University of Pittsburgh Department of Electrical and Computer Engineering
ECE 0301: ECE Problem Solving With C++

ECE 0301: Programming Problem 9

DUE SUNDAY NOVEMBER 3, 11:59 PM
LATE SUBMISSION WITH 30% PENALTY BY TUESDAY 11:59 PM FOLLOWING THE DUE DATE

In this problem you will develop a class **Image** that encapsulates memory management for a dynamic array of image data. Follow instructions below to implement the class, write your own tests for the class methods, and submit your code via Gradescope to check your solution against the autograder tests.

1. Download the starter code from Canvas and open the folder in VSCode with File > Open Folder. Run CMake: Configure.
2. The file **bitmap.hpp** contains a header-only library that defines several structures and functions for bitmap images. **Review the provided types and functions and ensure that you understand them before continuing**.
    a. Note the use of fixed width types uint8_t, uint16_t, etc. as defined in the <cstdint> library. The "u" indicates that these are unsigned integer types, and the number 8 or 16 indicates the number of bits (hence "fixed width" meaning a fixed number of bits). For example, the RGBPIXEL struct contains three fields of type uint8_t (aliased as _BYTE), meaning that each field is an 8-bit number, to represent a 24-bit pixel.
    b. Constants of type RGBPIXEL are defined for common colors that you should use in place of hardcoded pixel values. These pixel values are defined with the three 8-bit integer values in hexadecimal (hexadecimal 0xFF is decimal 255), defining the levels of blue, green, and red in the pixel.
        i. There are many online tools that you can use to translate a color value into a 24-bit RGB triplet, such as: https://www.rapidtables.com/web/color/RGB_Color.html
    c. Several operators are defined for the **RGBPIXEL** struct – for example, you can use "==" to compare variables of type RGBPIXEL.
    d. The __attribute__((__packed__)) used in several of the structs ensures that these values are spaced correctly in memory for writing bitmap image files (necessary if you attempt the bonus for this assignment).
3. The files Image.hpp and Image.cpp contain some provided methods and some empty method definitions. **Review the provided code in this class before continuing.**
    a. The private member image_data is a pointer to an array of RGBPIXEL values, representing the image data.

    b. Recall that private members are not accessible in test cases or in non-member functions, so they can only be accessed via provided set or get methods.
4. Implement the rest of the Image class methods according to the specifications below. All method declarations are provided in the Image.hpp file. You are to implement the method definitions marked "TODO" in the Image.cpp file. Add tests for your implementation in test_image.cpp.
    a. A destructor that deallocates the image data.
        i. Recall that calling delete on a nullptr is valid.
    b. A constructor that takes one integer value, and allocates an array to hold data for a square image of the specified size in pixels. All pixel values should be set to green.
        i. RGBPIXEL constants such as GREEN are defined in bitmap.hpp for common colors.
    c. A constructor that takes two integer values - width and height - and allocates an array to hold data for a rectangular image of the specified dimensions in pixels. All pixel values should be set to green.
    d. A copy constructor that constructs a new Image instance with the same size and pixel values as the Image argument passed to the copy constructor.
    e. A copy assignment operator, implemented using the **copy swap idiom.**
    f. An **overloaded addition operator (+)**, that takes one Image argument and one RGBPIXEL argument, and adds the RGBPIXEL value to each of the Image pixels.
        i. The addition should be implemented as a **non-member function.**
        ii. Recall that in bitmap.hpp, operators are defined for adding RGBPIXELs, so you should utilize these in your implementation.
        iii. Addition should work in either direction, that is, either RGBPIXEL+image or image+RGBPIXEL, so two overloads are defined.
        iv. In addition to the book and lecture examples, the following link is a good summary of operator overloading best practices and syntax: https://stackoverflow.com/questions/4421706/what-are-the-basic-rules-and-idioms-for-operator-overloading
5. Bonus (3 pts): Implement functions to read a bitmap image from a file, and write a bitmap image to a file, and use them to "decode" an image.
    a. Implement the public load_from_file member function in Image that takes a string argument that specifies the name of a bitmap file to use as input (for example, "input.bmp"). The image data read from the file should replace any current image data (be sure to deallocate and reallocate an array of the correct size). **Utilize the import_bmp function defined in bitmap.hpp.**
        i. The small bitmap image "smiley.bmp" is provided in the starter code for testing purposes.

b. Implement the public save_to_file member function in Image that takes a string argument that specifies an output file name, and writes the current image data to the file in bitmap format. **Utilize the export_bmp function defined in bitmap.hpp.**
c. Use Image class methods to load in the provided "bonus_test.bmp" image, remove the red pixel noise, and output the resulting image to a file. You may do this in a test case or in a main function in a separate file, but using a test case will not require changes to the CMakeLists.txt.
    i. The noise must be **completely removed** from the image to qualify for bonus.
    ii. You may use your overloaded addition operator, or you may implement and use subtraction operators (as declared in Image.hpp)
d. Demonstrate to the instructor or TA your implementation of reading the "bonus_test.bmp" image, modifying the image data, and writing the output image for the bonus checkoff.

Files to submit:

Submit the following files for this assignment:

1. Image.hpp
2. Image.cpp
3. test_image.cpp

You again have the option to generate a zip file for submission with the following steps:

1. Open your project folder in VSCode
2. Set the build target to "submission": Open the command palette, search for and run CMake: Set Build Target, and select "submission"
3. Run CMake: Build
4. A zip file should be created in your build/ folder. Upload this file to Gradescope
5. Ensure that you set the built target back to your executable if you continue testing
6. If you make changes to any files, you will need to build the submission target again to generate a new submission zip.