CS2110 Fall 2017

Homework 8

Author: **Elli Goebel, Henry Peteet**

Overview

The Gameboy Advance (GBA) uses its own format for displaying graphics, which is different from the formats of conventional image files (BMP, JPG, PNG, etc). It would be very helpful if we had a program capable of converting such images for us into a format usable by the GBA, wouldn't it?

You will be writing a program in C called bmptoc which will convert a bitmap file (.bmp) to a c (.c) file. To run the program, type the following into your command line.

```
./bmptoc something.bmp
```

This will create two files – `something.c`, and `something.h`. These files will be used by the GBA to draw images onto the screen.

We have provided the code that does the majority of the file IO. You have to complete the remaining parts which are:

1. getWidth() and getHeight()
    a. These functions will read the data from the BMP header and return the values for the height and width of the image.
2. main()
    a. There is a small part at the end of main that you will need to implement in order to call writeCFile() correctly. We encourage using writeHFile() as a reference.
3. writeCFile()
    a. This function will convert the image data from BMP format to GBA format and print the resulting pixels into a .c file.

You will be writing the code in **bmptoc/bmptoc.c**. You have been provided a Makefile that builds the bmptoc binary for you from the source file. To build the bmptoc binary, navigate to the folder in a terminal, and type:

```
make
```

Learning Objectives

Writing and running C programs
Command-line arguments in C
Using C library functions
Interpreting raw data
File I/O
Using pointers and buffers

Read everything before you start! Many questions that people often ask are answered in this PDF already.

BMP Header

The BMP file is divided into 2 sections: the header and the pixel data.

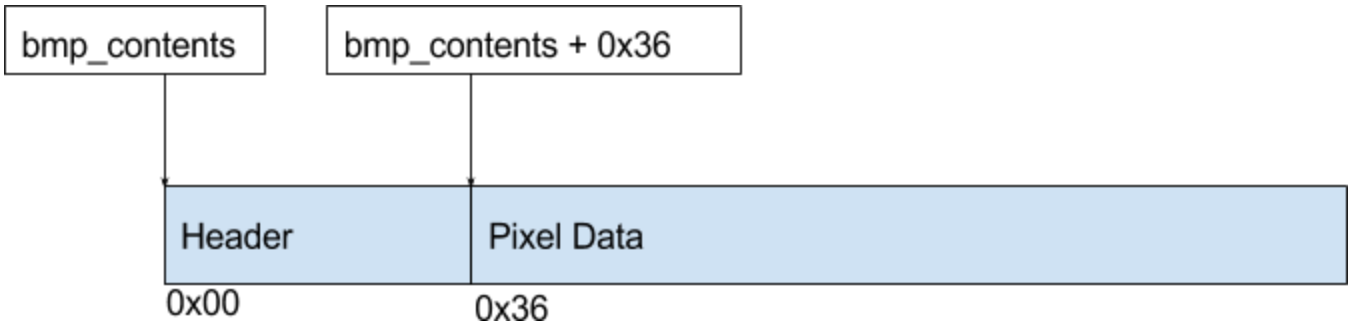


Figure 1: Layout of a BMP file

The header starts at the beginning of the file, is 54 (0x36) bytes long, and consists of useful information about the image such as its file size, dimensions, and pixel format. The only information from the header that we're concerned with is the image's dimensions which exist at the following offsets from the start of the file. Note that the offsets are in hex.

File offset	Length	Field
0x12	4 bytes	Image width
0x16	4 bytes	Image height

Figure 2: Addresses of the Height and Width Metadata Within the BMP File

These numbers are stored in **little endian format**. This means that in an int, the most significant byte is the rightmost byte. For instance, if you had the number 0x12345678, it would be stored in little-endian as 78 56 34 12.

If you read individual bytes to reconstruct the ints, you'll have to deal with the endianness (not

recommended). However if you get integer pointers instead (int*) to those offsets, you can dereference the integer values without worrying about endianness. **It is better form to use the int pointers, as endianness may not be consistent across different machines.**

BMP Pixel Data

The pixel data starts at offset 0x36 in the file, and contains an array of 4-byte entries where there are 4 bytes (32 bits) for each pixel. While you could read these entries as individual bytes you would have to worry about endianness just like in the header. It is recommended that you use integer pointers instead. Within a single 32-bit (4-byte) pixel the alpha, red, blue, and green channels will be laid out as shown in figure 3.

31:24	23:16	15:8	7:0
Alpha	Red	Green	Blue

Figure 3: The BMP Pixel Format

Since the GBA doesn't have an alpha channel in mode 3, you only need to worry about the red, green and blue bytes.

The pixels are ordered sequentially per each row, but **the rows are ordered in reverse**:

Image Data PixelArray [x,y]				
Pixel[0,h-1]	Pixel[1,h-1]	Pixel[2,h-1]	...	Pixel[w-1,h-1]
Pixel[0,h-2]	Pixel[1,h-2]	Pixel[2,h-2]	...	Pixel[w-1,h-2]
			⋮	
Pixel[0,9]	Pixel[1,9]	Pixel[2,9]	...	Pixel[w-1,9]
Pixel[0,8]	Pixel[1,8]	Pixel[2,8]	...	Pixel[w-1,8]
Pixel[0,7]	Pixel[1,7]	Pixel[2,7]	...	Pixel[w-1,7]
Pixel[0,6]	Pixel[1,6]	Pixel[2,6]	...	Pixel[w-1,6]
Pixel[0,5]	Pixel[1,5]	Pixel[2,5]	...	Pixel[w-1,5]
Pixel[0,4]	Pixel[1,4]	Pixel[2,4]	...	Pixel[w-1,4]
Pixel[0,3]	Pixel[1,3]	Pixel[2,3]	...	Pixel[w-1,3]
Pixel[0,2]	Pixel[1,2]	Pixel[2,2]	...	Pixel[w-1,2]
Pixel[0,1]	Pixel[1,1]	Pixel[2,1]	...	Pixel[w-1,1]
Pixel[0,0]	Pixel[1,0]	Pixel[2,0]	...	Pixel[w-1,0]

Figure 4: The Order of Pixels in a .bmp File

This means you will have to read the pixels in order left to right, but the rows must be read bottom to top. Figuring out the math with the indexes here will be somewhat difficult, we suggest drawing out a small array and working out the math before you begin coding.

GBA Mode 3 Pixel Format

In mode 3, the GBA stores pixels as 16 bits per pixel, with 5 bits each for r, g and b.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	Blue					Green					Red				

Figure 5: The GBA Mode 3 Pixel Format

To convert the BMP pixel data into the GBA Mode 3 pixel format, take each 8-bit color value and represent it with 5 bits. When doing this, make sure to preserve the five most significant bits. For example, 240 red from a bitmap would scale to 30 red for the GBA pixel.

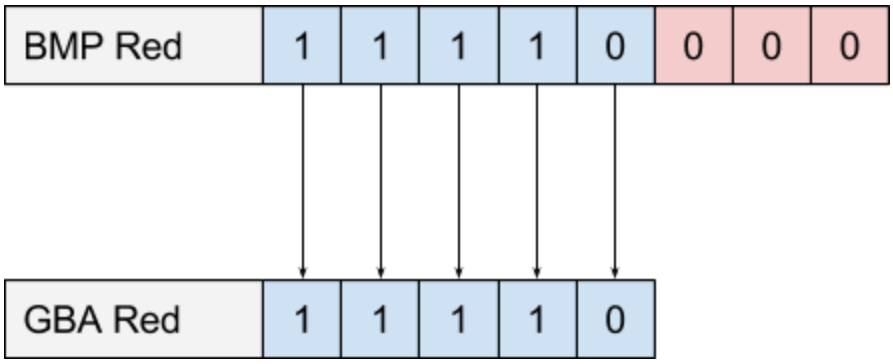


Figure 6: Compressing a 8-bit Color Channel (BMP) into a 5-bit Color Channel (GBA)

After calculating the 5-bit values for each of the colors, combine the three colors into a 16-bit short in the format shown in *Figure 5*. (Bit 15 is unused).

Header File

bmptoc will take the name of the input file, and create a file called `something.h`. This is just an example name; the header will have the same name as the input file, minus the original extension. The macros will be in all uppercase like in the example below. The file will contain two defines for the image width and height, as well as an array prototype for the data array:

```
#define SOMETHING_WIDTH 105
#define SOMETHING_HEIGHT 150
const unsigned short something_data[15750];
```

Data File

bmptoc will also create a file called `something.c` (again, just an example name) that contains the data array for the pixels as unsigned short values. It will look similar to the array below, except it will contain 15750 elements in total.

```
const unsigned short something_data[15750] = {
     0x04B2, 0x491C, 0x7F02, 0x6912, 0x4BBA, 0x1FD5,
     0x51BC, 0x43E7, 0x7244, 0x3AA6, 0x6309, 0x038D,
};
```

The whitespace and newlines between the numbers as well as their base do not matter; what matters is that the array is syntactically correct C code, and that the numbers are correct. Test your arrays and headers in the provided tester to see if the images work!

Image Tester

Initial install instructions:
To install the GBA emulator and the compiler toolchain, enter these commands into your command line one at a time (this requires you to be running Ubuntu):

```
sudo apt-get install gcc-arm-none-eabi
sudo apt-get install libnewlib-arm-none-eabi
sudo apt-get install cs2110-vbam-sdl
```

Testing
A tester has been provided, in which a GBA game is built with your image data and displayed on the screen; it even comes with sample generated code which you can use as a model target for array generation with your bmptoc program. You are not required to know everything about writing Gameboy games for this assignment; the tester is designed so you only have to change `drawimage.c` and `Makefile` to test your image generated by bmptoc. To test your generated image array from bmptoc:

1. Go to the imagetester directory in a terminal, and type:

   ```
   make vba
   ```

2. Now, place your own generated `something.c` and `something.h` files from bmptoc in the imagetester directory.

3. Edit the `drawimage.c` file, and use `#include "something.h"` to use the definitions in the code, similar to the example already in `drawimage.c`.

4. In `drawimage.c`, add a call to drawImage() using your definitions from `something.h`, similar to the example code already there.

5. Edit `Makefile`; under OFILES, add `something.o` to the list of files. Under HFILES, add `something.h` to that list. This is a list of object files necessary to build the Gameboy game. Make sure you also remove the references to diddy.o.

Implementation Hints

You shouldn't name any bitmaps you use "bmptoc.bmp", because it could generate a file called bmptoc.c, overwriting your source file.

If you get `segmentation fault (core dumped)` when running your program, **it means you tried to access a memory address not given to your program by the operating system**. Use Google and read about how to use gdb, and step through your code to verify that your variables have values you expected. It's often caused by an out-of-bounds array index.

To set breakpoints on line numbers and see code as you step through it in gdb, you will need to compile bmptoc using the debug flag -g, which the provided Makefile can do:

```
make debug
```

Be sure to use `make clean` when switching between debug and release builds, because otherwise it might not rebuild the program if it detects the source was not modified.

Deliverables

Submit ONLY the bmptoc.c file

Non-compiling submissions will get a zero!

Use the provided Makefiles to build your application instead of using gcc in the terminal, because if the compilation flags differ, it might not compile for us, and in that case you also get a zero!

References (extra information)

Command line arguments in C are passed into main as 3 arguments:

- int argc — The number of arguments, including invocation of the executable binary. For instance, passing in bar and baz to program foo with "./foo bar baz" would make argc be 3.
- char *argv[] — A pointer array of strings representing the arguments, including invocation of the executable binary. For instance, "./foo bar baz" would make the indices of argv contain "./foo", "bar", and "baz".
- char *envp[] — A pointer array of strings containing environment variables from your terminal. This is not necessary for bmptoc, and it (or any of these args) can actually be omitted from main's function header and it will still run.

A few functions you may find useful for this assignment: **fopen**, **fclose**, **fread**, **fwrite**, **strlen**, **strcpy**, **toupper**, **sprintf**, and **fprintf**. Using a terminal, read the man pages on any of these functions to see how they are called, what they do, and what they return:

```
man fopen
```

As you can see in the man pages, fopen returns NULL if there was an error opening a file. Sounds like it might be useful not only for opening a file, but also finding out if it exists!

You can use sprintf to format a string for writing/printing, using printf's format string syntax:

```
char buffer[50];
sprintf(buffer, "file %s is %d x %d pixels\n", filename,
     width, height);
printf(buffer);

"file something.bmp is 105 x 150 pixels"
```

You can use errno (which is mentioned in the man pages for some of the above functions) to get more detailed information about function invocations that caused an error:

```
printf("an error occurred: %s\n", strerror(errno));
```

Bitmap Format

There are several different formats for bitmap images: http://en.wikipedia.org/wiki/BMP_file_format

For simplicity and standardization purposes, we will only require that bmptoc can deal with **Windows 3.x BMP in bgra format**. Don't assume any random bitmap images you download are in this format, either! To convert any image to this format in Linux, use:

```
ffmpeg -i something.png -pix_fmt bgra somethingelse.bmp
```

If you run this command and it says "command not found", it means that you don't have ffmpeg installed; you will need to install ffmpeg with your package manager. There are ffmpeg installation instructions near the end of this PDF.

To verify that a bitmap file is indeed in the above format, you can use:

```
file something.bmp
```

And it should say something to the effect of (with the important parts underlined):

```
something.bmp: PC bitmap, Windows 3.x format, 105 x 150 x 32
```

If you need to resize an image, I would suggest that you use gimp (also available via the package manager), and converting whatever you save the image as with the aforementioned ffmpeg command to ensure that it is in the correct format.

For your convenience, three images already in this format have been provided in the assignment.

Installing ffmpeg

If you have Ubuntu 16, then you only need to do this step:

```
sudo apt-get install ffmpeg
```

However, ffmpeg is not available in the default repositories for Ubuntu 14 or Mint 17. If you aren't using Ubuntu 16, you must first add a personal package archive (ppa) that contains ffmpeg:

```
sudo add-apt-repository ppa:mc3man/trusty-media
```

Update your package lists with this new information:

```
sudo apt-get update
```

And now you can install ffmpeg as usual, using the first command above.

Try this tutorial if the above steps don't work for some reason

Rules and Regulations

General Rules

1. Starting with the assembly homeworks, any code you write must be clearly commented and the comments must be meaningful.

2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.

3. Please read the assignment in its entirety before asking questions.

4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.

5. If you find any problems with the assignment, please report them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere near the top all other files.

2. When preparing your submission, you may either submit the files individually to T-Square or you may submit an archive (zip or tar.gz only please) of the files (preferred). You can create an archive by right clicking on files and selecting the appropriate compress option on your system.

3. If you choose to submit an archive, please don't zip up a folder with the files. Only submit an archive of the files we want. (See **Deliverables**).

4. Do not submit compiled files that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.

5. Do not submit links to files. We will not grade assignments submitted this way as it is easy to change the files after the submission period ends.

Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know ***IN ADVANCE*** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via T-Square. When you submit the assignment you should get an email from T-Square telling you that you submitted the assignment. If you do not get this email that means that you did not complete the submission process correctly. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over T-Square.

3. There is a 6-hour grace period added to all assignments. You may submit your assignment without penalty up until 11:55PM, or with 25% penalty up until 5:55AM. *So what you should take from this is not to start assignments on the last day and plan to submit right at 11:54AM.* You alone are responsible for submitting your homework before the grace period begins or ends; neither T-Square, nor your flaky internet are to blame if you are unable to submit because you banked on your computer working up until 11:54PM. The penalty for submitting during the grace period (25%) or after (no credit) is non-negotiable.

Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class.
Quizzes, timed labs and the final examination are individual work.

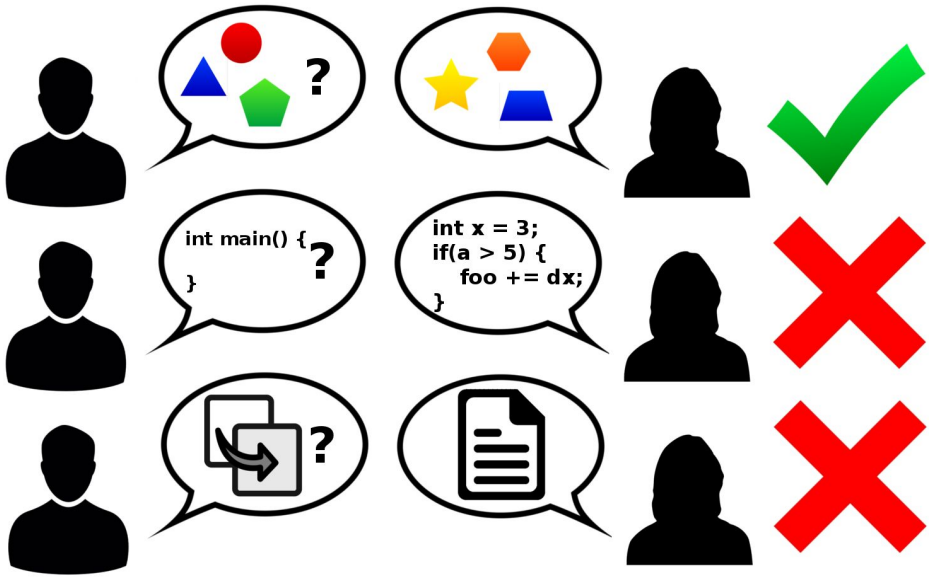
Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using electronic computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply e-mailing it to them so they can look at it. If you supply an electronic copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu

Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, as well as help each other debug code. What you shouldn't be doing, however, is paired programming where you collaborate with each other on a low level. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, and it is often the case that the recipient will simply modify the code and submit it as their own.


?
int main() {
}
?
int x = 3;
if(a > 5) {
foo += dx;
}
?