Announcements:

* Quiz on Wednesday. Will cover all material prior to today. It’ll be the first 15 minutes of class.

Notes:

* There are multiple macros and functions that we use repeatedly in our code, and they tend to clutter the code and it’s a pain to rewrite it repeatedly for each new file
* Solution to this is to create an external file that held all of this repetitive stuff and simply import it to use it
  + Importing in C uses the #include
    - #include “fileYouWrote.h”
    - #include <fileCcreatorsWrote.h>
      * For files like stdlib.h, that are from C creators libraries
* The external file used is a .h file (header file) instead of a .c file. This allows you to put prototypes and macros into your main.c file into an external library
  + Created a header file called mylib.h (typical name but you can change it)
    - setPixel, drawRect, delay (common function prototypes can go in)
    - #define REG\_DISPCTL, COLOR, OFFSET, MODE3 (macros)
    - Don’t include definitions for variables or functions in header files
  + Import it in your main file using
    - #include “mylib.h”
    - The preprocessor copies over everything over into the main.c file
  + Don’t put variables like videoBuffer in mylib.c, should go in a header file, only their declaration and not their definition
  + You’re also going to create a myLib.c file that imports myLib.h as well
  + DON’T #include main.c basically ever in anything
    - Will end up creating a sort of circular issue
  + Don’t want to keep executable code in header files
* Handling videoBuffer
  + videoBuffer, when declared in C at the top outside of all the functions, makes it a static variable
* Static and extern
  + Static variables are different in C than they are in Java
  + Variables declared or defined in a function exist throughout the functions lifetime
    - Declared outside makes them static, and they exist throughout the program execution
    - Not dynamically allocated or de-allocated
  + Some rules:
    - Variables with the same name declared outside any function in different files will be viewed by the linker as the same variable
      * Variables with the same name in a function across files will be viewed as different
    - In order to have the linker recognize a variable inside the function and the static variable with the same name outside a function in a different file, use the EXTERN keyword.
      * This tells the compiler that the variable is declared elsewhere
      * Must still declare the variable type:
        + extern int i;
    - If you place the keyword STATIC on a variable that is already declared outside any functions (therefore already static), the scope of the variable will instead be reduced to only existing in its current file
    - Functions have a global scope by default, and declaring them static reduces their scope to existing only in the file its in
    - Adding STATIC to a variable that exists inside a function will make it persist after the function its in finishes running, instead of expiring with the function
      * Any updates or changes to the local static variable will stay
      * Can have a static variable inside a non-static function
* videoBuffer:
  + videoBuffer is a variable that we don’t want to include in the header file (no executable code), so we use a shortcut called TYPEDEF
  + at the top of your myLib.h file:
    - typedef unsigned short u16;
      * u16 is kind of an abbreviation for unsigned short
    - extern u16 \*videoBuffer;
      * this tells the compiler you declared videoBuffer somewhere else
  + initialize videoBuffer in your myLib.c file
    - u16 \*videoBuffer =….
* Flicker:
  + Imagining the GBA screen as a CRT scan
    - The cathode ray is firing electrons at a screen, starting from the top left corner, all the way to the right, then stopping, shifting to the start of the next row and starting again
      * Repeat
      * The pause in repositioning is called horizontal blank
    - Also stops when it reaches the bottom of the screen and stays stopped while it repositions itself at the top left of the screen to begin again
      * Vertical blank
      * Much longer than the horizontal blank
  + Flicker caused by writing to memory at the same time the screen was accessing the memory
    - Can’t do both at once, so one process woul have to stop and let the other finish
  + Fixed this by writing to memory during the vertical blank
  + SCANLINE: is a location in memory that keeps track of and updates the row (scan line) you are in
    - When the vertical blank begins, you start writing to memory
    - Keep track of the scan line
    - Be sure to start as soon as the vertical blank begins to maximize your time in writing to memory
    - Also be sure to write quickly (have efficient code)
    - waitforBlank() function