer)	<p><b>Description</b>  Deep Copy constructor</p> <p><b>Inputs variables</b>  layername: name of created Layer instance  layer: Layer object to copy from</p> <p><b>Return value</b>  None</p> <p><b>Detailed Specification</b>  Keep in mind that when image data is loaded using stbi_load it is allocated using malloc, not new. So when allocating a new array to copy the data, use malloc for simplicity of the destructor.</p>
~Layer()	<p><b>Description</b>  Destructor of Layer</p>

	<p><b>Inputs variables</b> None</p> <p><b>Return value</b> None</p> <p><b>Detailed Specification</b> Keep in mind that when image data is loaded using stbi_load it is allocated with malloc (not new).</p>
int resizeLayer(int newWidth, int newHeight)	<p><b>Description</b> Resize the image to given width and height</p> <p><b>Inputs variables</b> newWidth: width of resized image newHeight: height of resized image</p> <p><b>Return value</b> Return 0 if success, else return 1 (newWidth or newHeight&lt;= 0 , error during resizing, etc.)</p> <p><b>Detailed Specification</b> Use stbir_resize_uint8() function in "stb_image_resize.h". For a detailed description read the comments in the header file or just google it.</p>
uint8_t getdata(int x, int y, int channel, int* status)	<p><b>Description</b> Return the image data value of given x, y, channel position.</p> <p><b>Inputs variables</b> x: x coordinate of data y: y coordinate of data channel: channel of data status: It is an address of integer value. The *status will be assign 0 if success else it will be assign 1 (if invalid x,y,channel)</p> <p><b>Return value</b> If success value of given query else 0</p> <p><b>Detailed Specification</b> None</p>
void setdata(int x, int y, int channel, uint8_t data, int* status)	<p><b>Description</b> Store the image data at given x, y, channel position.</p> <p><b>Inputs variables</b> x: x coordinate of data y: y coordinate of data channel: channel of data data: value to store at the designated position status: It is an address of integer value. The *status will be assign 0 if success else it will be assign 1 (if invalid x,y,channel)</p> <p><b>Return value</b> None</p> <p><b>Detailed Specification</b> None</p>

## Problem 2-2: Save the Scene [1 Mark]

**Objective:** Implement a function that stores the image in the Canvas of a Scene. (Scene.cpp)

**Description:**

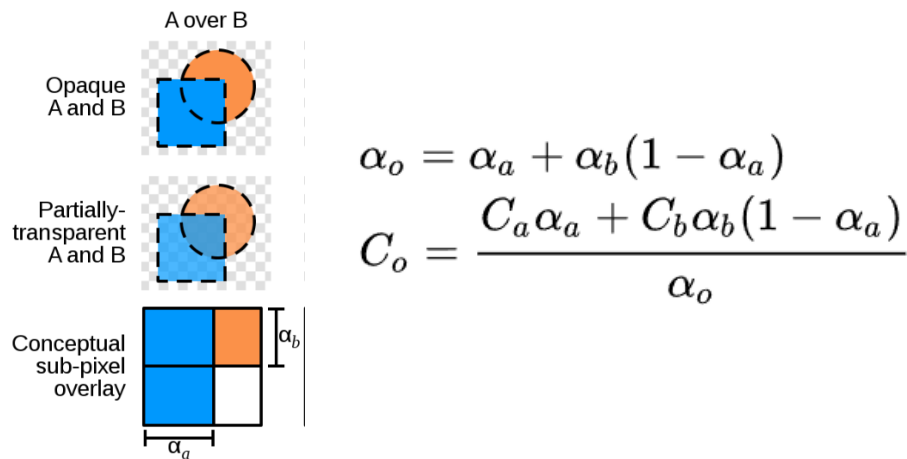


**Figure 2.** Example of Canvas and Layer bias example

Canvas is the visible region of the Scene. Only the section of images that lies in this region is saved. Figure 2 shows how the bias of Layers and Canvas works. When saving the Scene using `Scene::saveScene()` it saves the Canvas represented by the hatched region (from 0,0 to 2199,1499). Canvas is defined as a rectangle that has (0,0) and (width-1, height-1) as vertices. As mentioned in Problem 2.1 assume all width and height values of Scene or Layer are positive integers.

Bias of each Layer defines the starting coordinate of each Layer as shown in Figure 2. Only the image of a Layer that overlaps with the Canvas is accumulated to construct the saved image.

When merging Layers inside the Canvas it should be done according to their relative orders. (Remember 0 is the backmost image). Skip non-visible Layers (i.e. `Layer.visible = false`). Alpha is a transparency value, 255 means it is opaque and 0 means it is completely transparent. RGBA value of the saved image can be calculated by using the equation denoted at Figure 3.



**Figure 3.** Pixel value computation for A over B using alpha.

For more information please view [https://en.wikipedia.org/wiki/Alpha\\_compositing](https://en.wikipedia.org/wiki/Alpha_compositing).

Beware that in the equation in Figure 3, alpha values are float between 0 and 1 whereas in our alpha value of our image data is integer between 0 - 255. So, when using this equation, you should convert alpha values of our image data to 0-1 by dividing it with 255. C denotes value for each color channel, which in our case is RGB.

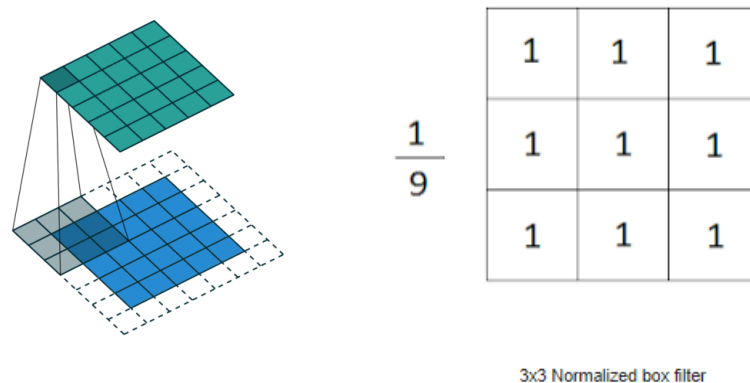
### Descriptions of Functions to Implement:

Table below are specifications of the function that you have to implement.

int saveScene(string filename)	<p><b>Description</b> Save an image of the Scene at given path</p> <p><b>Inputs variables</b> filename: filepath to saved image</p> <p><b>Return value</b> Return 0 if success else return 1 (invalid path)</p> <p><b>Detailed Specification</b> Saved Image size is (Scene.Width, Scene.Height). If none of the Layers overlap with the saved image, the saved image should be (0,0,0,0) for all pixels. This goes the same when there is no Layer at all. Use stbi_write_png() in "stb_image_write.h".</p>
--------------------------------	--

## Problem 2-3: Layer Manipulation Functions I [1 Marks]

**Objective:** Now we will manipulate image data stored in each Layers using a manipulation function. Implement Square Blur, LevelChanger, per-Channel control. (LayerManipulator.cpp)



**Figure 4.** Convolution and box filter example

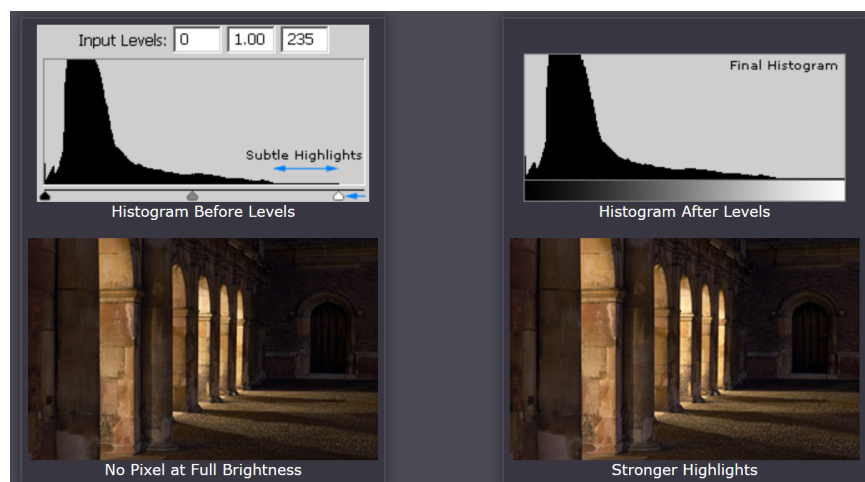
### Descriptions:

**BoxBlur** does a convolution with the image using a box filter which is the same as averaging all pixel values inside the kernel. Left image of Figure 4 is an example of convolution. The blue square is input data, green squares denote the output values, and gray square denotes the kernel. Here the kernel size is 3x3. Right Image is an example of a 3x3 boxblur filter (kernel).

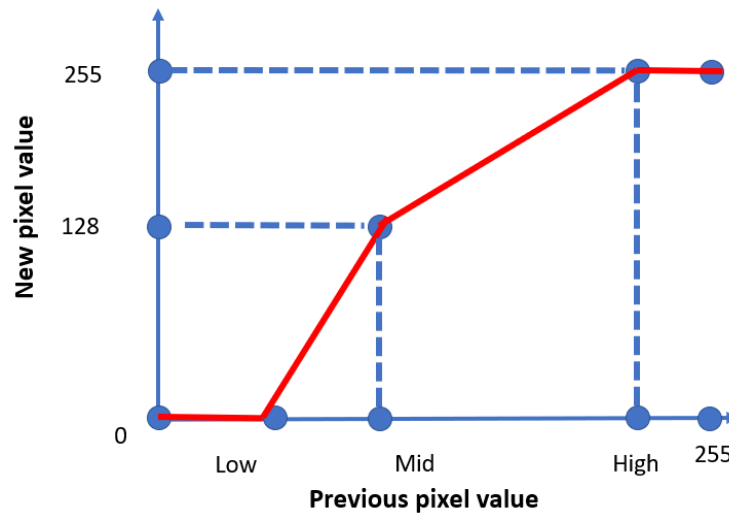
**LevelChanger** changes the pixel value based on designate the low, mid, high values. Figure 5 below demonstrates the effect of levelchanger. By lowering high, the bright region of an image gets brighter. Lowering mid values will make the overall image brighter.

If a pixel value is lower than low it clamps it to 0. If the pixel value is larger than high it clamps it to 255. A pixel with channel value equal to mid will become 128. Other values in between low-mid and mid-high can be calculated using linear interpolation. (Figure 6)

**ChannelScaling** scales the channel value of pixels by some floating value.



**Figure 5.** Effect of changing Level



**Figure 6.** Function that describes the effect of LevelChanger

**Descriptions of Functions to Implement:**

Table below are specifications of the function that you have to implement.

<pre>int BoxBlur(Layer* layer, int kernelSize, int rep)</pre>	<p><b>Description</b> Apply BoxBlur to the image of given layer</p> <p><b>Inputs variables</b> layer: pointer to the Layer that Boxblur will be applied kernelSize: box kernel size ( kernel is square =&gt; kernelSize x kernelSize ) <b>assume the kernel size is always an odd number.</b> rep: how many time to perform the blur (multiple boxblur can approximate gaussian blur)</p> <p><b>Return value</b> Return 0 if success else return 1 (layer == NULL, kernelsize&lt;1, rep &lt;0)</p> <p><b>Detailed Specification</b> When blurring near the edges, where some parts of the kernel don't overlap with the image, only average overlapped values of image instead of averaging it for the whole kernel. For example, the blurred value at a will be (a+b+d+e)/4 instead of (a+b+d+e)/9.</p> <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td>a</td><td>b</td><td>c</td></tr><tr><td></td><td>d</td><td>e</td><td>f</td></tr><tr><td></td><td>g</td><td>h</td><td>i</td></tr></table>						a	b	c		d	e	f		g	h	i
	a	b	c														
	d	e	f														
	g	h	i														
<pre>int LevelChanger(Layer *layer, int channel, int low, int mid, int high)</pre>	<p><b>Description</b> Transform channel values of pixels based on given low, mid, high value</p>																

	<p><b>Inputs variables</b>  layer: pointer to the Layer that levelchanger will be applied  channel: channel to operate  low: if lower than low pixel values are clamped to 0  mid: pixel value with mid is set to 128  high: pixel value exceeding high will be clamped to 255</p> <p><b>Return value</b>  Return 0 if success else return 1 (layer == NULL, invalid low/mid/high value, invalid channel)</p> <p><b>Detailed Specification</b>  Change the pixel channel value based following figure 6. Use linear interpolation between low-mid and mid-high points. Compute using floating point for better accuracy.  Valid range of low/mid/high values: (0 &lt;= low&lt;=mid&lt;= high&lt;= 255 &amp;&amp; low&lt; high)</p>
<p>int ChannelScaling(Layer *layer, int channel, float scaleRatio)</p>	<p><b>Description</b>  Scale selected channel value of given layer by given scaleRatio</p> <p><b>Inputs variables</b>  layer: pointer to the Layer that scaling will be applied  channel: channel to scale  scaleRatio: amount to scale</p> <p><b>Return value</b>  Return 0 if success else return 1 (layer == NULL, scaleRatio&lt; 0, invalid channel)</p> <p><b>Detailed Specification</b>  If the scaled value gets larger than 255 it should be clamped to 255.  Final channel value of pixel = current channel value * scaleRatio.  Use float to multiply and round it to unsigned char. (use std::round() )</p>

## Problem 2-4: Layer Manipulation Functions II & Stamp [1.5 Marks]

**Objective:** Implement two layer manipulation functions Channel Mask, Color Matcher and a Stamp class.

### Descriptions:

**Channel Mask** masks a target Layer based on a channel of mask Layer. It changes the alpha value of a pixel in target Layer to  $(255 - \text{channel value in mask Layer})$ . If the channel value of mask Layer is 255, which means that it is fully masked, the corresponding alpha value will be 0 turning that pixel fully transparent, vice versa.



**Figure 7.** Example of Masking. Black and white colors in the Mask denote channel value of 255 and 0, respectively. As the Star shape is not masked the output is star shape with masked region transparent.

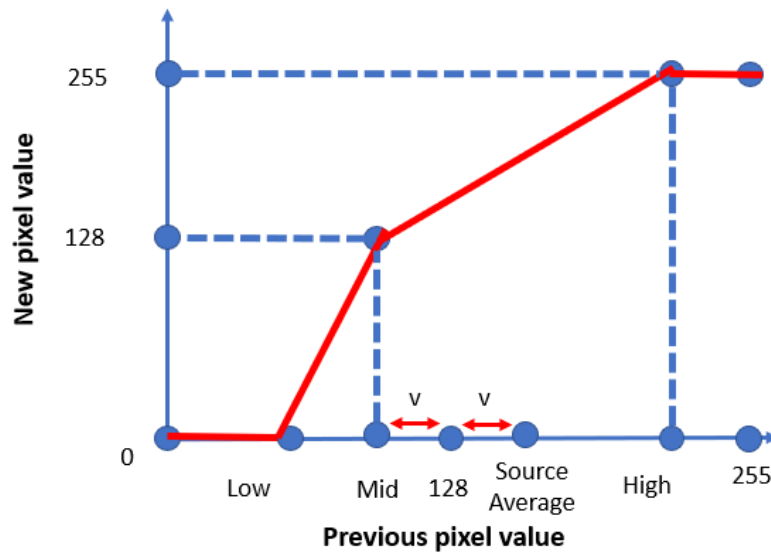
**Color Matcher** changes the color level of a target Layer to match the color distribution of the source Layer. For example, at sunset the light has a higher red value compared to normal lighting conditions. Thus, without color matching if we use an image sunset as background for an object with normal lighting the final image will look inauthentic. There are many different ways to do this but for sake of simplicity we will use an inaccurate but simple heuristic.

First, obtain min / average / max value for RGB channels of the source image. Then change the channel level of the target image using LevelChanger. For low and high values use min and max channel value of the source image.

For mid value we will calculate the absolute difference between the average of the source image and 128 which we will denote as  $v$ . If the average of source  $> 128$ , the mid value will be  $128 - v$  otherwise it will be  $128 + v$ . (mid =  $256 - \text{average of source}$ )

This is based on the intuition that if the average channel value of source image is greater than 128, the target image will need to increase the channel values.



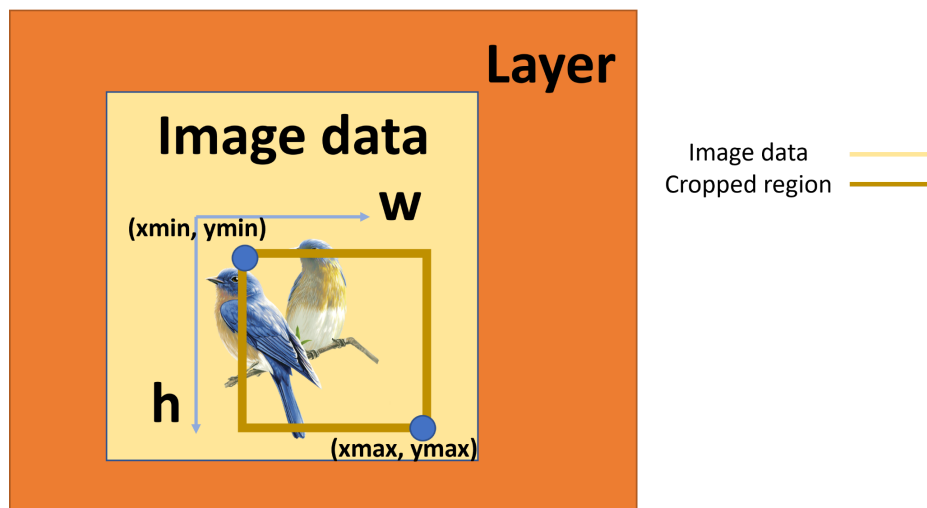


**Figure 8.** How to select Mid value for Color Matcher.

**Stamp** class is a class of tool that 1) crop image from a Layer and 2) stamp the cropped image on top of another image.

Stamp class has a single variable which is a Layer pointer and can be viewed as a wrapper class of the Layer class. Some difference is that the Layer in a Stamp instance is created from cropping another Layer.

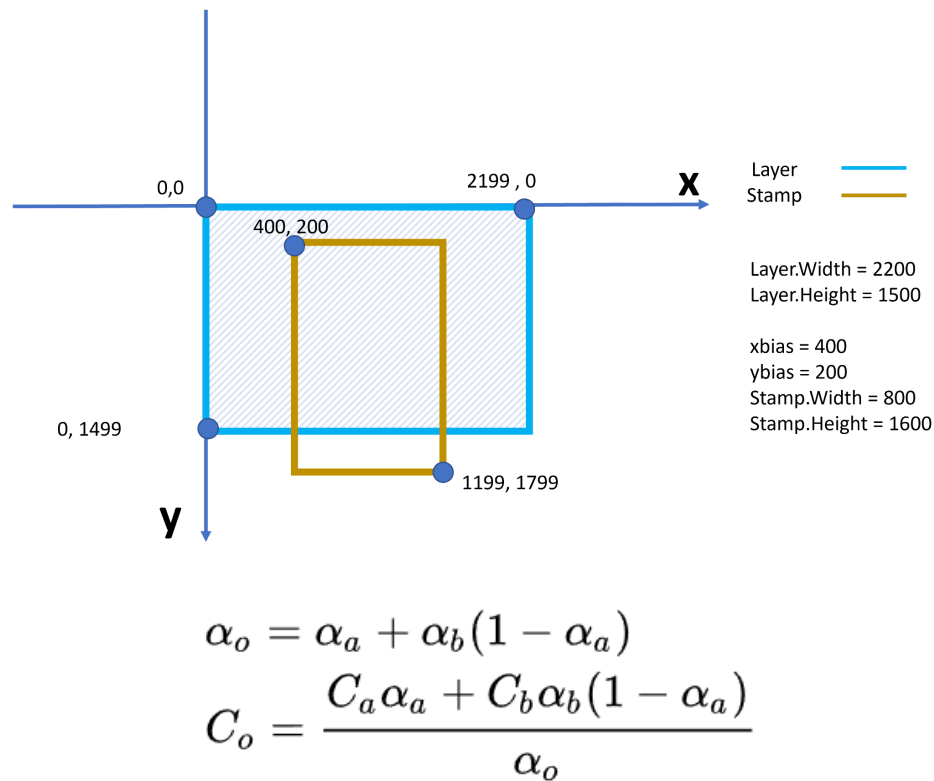
Figure 9 shows how a Stamp instance crops a Layer. It crops a rectangular region from the image data of a given Layer. The area to crop is given as two pairs x,y coordinate. (xmin, ymin) and (xmax, ymax).



**Figure 9.** Logic of `int Stamp::cropLayer()`

Figure 10 shows how a Stamp instance stamps on another Layer. It is similar to Problem 2-2. The Stamp instance gets (x,y) coordinate that marks the start of the stamp image similar to the

bias in Problem 2-2. When the stamp overlaps with the Layer image it uses the equation on the bottom of Figure 10, which is used in problem 2-2 to overwrite the image of the Layer.



**Figure 10.** Example of `int Stamp::StampOnLayer(layer, xbias, ybias)` and equation to use for calculating the pixel value.

### Descriptions of Functions to Implement:

Table below are specifications of the function that you have to implement.

<pre>int ChannelMask(Layer* mask, Layer* target, int channel);</pre>	<p><b>Description</b> Based on the selected channel value of the mask Layer, modify the alpha value of the target Layer. New alpha = (255 - channel value of mask layer)</p> <p><b>Inputs variables</b> mask: Layer to use as mask target: Layer to apply mask on channel: channel to use for masking</p> <p><b>Return value</b> Return 0 if success else return 1 (mask,target == NULL, invalid channel, invalid mask or target)</p> <p><b>Detailed Specification</b> Dimension of the target and mask image should be the same. Else it returns 1.</p>
<pre>int ColorMatcher(Layer* targetlayer, Layer* sourcelayer)</pre>	<p><b>Description</b> Change the level of the target Layer to make the color level similar to a source Layer. Obtain the min/average/max of each channel value for the source Layer.</p>

	<p>Change the level of the target Layer as described above.</p> <p><b>Inputs variables</b>  targetlayer: Layer to change the channel level  sourcelayer: Layer to obtain channel level</p> <p><b>Return value</b>  Return 0 if success else return 1 (targetlayer/sourcelayer == NULL)</p> <p><b>Detailed Specification</b>  If the mid value for the targetlayer is greater than the max it should be clamped to max. If mid value is smaller than min it should be clamped to min. Mid should be less or equal to 255. If mid &gt; 255 it should be clamped to 255. ColorMatcher is applied on RGB values.</p>
--	--

Functions to Implement in Class Stamp	
int cropLayer(Layer *layer, int xmin, int ymin, int xmax, int ymax)	<p><b>Description</b>  Get a rectangular cropped image of a given layer. 2 pairs of coordinates are given which are (xmin, ymin) and (xmax, ymax)</p> <p><b>Inputs variables</b>  layer: Layer to crop from  xmin, ymin, xmax, ymax: coordinate of Rectangle (Check Figure 10)</p> <p><b>Return value</b>  Return 0 if success else return 1 (layer == NULL, invalid coordinates)</p> <p><b>Detailed Specification</b>  Create new Layer and store the Layer* pointer at Stamp.stamplayer  Cropped region should be inside the image. You can freely name the layer created by cropLayer(). Delete previously existing Stamp.stamplayer.  (xmin &gt;= 0 , ymin &gt;=0 , xmax &lt; W, ymax &lt; H, xmin &lt;= xmax, ymin &lt;= ymax)</p>
int StampOnLayer(Layer* layer, int xbias, int ybias)	<p><b>Description</b>  Stamp on the stamp image at the image of the Layer.</p> <p><b>Inputs variables</b>  layer: Layer to stamp on  (xbias, ybias): start coordinate of stamp</p> <p><b>Return value</b>  Return 0 if success else return 1 (layer == NULL, stamplayer==NULL)</p> <p><b>Detailed Specification</b>  Stamped image is in front of the image from layer. Use equation in Problem 2-2.</p>

## Problem 2-5: Have some fun [0.5 Marks]

**Objective:** Use implemented functions and classes to generate an image of your choice. Implement it at `void yourImageGenerator(void)` at "yourimage.cpp/.h". You can import any standard libraries and local headers. Use at least 3 different types of Layer Manipulation Functions or Stamp. Also you should use at least 2 different images. Don't forget to include your source images in the `./data` directory. The images should be at `./data/p25_X.yyy` where `x` denotes the id of that image and `yyy` some filename extension. If three images are used for this problem, the file path to each image should be `./data/p25_0.yyy`, `./data/p25_1.yyy`, `./data/p25_2.yyy`. Name of the output image should be `./data/p25_output.png`. Also make sure the **size of ./data directory is less than 40MB**. This restriction is to reduce the storage burden on TA's computer. **Do not use offensive or inappropriate images**. Just follow common sense.

## References: Standard Template Library (STL) for Assignment 6

Consider using the following data structures provided by STL. In particular, refer to the following.

1. `class template <class T> std::vector` in `<vector>` (for assignment 6-1 and 6-2)
  - a. overview: <http://www.cplusplus.com/reference/vector/vector/>
  - b. `void push_back(const T& val)` method
    - i. Add items to the end of the sequence.
    - ii. [https://www.cplusplus.com/reference/vector/vector/push\\_back/](https://www.cplusplus.com/reference/vector/vector/push_back/)
  - c. `iterator begin()` and `iterator end()` methods
    - i. You can iterate the vector with `iterator begin()` and `iterator end()` methods. See an example here:
    - ii. <http://www.cplusplus.com/reference/vector/vector/begin/>
2. `class template <class T> std::unordered_set` in `<unordered_set>` (for assignment 6-1)
  - a. overview: [http://www.cplusplus.com/reference/unordered\\_set/unordered\\_set/](http://www.cplusplus.com/reference/unordered_set/unordered_set/)
  - b. `pair<iterator, bool> insert(const_value_type& val)` method
    - i. Inserts a new element to the `unordered_set`.
    - ii. [https://www.cplusplus.com/reference/unordered\\_set/unordered\\_set/insert/](https://www.cplusplus.com/reference/unordered_set/unordered_set/insert/)