asgn 5: DESIGN - Color Blindness Simulator

Eric Wang

May 28, 2023

Purpose:

For this assignment, we are to write an image-processing program that allows people with normal color vision to experience what people, say, with deuteranopia (a form of red green color blindness) see everyday. The basic needs for this assignment are, Use “unbuffered file-I/O” functions to read and write binary files. We’re also required to use simple shell scripts to avoid typing complex commands for the program. Much like the previous assignments, in our main function, we are also required to have command line options. This time the options are -i, -o, -h. Where -i sets the name of the input file, -o sets the name of the output file, and -h prints the help message, respectively.

How to use the program:

Download the required .c and .h files: bmp.c, bmp.h, io.c, io.h, colorb.c, Makefile, as well as the required bmp test files.
Once you have the required files, run make in your unix terminal and get the binary file.
Next type in this syntax ‘./colorb -i (input file name - the original bmp file) -o (output file name - new bmp file)”
To view the syntax again, use -h to display the help message.
(Sample results in the results section)

Pseudocodes:
io.c

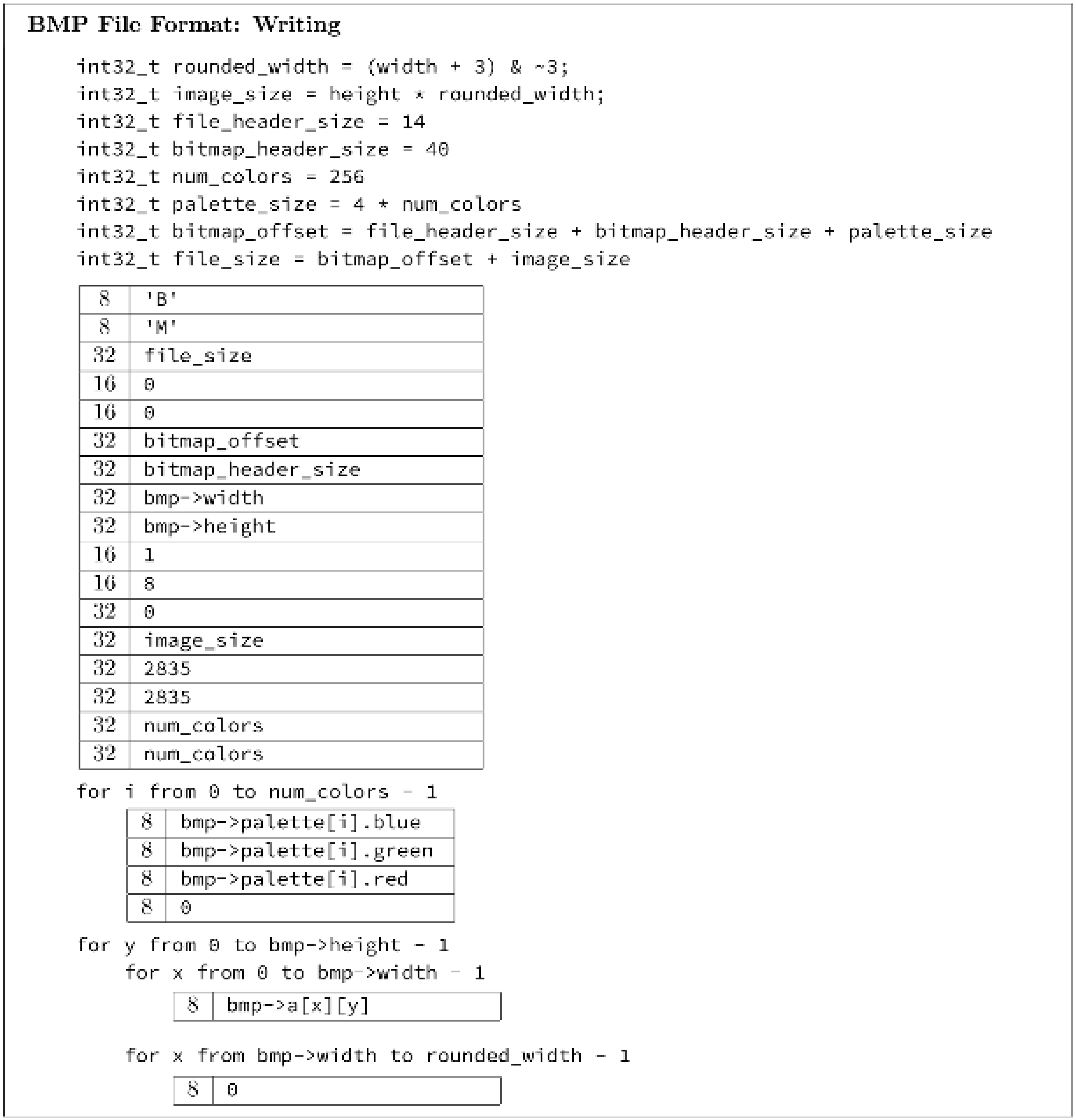
```
Def *read_open(*filename)
    Open file
    If file value is less than 0
        Return null
    Otherwise create a new buffer and allocate the memories for it
    Set file destination to file value
    Set offset to 0
    Set number remaining to 0
    Return buffer

Def read_close(**pbuf)
    Call close((*pbuf)->fd)
    Free buffer
    *pbuf = NULL
```

```
Def read_uint8(*buf, *x)
    Check if number remaining is equal to 0
        Read bytes of the size buffer size
        Return error if it’s less than 0
        Return false if it’s equal to 0
        Set number remaining to the bytes read
        Set offset to 0
    Increment offset
    Decrement number remaining
    Return true

Def read_uint16(*buf, *x)
    Create two uint8_t variables
    Call read_uint8 twice with the two variables
    If either of them is false, return false
    Create a uint16 variable
    Take the second uint8 variable and shift it to the left 8 bits
    And binary or it with the first uint8 variable
    Set x equal to the uint16 variable

    Return true

Def read_uint32(*buf, *x)
    Same as read uint16 but you create two uint16_t variables and at last create a uint32_t variable

Def *write_open(*filename)
    Create a variable using filename and size 0664
    Check if the variable is less than 0, if so, return NULL
    Create a new buffer and allocate the size for it
    Set the file destination to the variable created
    Set offset to 0
    Set number remaining to 0

    Return buffer

Def write_close(**pbuf)
    Create a start array that points to the buffer array
    Create a int variable bytes that points to the buffer offset

    In the do while loop:
        Write in the bytes
        If the bytes are less than 0, print out an ERROR
        Assigning bytes to the array
```

```
                Decrement bytes from offset
        Set offset to 0
        Close buffer
        Free buffer
        Set buffer to NULL

Def write_uint8(*buf, x)
        If the offset is equal to buffer size
                Create a start array that points to the buffer array
                Create num_bytes that points to the buffer offset

                In the do while loop
                        Write in bytes
                        If the bytes are less than 0, print out error
                        Assigning bytes to the start array
                        Decrement bytes from num_bytes
                Set offset to 0
        Assign buffer array to x
        Increment offset

Def write_uint16(*buf, x)
        Call write_uint8 twice, once with x, once with x >> 8.

Def write_uint32(*buf, x)
        Call write_uint16 twice, once with x, once with x >> 16.
```

bmp.c
(the following pseudocodes are taken from Dr. Veenstra’s assignment document)

Def bmp_write(*bmp, *buf)



BMP File Format: Writing

```
int32_t rounded_width = (width + 3) & ~3;
int32_t image_size = height * rounded_width;
int32_t file_header_size = 14
int32_t bitmap_header_size = 40
int32_t num_colors = 256
int32_t palette_size = 4 * num_colors
int32_t bitmap_offset = file_header_size + bitmap_header_size + palette_size
int32_t file_size = bitmap_offset + image_size
```

8	'B'
8	'M'
32	file_size
16	0
16	0
32	bitmap_offset
32	bitmap_header_size
32	bmp->width
32	bmp->height
16	1
16	8
32	0
32	image_size
32	2835
32	2835
32	num_colors
32	num_colors

```
for i from 0 to num_colors - 1
```

8	bmp->palette[i].blue
8	bmp->palette[i].green
8	bmp->palette[i].red
8	0

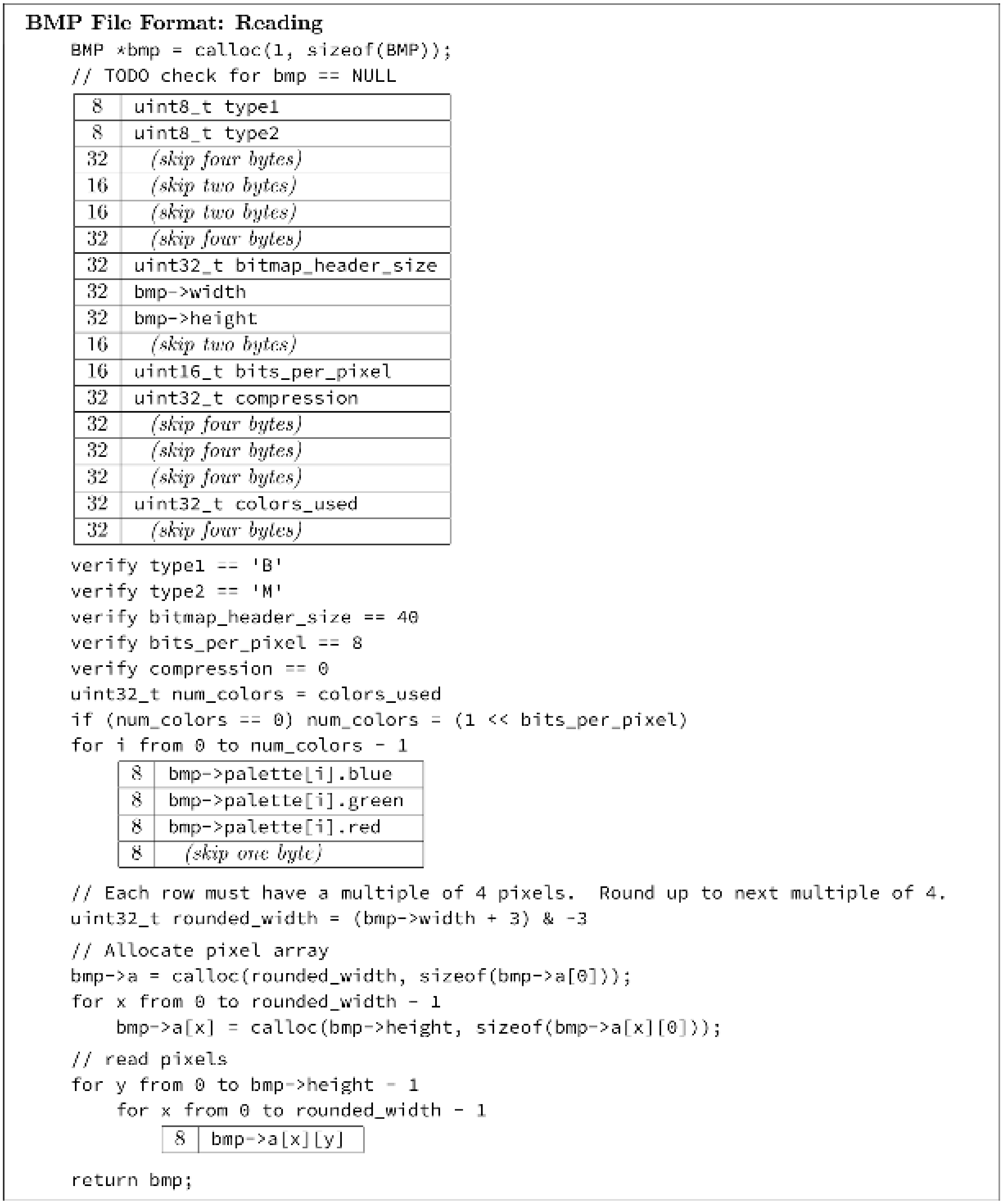
```
for y from 0 to bmp->height - 1
    for x from 0 to bmp->width - 1
```

8	bmp->a[x][y]

```
    for x from bmp->width to rounded_width - 1
```

8	0

Def bmp_create(*buf)



BMP File Format: Reading

```
BMP *bmp = calloc(1, sizeof(BMP));
// TODO check for bmp == NULL
```

8	`uint8_t type1`
8	`uint8_t type2`
32	*(skip four bytes)*
16	*(skip two bytes)*
16	*(skip two bytes)*
32	*(skip four bytes)*
32	`uint32_t bitmap_header_size`
32	`bmp->width`
32	`bmp->height`
16	*(skip two bytes)*
16	`uint16_t bits_per_pixel`
32	`uint32_t compression`
32	*(skip four bytes)*
32	*(skip four bytes)*
32	*(skip four bytes)*
32	`uint32_t colors_used`
32	*(skip four bytes)*

```
verify type1 == 'B'
verify type2 == 'M'
verify bitmap_header_size == 40
verify bits_per_pixel == 8
verify compression == 0
uint32_t num_colors = colors_used
if (num_colors == 0) num_colors = (1 << bits_per_pixel)
for i from 0 to num_colors - 1
```

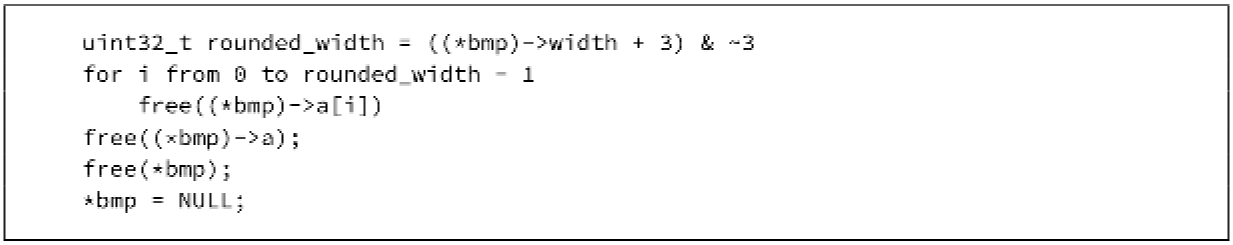
8	`bmp->palette[i].blue`
8	`bmp->palette[i].green`
8	`bmp->palette[i].red`
8	*(skip one byte)*

```
// Each row must have a multiple of 4 pixels.  Round up to next multiple of 4.
uint32_t rounded_width = (bmp->width + 3) & -3

// Allocate pixel array
bmp->a = calloc(rounded_width, sizeof(bmp->a[0]));
for x from 0 to rounded_width - 1
    bmp->a[x] = calloc(bmp->height, sizeof(bmp->a[x][0]));

// read pixels
for y from 0 to bmp->height - 1
    for x from 0 to rounded_width - 1
```

8	`bmp->a[x][y]`

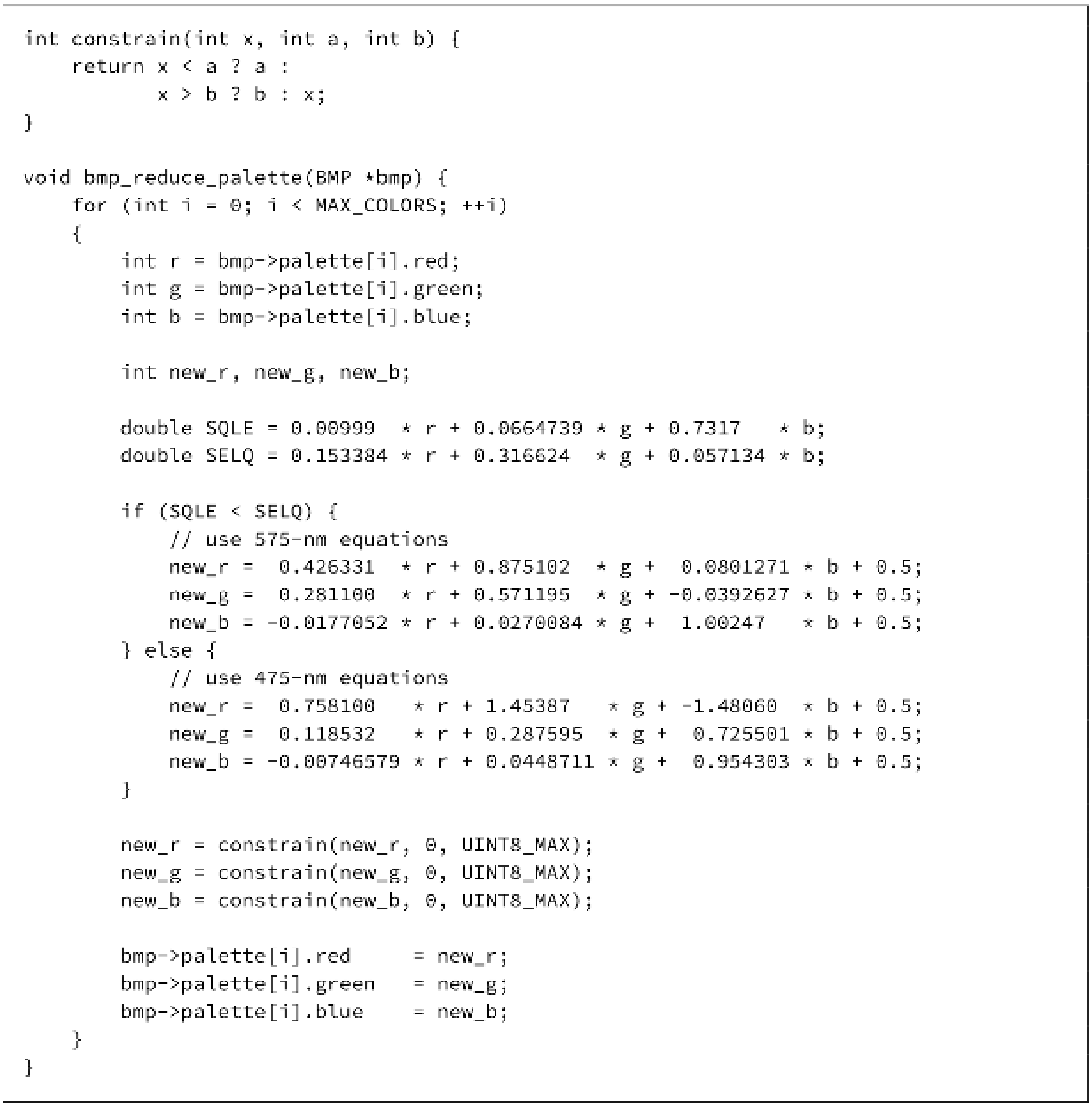
```
return bmp;
```

Def bmp_free(**bmp)



```
uint32_t rounded_width = ((*bmp)->width + 3) & ~3
for i from 0 to rounded_width - 1
    free((*bmp)->a[i])
free((*bmp)->a);
free(*bmp);
*bmp = NULL;
```

Def bmp_reduce_palette(*bmp)

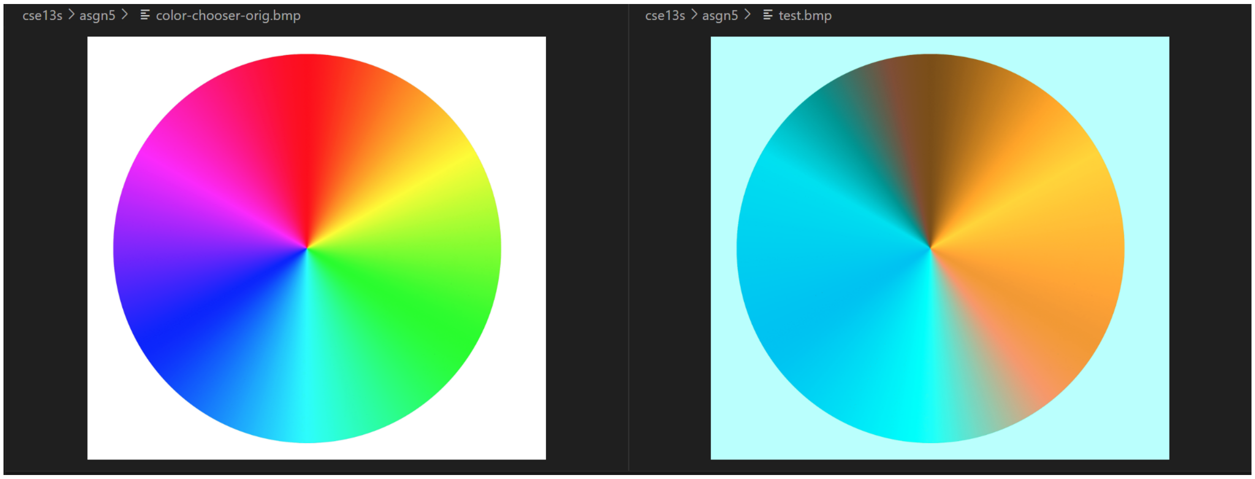


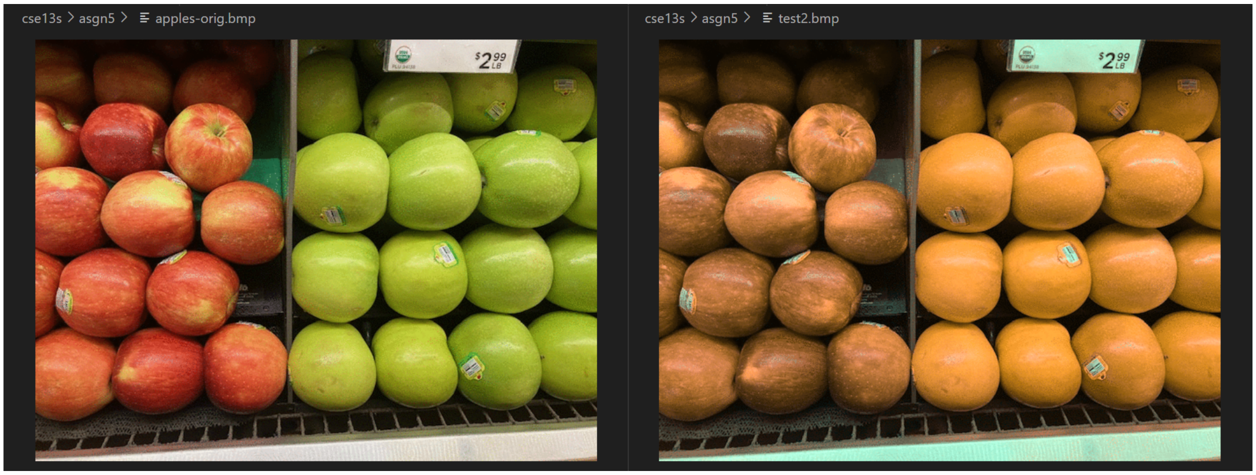
```
int constrain(int x, int a, int b) {
    return x < a ? a :
           x > b ? b : x;
}

void bmp_reduce_palette(BMP *bmp) {
    for (int i = 0; i < MAX_COLORS; ++i)
    {
        int r = bmp->palette[i].red;
        int g = bmp->palette[i].green;
        int b = bmp->palette[i].blue;

        int new_r, new_g, new_b;

        double SQLE = 0.00999  * r + 0.0664739 * g + 0.7317   * b;
        double SELQ = 0.153384 * r + 0.316624  * g + 0.057134 * b;

        if (SQLE < SELQ) {
            // use 575-nm equations
            new_r =  0.426331  * r + 0.875102  * g +  0.0801271 * b + 0.5;
            new_g =  0.281100  * r + 0.571195  * g + -0.0392627 * b + 0.5;
            new_b = -0.0177052 * r + 0.0270084 * g +  1.00247   * b + 0.5;
        } else {
            // use 475-nm equations
            new_r =  0.758100   * r + 1.45387   * g + -1.48060  * b + 0.5;
            new_g =  0.118532   * r + 0.287595  * g +  0.725501 * b + 0.5;
            new_b = -0.00746579 * r + 0.0448711 * g +  0.954303 * b + 0.5;
        }

        new_r = constrain(new_r, 0, UINT8_MAX);
        new_g = constrain(new_g, 0, UINT8_MAX);
        new_b = constrain(new_b, 0, UINT8_MAX);

        bmp->palette[i].red     = new_r;
        bmp->palette[i].green   = new_g;
        bmp->palette[i].blue    = new_b;
    }
}
```

```
colorb.c
Define the options “i:o:h”
Def main(argc, **argv)
    Set opt to 0
    Create 3 boolean values for each option, set them to false
    Create 2 char values, in and out, set them equal to null

    Write the getopt
    Case i
        Set in to optarg
        Set the test for case i as true
    Case o
        Set out to optarg
        Set the test for case o as true
    Case h
        Set the test for case h as true

    Create new read buffer and call read open with “in” variable
    Create new write buffer and call write open with “out” variable

    Creat a new read bmp and call bmp create with “read buffer”
    Call bmp_reduce_palette with “read bmp”
    Call bmp_write with “read bmp” and “write buffer”

    Close read buffer
    Close write buffer
    Free bmp
    Return 0
```

Results:

Here are the sample results


cse13s > asgn5 > color-chooser-orig.bmp
cse13s > asgn5 > test.bmp



Error Handling:
We have to think about how if a user gives invalid options or if no files are specified, we have to handle the error accordingly. With that in mind, I created if statements in the main functions that if any of the test boolean values is false, print out the error message and the help message again. As we can see from the pseudocode in bmp.c there are a bunch of verify steps in the bmp_create() function. For each of the verify steps, I created if statements that if they're false, return an error message and exit the program using the exit(1) function, as exit(1) exits the program because an error is present.

Credit:
Dr. Kerry Veenstra's asgn5 document