Assignment 6 – Color Blindness Simulator

Sebastian Avila

CSE 13S – Fall 2023

Purpose

The purpose of this program is to process an image so that someone with normal color vision can appreciate the range of colors experienced by someone who has deuteranopia.

How to Use the Program

To use this program you will need to have several files in the same directory. The files are `colorb.c`, `bmp.c`, `bmp.h`, `io.c`, `io.h`, and `Makefile`. With these files in the same directory, use command `make clean` to remove any preexisting object files. Then, use command `make format` to format all the header and source files. Then use command `make` to compile the program. You will also need a bit mapped image file in the directory or in a sub-folder. Then run the program using `./colorb -i filename.bmp -o filename2` with any desired command line arguments. The possible command line arguments and their meanings are as follows:

- `-i` : Sets the input file. It requires a file name as an argument. It is required to run the program.
- `-o` : Sets the output file. It requires a file name as an argument. It is required to run the program.
- `-h` : Prints a help message to `stdout`.

The program uses several optional compiler flags:

- `-Wall`: This flag enables all warning messages.
- `-Werror`: This flag turns all warnings into errors.
- `-Wextra`: This flag enables extra warning flags that are not enabled by `-Wall`.
- `-Wstrict-prototypes`: This flag warns if a function is declared or defined without specifying the argument types.
- `-pedantic`: This flag issues all the warnings demanded by strict ISO C and ISO C++.
- `-lm`: This flag links the math.h library. This allows the program to access and use the functions from the math.h library.

Program Design

The program begins by initializeing two boolean variable to represent whether the command line arguments `-i` and `-o` have been entered. It then checks the command line arguments. When the arguments `-i` or `-o` are read, the program sets the corresponding boolean variable to true and attempts to open the input file or output file depending on the argument and assign file pointers to them. It checks if the file pointers are null; if either of them are, it prints an error message and the usage message then terminates. If the program receives an unknown argument, it prints an error message and usage message and terminates. It it recieves

`h`, it prints the usage message and ends. After checking the command line areguments, it checks the values of the boolean variables initialized earlier. If either of them are false it prints out a corresponding error message and the usage message and then it terminates. Otherwise, it sends the input file to a function to create a BMP struct. After this, the input file is closed. This struct is then sent to a function that alters the the color palette of the BMP struct to simulate deuteranopia. The BMP struct is then sent to a function that writes a BMP file to the outfile based on the BMP struct. The program then closes the output file. After that, it frees the memory allocated for the BMP struct and terminates.

Data Structures

MAX_COLORS

This function defines a constant global integer variable `MAX_COLORS`. This variable equals 256 and represents the maximum number of colors possible.

Color

The program defines a type `Color`. It is a struct with three unsigned 8 bit integer variables:

- `red` : The red value of an RGB color.
- `green` : The green value of an RGB color.
- `blue` : The blue value of an RGB color.

```
typedef struct color {
    uint8_t red;
    uint8_t green;
    uint8_t blue;
} Color;
```

BMP

The program defines a type `BMP`. It is a struct with two unsigned 32 bit integer, an array of `Color` variables, and a pointer to an array of pointers to unsigned 8 bit integers:

- `height` : This unsigned 32 bit integer represents the height of the bit mapped image.
- `width` : This unsigned 32 bit integer represents the width of the bit mapped image.
- `palette` : This array of `Colors` represents the color palette of the bit mapped image.
- `**a` : This double pointer is a 2d array of unsigned 8 bit integers the map of the bits.

```
typdef struct bmp {
    uint32_t height;
    uint32_t width;
    Color palette[MAX_COLORS];
    uint8_t **a;
} BMP;
```

Algorithms

Round up

The program includes a function in which a number `x` is rounded up to the next multiple of some number `n`. To accomplish this, the program adds 1 to `x` until the remainder of `x` divided by `n` is equal to 0. Information on the function implementation of this algorithm is in the function descriptions section of this report.

Constrain

The program utilizes an algorithm in which a number $x \in \mathbb{R}$ is rounded to the nearest integer and then made to fit the constraint $0 \leq x \leq 255$. It does this by checking three cases (assuming x is rounded to the nearest integer):

$$x = \begin{cases} 0, & \text{for } x < 0 \\ x, & \text{for } 0 \leq x \leq 255 \\ 255, & \text{for } x > 255 \end{cases} \tag{1}$$

Function Descriptions

io

- `void read_uint8(FILE *fin, uint8_t *px)` : This function takes in a file pointer `fin` and an unsigned 8 bit integer pointer `px`. It does not return anything. Its purpose is to read an unsigned 8 bit integer from the file `fin` and assign the value to `px`. If `EOF` is read, it will report a "fatal error." Psuedocode was given for this function in the assignment instructions[1].

```
void read_uint8(FILE *fin, uint8_t *px)
    result = fgetc(fin)
    if result = EOF then
        report fatal error
    px = 8 bits of result
```

- `void read_uint16(FILE *fin, uint16_t *px)` : This function takes in a file pointer `fin` and an unsigned 16 bit integer pointer `px`. It does not return anything. Its purpose is to read an unsigned 16 bit integer from the file `fin` and assign the value to `px`.

```
void read_uint16(FILE *fin, uint16_t *px)
    read_uint8(fin, px)
    uint_t px2
    read_uint16(fin, px2)
    px2 = px2 << 8
    px = px OR px2
```

- `void read_uint32(FILE *fin, uint32_t *px)` : This function takes in a file pointer `fin` and an unsigned 32 bit integer pointer `px`. It does not return anything. Its purpose is to read an unsigned 32 bit integer from the file `fin` and assign the value to `px`.

```
void read_uint32(FILE *fin, uint32_t *px)
    read_uint16(fin, px)
    uint32_t px2
    read_uint16(fin, px2)
    x  = x << 16
    *px = px OR x
```

- `void write_uint8(FILE *fout, uint8_T x)` : This function takes in a file pointer `fout` and an unsigned 8 bit integer `x`). It does not return anything. Its purpose is to write the integer `x` to the file `fout`. If an error occurs, it reports it. Psuedocode was given for this function in the assignment instructions[1].

```
void write_uint8(FILE *fout, uint8_T x)
    result = fputc(x, fout)
    if result = EOF then
        report fatal error
```

- `void write_uint16(FILE *fout, uint16_T x)` : This function takes in a file pointer `fout` and an unsigned 16 bit integer `x)`. It does not return anything. Its purpose is to write the integer `x` to the file `fout`.

```
void write_uint16(FILE *fout, uint16_T x)
    write_uint8(fout, x)
    write_uint16(fout, x >> 8)
```

- `void write_uint32(FILE *fout, uint32_T x)` : This function takes in a file pointer `fout` and an unsigned 32 bit integer `x)`. It does not return anything. Its purpose is to write the integer `x` to the file `fout`.

```
void write_uint32(FILE *fout, uint32_T x)
    write_uint16(fout, x)
    write_uint16(fout, x >> 16)
```

bmp

- `uint32_t round_up(uint32_t x, uint32_t n)` : This function takes in two unsigned 32 bit integers: `x` and `n`. It returns an unsigned 32 bit integer. Its purpose is to round `x` up to the next multiple of `n`. Psuedocode was given for this function in the assignment instructions[1].

```
uint32_t round_up(uint32_t x, uint32_t n)
    while x % n != 0
        x = x + 1
    return x
```

- `BMP *bmp_create(FILE *fin)` : This function takes in a file pointer `fin`. It returns a pointer to a BMP struct. It purpose is to create a new BMP struct, read a BMP file into it, and return a pointer to the new struct. Psuedocode was given for this function in the assignment instructions[1].

```
BMP *bmp_create(FILE *fin)
    BMP *pbmp = calloc(1 element of sizeof(BMP)
    if pmbp = NULL then
        report fatal error

    read_uint8(fin, type1)
    read_uint8(fin, type2)

    read_uint32(fin, skip)
    read_uint16(fin, skip)
    read_uint16(fin, skip)
    read_uint32(fin, skip)

    read_uint32(fin, bitmap_header_size)
    read_uint32(fin, pbmp width)
    read_uint32(fin, pbmp height)

    read_uint16(fin, skip)

    read_uint32(fin, bits_per_pixel)
    read_uint32(fin, compression)

    read_uint32(fin, skip)
    read_uint32(fin, skip)
    read_uint32(fin, skip)

    read_uint32(fin, colors_used)
```

```
    read_uint32(fin, skip)

    assert(type1 = B)
    assert(type2 = M)
    assert(bitmap_header_size = 48)
    assert(bits_per_pixel = 8)
    assert(compression = 0)
    uint32_t num_colors = colors_used
    if num_colors = 0 then
        num_colors = (1 << bits_per_pixel)
    for i, 0, i < num_colors
        read_uint8(fin, pbmp pallete[i].blue)
        read_uint8(fin, pbmp pallete[i].green)
        read_uint8(fin, pbmp pallete[i].red)
        read_uint8(fin, skip)

    uint32_t rounded_width = round_up(pbmp width, 4)

    pbmp a = calloc(pbmp width elements of size of(pbmp a[0]))
    for x, 0 to x < pbmp width
        pbmp a[x] = calloc(pbmp height elements of sizeof(pbmp a[x][0]))

    for y, 0 to y < pbmp height
        for x, 0 to x < pbmp width
            read_uint8(fin, pbmp a[x][y])

        for x, pmb width to x < rounded_width
            read_uint8(fin, skip)

    return pbmp
```

- `void bmp_write(const BMP *pbmp, FILE *fout)` : This function takes in a pointer to a constant BMP struct `pdmp` and a file pointer `fout`. It does not return anything. Its purpose is it write a BMP image from the bmp struct `pbmp` to the file `fout`. Psuedocode was given for this function in the assignment instructions[1].

```
void bmp_write(const BMP *pbmp, FILE *fout)
    uint32_t rounded_width = round_up(pbmp width, 4)
    uint32_t image_size = pbmp height * rounded_width
    uint32_t file_header_size = 14
    uint32_t bitmap_header_size = 40
    uint32_t num_colors = MAX_COLORS
    uint32_t palette_size = 4 * num_colors
    uint32_t bitmap_offset = file_header_size + bitmap_header_size + palette_size
    uint32_t file_size = bitmap_offset + image_size

    write_uint8(fout, 'B')
    write_uint8(fout, 'M')
    write_uint32(fout, file_size)
    write_uint16(fout, 0)
    write_uint16(fout, 0)
    write_uint32(fout, bitmap_offset)
    write_uint32(fout, bitmap_header_size)
    write_uint32(fout, pbmp width)
    write_uint32(fout, pbmp height)
    write_uint16(fout, 1)
    write_uint16(fout, 8)
    write_uint16(fout, 0)
```

```
    write_uint32(fout, image_size)
    write_uint32(fout, 2835)
    write_uint32(fout, 2835)
    write_uint32(fout, num_colors)
    write_uint32(fout, num_colors)

    for i, 0 to i < num_colors
        write_uint8(fout, pbmp palatte[i].blue)
        write_uint8(fout, pbmp palatte[i].green)
        write_uint8(fout, pbmp palatte[i].red)
        write_uint8(fout, 0)

    for y, 0 to y < pbmp height
        for x, 0 to x < pbmp width
            write_uint8(fout, pbmp a[x][y])

        for x, pbmp width to x < rounded_width
            write_uint8(fout, 0)
```

- `void bmp_free(BMP **ppbmp)` : This function takes in a pointer to a pointer to a BMP struct. It does not return anything. Its purpose is to free all memory used by the BMP struct. Psuedocode for this function was given in the assignment instructions[1].

```
void bmp_free(BMP **ppbmp)
    for i, 0 to i < (*ppbmp) width
        free((*ppbmp) a[i])
    free((*ppbmp)->a)
    free(*ppbmp)
    *ppbmp = NULL
```

- `uint8_t constrain(double x)` : This function takes in a double x. It returns an unsigned 8 bit integer. Its purpose is to round the double `x` to an integer, constrain the value to a range that fits in an unsigned 8 bit integer, and return it. Psuedocode for this function was given in the assignment instructions[1].

```
uint8_t constrain(double x)
    x = round(x)
    if x < 0 then
        x = 0
    if x > UINT8_MAX:
        x = UINT8_MAX
    return (uint8_t) x
```

- `void bmp_reduce_palette(BMP *pbmp)` : This function takes in a pointer to a BMP struct `pbmp`. It does not return anything. Its purpose is to adjust the color palette of a bitmap image `pbmp` to simulate deuteranopia. Psuedocode for this function was given in the assignment instructions[1].

```
void bmp_reduce_palette(BMP *pbmp)
    for i, 0 to i < max colors
        r = pbmp palette[i].red
        g = pbmp palette[i].green
        b = pbmp palette[i].blue

        SqLe = 0.00999 * r + 0.0664739 * g + 0.7317 * b
        SeLq = 0.153384 * r + 0.316624 * g + 0.057134 * b

        if SqLe < SeLq then
            r_new = constrain( 0.426331 * r + 0.875102 * g + 0.0801271 *b)
```

```
            g_new = constrain( 0.281100 * r + 0.571195 * g + -0.0392627 * b)
            b_new = constrain(-0.0177052 * r + 0.0270084 * g + 1.00247 * b)
        else
            r_new = constrain( 0.758100 * r + 1.45387 * g + -1.48060 *b)
            g_new = constrain( 0.118532 * r + 0.287595 * g + 0.725501 *b)
            b_new = constrain(-0.00746579 * r + 0.0448711 * g + 0.954303 *b)

        pbmp palette[i].red = r_new
        pbmp palette[i].green = g_new
        pbmp palette[i].blue = b_new
```

Psuedocode

```
main
    FILE *fin = NULL
    FILE *fout = NULL

    int opt
    opterr = 0
    while (opt = getopt) != -1
        switch opt
            case 'h':
                print usageg message to stdio
                return 0
                break
            case 'i':
                dash_i = true
                FILE *fin = fopen(optarg, read binary)
                if fin == NULL then
                    print error and usage menu to stderr
                    return 1

            case 'o':
                dash_o = true
                FILE *fout = fopen(outfile_name, write binary)
                    if outfile == NULL then
                        print error and usage menu to stderr
                        return 1

            default:
                print error and usage message to stderr
                return 1

    if fin = NULL then
        print error and usage message to stderr
                return 1
    if fout = NULL then
        print error and usage message to stderr
                return 1

    bmp = bmp_create(fin)
    fclose(fin)

    bmp_reduce_palette(bmp)

    bmp_write(bmp, fout)
     fclose(fout)
```

```
bmp_free(&bmp)

return 0
```

Error Handling

- Invalid command line arguments : If an invalid command line argument is given, the usage message will be printed and the program will be terminated.
- Missing arguments : If the program does not recieve the `-i` and `-o` arguments, it will print an error message and the usage message and then terminate.
- Input file is compressed : If the program discovers that the file is compressed while reading in input, it will print an error message and terminate.
- Invalid input file: If the program cannot open the input file, it will print an error message and the usage message and terminate the program.
- Errors will reading program : If the program reads an `EOF` before expected, it will print and error message and terminate the program.
- Invalid out file : If the output file cannot be opened, an error message will be printed and the program will terminate.

Testing

To ensure no memory leaks, the program was run with valgrind using the commands `make debug` and `valgrind ./colorb -i bmps/some-image.bmp -o outputfile.bmp`. The result of this is shown in Fig. 1.It was also compiled with scan-build. This was done using the `Makefile`. I added a rule to the `Makefile` to first clean directory of any object files or executable, then execute the command `scan-build --use-cc=clang make`. This command compiles the files with scan-build and specifies clang as the compiler. `make scan-buld` is the command used to access this `Makefile` rule. This is shown in Fig. 2.

To confirm the output of the program is correct, the `diff` command is utilized. To do this, the binary given in the resources repository will be run and the output will be directed into a file. For this program, this can be done using the commands `./cb.sh` and `./cb_ref.sh`. These executable files run the program and the reference binary with each file in the `bmps` folder. The output files of the reference binary and this program's can then be compared using the `diff` command.

The program is also tested using another program `iotest.c`. This c file was given in the cse13s resources repository[2]. This program was given containing several calls to the read functions implemented in `bmp.c` and checks to ensure they function correctly. I added test for the write functions from `bmp.c` to this program to ensure they work correctly.

Results

The program works as intended. It produces error messages when an error occurs. It responds to all the command line arguments correctly. Figure 3 shows the programs help message and the use of `cb.sh`; an executable that runs `colorb` with all the bit-mapped images in the directory `bmps`. The program produces a changed bit-mapped image that simulates deuteranopia. The results of this can be seen in Figures 4 and 5. Figure 4 shows the original image. Figure 5 shows the altered image made to simulate deuteranopia. To convert the bit-mapped images to png files, I used the command `convert`. This command was given in the assignment instructions[1].

```
savila35@cse13s-vm:~/cse13s/asgn6$ valgrind ./colorb -i bmps/apples-orig.bmp -o bmps/apples-colorb.bmp
==13489== Memcheck, a memory error detector
==13489== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==13489== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==13489== Command: ./colorb -i bmps/apples-orig.bmp -o bmps/apples-colorb.bmp
==13489==
==13489==
==13489== HEAP SUMMARY:
==13489==     in use at exit: 0 bytes in 0 blocks
==13489==   total heap usage: 646 allocs, 646 frees, 322,240 bytes allocated
==13489==
==13489== All heap blocks were freed -- no leaks are possible
==13489==
==13489== For lists of detected and suppressed errors, rerun with: -s
==13489== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 1: Valgrind testing

```
savila35@cse13s-vm:~/cse13s/asgn6$ make scan-build
rm -f colorb io.o bmp.o colorb.o iotest.o iotest
scan-build --use-cc=clang make
scan-build: Using '/usr/lib/llvm-14/bin/clang' for static analysis
make[1]: Entering directory '/home/savila35/cse13s/asgn6'
/usr/share/clang/scan-build-14/bin/../libexec/ccc-analyzer -Wall -Wextra -Wstrict-prototypes -Werror -pe
dantic -gdwarf-4 -c io.c
/usr/share/clang/scan-build-14/bin/../libexec/ccc-analyzer -Wall -Wextra -Wstrict-prototypes -Werror -pe
dantic -gdwarf-4 -c bmp.c
/usr/share/clang/scan-build-14/bin/../libexec/ccc-analyzer -Wall -Wextra -Wstrict-prototypes -Werror -pe
dantic -gdwarf-4 -c colorb.c
/usr/share/clang/scan-build-14/bin/../libexec/ccc-analyzer io.o bmp.o colorb.o -lm -o colorb
/usr/share/clang/scan-build-14/bin/../libexec/ccc-analyzer -Wall -Wextra -Wstrict-prototypes -Werror -pe
dantic -gdwarf-4 -c iotest.c
/usr/share/clang/scan-build-14/bin/../libexec/ccc-analyzer io.o bmp.o iotest.o -lm -o iotest
make[1]: Leaving directory '/home/savila35/cse13s/asgn6'
scan-build: Analysis run complete.
scan-build: Removing directory '/tmp/scan-build-2023-11-20-020922-13500-1' because it contains no report
s.
scan-build: No bugs found.
```

Figure 2: Scan-build testing

```
savila35@cse13s-vm:~/cse13s/asgn6$ ./colorb -h
Usage: colorb -i infile -o outfile
       colorb -h
savila35@cse13s-vm:~/cse13s/asgn6$ ./cb.sh
savila35@cse13s-vm:~/cse13s/asgn6$
```

Figure 3: Program output

Figure 4: Original picture

Figure 5: Colorb picture

References

[1] Dr. Keery Veenestra and TAs. Assignment 6: Color blindness simulator. `https://git.ucsc.edu/cse13s/fall-2023-section-01/resources`, Fall 2023.

[2] Dr. Keery Veenestra. Cse 13s resources. `https://git.ucsc.edu/cse13s/fall-2023-section-01/resources`, Fall 2023.