CS2110 Summer 2017
Homework 09

Authors: Kevin Berry

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 11:54PM.* 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.

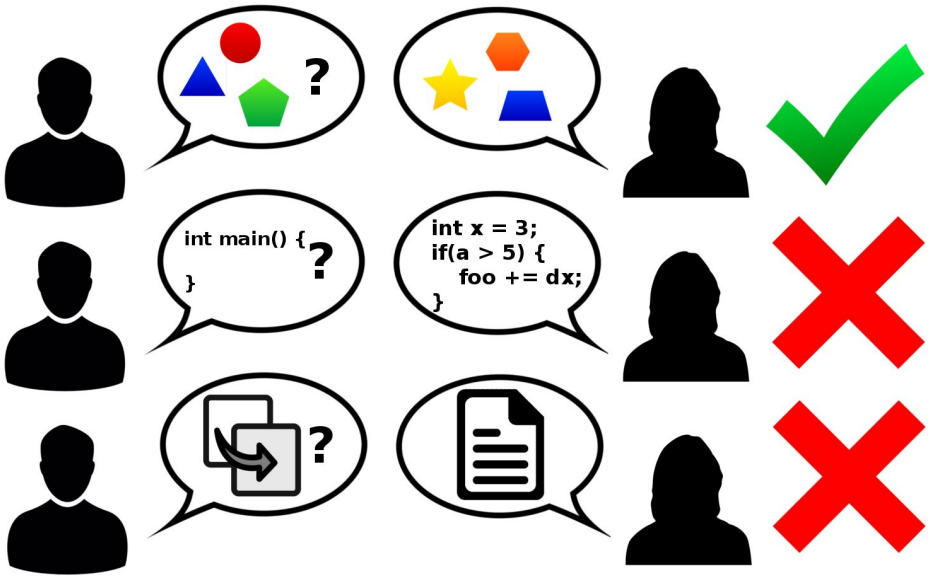
Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using electronic computer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



Objective:

The goal of this assignment is to make a C program, a game. Your game should include everything in the requirements and be written neatly and efficiently! Your main.c file should be something different from lecture code, since in this homework you will be creating your own game, but keep the core setup with REG_DISPCNT, videoBuffer, mode 3 use, waitForVBlank, etc. Prototypes, defines and extern declarations should be put into a **myLib.h**. It is also optional for you to use other .c/.h files to organize your game logic if you wish, **just make sure you include them in your submission and your Makefile**.
Additionally, **do not simply rehash lecture code in your game**. This means that you are not allowed to just slightly modify lecture code and to call it a day. If we open your game and we only see several boxes flying in random directions, that will be a very bad sign, and you will not receive a very pleasant grade. Also, please do not make Pong. Everyone asks if they can make pong, and it's just a boring, low-effort game in general. (Arkanoid/Brick Breaker is okay though)

Requirements:

1. Must be in Mode 3
2. You must also implement **drawImage3** with **DMA**. The prototype and explanation are later in the assignment.
3. Images – You must use at least 3 distinct images in your game, all drawn with DMA
    a. A title screen sized 240x160
    b. A game over screen sized 240x160
    c. A third image which will be used during game play. The width of this image may not be 240 and the height of this image may not be 160.
    d. Note: all image files should be included in your submission
4. You must be able to reset the game to the title screen AT ANY TIME using the **select** key. This should correspond to the **backspace** key on your keyboard.
5. You must create a header (**myLib.h**), and move any #defines, function prototypes, and typedefs to this file from your game code, along with your extern videoBuffer statement if you wish to use videoBuffer in other files. Remember that function and variable definitions should not go in header files, just prototypes, struct definitions, and extern variable declarations. **We will deduct points if we find these in your .c files, so make sure they are only in your headers.**
6. You must use at least one **struct**.
7. **Button input** should visibly and clearly affect the flow of the game
8. You must have **2-dimensional movement** of at least one entity. One entity moving up/down and another moving left/right alone does not count.
9. You should implement some form of **object collision**. For games where application of this rule is more of a gray area (like Minesweeper), core functionality will take the place of this criteria, such as the numbers for Minesweeper tiles calculated correctly, accurate control, etc.
10. You must implement **waitforVBlank** and the scanlinecounter declaration.
11. Use text to show progression in your game. Use the example files from lecture, and you

can look into it more in Tonc: http://www.coranac.com/tonc/text/text.htm

12. **There must be no tearing in your game. Make your code as efficient as possible!**
13. Include a **readme.txt** file with your submission that briefly explains the game and the controls.

What Game to Make?

You may either create your own game the way you wish it to be as long as it covers the requirements, or you can make games that have been made before with your own code. **However, your assignment must be yours entirely and not based on anyone else's code. This also means that you are not allowed to base your game off the code posted from lecture. Games that are just slightly modified versions of Dan's code are subject to heavy penalties.** Here are some previous games that you can either create or use as inspiration:

Galaga: http://en.wikipedia.org/wiki/Galaga
Requirements:

1. Accurate, efficient O(1) rectangular collision detection implemented as a function.
2. Lives displayed with text or images.
3. Game ends when all lives are lost. Level ends when all aliens are gone.
4. Different types of aliens – there should be one type of alien that rushes towards the ship and attacks it
5. Smooth movement (aliens and player)

The World's Hardest Game (Challenge our skill with your game):
http://www.addictinggames.com/action-games/theworldshardestgame.jsp
Requirements:

1. Accurate, efficient O(1) rectangle collision detection as a function.
2. Smooth motion for enemies and player (no jumping around)
3. Restriction to the boundaries of the level.
4. Enemies moving at different speeds and in different directions.
5. Sensible, repeating patterns of enemy motion
6. Enemies and the Player represented by Structs

Frogger: http://en.wikipedia.org/wiki/Frogger
Requirements:

1. The Frog and the logs/lily pads must be represented by Structs internally.
2. O(1) collision detection with any object. If the frog collides with traffic, it dies. If it does not land on a lilypad/log, then it also dies. The lily pads and logs in the river must move the froggy along with them.
3. There is a time limit in which the frog must get to his home. If time expires then the frog also dies (hint there are about 60 vblanks per second.)
4. Once a Froggy occupies a home, another frog cannot occupy that home.
5. The game must display a set amount of lives the player can lose before losing. The game is lost when all of the lives are lost. The game is won if all frogs get to their homes.

Some common game ideas we are asked that we **don't recommend** for this homework are platformers (Super Mario Bros), Pac-Man, RPGs (Pokemon), or turn-based strategy (Fire Emblem). These types of games have usually proven very difficult to implement well given the time constraints for this homework.

Images

As a requirement you must use at least 3 images in your game and you must draw them all using drawImage3. If your bmptoc submission works, you may use that to process image files (provided you first use ffmpeg to convert it to the correct format from HW08). If your bmptoc doesn't work or you would prefer the convenience of not having to convert between different image encodings, Brandon has built a tool called **nin10kit** which he has described how to install on Piazza. Or, you may use our provided tool that will make this task easier for you:

CS2110ImageTools.jar – available on TSquare in CS-2110 Resources/GBA

CS2110ImageTools.jar reads in, converts, and exports an image file into a format the gba can read – .c/.h files! It also supports resizing the images before they are exported.

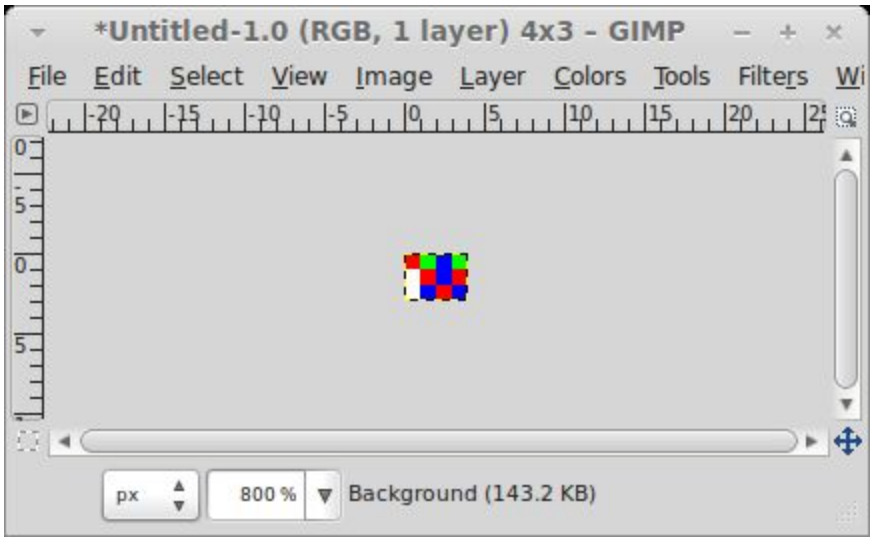
You may run the program from terminal in the directory where you downloaded it like this:

```
java -jar CS2110ImageTools.jar
```

CS2110ImageTools.jar will give you a graphical interface to load image files and save their converted files. The output of this program will be a .c file and a .h file. In your game you will #include the header file. It contains an extern statement so any file that includes it will be able to see the declarations given in the .c file CS2110ImageTools.jar exported. Inside the exported .c file is a 1D array of colors which you can use to draw to the screen.

Example

For instance take this 4x3 image drawn in GIMP



When this file is exported it will generate a c array like this:

```
const unsigned short example[12] =
{
      // first row red, green, blue, green
      0x001f, 0x03e0, 0x7c00, 0x03e0,
      // white, red, blue, red
      0x7fff,0x001f, 0x7c00, 0x001f,
      //white, blue, red, blue
      0x7fff, 0x7c00, 0x001f, 0x7c00,
}
```

This 1D array has 12, or 4 times 3, entries. Each row from the image is stored right after the other. So if you want to access coordinate (row = 1, col = 3) then you should get the value at index 7 from the array, which will be the color data for drawing a red pixel.

DMA / drawImage3

In your game you must use DMA to code drawImage3.

DMA stands for Direct Memory Access and may be used to make your rendering much faster. If you want to learn more about DMA, you can read these pages from tonc, up until 14.3.2.

You must not use DMA to do single pixel copies (Doing this defeats the purpose of DMA and is slower than just using setPixel!). Solutions that do this will receive no credit for drawImage3.

The prototype and parameters for drawImage3 are as follows:

```
/* drawImage3
 * A function that will draw an arbitrary sized image
 * onto the screen (with DMA).
 * @param r row where the top of the image will be drawn
 * @param c column where the left edge of the image will be drawn
 * @param width width of the image
 * @param height height of the image
 * @param image Pointer to the first element of the image.
 */
void drawImage3(int r, int c, int width, int height, const u16* image)
{
    // @todo implement :)
}
```

Tip: if your implementation of this function does not use all of the parameters that are passed in then **YOU'RE DOING IT WRONG**.

Here is a hint for this function. You should know that DMA acts as a for loop, but it is done in hardware. You should draw each row of the image and let DMA handle drawing a row of the image.

C coding conventions:

1. Do not jam all of your code into one function (i.e. the main function)
2. Split your code into multiple files (for example, you can have game logic in your main file, library functions in mylib.c with declarations in mylib.h, game specific functions in game.c)
3. Comment your code, and comment what each function does. The better your code, is the better your grade will be!

Warnings

1. Do not use floats or doubles in your code. Doing so will SLOW your code down GREATLY. The ARM7 processor the GBA uses does not have a Floating Point Unit which means floating point operations are SLOW and are done in software, not hardware. All that crazy IEEE floating point stuff Dan covered in lecture? Anywhere you use floats, gcc has to insert assembly code to convert integers to that format. If you do need such things then you should look into fixed point math (google search).
2. Only call waitForVBlank once per iteration of your while/game loop
3. Keep your code efficient. If an O(1) solution exists to an algorithm and you are using an $O(n^2)$ algorithm then that's bad (for larger values of n)! Contrary to this, only worry about efficiency if your game is showing signs of tearing!

WE STRONGLY ADVISE that you go **ABOVE AND BEYOND** on this homework. **PLEASE do not just clutter your screen with squares that run around with no meaning in life** – such submissions will **lose** points. Make some catchy animations, or cut scenes! These games are always fun to show off after the class, and you can even download emulators on your phone to play them. Also, **remember that your homework will be partially graded on its creative properties and evidence of effort**. You are much more likely to make your TAs very happy and to have a better demo/grade if you go above and beyond what is required.

You may research the GBA Hardware on your own. You can read Tonc, however you may not copy large swaths of code, only use it as a reference. The author assumes you are using his gba libraries, which you are not and may not.

Here are the inputs from the GameBoy based on the keyboard for the default emulator vbam:

```
GameBoy | Keyboard
--------|----------
Start   | Enter
Select  | Backspace
A       | Z
B       | X
L       | A
R       | S
Left    | Left Arrow
Right   | Right Arrow
```

```
Up        | Up Arrow
Down      | Down Arrow
```

Note: The alternate emulator we support, mednafen, has different (configurable) controls. Holding the spacebar will make the emulator run faster. This might be useful in testing, but the player should never have to hold down spacebar for the game to run properly since there is no magical space bar speedup on an actual Gameboy.

You can learn more about button inputs on this site: http://www.coranac.com/tonc/text/keys.htm

If you want to add randomness to your game then look up the function rand in the man pages. Type man 3 rand in a terminal.

Demo

This assignment will be demoed. Some time around the due date, a T-Square announcement will be made to signal demo slot availability. As usual, sign up for a demo slot early to guarantee that you get a demo slot. If none of the available slots will work for you, then you may e-mail the head TA **before Monday on the week of demos being held** to request accommodation. If you wait until Monday to sign up for a demo slot and find that there are no times that you can make, then you'll just have to miss one of your classes to sign up for an available slot, or get a 0 on HW09 if there are no slots left.

Deliverables

A tar.gz archive containing the following files:

myLib.h
All of your c files
All of your h files
Makefile (your modified version)
Your README
and any other files that you chose to implement

You can create the submission archive with **tar -czvf hw09_submission.tar.gz hw09_folder** (where hw09_folder is the name of your folder, of course)

Note: After submitting, make sure to re-download your code into an empty folder and check that it compiles with the command:

```
make
```

If your submitted program doesn't compile when we try it, you will receive a zero (0).

Good luck, and have fun!