CSE 241 Programming Assignment 3

In this assignment, you are going to extend the implementation in PA2.

ppmImage class

If you didn't implement a class for a ppm image (this was a requirement of PA2), you have to first create a class for ppm data. You can use the following guideline:

`ppmImage` class will have the following methods:

- Necessary constructors.
  - A constructor which takes a `ppm` file name as argument and creates object form file.
  - A constructor which creates an object according to the given dimensions. The image data should be allocated an initialized in a way to represent a blank (white) image.
  - The default constructor
  - Your constructors should check for the validity of the image data.
- Accessor and mutator functions for private member data
- A member function in order to save `ppm` image to a file.
- A member function to read `ppm` image from a file.
- A member function which prints dimensions of the ppm image.
- A Member function which returns individual pixel information. This function will take some index value and return information about the pixel pointed by that index for each color. (it basically takes a parameter and according to that parameter, it returns red, green or blue value of that particular pixel)
- A member function which changes individual pixel values.
- You can overload these functions if you need to.
- Any other member function you want your class to have.
- Check for validity of the data if you modify the members.
- Try to use `const` parameter modifier if the parameter shouldn't be modified.
- Divide your definition in `public` and `private` sections.
- Member data:
  - Dimension information
  - Image data: you can use `std::vector`. each element can be about a pixel. You can define a struct which represents a pixel.
  - Any other member data you want your class to have

Additional Functionality of ppmImage class

Implement the following (you can use friend functions or you can implement them as member functions if possible)

- `operator +`: Adds two ppmImage objects. Adds them pixel-by-pixel. They have to be the same size otherwise it returns an empty ppmImage object. (empty objects represents an image which has 0 number of pixels.).

Color channels may reach to saturation. Don't go beyond the max value written in the image data. You can assume that both images will have the same max value.

- `operator -`: Similar to `operator +`. subtracts one image from the other. Color data cannot go below `0`.
- `operator <<`: Prints image details and image data to stdout. This is similar to saving the image to a file. Allows cascading.
- `operator ()`: Function-call operator. This operator takes three parameters. The first parameter is the index of the row number. The second parameter is the index of the column number. The third parameter is the color channel. It can either `1`, `2` or `3`. The upper left corner is `(0,0)`. This operator returns a reference to the pixel value.

Test Your Class

Write a main function and test all of the functions of your class. Create objects, read from file, write to a file. change individual pixels, read individual pixels etc...

Implement the following functions in order to test your class implementation

Although you can define and use other class internally in ppmImage class, for the following functions, you are not allowed to use those classes you defined. You can only use `ppmImage` class in these functions.

Standalone Functions

```
// returns 1 if the operation is successful. otherwise, returns 0.
// reads ppm data from file named as source_ppm_file_name. stores data in destination_object
↪  which is already created outside of the function.
int read_ppm(const string source_ppm_file_name, ppmImage& destination_object);

// returns 1 if the operation is successful. otherwise, returns 0.
// writes ppm data from source_object to the file named destination_ppm_file_name.
int write_ppm(const string destination_ppm_file_name, const ppmImage& source_object);

// returns 1 if the operation is successful. otherwise, returns 0.
// reads images from filename_image1 and filename_image2. Adds them and saves the resulting
↪  image to filename_image3
int test_addition(const string filename_image1, const string filename_image2, const string
↪  filename_image3);

// returns 1 if the operation is successful. otherwise, returns 0.
// reads images from filename_image1 and filename_image2. Subtracts filename_image2 from
↪  filename_image1 saves the resulting image to filename_image3
int test_subtraction(const string filename_image1, const string filename_image2, const string
↪  filename_image3);

// returns 1 if the operation is successful. otherwise, returns 0.
// reads images from filename_image1 and prints it to stdout
int test_print(const string filename_image1);

// implement this using the function-call operator overloaded.
// this function swaps the color values of every pixel in a given ppm image.
// this function does not create a new object but modifies the given one.
// if swap_choice is 1: swaps red and green
// if swap_choice is 2: swaps red and blue
// if swap_choice is 3: swaps green and blue
// if swap_choice is not 1, 2 or 3: no swaps (this does not mean that the operation is not
↪  successful. the function should return 1 in this case if everything is normal)
// returns 1 if the operation is successful. otherwise, returns 0.
int swap_channels(ppmImage& image_object_to_be_modified, int swap_choice);
```

```
// implement this using the function-call operator overloaded.
// creates and returns a copy of a new ppmImage object which stores only one color at each
↪  pixel. This simply takes the source and copies only one color information and stores it in
↪  a new object. The other color channels are simply going to be zeros.
//if color_choice is 1: red channel is preserved
//if color_choice is 2: green channel is preserved
//if color_choice is 3: blue channel is preserved
ppmImage single_color(const ppmImage& source, int color_choice);
```

Main Function

```
// Use this main function skeleton otherwise you will get zero

int main(int argc, char** argv)
{
 // check for number of command line arguments
 // the first argument is going to be choice number
 // the second argument is going to be a ppm_file_name1
 // the third argument is going to be ppm_file_name2  (this is optional)
 // the third argument is going to be ppm_file_name3  (this is optional)

 // if choice number is 1
 // check the existence of the optional arguments. If they are not given, exit
 int test_addition(ppm_file_name1, ppm_file_name2, ppm_file_name3);

 // if choice number is 2
 // check the existence of the optional arguments. If they are not given, exit
 int test_subtraction(ppm_file_name1, ppm_file_name2, ppm_file_name3);

 // if choice number is 3
 // read ppm_file_name1 using function read_ppm
 // swap red and blue channels
 // write the updated data to a file named "ppm_file_name2". use write_ppm function.

 // if choice number is 4
 // read ppm_file_name1 using function read_ppm
 // swap green and blue channels. use swap_channels function
 // write the updated data to a file named "ppm_file_name2". use write_ppm function.

 // if choice number is 5
 // read ppm_file_name1 using function read_ppm
 // create a new object which only contains red channel data of the file read. use single_color
 ↪  function
 // write the data of the new object to a file named "ppm_file_name2". use write_ppm function.

 // if choice number is 6
 // read ppm_file_name1 using function read_ppm
 // create a new object which only contains green channel data of the file read. use
 ↪  single_color function
 // write the data of the new object to a file named "ppm_file_name2". use write_ppm function.

 // if choice number is 7
 // read ppm_file_name1 using function read_ppm
 // create a new object which only contains blue channel data of the file read. use
 ↪  single_color function
```

```
 // write the data of the new object to a file named "ppm_file_name2". use write_ppm function.

}
```

Remarks

- Error checking is important.
- Your program should be immune to the whitespace before any user input.
- Do not submit your code without testing it with several different scenarios.
- Write comments in your code. Extensive commenting is required. Comment on every variable, constant, function and loop. (`10pts`)
- Do not use `#Define` and define macros. Instead use `constant` keyword and define constant variables. If you use macros, you will loose `5` points for each of them.
- Be very careful about the input and output format. Don't print anything extra(including spaces).