CS 201 Homework 7

This assignment is focused on *Classes*, *Graphics,* and *File Streams*.

Note: If you encounter difficulties, please discuss on Discord #cs201.

Grading

- (Pass/Fail) Programs compile and run
- (Pass/Fail) GitHub Project
- (Pass/Fail) Overleaf Report written in LaTeX
- (Pass/Fail) Report submitted on Blackboard as a PDF (lastname.firstname-hwN.pdf)
- (Pass/Fail) Report contains link to GitHub repository
- (50 pts) Report contains *design* paragraph (~100 words)
- (50 pts) Report contains *post mortem* paragraph (~100 words)
- (50 pts) Report questions
- (50 pts) Report contains *sample run*
- (50 pts) Source Code has *Makefile* or Visual Studio Solution *.sln*
- (50 pts) Source Code is neat and documented
    - Name, Class, Date, and Description at top
    - All functions and classes have documentation (besides `main()`)
- (100 pts) 2 of the *additional programs* are completed
- (100 pts) GitHub frequent commits (ensure you use the program name in your text)

General Requirements

Programs that do not compile will not be graded; there is no point in turning in code that does not compile. For full credit, turn in only what was requested—no project files, executables, etc. As always:

- Code should compile, execute and work correctly.
- It should be clear to the user what the program is doing and what the user is expected to do.
- The source code should begin with comments indicating the source file's filename, author, date, and purpose.
- The source code should be neat and readable.
- Functions should be commented with their purpose.
- Other comments should be used anywhere things might not be completely clear.

Questions

1. What is a *file*? and What does it mean to *open* a file?
2. What should a program *always* do immediately after attempting to open a file?
3. When a file has been opened, it will eventually need to be closed. But when we use C++ file streams, we mostly do not need to worry about closing files. Why not?

4. Why is error checking more important when doing input than it is when doing output?
5. When we do input, we may receive valid data, or we may not. In this context, what is *valid* data?
6. Your instructor says that code should only check for end-of-file when it has already determined that there was an error on the stream. Why is this?
7. What is a *pointer*? And in the context of pointers, what is an *address*? What is a *null pointer* and why do null pointers exist?
8. How do we declare a pointer in C++? (For example, write a declaration of a variable of type pointer-to-**`double`**.) How would we *dereference* this pointer?
9. Given that the **`vector`** class is not built into C++ but *Arrays* are, why do we prefer **`vector`**?
10. If we have a pointer to an array item, how do we move the pointer to a different array item that is some given *distance* from the first? What do we mean by the *distance* between two pointers? And can the distance be calculated if they are different pointer types?

Main Program (asciiart.cpp)

Write a C++ class library to save and load Portable Pixmap Format (PPM) images. The Portable Pixmap Format (PPM) is a very easy to use image format that can be opened natively on many operating systems or with IrfanView or GIMP on Microsoft Windows. The file has the following scheme

```
P3
3 2
255
# The part above is the header
# "P3" means this is a RGB color image in ASCII
# "3 2" is the width and height of the image in pixels
# "255" is the maximum value for each color
# A comment is a line starting with a '#' symbol
# The part below is image data: RGB triplets
255   0   0
  0 255   0
  0   0 255
255 255   0
255 255 255
  0   0   0
```

A 2D image of dimensions **(w, h)** can be represented by a single vector and addressed with a set of coordinates **(x, y)** by multiplying the y coordinate by the width and adding x as follows:

Pixel address = y * w + x

Modify Color3.cpp and Image3.cpp and complete these Requirements:

- Finish implementing the C++ class **Color3** inside of Color3.hpp and Color3.cpp which represents an RGB color using three members **r**, **g**, and **b** which may take values of 0 to 255.

- Finish implementing the C++ class **Image3** inside of Image3.hpp and Image3.cpp which represents a 2D image.
- Write a Program that reads a PPM image and outputs an ASCII Art version of that image. Use the **Color3** method **weightedSum()** to map RGB values to luminance. Then map the luminance (which is in the range 0 to 255) to at least 16 ASCII characters ranging in brightness where SPACE ' ' might represent WHITE and 'Q' or 'W' might represent BLACK.
- Test your program with Parrot.ppm and other images. Include sample output in your project report.

You may find the web page https://en.wikipedia.org/wiki/Netpbm_format useful.

Example Output:

Program (mandelbrot.cpp)

The Mandelbrot Set is a famous fractal named after Benoit Mandelbrot. The set contains values for which the following condition remains **bounded**:

$$z_{n+1} = z_n^2 + c$$

For example, if we start with $z_0 = 0$ and we set the complex number $c = 1$, then the values produced are 0, 1, 2, 5, 26, ... which is unbounded, so it is not in the set. If we set $c = -1$, then the values produced are 0, -1, 0, -1, 0, ... which is bounded and therefore -1 belongs to the Mandelbrot set. It is important to note that the number c is a complex number and has a real and imaginary component. So we can draw a picture of this set by setting the Imaginary value of c or $Im[C]$ to the y axis and the Real value of c or $Re[c]$ to the x axis. The calculation for a point (x, y) is then

$$x = Re(z^2 + c) = x^2 - y^2 + x_0$$
$$y = Im(z^2 + c) = 2xy + y_0$$

The following Pseudocode from Wikipedia gives an overview of the process:

```
For each pixel (Px, Py) on the screen, do:
{
  x0 = scaled x coordinate of pixel (scaled to lie in the Mandelbrot X scale (-2.5, 1))
  y0 = scaled y coordinate of pixel (scaled to lie in the Mandelbrot Y scale (-1, 1))
  x = 0.0
  y = 0.0
  iteration = 0
  max_iteration = 1000
  while (x*x + y*y <= 2*2  AND  iteration < max_iteration) {
    xtemp = x*x - y*y + x0
    y = 2*x*y + y0
    x = xtemp
    iteration = iteration + 1
  }
  color = palette[iteration]
  plot(Px, Py, color)
}
```

Write a program which uses the **Image3** and **Color3** classes above to write out the Mandelbrot set. Be creative and try and color the set. Include sample output in your homework report.

You may find the web page https://en.wikipedia.org/wiki/Mandelbrot_set useful.

Program (rule30.cpp)

Write a program to evaluate Rule 30, which is a cellular automaton discovered by Stephen Wolfram. You will want to use a **vector<int>** to implement this program. Use the **int argc** and **char** argv** arguments in your **main()** function to take two optional integers to specify the number of columns and rows to process. The default should be 40 columns and 20 rows. This kind of cellular automata is initialized by setting the middle cell to **1** and the other cells to **0**. Then you will evaluate every cell according to these rules and formula:

Current	111	110	101	100	011	010	001	000
New	0	0	0	1	1	1	1	0

It is called rule 30 because if you convert binary 00011110 to decimal, you get 30. You should read from cells **i-1**, **i**, and **i+1**. Let's call them left, center, and right. Assume that a cell out of range is a 0. Apply the following formula

New = Left XOR (Center OR Right)

After each iteration, print out the vector to a file called rule30.txt. Use the Image3 class to generate an image of at least resolution 1080x1920. Include sample output in your homework report.

Please see https://en.wikipedia.org/wiki/Rule_30 for more information.

Program (caesar-cypher.cpp)

A **Caesar Cypher** is a simple encryption/decryption method that shifts letters by some given number of steps. For example, a shift of 1 takes A to B, B to C, and so on. It takes Y to Z, while Z rotates around to A. A shift of 2 takes A to C, B to D, and so on. It takes W to Y, X to Z, Y to A, and Z to B. Only letters are modified. Upper-case letters stay upper-case, and lower-case letters stay lower-case. So, a Caesar cypher with a shift of 3, applied to the string, "Hello, everyone!" would become "Khoor, hyubrqh!".

Write a C++ program that repeatedly inputs a line of text and an integer to be used as a shift. It applies the Caesar cypher with that shift to the text and outputs the result. There should be some way the user can end the program. Proper error handling should be done. If the user is asked to enter a number, and something else is typed, then the program should notify the user and retry the input.

A run of the program might look like this. Boldface text is the user input.

```
Caesar Cypher

Enter a message to cypher (blank line to end): Hello, everyone!
Enter an integer to use as the shift: 3

Result: Khoor, hyhubrqh!

Enter a message to cypher (blank line to end): Khoor, hyhubrqh!
Enter an integer to use as the shift: -3

Result: Hello, everyone!

Enter a message to cypher (blank line to end):

Program Complete
```