CS 201 Homework 7

Solomon Himelbloom

November 30, 2020

- Repository Link: `https://github.com/techsolomon/cs201`
- Git Commits: `https://github.com/techsolomon/cs201/commits`
- This homework took approximately 8 hours to complete.

1 Design

This eighth homework assignment was designed with future reference and refactoring in mind. I also followed the directions of including the file name in the commit message when code was added or deleted with functions broken down into much smaller parts than before. This made it simple to add references to classes and other important functions across multiple files.

The hw7 folder contains the main program (main – ASCII art), two additional programs (Caesar cypher and rule30), and a practice folder (for extra credit, such as Mandelbrot). Header files were created to allow for linking multiple files and removing repetition for defining each function. Once again, each .cpp file starts with the author's name, the computer science class code, the date file was created or last edited, and finally, a short description with an included Makefile.

2 Post Mortem

During this homework assignment, I learned more about classes and the fundamentals of extrapolating information from files. For style and design, I enjoyed using detailed comments and descriptive functions to better organize and reference programs on the next assignments.

Some changes that I hope to add for the next homework assignment include reviewing textbooks, prior homework assignments, and lab submissions before diving into the larger programs. In conclusion, I also hope to ask for assistance when clarifications are needed, in addition to referencing online C++ guides, before submitting my final draft of the project.

3 Answers to Questions

1. A file, which is commonly referred to as a document, is simply a sequence of characters. Two basic kinds of files include text and binary. To open a file is to create a connection with the aforementioned file.

2. Immediately upon attempting to open a file, we should always check for errors. Also, such C++ code to attempt a file opening should prevent further use of the file if an error is detected. Where appropriate, we should also be checking for valid data – information that is usable and meaningful to the end-user or computer program.

3. To begin, the C++ Standard Library (STL) declares stream types that do input and output work on a file known as file streams. Since a file stream automatically considers a "close" when it goes away, we do not need to worry about closing files.

4. Although error checking is crucially important in all aspects of computer programming, it is interestingly more important to be completed while doing input over, for example,

output. This is because when our programs complete input, data transfer is coming from the outside world (i.e. a source that we should not be immediately trusting). Because of this important realization, error handling and the verification of values are considered much more important for input rather than output.

5. In this context, valid data means meaningful and useable data. It is worthwhile mentioning that specific types of values that are valid can change depending on the specifics of the application or computer program.

6. The code that we write should only check for the end-of-file (EOF) when we have already determined that there is an error on the stream because of how the EOF items are processed. When checking for EOF when a stream error has not occurred can, unfortunately, result in the last item in a file being skipped thus not being accounted for in a designated final value or total.

7. Put simply, a pointer is a piece of data that expresses where in memory another piece of data is being stored. Since pointers are only as good as what they point to, the pointer value returned with an ampersand () operator is called the address of the variable.

   Null pointers originate when we declare a pointer without initializing it. One common guideline is to, "Never dereference a pointer that does not point to anything." Null pointers (a special value that does not point to anything) exist to check whether a pointer points to anything, where in some cases it can point to nothing. Finally, in our programs, we can represent null pointers as nullptr.

8. We can declare a pointer in C++, or write a declaration of a variable of type pointer-to-double, an asterisk (*) is placed after the pointed-at-type. For example, the phrase "double * dp;" essentially means the same as a pointer to double. Similar to how we receive the value of a pointer reference with an iterator, we can deference the pointer with the placement

of an indirection operator (*) to the left of what we are planning on dereferencing.

9. While the vector class is not built into C++ analogous to arrays, there are numerous reasons why we as programmers might prefer vectors over other storage devices, such as arrays. One main example is that the sizes of vectors are resizable, whereas arrays are fixed. This means that if our application were to be allocated more memory, arrays would be considered to be hard-coded into our program meaning that it is fixed and unable to be easily dynamically scaled at run-time.

10. In this scenario, we can use pointer arithmetic to move the pointer around in the array thus making it easier to point to other array items. If two pointers point to numbers in the same array, we can subtract one from the other to get the elapsed distance between the two pointers. The distance between two pointers can be considered the amount of space traveled during each iteration of the array or vector item. It is worthwhile mentioning that although this trick works for other kinds of iterators (such as vectors) we can only subtract iterators and pointers that reference items in the same containers.

4 Program 1

4.1 Sample Output/Screenshot

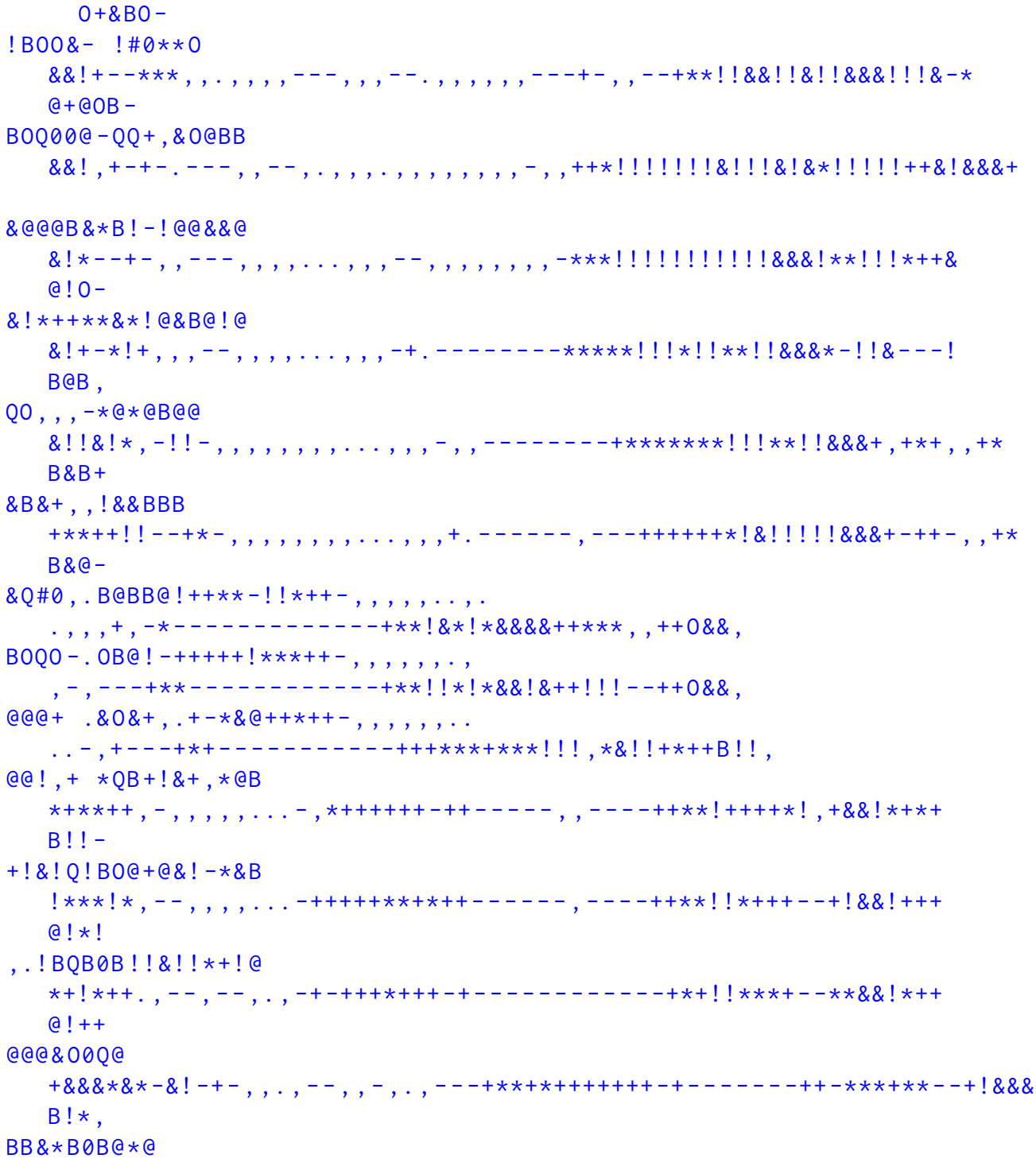
Listing 1: Sample Program Output

```
Here are some images:

Image #1: [parrot.ppm image]

Opened parrot.ppm
Found PPM (text).
Currently ignoring the PPM file comment.
```

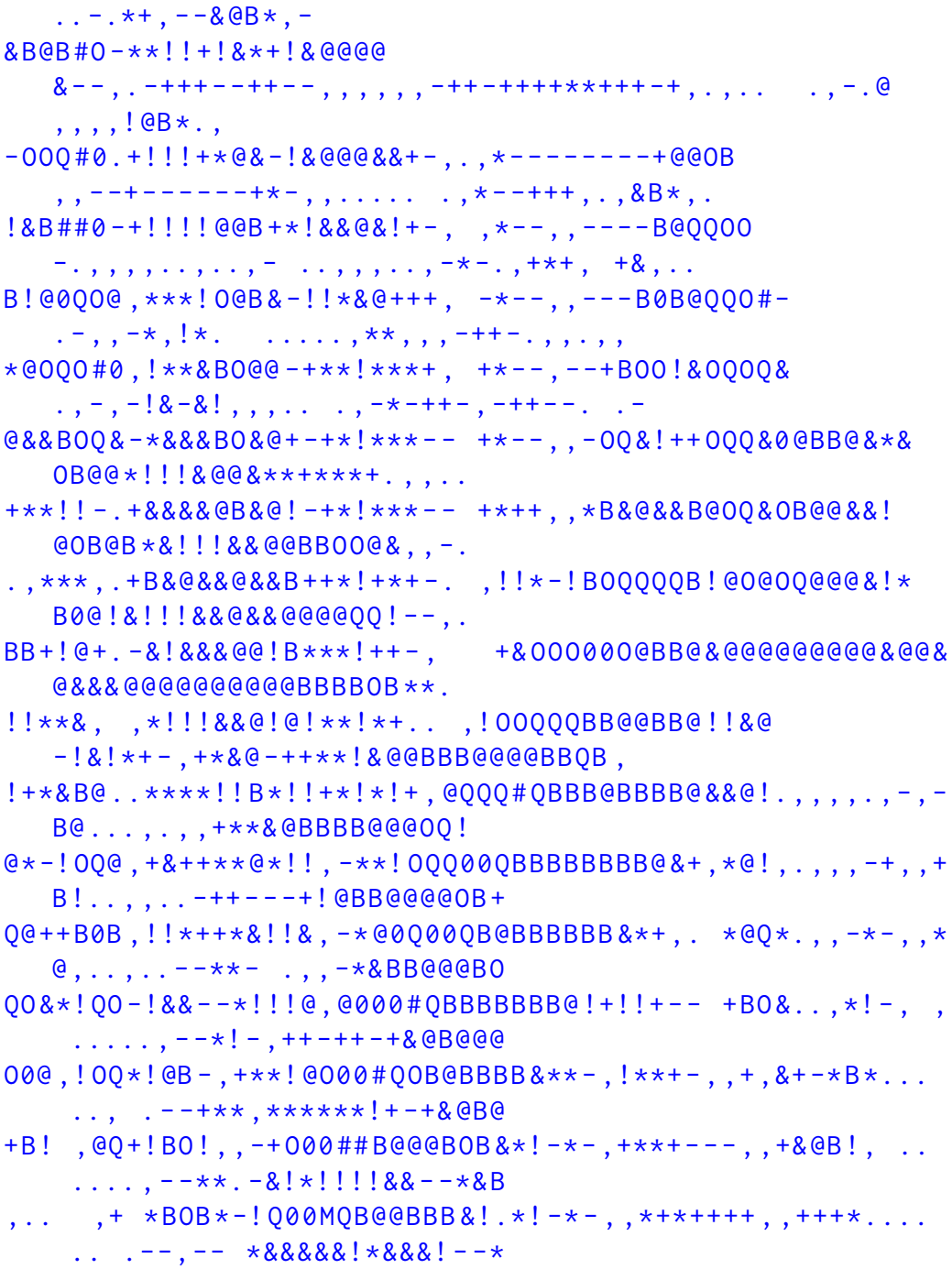

Image Size: 80x80
Maximum Value: 255



```
   &*+&&**!*-,,,,..,,,,-,.,---+!-*+++++++++++---,--++--+++++++---!&&&**-
   @*!*
B@&*&0B!&@
   **!****&!++,,,....,,..----++++++++++++++++++--------,+---+++-,++
   @&&!!*&*&!
&--**QB*@
   !!+!!*&&&&+*,.........,----++++++++++++++++++--------++++-+-----+++!&&

-    .0@!B@@+!!*@@@&+*+...
   .....,---+++++++++!++-+++-++------++------+-----*!!&****

.     Q&+B@**!!*!@@!**!,..
   .,--++++++++++++++++--------+---+-+++-,-+*!!&&*!*
++.  .O!*&&&!@@!*!@&!+*-..
   ,,--++++++++!**********---+------++-+++,--+*!!!&!&B
&@+..B!&@&&+B@!!+!&@***+,.
   ,---++++++++!*++++--++-+------,,-+++,+++,.-.***!&!BO
@@*,.B*!!++!BB@!+!@@&**..
   .,+-+++++++++!*+-+----+----,-,-+++-.-+-....,+!&!-&O
&B++-@*!!@*&&OO!+!&@@*!..
   ,,--++++!++++!!*,+-*+-+----,----+-,.+--..  ,  ,&!,,-&
BO++,&-*@&!&!OO!**&@B!&...
   ,,+,-+++!*++++!!+-++++-*----,,-----,.**-....
   .+++++-+
!B+,,&-&Q&&@&&@+!*!@B@&-..
   ++*--++++++++++++,,----+-,-,,------,.**-.,-
   .,+&!!!-
*!-.-&+&&&@&&!!*!!!&BB!*..
   .***--++++++-++**---+-+--+,,,,-,-+---  .+*-,,.+!.!@!&
   B!
**-  +&+B&&@&!&&*&&!&@B!!,.
   -!++--+++--+-+++*,--*--+------------,   --,,.+&+,&@
   &*&+
+-,  *!+B+!&@****!&&&@B
   &+-..****,-++----.-,,,-+-++++----+++-+,,.
   ..,-,-!,-!&@*+-
B***O&+@*!!&&!+*!@&&&@
   &--,.*+-+--++---,...,----++++++++++---,
```



```
Done: the pixels2read task has been completed.
```

```
PRESS ENTER to quit
```

4.2 Git Commit Messages

Date	Message
2020-12-01	draft: split asciiart into multiple files
2020-12-01	refactor: split file into read function
2020-12-01	refactor: combine read-parrot into asciiart file
2020-12-01	refactor: read-parrot.cpp toner and file name
2020-11-30	add: draft read-parrot.cpp output before refactor
2020-11-30	add: value map luminance in read-parrot.cpp
2020-11-30	add: read-parrot.cpp (v1) RGB blueprint
2020-11-30	add: read-parrot.cpp input for parrot.ppm
2020-11-30	add: LoadPPMImages class to asciiart.cpp
2020-11-29	add: asciiart.cpp file input pseudocode
2020-11-29	add: hw7 main ASCII supporting files
2020-11-28	add: hw7 (initial files)

4.3 Source Code

4.4 asciiart.cpp

```cpp
// asciiart.cpp
// Solomon Himelbloom
// 28 November 2020
// ASCII art example for CS 201.

#include <iostream>
#include <stdio.h>
#include <string>
#include <map>
#include <fstream>
using std::cin;
using std::cout;
using std::endl;
using std::ifstream;
using std::map;
using std::string;

// Sample parrot.ppm file:
// P3
// # CREATOR: GIMP PNM Filter Version 1.1
// 80 80
// 255
// 0
```

```
// 0
// 1

// Input and output from the attached Portable Pixmap Format (PPM) image.
void readPPM() {
    const string file_name = "parrot.ppm";
    ifstream fin(file_name);
    if (!fin) {
        cout << "Error opening " << file_name << endl;
        exit(1);
    }
    cout << "Opened " << file_name << endl;

    // Read and verify the magic number.
    string line;
    getline(fin, line);
    if (line[0]=='P' && line[1]=='3') {
        cout << "Found PPM (text)." << endl;
    }
    else {
        cout << "Unable to read magic number P3." << endl;
        exit(2);
    }

    // Process the comment.
    getline(fin, line);
    if (line[0]=='#') {
        cout << "Currently ignoring the PPM file comment." << endl;
    }

    // Input (and print) the x + y image size and maximum value.
    int xres, yres, maxval;
    fin >> xres >> yres >> maxval;
    if (!fin) {
        cout << "Error reading resolution." << endl;
        exit(3);
    }
    cout << "Image Size: " << xres << "x" << yres << endl;
    cout << "Maximum Value: " << maxval << endl;

    int r, g, b, y;
    int pixels2read = xres * yres;
    for (int i = 0; i < pixels2read; i++) {
        // Check status for end-of-file (EOF) errors.
        fin >> r >> g >> b;
        if (!fin) {
            cout << "Error reading the pixels." << endl;
            exit(4);
        }
        // Push to back of vector (6,400 pixels; each has an R,G,B value).
        y = 0.2126 * r + 0.7152 * g + 0.0722 * b;
        // Make sure the y-value is [0, 255].
        if (y < 0 || y > 255) {
            cout << "ERROR: the y-value is out of range." << endl;
            exit(6);
        }
        // Map the y-value to a character.
        const char values[] = " .,-+*!&@BOQ0#MW";
        int val_map = y / 16;

        cout << values[val_map];
```

```
        if (i % 80 == 79) {
            cout << endl;
        }
    }
}

// Class Load Portable Pixmap Format (PPM) Images
class LoadPPMImages {

public:
    // ***** LoadPPMImages class: constructors *****
    const string file_name = "parrot.ppm";
    // ***** LoadPPMImages: general public member functions *****

    void printPPM() const {
        cout << file_name << " image";
    }

    // ***** LoadPPMImages: data members *****
private:
    // 2D image dimensions:
    int _length;  // PPM total length (L -> Y)
    int _width; // PPM total width (W -> X)
    int _height; // PPM total height (H -> Y)

}; // End class LoadPPMImages


int main() {
    // Print header.
    cout << "Here are some images:" << endl;
    cout << endl;

    // Make an image object and print it.
    int i = 1;
    LoadPPMImages i1;
    cout << "Image #" << i << ": [";
    i1.printPPM();
    cout << "]" << endl;
    cout << endl;

    readPPM();
    cout << "\nDone: the pixels2read task has been completed." << endl;

    // Wait for user
    cout << "\nPRESS ENTER to quit ";
    while (cin.get() != '\n');
}
```

4.5 Colors3.cpp

```
// Color3.cpp
// Solomon Himelbloom
// Implementation for Color3 class
// Original Author: Jonathan Metzgar
// CS 201 course
#include <iomanip>
#include "Color3.hpp"

#include <iostream>
```

```cpp
#include <stdio.h>
#include <string>
#include <map>
#include <fstream>
using std::cin;
using std::cout;
using std::endl;
using std::ifstream;
using std::map;
using std::string;
using std::setw;

// Ensure values are in the range 0 to maxvalue
constexpr int saturate(int x, int maxvalue) {
  return x < 0 ? 0 : x > maxvalue ? maxvalue : x;
}

Color3::Color3()
  : r(0), g(0), b(0)
{ }

Color3::Color3(int R, int G, int B) {
  r = (unsigned char)saturate(R, 255);
  g = (unsigned char)saturate(G, 255);
  b = (unsigned char)saturate(B, 255);
}

int Color3::weightedSum() const {
  int r, g, b, y;
  // Implement Y = 0.2126R + 0.7152G + 0.0722B
  y = 0.2126 * r + 0.7152 * g + 0.0722 * b;
  // Ensure values are inside the range 0 to 255
  if (y < 0 || y > 255) {
            cout << "ERROR: the y-value is out of range." << endl;
            exit(6);
        }
  return 0;
}

char Color3::asciiValue() const {
  // Use at least 16 characters, sort these from dark to light
  // or light to dark and then map the weightedSum() to the range
  // 0 to 15. Please pick your own characters
  const char values[] = " .,-+*!&@BOQ0#MW";
  unsigned darkness = 0;
  return values[darkness];
}

// Stream Operators for input and output

std::ostream& operator<<(std::ostream& ostr, const Color3& color) {
  ostr << setw(3) << (int)color.r << " ";
  ostr << setw(3) << (int)color.g << " ";
  ostr << setw(3) << (int)color.b << " ";
  return ostr;
}

std::istream& operator>>(std::istream& istr, Color3& color) {
  // Implement your own input for a Color3
  return istr;
}
```

4.6 Color3.hpp

```cpp
// Color3.hpp
// Interface for Color3 class
// Original Author: Jonathan Metzgar
// CS 201 course
#ifndef COLOR3_HPP
#define COLOR3_HPP

#include <iostream>

class Color3
{
public:
  Color3();
  Color3(int R, int G, int B);

  // Returns 0.2126R + 0.7152G + 0.0722B
  int weightedSum() const;

  // Returns an ASCII char representing darkness
  // e.g. ' ' = WHITE and 'W' = BLACK
  char asciiValue() const;

  unsigned char r;
  unsigned char g;
  unsigned char b;
};

std::ostream& operator<<(std::ostream& ostr, const Color3& color);
std::istream& operator>>(std::istream& istr, Color3& color);

#endif
// Color3.hpp
// Solomon Himelbloom
// Interface for Color3 class
// Original Author: Jonathan Metzgar
// CS 201 course
#ifndef COLOR3_HPP
#define COLOR3_HPP

#include <iostream>

class Color3
{
public:
  Color3();
  Color3(int R, int G, int B);

  // Returns 0.2126R + 0.7152G + 0.0722B
  int weightedSum() const;

  // Returns an ASCII char representing darkness
  // e.g. ' ' = WHITE and 'W' = BLACK
  char asciiValue() const;

  unsigned char r;
  unsigned char g;
  unsigned char b;
};

std::ostream& operator<<(std::ostream& ostr, const Color3& color);
std::istream& operator>>(std::istream& istr, Color3& color);

#endif
```

4.7 Image3.cpp

```cpp
// Image3.hpp
// Solomon Himelbloom
// Implementation for Image3 class
// Original Author: Jonathan Metzgar
// CS 201 course

#include <iostream>
#include <stdio.h>
#include <string>
#include <map>
#include <fstream>
using std::cin;
using std::cout;
using std::endl;
using std::ifstream;
using std::map;
using std::string;
#include "Image3.hpp"

// Image3 Constructor
Image3::Image3(unsigned width, unsigned height) {
  int w, h;
  w = width;
  h = height;
  pixels.resize(height * width);
}

// Return a pixel from the image
const Color3& Image3::getPixel(unsigned x, unsigned y) const {
  int fin, xres, yres, maxval;
    // fin >> xres >> yres >> maxval;
    if (!fin) {
        cout << "Error reading resolution." << endl;
        exit(3);
    }
    cout << "Image Size: " << xres << "x" << yres << endl;
    cout << "Maximum Value: " << maxval << endl;

  return pixels[y * w + x];
}

void Image3::setPixel(unsigned x, unsigned y, const Color3& color) {
  int r, g, b, pixels2read;
    for (int i = 0; i < pixels2read; i++) {
        // Map the y-value to a character.
        const char values[] = " .,-+*!&@BOQ0#MW";
        int val_map = y / 16;

        cout << values[val_map];
        if (i % 80 == 79) {
            cout << endl;
        }
    }
}

bool Image3::savePPM(const std::string& path) const {
  // REQUIREMENT: Use the STREAM operators for the file contents
  return false;
}

```

```cpp
bool Image3::loadPPM(const std::string& path) {
  // REQUIREMENT: Use the STREAM operators for the file contents
  return false;
}

void Image3::printASCII(std::ostream& ostr) const {
  int fin, r, g, b, xres, yres, vector;
    int pixels2read = xres * yres;
    for (int i = 0; i < pixels2read; i++) {
        // Check status for end-of-file (EOF) errors.
        if (!fin) {
            cout << "Error reading the pixels." << endl;
            exit(4);
        }
        // Push to back of vector (6,400 pixels; each has an R,G,B value).
        vector = 0.2126 * r + 0.7152 * g + 0.0722 * b;
        // Make sure the y-value is [0, 255].
        if (vector < 0 || vector > 255) {
            cout << "ERROR: the y-value is out of range." << endl;
            exit(6);
        }
    }
}

// STREAM OPERATORS for IMAGE3 class

std::ostream& operator<<(std::ostream& ostr, const Image3& image) {
  return ostr;
}

std::istream& operator>>(std::istream& istr, Image3& image) {
  return istr;
}
```

4.8 Image3.hpp

```cpp
// Image3.hpp
// Solomon Himelbloom
// Interface for Image3 class
// Original Author: Jonathan Metzgar
// CS 201 course
#ifndef IMAGE3_HPP
#define IMAGE3_HPP

#include <iostream>
#include <vector>
#include "Color3.hpp"

class Image3
{
public:
  Image3(unsigned width, unsigned height);

  const Color3& getPixel(unsigned x, unsigned y) const;
  void setPixel(unsigned x, unsigned y, const Color3& color);

  bool savePPM(const std::string& path) const;
  bool loadPPM(const std::string& path);

  void printASCII(std::ostream& ostr) const;
```

```
25
26   std::vector<Color3> pixels;
27   unsigned w;
28   unsigned h;
29   Color3 borderColor;
30 };
31
32 std::ostream& operator<<(std::ostream& ostr, const Image3& image);
33 std::istream& operator>>(std::istream& istr, Image3& image);
34
35 #endif
```

5 Program 2

5.1 Sample Output/Screenshot

Listing 2: Sample Program Output

```
Caesar Cypher

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

Enter an integer to use as the shift: 0

RESULT
Message Input = Solomon
Integer Input = 0

Program Complete
```

5.2 Git Commit Messages

Date	Message
2020-12-02	add: generic alphanumeric default cypher
2020-12-02	add: validate user input in caesar-cypher.cpp
2020-12-01	add: user input and userRequestedInput func
2020-11-28	add: hw7 (initial files)

5.3 Source Code

5.4 caesar-cypher.cpp

```
// caesar-cypher.cpp
// Solomon Himelbloom
// 28 November 2020
// Caesar cypher example for CS 201.

#include <iostream>
#include <stdio.h>
#include <string>
#include <map>
#include <vector>
#include <algorithm>
#include <fstream>
#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::map;
using std::vector;
using std::sort;
using std::reverse;
using std::ofstream;
using std::ifstream;

void userRequestedInput() {
    std::string program_line;
    std::string message_input = "";
    int integer_input = 0;

    cout << "\nEnter a message to cypher (blank line to end): ";
    getline(cin, program_line);
    cout << "" << endl;

    cout << "Enter an integer to use as the shift: ";
    cin >> integer_input;

    // Validate alphanumeric user input.
    for (int i = 0; i < program_line.size(); ++i){
        if ((program_line[i] >= 'A' && program_line[i] <= 'Z')
        || (program_line[i] >= 'a' && program_line[i] <= 'z')) {
            message_input += program_line[i];
        }
    }

    // Generic shift for character cyphers.
    const char upper_case[]
        = "ABCDEFGHIJKLMNOPQURTUVWXYZABCDEFGHIJKLMNOPQURTUVWXYZABCDEFGHIJKLMNOPQURTUVWXYZ";
  const char lower_case[]
        = "abcdefghijklmnopqurtuvwxyzabcdefghijklmnopqurtuvwxyzabcdefghijklmnopqurtuvwxyz";

    // Print the results.
    cout << "\nRESULT";
    cout << "\nMessage Input = " << message_input;
    cout << "\nInteger Input = " << integer_input;
}

int main(int argc, char *argv[]) {
    cout << "Caesar Cypher" << endl;
```

```
59      userRequestedInput();
60      cout << "\n\nProgram Complete" << endl;
61
62      // // Wait for user for end of program.
63      // while (cin.get() != '\n') {
64      //     cout << ">>";
65      //     userRequestedInput();
66      // }
67
68      return 0;
69  }
```

---

6 Program 3

6.1 Sample Output/Screenshot

Listing 3: Sample Program Output

```
Hello, rule 30.

    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0

Printing process to rule30.txt complete.
```

6.2 Git Commit Messages

Date	Message
2020-12-02	refactor: rule30.cpp binary-like file output
2020-12-02	add: program output to rule30.txt
2020-12-01	add: color and image template files
2020-12-01	add: comments for rule30 mandelbrot
2020-11-28	add: hw7 (initial files)

6.3 Source Code

6.4 rule30.cpp

```cpp
// rule30.cpp
// Solomon Himelbloom
// 28 November 2020
// Rule 30 example for CS 201.

#include "Color3.hpp"
#include "Image3.hpp"
#include <iostream>
#include <stdio.h>
#include <string>
#include <map>
#include <vector>
#include <algorithm>
#include <fstream>
#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::map;
using std::vector;
using std::sort;
using std::reverse;
using std::ofstream;
using std::ifstream;

/*
    Rule 30 Formula: New = Left XOR (Center OR Right)
    Current: 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000
    New: 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0
*/

// Print 10x10 formatted block of intergers.
void printIntegersToFile() {
    const string file_name = "rule30.txt";

    std::vector<int> square;
    ofstream output (file_name, ofstream::out);

```

```
    for (int i = 0; i < 100; i++) {
        square.push_back(0);
    }

    for (std::size_t i = 0; i < square.size(); i++) {
        output << std::setw(4) << square[i];

        if ((1 + i) % 10 == 0) {
            output << "\n";
        }
    }
}

// Check for errors on file open and print desired text.
void printTextToFile() {
    const string file_name = "rule30.txt";

    ofstream fout(file_name);
    fout << "Hello, input file." << endl;

    if (!fout) {
        cout << "Error opening file." << endl;
    }
}

int main() {
    const string file_name = "rule30.txt";

    std::cout << "Hello, rule 30." << endl;
    printIntegersToFile();
    std::cout << "\nPrinting process to " << file_name << " complete." << endl;
}
```

6.5 Colors3.cpp

```
// Color3.cpp
// Solomon Himelbloom
// Implementation for Color3 class
// Original Author: Jonathan Metzgar
// CS 201 course
#include <iomanip>
#include "Color3.hpp"

using std::setw;

// Ensure values are in the range 0 to maxvalue
constexpr int saturate(int x, int maxvalue) {
  return x < 0 ? 0 : x > maxvalue ? maxvalue : x;
}

Color3::Color3()
  : r(0), g(0), b(0)
{ }

Color3::Color3(int R, int G, int B) {
  r = (unsigned char)saturate(R, 255);
  g = (unsigned char)saturate(G, 255);
  b = (unsigned char)saturate(B, 255);
}

```

```
int Color3::weightedSum() const {
  int y;
  // Implement Y = 0.2126R + 0.7152G + 0.0722B
  y = 0.2126 * r + 0.7152 * g + 0.0722 * b;
  // Ensure values are inside the range 0 to 255
  if (y < 0 || y > 255) {
            // cout << "ERROR: the y-value is out of range." << endl;
            exit(1);
        }

  return 0;
}

char Color3::asciiValue() const {
  // Use at least 16 characters, sort these from dark to light
  // or light to dark and then map the weightedSum() to the range
  // 0 to 15. Please pick your own characters
  const char values[] = "WM#0QOB@&!*+-,. ";
  unsigned darkness = 0;
  return values[darkness];
}
// Stream Operators for input and output

std::ostream& operator<<(std::ostream& ostr, const Color3& color) {
  ostr << setw(3) << (int)color.r << " ";
  ostr << setw(3) << (int)color.g << " ";
  ostr << setw(3) << (int)color.b << " ";
  return ostr;
}

std::istream& operator>>(std::istream& istr, Color3& color) {
  int r, g, b;

  istr >> r;
  istr >> g;
  istr >> b;

  color.r = saturate(r, 255);
  color.g = saturate(g, 255);
  color.b = saturate(b, 255);

  return istr;
}
```

6.6 Color3.hpp

```
// Color3.hpp
// Solomon Himelbloom
// Interface for Color3 class
// Original Author: Jonathan Metzgar
// CS 201 course
#ifndef COLOR3_HPP
#define COLOR3_HPP

#include <iostream>

class Color3
{
public:
```

```
  Color3();
  Color3(int R, int G, int B);

  // Returns 0.2126R + 0.7152G + 0.0722B
  int weightedSum() const;

  // Returns an ASCII char representing darkness
  // e.g. ' ' = WHITE and 'W' = BLACK
  char asciiValue() const;

  unsigned char r;
  unsigned char g;
  unsigned char b;
};

std::ostream& operator<<(std::ostream& ostr, const Color3& color);
std::istream& operator>>(std::istream& istr, Color3& color);

#endif
```

6.7 Image3.cpp

```
// Image3.hpp
// Solomon Himelbloom
// Implementation for Image3 class
// Original Author: Jonathan Metzgar
// CS 201 course
#include "Image3.hpp"
#include <iostream>
#include <stdio.h>
#include <string>
#include <map>
#include <fstream>
using std::cin;
using std::cout;
using std::endl;
using std::ifstream;
using std::map;
using std::string;

// Image3 Constructor
Image3::Image3(unsigned width, unsigned height) {
  int w, h;
  w = width;
  h = height;
  pixels.resize(height * width);
}

// Return a pixel from the image
const Color3& Image3::getPixel(unsigned x, unsigned y) const {
  int fin, xres, yres, maxval;
    // fin >> xres >> yres >> maxval;
    if (!fin) {
        cout << "Error reading resolution." << endl;
        exit(3);
    }
    cout << "Image Size: " << xres << "x" << yres << endl;
    cout << "Maximum Value: " << maxval << endl;

```

```
  return pixels[y * w + x];
}

void Image3::setPixel(unsigned x, unsigned y, const Color3& color) {
  int r, g, b, pixels2read;
    for (int i = 0; i < pixels2read; i++) {
        // Map the y-value to a character.
        const char values[] = " .,-+*!&@BOQ0#MW";
        int val_map = y / 16;

        cout << values[val_map];
        if (i % 80 == 79) {
            cout << endl;
        }
    }
}

bool Image3::savePPM(const std::string& path) const {
  // REQUIREMENT: Use the STREAM operators for the file contents
  return false;
}

bool Image3::loadPPM(const std::string& path) {
  // REQUIREMENT: Use the STREAM operators for the file contents
  return false;
}

void Image3::printASCII(std::ostream& ostr) const {
  int fin, r, g, b, xres, yres, vector;
    int pixels2read = xres * yres;
    for (int i = 0; i < pixels2read; i++) {
        // Check status for end-of-file (EOF) errors.
        if (!fin) {
            cout << "Error reading the pixels." << endl;
            exit(4);
        }
        // Push to back of vector (6,400 pixels; each has an R,G,B value).
        vector = 0.2126 * r + 0.7152 * g + 0.0722 * b;
        // Make sure the y-value is [0, 255].
        if (vector < 0 || vector > 255) {
            cout << "ERROR: the y-value is out of range." << endl;
            exit(6);
        }
    }
}

// STREAM OPERATORS for IMAGE3 class

std::ostream& operator<<(std::ostream& ostr, const Image3& image) {
  return ostr;
}

std::istream& operator>>(std::istream& istr, Image3& image) {
  return istr;
}
```

6.8 Image3.hpp

```cpp
// Image3.hpp
// Solomon Himelbloom
// Interface for Image3 class
// Original Author: Jonathan Metzgar
// CS 201 course
#ifndef IMAGE3_HPP
#define IMAGE3_HPP

#include <iostream>
#include <vector>
#include "Color3.hpp"

class Image3
{
public:
  Image3(unsigned width, unsigned height);

  const Color3& getPixel(unsigned x, unsigned y) const;
  void setPixel(unsigned x, unsigned y, const Color3& color);

  bool savePPM(const std::string& path) const;
  bool loadPPM(const std::string& path);

  void printASCII(std::ostream& ostr) const;

  std::vector<Color3> pixels;
  unsigned w;
  unsigned h;
  Color3 borderColor;
};

std::ostream& operator<<(std::ostream& ostr, const Image3& image);
std::istream& operator>>(std::istream& istr, Image3& image);

#endif
```

7 Program 4

7.1 Sample Output/Screenshot

Listing 4: Sample Program Output

```
Mandelbrot set.
```

7.2 Git Commit Messages

Date	Message
2020-12-01	add: color and image template files
2020-12-01	add: comments for rule30
mandelbrot	
2020-11-28	add: hw7 (initial files)

7.3 Source Code

7.4 mandelbrot.cpp

```
// mandelbrot.cpp
// Solomon Himelbloom
// 28 November 2020
// Mandelbrot set example for CS 201.

#include <iostream>
#include <stdio.h>
#include <string>
#include <map>
using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::map;

/*
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)
}
*/

int main() {
    std::cout << "Mandelbrot set." << endl;
}
```