CSE 241 Programming Assignment 2

DUE

April 16, 2021, 23:55

Description

- This is an individual assignment. Please do not collaborate.
- If you think that this document does not clearly describes the assignment, ask questions before its too late.

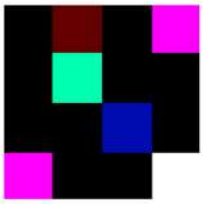
You are going to implement a PPM image library and test it.

Image File Format

Your program will basically work on PPM format(Ascii format). Particularly P3 format. Read the description here: http://netpbm.sourceforge.net/doc/ppm.html

Many image editors can read and write ppm files. I advise you to use `GIMP`. Use `Ascii` format while exporting, otherwise you cannot read it.

PPM example



```
P3
4 4
255
0   0   0     100 0   0         0   0   0     255   0 255
0   0   0       0 255 175       0   0   0       0   0   0
0   0   0       0   0   0       0  15 175       0   0   0
255 0 255     0   0   0         0   0   0     255 255 255
```

Class Definition and Requirements

You are going to define a class named `ppmImage`. In order to design your class first understand the format and decide on the member data required. For instance, you may want to keep the dimension information of the image. You need another field in order to store the data. You can use `std::vector` for data storage. You also need to decide the structure of the data. Since you are going to store `2D` data, you may want to define auxiliary classes. For instance, you can define a class which represents a pixel. If you define a class `pixel` then `ppmImage` may have a member in this form: `vector<vector<pixel> >`. Or you can just define a `1D` vector and do some index calculations in order to access particular row and columns. You can define additional structs and class if you want. You have to at least define the class `ppmImage`.

`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
  - Any other member data you want your class to have

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);

// 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);

// 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_name

 // if choice number is 1
 // read ppm_file_name using function read_ppm
 // write the same data without changing anything to a file named "o1.ppm". use write_ppm
 ↪  function.

 // if choice number is 2
 // read ppm_file_name using function read_ppm
 // swap red and green channels
 // write the updated data to a file named "o2.ppm". use write_ppm function.

 // if choice number is 3
 // read ppm_file_name using function read_ppm
 // swap red and blue channels
 // write the updated data to a file named "o3.ppm". use write_ppm function.

 // if choice number is 4
 // read ppm_file_name using function read_ppm
 // swap green and blue channels. use swap_channels function
 // write the updated data to a file named "o4.ppm". use write_ppm function.

 // if choice number is 5
 // read ppm_file_name using function read_ppm
 // create a new object which only contains red channel data of the file read. ue single_color
 ↪  function
 // write the data of the new object to a file named "o5.ppm". use write_ppm function.

 // if choice number is 6
 // read ppm_file_name using function read_ppm
 // create a new object which only contains green channel data of the file read. ue
 ↪  single_color function
 // write the data of the new object to a file named "o6.ppm". use write_ppm function.

 // if choice number is 7
 // read ppm_file_name using function read_ppm
 // create a new object which only contains blue channel data of the file read. ue single_color
 ↪  function
 // write the data of the new object to a file named "o7.ppm". use write_ppm function.

}
```