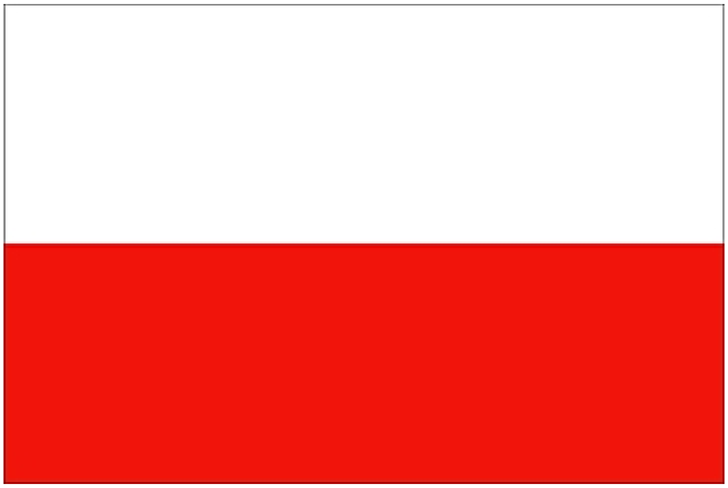
Lab 4: Image file creation

Goals

Construct a C language program that will produce an image of the flag of Poland.



Background

Image files

- Images (e.g. digital photos) consist of a rectangular array of *discrete picture elements* called *pixels.*

- An image consisting of 200 rows of 300 pixels per row contains 300 x 200 = 60,000 individual pixels. The *height* of the image in pixels is the number of pixel rows and the *width* is the number of pixels in each row. A color image requires 3 bytes per pixel or 180,000 total bytes!

- The Portable PixMap *(.ppm)* format is a particularly simple way used to encode a rectangular image (picture) as uncompressed data file.

- The *.ppm* file can viewed with a number of tools including *xv* and *gimp.* M$ Window$ systems have no built-in .ppm viewer. Several freeware viewers are available on the web though. See (www.irfanview.com) for example.

- Other well known formats include JPEG (.jpg), TIFF (.tif), GIF (.gif), bitmap (.bmp), and PNG (.png)

PPM file format

- **ppm files:** *ppm* is a very simple image file format. A *ppm* image consists of two components:

  1. **The header:** the header contains
     1) a *label* to identify the file format as a color ppm file ("P6"),
     2) the *width* of the image in pixels,
     3) the *height* of the image in pixels and
     4) the maximum pixel value (*always 255*).
     5) the header ends with a *single \n* newline character.

  2. **Binary image data:** the data consists of unsigned char (8 bit) binary values defining the color of each pixel.

- **Example ppm header**: The header of a color image of 800 pixels wide and 600 pixels high has the following format

  ```
  P6
  800 600 255
  ```

- **Pixel data format:**

  - each pixel consists of 3 unsigned char (byte) values
  - the first three bytes in the file define the color of the pixel in the upper left corner of the image
  - the last three define the color of the pixel in the lower right
  - pixel values defining each horizontal row of the image are adjacent
  - *NO spaces, tabs, newlines may be embedded in the file.*

- **red/green/blue image encoding**
  - this format is called RGB. The three bytes of each pixel represent the color intensities of the:
    - red component
    - green component
    - blue component
  - (255, 0, 0) is bright red
  - (0, 255, 0) is bright green
  - (0, 0, 255) is bright blue

- **Colors are additive**

  - (255, 255, 0) = red + green = bright yellow
  - (255, 0, 255) = red + blue = magenta (purple)
  - (0, 255, 255) = blue + green = cyan (turquoise)
  - (255, 255, 255) = red + green + blue = white
  - when *red* == *green* == *blue* a gray “color” is produced

- **Writing a pixel value**

  - The *%c* format code tells fprintf() *NOT to convert* the value being printed to ASCII format.
  - Since pixel values are binary, the *%c code must be used*.
  - The following statement can be used to write a red pixel

```
fprintf(stdout, "%c%c%c", 255, 0, 0);
```

  - NO spaces or newlines are permitted in the format string!!

Assignment:

Write a program called lab4.c that creates an image of the Polish flag that is 640 pixels wide and 480 pixels tall. This should be done in two steps.

Step 1:

Write a program that writes a correct .ppm header for the flag to the standard output, and when you think it is correct show it to one of the lab TA's. You and the TA can verify you have no missing or extraneous newlines by piping the output of the program through the *od* program in the following way:

```
 ==> ./a.out | od -t x1
```

A correct header will look something like the following (The hex representation of each byte is in blue with the corresponding ASCII code shown below it in black.

```
0000000 50 36 0a 36 34 30 20 34 38 30 20 32 35 35 0a
        P  6  \n 6  4  0  sp 4  8  0  sp 2  5  5  \n
```

Step 2:

Add the code to write the image data. This section of the code must contain a while loop that iterates 640 * 480 times!

During the first half of the iterations a white pixel must be written and during the second half a red pixel must be written.

When running the program be sure to redirect the standard output to a file.

```
./a.out > poland.ppm
```

Use the xv image viewer to verify that your image is correct. You can also use od to view the image data in hex as follows:

```
od -t x1 poland.ppm | more
```

```
0000000 50 36 0a 36 34 30 20 34 38 30 20 32 35 35 0a ff
0000020 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

Turn In Work

Show your TA that you completed the assignment. Then turn in your lab4.c program using the command:

```
sendlab.101.section_number 4 lab4.c
```