Assignment 11 - Unscrambling Dutch National Flag

Due May 16 by 11:59pm **Points** 20 **Submitting** a file upload **File Types** h and cpp **Available** until May 17 at 12:01am

This assignment was locked May 17 at 12:01am.

ST. CHARLES COMMUNITY COLLEGE

CPT-182 - Programming in C++

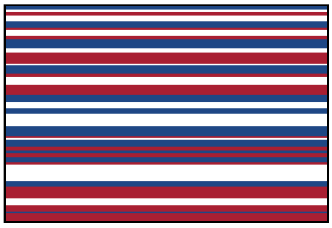
Programming Assignment - Unscrambling Dutch National Flag (20 Points)

(Number in Question Bank: Assignment **11.1**)

Program Overview

The **Dutch National Flag Problem** is one of the most popular programming problems proposed by Dijkstra. The flag consists of three stripes that are colored (from top to bottom) red, white, and blue (see below).

Unfortunately, when the flag arrived, it looked like the figure below; threads of each of the colors were all scrambled together.



Fortunately, we have a machine that can unscramble it, but it needs software.

In this assignment, you are going to write a C++ program that reads in a scrambled flag in PPM format, reorganizes the pixels in the image to unscramble the flag, and writes the unscrambled flag to an output file in PPM format.

The Unscrambling Machine

- The machine can look at one pixel in the flag and determine its color.
- The machine can swap the position of two pixels in the flag.
- The machine can execute any sequential statements.
- The machine can execute `if`/`else` statements and `while` loops. It **cannot** ~~execute other conditional/looping structures~~.
- Please note that you can only **move**/**swap** pixels in the image; you **cannot** ~~create, delete, or change the color of any pixels~~.

The Input PPM Image

In this assignment, your program will read in a **PPM image** as the input file (filename: `scrambled_flag.ppm`). PPM is an image format which uses text to represent images. Please see an example below:

```
1  P3  // This is a "magic number" to indicate that the file is a PPM image.
2  1350 900  // Means that in this image, there are 1350 pixels per row and 900 rows.
3  255  // This is the color depth, which means each color value is between 0 and 255.
4  30  // Red value of the first pixel
5  71  // Green value of the first pixel
6  133  // Blue value of the first pixel
7  169  // Red value of the second pixel
```

```
31  // Green value of the second pixel
50  // Blue value of the second pixel
...
...
...
```

In PPM images, each pixel's color is the combination of **3** values, red, green, and blue. Each value is an integer between **0** and **255**. This way to represent the color of a pixel is called **RGB Color**.

In the first line, the "**P3**" is a "magic constant" identifying this file as a PPM image. The second line contains **2** integers, which represent the dimensions of the image. In this example, the image has **1350** pixels per row and **900** rows. The third line contains a single integer, which represents the **color depth**. In this example, the red, green, and blue values for each pixel range from **0** to **255** (inclusive).

Starting from the next line, the RGB colors of the pixels are listed, which are in row-major order: red value of the first pixel, green value of the first pixel, blue value of the first pixel, red value of the second pixel, green value of the second pixel, blue value of the second pixel, and so on. All values are separated by whitespaces. Please note that the PPM standard does **not** require line breaks any place in particular (or at all); spaces, tabs, or newlines, in any combination, can separate values.

In this assignment, all input images have dimensions of **1350**-by-**900**, **1350** pixels per row and **900** rows.

The Output Image

- The output image is a PPM image with dimensions **1350**-by-**900** (filename: **unscrambled_flag.ppm**).
- After your program unscrambles the pixels read from the input image, it writes the pixels to the output file to generate the unscrambled flag.

The `Pixel` Class

- The class has three **private** data fields, **red**, **green**, and **blue**, which store the RGB color values of the pixel.
- The class has a default constructor (**no** ~~arguments~~) that initializes all the data fields with **0**.
- Write a **friend** function to overload the **stream extraction operator** (">>") for the class. The operator will read the red, green, and blue colors (in that order) from the input stream into the pixel's data fields.
- Write a **friend** function to overload the **stream insertion operator** ("<<") for the class. The operator will write the red, green, and blue colors (in that order) to the output stream. Please separate the color values by newline character (**endl**).
- The class has a **const** class-member function, **get_color()**, that takes **no** ~~arguments~~. The functions returns a string representing the color of the pixel. Please see the table below to understand the three colors in the Dutch national flag.

Value of red	Value of green	Value of blue	Function Return Value
169	31	50	"**red**"
255	255	255	"**white**"
30	71	133	"**blue**"

- You do **not** need to write anything besides what mentioned above in the class.

The "Unscrambling Machine"

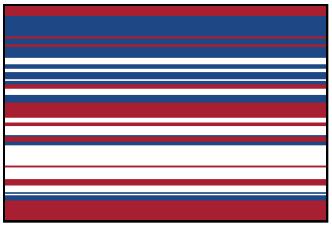
- Please write a header file, **machine.h**, that contains only one function, **unscramble()**.
- The **unscramble()** function takes **a vector of pixels** (type **vector<Pixel>**) as its only argument. Please note that the argument should be passed by reference, **not** ~~**const** reference~~ (since we are changing the vector).
- The **unscramble()** function returns **void**.
- The **unscramble()** function reorganizes the pixels in the vector by swapping so that a) all red pixels are moved to the beginning of the vector, b) all blue pixels are moved to the end of the vector, and c) all white pixels are moved to the middle of the vector.
- The basic algorithm to move pixels is keeping three variables: **end_of_red** (representing the end of the red section), **start_of_blue** (representing the beginning of blue section), and **current** (current pixel to examine).
  1) Initially, **end_of_red** is **0** (red section is empty), **start_of_blue** is "size of vector − **1**" (blue section is empty), and **current** is **0** (starting from the first pixel to examine).
  2) If current pixel's color is red, you need to swap the pixels at **current** and **end_of_red**, then increase both **current** and **end_of_red** by **1**.
  3) If current pixel's color is white, you need to do **nothing** but increase **current** by **1**.
  4) If current pixels' color is blue, you need to swap pixels at **current** and **start_of_blue**, then decrease **start_of_blue** by **1**.
  5) You need to keep examining the pixels one-by-one, until **current** > **start_of_blue**, under which your function should return and stop any further processing, since all pixels are already unscrambled.

- Do **not** forget that you can only use sequential statements, **`if`**/**`else`** statements, and **`while`** loops in the function, since these are what the machine can execute.
- Do **not** forget that you can only **swap** pixels (move pixels) in the image; you **cannot** ~~create, delete, or change the color of any pixels~~.

The `main()` Program

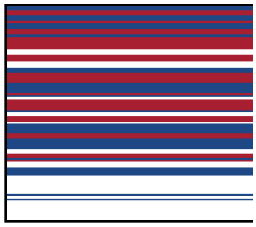
- After you successfully open the input image, you need to read the "header" of the image, the "**`P3`**", dimensions, and color depth.
- Create a vector of pixels (type **`vector<Pixel>`**) to store the pixels read from the input file.
- Using the overloaded "**`>>`**" operator, read in all the pixels from the input image and push them into the vector.
- Call the **`unscramble()`** function to unscramble the flag.
- Write the "header" of PPM image to the output file, then write all the pixels in the vector (already unscrambled) using the overloaded "**`<<`**" operator.

Sample Input and Output Files (Click the Filename to Download)



Sample Input File 1 (https://drive.google.com/uc?export=download&id=1XX5azCzj32rA4IjnFXujpOxVcbuAUwsF)

Sample Output File 1 (https://drive.google.com/uc?export=download&id=1XczcVtqS0UwfKfDFlEu0PmtkS7Ov0Vqm)



Sample Input File 2 (https://drive.google.com/uc?export=download&id=1XRhuP11zAXWK7oGxo)

Online PPM Image Viewer

Unfortunately, Windows operating systems **cannot** open PPM images directly (macOS can). The link below is an online PPM image viewer. You can use it to view your PPM files as images.

http://paulcuth.me.uk/netpbm-viewer (http://paulcuth.me.uk/netpbm-viewer)

Assignment Submission and Grading (Please Read)

- Please upload all your **`.h`** (if any) and **`.cpp`** files (**not** the ~~entire Microsoft Visual Studio project folder~~) on Canvas.
- Before the assignment deadline, you can submit your work unlimited times. However, only your latest submission will be graded.
- At least **20**% of your code should be **comments**. All variable, function (if any), and class (if any) names should "make good sense". You should let the grader put **least effort** to understand your code. Grader will **take off points**, even if your program passed all test cases, if he/she has to put extra **unnecessary** effort to understand your code.
- Please **save a backup copy** of all your work in your computer hard drive.
- Your program will be graded (tested) using another valid input file (still named **`scrambled_flag`**) to check whether it can generate the expected (correct) output file (with correct format and correct output values in it). As long as the input file is valid, your program should generate a correct output file. In other words, your program should work for **any** valid input file, **not** just the sample input files provided in the assignment instructions.
- In this class, you can assume that the input file (input data) is always **valid** and **has correct format**. You do **not** need to deal with ~~invalid input~~ or ~~error handling~~.
- Your work will be graded after the assignment deadline. All students will receive their assignment grades at (almost) the same time.