

CS2110 Fall 2013

Homework 8

This assignment is due by:

Day: October 23rd, 2013

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. You are also responsible for ensuring that what you turned in is what you meant to turn in. No excuses, 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.

There is a random grace period added to all assignments and the TA who posts the assignment determines it. The grace period will last **at least one hour** and may be up to 6 hours and can end on a 5 minute interval; therefore, you are guaranteed to be able to submit your assignment before 12:55AM and you may have up to 5:55AM. As stated it can end on a 5 minute interval so valid ending times are 1AM, 1:05AM, 1:10AM, etc. **Do not ask us what the grace period is we will not tell you.** *So what you should take from this is not to start assignments on the last day and depend on this grace period past 12:55AM.* There is no late penalty for submitting within the grace period. If you can not submit your assignment on T-Square due to the grace period ending then you will receive a zero, no exceptions.

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.

Submission Conventions

1. **Failure to follow these may result in a max of 5 points taken off**
2. 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.
3. 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.
4. 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).
5. 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.
6. 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. You will need to install several packages: cs2110-tools cs2110-tools-emulator. More information can be found on 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 TA 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 three 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)`
`void drawHollowRect(int r, int c, int width, int height,`
`u16 color)`
`{`
 `// @todo implement :)`
`}`

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!
- **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)`.

In your main function you must satisfy **one** of the following three requirements

1. Draw a picture to the screen using the functions you have defined above (define more if you want!) Your picture must consist of **5** or more colors (black should not be one of the 5 main colors, but you may still use it) and your image must take up at least **66%** (or more) of the screen! Think kindergarten or first grade art here – you are not required to draw a masterpiece.
2. Have an object move (you are not required to accept button input), but unlike option 1 you may draw on **33%** (or more) of the screen. You must still use a minimum of **5** colors.

3. Create a cool animating image or something equally awesome. If you implement a full animation (several frames of animation), you may use 3 or more colors. A few semesters ago, someone made an animating Nyan Cat, and last semester someone made a running Pikachu. Yet another student made an interactive ETCH-A-SKETCH. The idea is for you to make something awesome and to experiment with the GBA environment! If you implement any additional hardware features from the GBA here then you will satisfy this requirement. However your demo must DO SOMETHING and DRAW SOMETHING to the screen. Do not do this option if what you are going to make is not of the caliber of the things I have listed people made last semester! I want to see something cool!

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).
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. Draw something amazing, or implement something really cool! The next assignment you will be making a full game for us to play. The more you do now the better off you will be. Even though we aren't asking much for this assignment, the next assignment will be a little more involved. Also, even though your submission will not be graded on its creative properties, you are still more likely to receive a higher grade or more partial credit if you go above and beyond what is required.

You may research the GBA Hardware on your own. Here are some good things to read if you want to do option 3. You may 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.

A good thing to research how to do on your own is Button Input which is explained here (<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. Lastly if you want to use an image (from the internet) in your submission, you may download and install `brandontools` (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 future assignments.

Note: You will be using buttons, random numbers, and images in the next 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 your 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
```


Debugging

The cs2110- tools-debug package allows you to print things to the terminal from within a gba game. (Remember that the GBA does not have a terminal to print things to).

I have enabled this functionality, by modifying the emulator's source code. To use this functionality here are the steps.

1. `#include <debugging.h>` at the top of any file you need to print something.
2. Use the Macro `DEBUG_PRINTF(format_string, varags)` to print using a format string
3. Use the Macro `DEBUG_PRINT(string)` to just print a string out.
4. Examples. `DEBUG_PRINTF("My age is %d", 25);` `DEBUG_PRINT("HELLO WORLD");`
5. To compile your code with debugging support you must type the following (6-7) in the terminal
6. `make clean`
7. `make debugvba` (or `make debugwxvba`)
8. [IMPORTANT] When you are done debugging type 9. into a terminal
9. `make clean && make vba`
10. Don't forget to do step 8. If you do forget, then you will still be running the debug version.

Likewise, if you fail to type the command in 6 you will still be running the non-debug version.

DO NOT CALL PRINTF DIRECTLY FROM GBA CODE. USE THE MACROS `DEBUG_PRINTF` AND `DEBUG_PRINT`.

Deliverables

main.c
mylib.c
Makefile (your modified version)

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`

ANY code that does not compile will receive a score of 0. No exceptions!