Assignment 1 – Rotating an Image File

CPSC 8700 Fall 2022

Introduction

In this assignment you will use the following skills:

1. Use of classes and classes containing objects as members.
2. Use of a Creational Design Pattern *Singleton*.
3. Use of command line arguments and file I/O streams.
4. Use of zip/compression utilities and makefiles

Description

This project is done in C++ in *teams of 2 (and no more than 2), or individually*. You are NOT allowed to discuss your solution or share your files with other teams/students, except your own partner (if you have one). Only one person from each team should submit the files, and all files (except makefile) should have author/s names in the comments on top of the file.

You will work with .ppm and .pgm image files in P2 and P3 ASCII format and P5 and P6 binary formats. You cam use the provided image files, or you can create your own image files using Gimp image editor for Linux. Gimp is a free open source software that can be downloaded free and is available for all platforms (PC, Mac, and Linux). It is very powerful, supports many file extensions, and can convert between different image formats. We will be using Gimp while grading to verify that your file is rotated correctly. Your program will read data from a .ppm (or .pgm) file and produce a rotated copy of the original. It will be rotated 90, 180, or 270 degrees to the right or to the left, as specified by the user on the command line. You will have to devise a clever way to achieve this.

Implementation Requirements

1. Do online research and learn about .ppm and .pgm files and their structure. Please keep in mind that pixel values in files with magic numbers P2 and P3 are in ASCII format, and in files with P5 and P6 in byte format (also called raw, or binary). To print to a file in binary format you need to research the ostream options, specifically ios:: binary.

2. You will create classes ColorPixel (*colorPixel.h*) that will contain the red, green, and blue values of each color pixel, and class GrayPixel (*grayPixel.h*) to contain values of the gray pixel. Again, provide default and parameterized constructors, destructors, and getter methods for each class. If you have a parameterized constructor, you will not need setter methods to save the pixel values, as the only time you will need to set values of the pixel is when you are creating an instance of that pixel. This will be a very small class with most, if not all functions inlined. Files in binary format may need to be treated differently, and it is up to you to devise a way to handle them.

3. You will create a class Ppm and a class Pgm. Each class will have a header file (named *ppm.h* and *pgm.h*) and an implementation file (*ppm.cpp* and *pgm.cpp*). You will need member variables for storing magic number, height, width, max_value for each pixel, (and other members, if needed), and a vector of ColorPixel or GrayPixel objects, correspondingly. Please remember that you have to handle both ASCII and binary formats of .ppm and .pgm files, so you will need to create the necessary methods for that. It is up to you to figure out the best way to do it.

4. Your driver will be a singleton class. Singleton class is a design pattern that allows only one instance of that class to be instantiated. (please carefully study the code examples on Canvas to understand how they work). This class will be called Rotator (and will live in *rotator.cpp*), since it rotates image files. It will contain a static public pointer to an instance of class Rotator, and will have private constructor. You will use a static method Instance() to create an instance of a Rotator class, and you will access all the class members/methods via that static public pointer. Your driver will take several command line parameters – the original image file, the output image file, direction to rotate and degrees. For example: the command below will rotate the original file *chips.ppm* 90 degrees to the right and save the result in the file chips2.ppm.

   $$ rotator chips.ppm chips2.ppm -r 90

5. Create a simple makefile and provide the functionality to remove object files (*clean*).

6. Create a tarball containing all the source code and the makefile, but *not* the object code. You can use the command line *tar* utility for that. Please run *clean* to remove object fuiles before creating a tarball. Your tarball should contain no subdirectories. Do not submit image files. Grader/s will use their own image files to test your program.

7. Submit to canvas before the deadline listed in Canvas.

NOTE: If you submit the archive that cannot be open, or is corrupt, you will not be given the opportunity to redo the work and resubmit. You have one shot to get it right.

How to create your own .ppm and .pgm files in GIMP (so you can use them to test your program)

Two *.ppm* and two *.pgm* files (ASCII and raw format) are provided for you. They can be found in Canvas: chips.ppm, chips.pgm, potatochips.ppm, potatochips.pgm. BUT if you would like to test your program with some other files, you can convert your favorite image to *.ppm* or *.pgm* format in GIMP. Open you image in GIMP, then click on Files-> Export As, give it a name, then on the bottom click on Select File Type (by Extension), scroll down, select .ppm (or .pgm). After that it will bring up a box with options to save in ASCII or raw format. Select format and save it.

Part III. Where to Get Help

1. First of all, you can use textbooks, online sources and search engine of your choice to look at code examples, programming concepts explanation, etc. You need to be able to use all available resources to help yourself.

2. You can ask your teacher of TA any assignment related questions. Though we will answer your questions, we will not help you to debug your code, it is your job.

3. Many questions can be answered in e-mail. Please do not e-mail code unless you were asked to do so. Do this ahead of time, not at the last moment. You are still responsible for submitting your assignment on time, whether you got your question answered or not at the last moment.

   Most importantly: make a plan A and a plan B. You should always have a plan A and a plan B!! Unexpected circumstances happen, and it is better to be safe, than sorry. Start working right away, DO NOT WAIT TILL THE LAST MOMENT. Make frequent backups!!!!