CS2110 Fall 2014
Homework 8

This assignment is due by:

Day: Tuesday October 21, 2014
Time: 11:54:59pm

Rules and Regulations

Academic Misconduct

Academic misconduct is taken very seriously in this class. Homework assignments are collaborative. However, each of these assignments should be coded by you and only you. This means you may not copy code from your peers, someone who has already taken this course, or from the Internet. You may work with others **who are enrolled in the course,** but each student should be turning in their own version of the assignment. Be very careful when supplying your work to a classmate that promises just to look at it. If he/she turns it in as his own you will both be charged.

We will be using automated code analysis and comparison tools to enforce these rules. **If you are caught you will receive a zero and will be reported to Dean of Students.**

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. Make sure that you submit and demo your assignment by the last demo date of the class. We will not demo anyone past that date, and all unfinished homework assignments will receive a score of zero.

General Rules

1. In addition 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.
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.

<u>Submission Conventions</u>

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 in 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.

Overview

The goal of this assignment is to get you familiar with the GBA environment. If you have not already, you will need to install several packages: cs2110-tools and cs2110-tools-emulator. More information can be found in the recent announcements. T-Square in Resources / GBA / GBA tools installation instruction.htm.

Part 1

It's time to create a library file for your GBA game. You will be using sets of these files to create your games for the next few assignments. Please keep the following in mind: We are going to create two .c files: a main.c and a mylib.c. The purpose of the main.c file is to write all of your game logic. For instance, you may have a neat function that will initialize all of the bricks' locations for a Breakout game; this would go in your main.c file. Other useful functions that would apply to any game you could write, for example, a function that draws a rectangle on the screen would go in mylib.c. Please keep your files organized, as it will make your TAs happy.

In addition, if you do not understand a concept, then you should be reading the required C book for this class. We will be moving quickly through this stuff and if you fall behind your grade will hurt. Terms you should know by the end of this assignment are italicized.

We will be creating the following files: mylib.c and main.c

We will be using a slightly modified version of this mylib.c for your next assignment.

mylib.c

Please add the following pre-processor declarations to this file:

- **u16** – This will be an alias of the unsigned short type. You don't want to type `unsigned short` all of the time so you should use *typedef* to make u16 an alias of unsigned short, e.g. you could write u16* videoBuffer instead of typing out unsigned short* videoBuffer.
- **videoBuffer** - A **global variable** pointing to the start of video memory. This is located at 0x6000000 and should **point** to a u16.
- ***Function prototypes*** – for each function you have to declare in mylib.c. These should be located above the actual function implementations. These are forward declarations of functions without defining them, so the compiler knows their signature ahead of time. You will see in Part 2 why this is useful.
- ***Functions*** – You need to implement these four functions. (You may switch row and column with x and y. I personally prefer (x, y), Bill likes (r, c)).

```
// A function to set pixel (r, c) to the color passed in.
void setPixel(int r, int c, u16 color)
{
     // @todo implement :)
}
```

```
// A function to draw a FILLED rectangle starting at (r, c)
void drawRect(int r, int c, int width, int height, u16
color)
{
                // @todo implement :)
}


// A function to draw a HOLLOW rectangle starting at (r,
c)
// NOTE: It has to run in O(w+h) time.
void drawHollowRect(int r, int c, int width, int height,
u16 color)
{
// @todo implement :)
}
```

The fourth function that you will need to write is an implementation of the **Bresenham's Line Algorithm**. It is a famous algorithm that is commonly used to draw lines to a computer screen. What is special about it is that it can approximate lines between two points using only integer addition, subtraction, and bit shifting. The GBA does not have dedicated hardware for floating point arithmetic, and so any calculations using floats are done in-software and are generally very slow. That is where this algorithm is really useful, since it uses relatively cheap computations. You can read more about the algorithm here: http://en.wikipedia.org/wiki/Bresenham's_line_algorithm. Below you will find pseudocode for the algorithm. Please only use it as a guide, and note the following:
NOTE: You may **only** use addition, subtraction, and bit shifting operations in your implementation. Multiplication and division are **not** allowed (that's the point!).

```
void plotLine(int x0, int y0, int x1, int y1, u16 color)
    dx=x1-x0
    dy=y1-y0

    D = 2*dy - dx
    plot(x0,y0)
    y=y0

    for x from x0+1 to x1
        if D > 0
            y = y+1
            plot(x,y)
            D = D + (2*dy-2*dx)
        else
            plot(x,y)
            D = D + (2*dy)
```

Part 2

main.c

For main.c, we will need to include a few declarations as well. Notice that some of the following declarations are duplicates from the mylib.c file. In your next assignment, we will be using header (.h) files which allow you to remove much of this redundancy. If you know how to use header files, feel free to go ahead and implement a mylib.h file, but this is not required for the assignment.

- **REG_DISPCNT** - The display control register, located at 0x4000000. This will be a ***symbol*** that will **access** the memory location 0x4000000 and get the **unsigned short** located at 0x4000000 for us.
- **RGB** - A *macro* you should #define at the beginning of your file. This macro takes three integers representing the red, green, and blue components of the color and returns the corresponding color value. Your macro should work in all cases e.g. RGB(2+2, 4+4, g+2) and should respect order of operations, based on the examples from class! (Hint: "Encapsulation")
- **u16** – This will be an alias of the unsigned short type. You don't want to type unsigned short all of the time so you should use *typedef* to make u16 an alias of unsigned short.
- **videoBuffer** - The global variable videoBuffer declared in mylib.c should be visible in any file that needs to use it. To do this, extend the scope of videoBuffer to main.c by using the extern keyword when referring to the variable from inside main.c.
- ***Function prototypes*** – for each function you had to declare in mylib.c. This is so the compiler knows how to link between mylib.c and main.c.
- **A main function as defined below**:

```
int main(void)
{
        // Put your code here
}
```

In your main function you should set up REG_DISPCNT as shown in class.
Remember that we want mode 3 and to enable background 2.

You will then have a busy loop somewhere in your code to keep the program from exiting. For example, you could use while(1). You would set up your drawing inside of the while-loop.

Your Goal:

The goal of this assignment is for you to get familiar with the GBA environment by setting pixels and drawing an image to the screen. After implementing the functions above, you will have a toolkit for drawing to the screen. You will be using these function and any additional functions that you wish to implement. Feel free to implement functions to draw other shapes, like triangles or circles!

We are not expecting amazing works of art from you, but at the same time we are expecting a bit more than just random boxes placed around the screen for no reason. **A part of the grading criteria is dedicated to the creativity and the demonstrated work ethic of your submission**, so please do not try to rush through the assignment. You are much more likely to receive a great grade from your TA if it is clear that you have put in some time and thought into your homework.

This is one of the most open-ended assignments of this course. The GBA screen is your canvas, and there are many different ways that you could go. That said, you must follow the rules listed below in order to not lose major points. Additionally, we want to make one point very clear: **please do not rehash lecture code in your game**. The work that you produce should be substantially different from the examples shown in lecture and from the works of others.

Requirements:

1. Implement all the declarations that are listed above.
2. At least ¾ (three fourths) of the pixels on the screen need to be set. (be non-black)
3. You must use at least 5 different colors, **not counting** black
4. Write and use setPixel() somewhere in your code.
5. Write and use drawRect() somewhere in your code.
6. Write and use drawHollowRect() somewhere in your code. **NOTE: It must run in O(w+h) time.**
7. Write and use plotLine() in your code. **NOTE: Addition, Subtraction, and Bit shifting only.**
8. Creativity and clearly-demonstrated work ethic. Put some time into this! Do not just draw boxes all over the screen, as that will decrease your grade.

<u>Optional</u>: Animation and Button Inputs

We will be covering the following in a future homework, and they are **not required** for now.
Still, if you would like to *really* get into GBA and to "wow" your TAs, you could try creating an animation in your drawing. This would essentially require you to have a busy loop of some kind as a timer, and every time it is ready to update, you should draw a different frame of an image (you could update a part of the screen, not the whole screen). It does not have to be exceptionally complex, but it is something that would make your TAs very happy.

Additionally, you may want to look ahead into button inputs. The button layout is as follows:

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

Holding the space bar will make the emulator run faster. This might be useful in testing, but the player should never have to hold down space bar for the game to run properly and furthermore there is no space bar on the actual GBA.

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

Warning

Two things to note for now

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. If you do need such things then you should look into fixed point math (google search). **NOTE: Hence only Bit shifting in plotLine().**
2. Do not do anything very intensive, you can only do so much before the GBA can't update the screen fast enough. You will be fixing this problem in the next assignment.

I **STRONGLY ADVISE** that you go **ABOVE AND BEYOND** on this homework. **PLEASE do not just sprinkle your screen in boxes** – such submissions will **lose** points. Draw something amazing, or implement something really cool! In the next assignment you will be making a game for us to play. The more you do now, the better off you will be. Also, **remember that your homework will be partially graded on its creative properties and work ethic**. 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 (available at http://www.coranac.com/tonc/text/), however you may not copy code wholesale from this site. The author assumes you are using his GBA libraries, which you are not.

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. Lastly if you want to use an image (from the internet) in your submission, you may download and install CS2110ImageTools.jar (also available in Resources / GBA). This program will take an image and convert it into an array of colors for you to draw on the screen. However, this will require knowledge of header files and multiple C files. We will get to all of this in the next assignment, but it is great to look ahead to those things and try them out in this homework.

Part 3 - Makefile

The makefile is a file used by make to help build your gba file. Normally you would use gcc directly to compile you C programs. However the compiling process to make a .gba file is a little complex for you to compile by hand. This is why we have provided for you a Makefile that will *compile*, *link*, and run your program on the emulator with one little command “make vba”.

However the only thing that you must do is set up our Makefile to build your program. So you must edit the Makefile and change the PROGNAME and OFILES to compile your program. Quick summary of what you must do

PROGNAME should be the name of the generated .gba file. EXAMPLE: HW8.

OFILES should be a list of .o files (space separated) needed to be linked into your gba file. For each .c file you have put it in this list and replace the .c extension with .o. EXAMPLE: main.o lol.o hi.o

To use the Makefile, place it in the same folder as your other game files, open a terminal and navigate to the directory where the Makefile is and then execute the following command

```
make vba
```

Alternatively you can execute this command to use wxvbam (another GBA emulator)

```
make wxvba
```

Deliverables

main.c
mylib.c
Makefile (your modified version)
and any other files that you chose to implement

Or an archive containing ONLY these files and not a folder that contains these files. DO NOT submit .o files as these are the compiled versions of your .c files

Note: make sure your code compiles with the command

```
make vba
```

Make sure to double check that your program compiles, as **you will receive a zero (0)** if your homework does not compile.

Good luck, and have fun!