CS2110 Summer 2016

Homework 7

Author: **Kevin Simon**

Rules and Regulations

General Rules

1. Starting with the assembly homeworks, Any code you write (if any) must be clearly commented and the comments must be meaningful. You should comment your code in terms of the algorithm you are implementing we all know what the line of code does.

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 it would be greatly appreciated if you reported 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 in the file, near the top, for other files unless otherwise noted.

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.

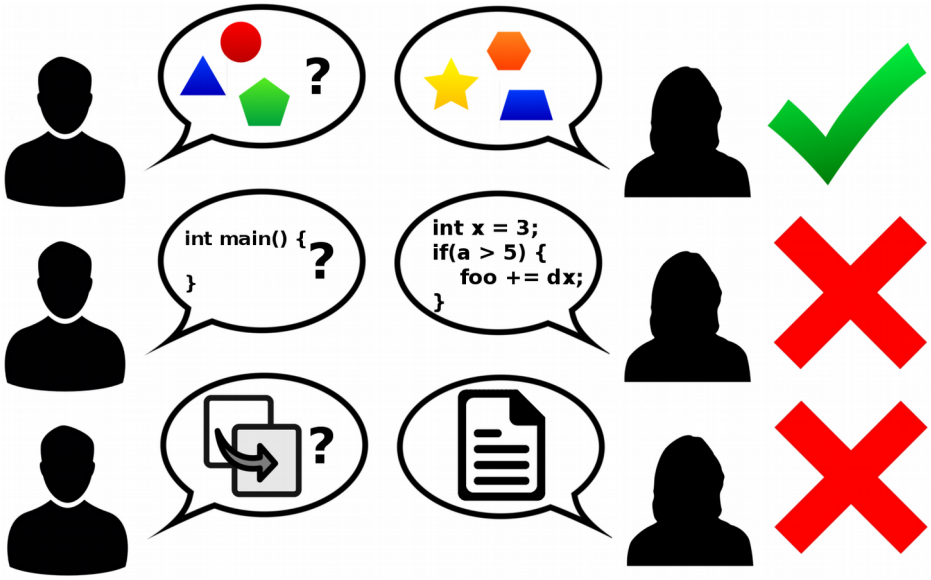
Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using electronic computer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



Overview

Important: We have already implemented some portions of bmptoc that this PDF refers to. In previous semesters we had students write all the file I/O themselves, but realized this added a great deal of complexity when file I/O isn't the main point of this assignment. The comments in the bmptoc source file explain what we have provided, and what still needs to be done, so please read this PDF and the comments in the code provided very carefully.

The Gameboy Advance 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 be able to do just that, with bitmap files; running the program with:

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

Bmptoc will create two files – `something.c`, and `something.h`. If `something.bmp` doesn't exist, bmptoc should tell the user that the file doesn't exist, and then exit without doing anything. If the wrong number of arguments (zero or more than 1) is passed, bmptoc should tell the user and exit. bmptoc should work for any files with a reasonable file name length (we won't test really long file names, but 100 characters should be fine for most situations).

You should be able to include the generated header in a source code file for a GBA game, and use the data array from the .c file to create an image that will display correctly in mode 3 on the GBA.

You will be writing the code in bmptoc.c. You have been provided a Makefile that builds the bmptoc binary for you from the source file; all you need to do is navigate to the folder in a terminal, and type:

```
make
```

Finally, you will write some simple code for a GBA game that will draw the image. In the drawimage folder, type:

```
make vba
```

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
- Drawing to the screen in GBA

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

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 of course means that you don't have ffmpeg installed; you will need to install ffmpeg with your package manager.

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.

BMP Header

The Windows 3.x bgra bitmap header is 54 bytes long, and consists of useful information about the image such as its file size, dimensions, and pixel format. Since we're standardizing the format, the only information we're concerned with are the image's dimensions:

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

The numbers are stored in **little endian format**. This means that the order of the bytes is reversed from what one would consider a human-readable int, but this format is more readable by your computer. 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. However, if you instead get int pointers to those offsets, you can dereference the integer values without worrying about endianness since little endian is the default way numbers are read on most architectures.

BMP Pixel Data

The pixel data starts at offset 0x36 in the file, and contains an array of 4-byte entries, one for each pixel. You could read these entries as individual bytes (and they'll be in the following order), or you could read all four as a single int if you use an int pointer. If you do the latter, you must consider endianness; this means blue would be the lowest 8 bits of the int, and alpha would be the highest 8 bits.

Since the GBA doesn't have an alpha channel in mode 3, you only need 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]

This means you will have to read the pixels in order per row, but the rows in reverse order.

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

You will have to scale down the 8-bit values from the bitmap's components to 5 bits each for the GBA pixel, and combine them properly into one combined value. (bit 15 is unused). Each value of 8 as a bitmap's RGB component is worth 1 as a GBA color component; for instance, 240 red from a bitmap would scale down to 30 red as a component for a GBA pixel. Discard fractional parts in the division.

Format for the Output Files

For the sake of example, let us consider a file called `something.bmp`, having a width of 105 px and height of 150 px. Your bmptoc should be generic enough to handle files with names different from “something.bmp” of course.

Header File

bmptoc should take the name of the input file, and create a file called `something.h`. **This is just an example name**; the header should have the same name as the input file, minus the original extension. The **macros should be in all uppercase** like in the example below; it 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 should also create a file called `something.c` (again, just an example name), which just contains the data array for the pixels, as unsigned short values (array truncated for brevity; there are actually 15750 numbers):

```
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

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. If you haven't already done so, install the `cs2110-tools` and `cs2110-tools-emulator` packages, which are available on T-square > Resources > GBA. Download the amd64 package if you're on 64-bit Linux, or the i386 version if you're on 32-bit Linux.

   ```
   sudo dpkg -i cs2110-tools_1.0-1_amd64.deb
   sudo dpkg -i cs2110-tools-emulator-1.6.0-amd64.deb
   ```

   Note: If you are on Ubuntu 15.04/15.10 use cs2110-tools-emulator-1.5.0-amd64.deb instead.

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

   ```
   make vba
   ```

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

4. 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`.

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

6. 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.

7. Repeat step 2.

Implementation Hints

- You probably shouldn't name any bitmaps you use “bmptoc.bmp”, because it could generate a file called bmptoc.c, overwriting your source file. I did this once while writing this!

- 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));
    ```

- 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.

- For those of you who have an easier time reading code than APIs, here's an example that reads up to 10 bytes from `file1.txt`, then creates `file2.txt` and copies to it. As a word of warning though, this code does not properly check that `file1.txt` exists! It could crash:

  ```
  char buffer[10];
  FILE *infile = fopen("file1.txt", "r");
  int bytes_read = fread(buffer, 1, 10, infile);
  fclose(infile);
  FILE *outfile = fopen("file2.txt", "w");
  fwrite(buffer, 1, bytes_read, outfile);
  fclose(outfile);
  ```

Image Viewer

Finally, you will be writing a small amount of code to draw an image exported by bmptoc to the GBA screen. In the drawimage directory, edit the main.c file to complete the code for drawing.

Be mindful of the warnings in the main.c file about what it should contain when you submit it; the only way for us to properly grade this file is for it to contain headers and function calls that we expect, so **it must only contain #include “diddy.h” (no other includes) and the call to drawImage3 must use the parameters from diddy.h** when you submit the file. If you test drawImage3 using other files, make sure you revert this file back to using only the diddy stuff or you won't get credit for this part.

Deliverables

Submit ONLY the following file, which can be generated in the bmptoc directory by using the command `make submit`:

- bmptoc_submission.tar.gz

Non-compiling submissions will get a zero! Furthermore, 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.