Announcements:

* Lab01 grades are out
* HW1 is due Thursday
* HW2 released tonight and due next Tuesday
* Make sure your homework 1 is not procedural, instead frame by frame is wanted

Animation in C and GBA:

* Everything needs to be dictated by you in the code, there’s not really any shortcuts
* Animation in Mode 3 works like the whiteboard animations (Draw My Life)
* Draw, erase, draw, erase, etc
* Key issue is slowing it down so you can see it:
  + Programming languages designed to be as fast as possible
  + To slow down, you want to give the comp something to do between frames
  + Like **delay** functions
  + Recall that the compiler will check over functions and variables to see if they are actually important in the program and will trash unimportant things (optimization).

void delay (int amt) {

int t = 0;

for (int i = 0; i<amt\*1000; i++) {

t++;

}

}

* + Function above doesn’t do anything, compiler ignores it in order to optimize your code. It sees that you don’t use that function in your program.
  + VOLATILE keyword tells the compiler not to ignore this and run the function

void delay (int amt) {

**volatile** int t = 0;

For (int I = 0; i<amt\*1000; i++) {

T++;

}

}

* + This will create the delay you want between drawing and erasing
  + Don’t abuse the volatile keyword
* Animating moving objects
  + Draw, then delay, erase, draw, repeat
    - Put delay after the draw so that the user can actually see what you’ve drawn
    - Or erase, draw, delay, repeat (better combo)
  + Find a balance of movement, speed and delay
    - speed DELAY – too slow
    - SPEED delay – too fast
    - speed delay – flicker
    - SPEED DELAY – “low framerate”
      * “low framerate” – smooth and choppy animation. Frequency at which frames are displayed.
  + Want it balanced, need to figure out what it should look like per project.
* Moving Objects with Code
  + Just like animating:
    - Delay, erase, draw, repeat
  + Put erase-draw-delay cycle in a loop
  + Make the drawing parameters into variables
  + Manipulate the variables as you see fit
* C Tips for Better Code
  + Main files get very large and less manageable if we stick everything in there
  + Need better code organization
    - Some code use the functions and macros multiple times
  + Solve this with the mylib.c file
    - Contains the COLOR macros, OFFSET, setPixel, delay, videoBuffer
      * Note that videoBuffer will not go in a header file
    - *Not* the main function, or any functions that aren’t usable for everything (functions should be very general)
  + Make sure to add it to your MakeFile SOURCES line of code, or you won’t get
    - Makefile is what tells what .c files are compiled
    - The files are separated by spaces, not commas
  + Header files:
    - Special files that only the preprocessor touches, not the compiler
    - You can only put “preprocessor things”
      * So no memory affecting code in header files
      * This is why videoBuffer won’t go in the header file (myLib.h)
      * Don’t put like

int a = 7

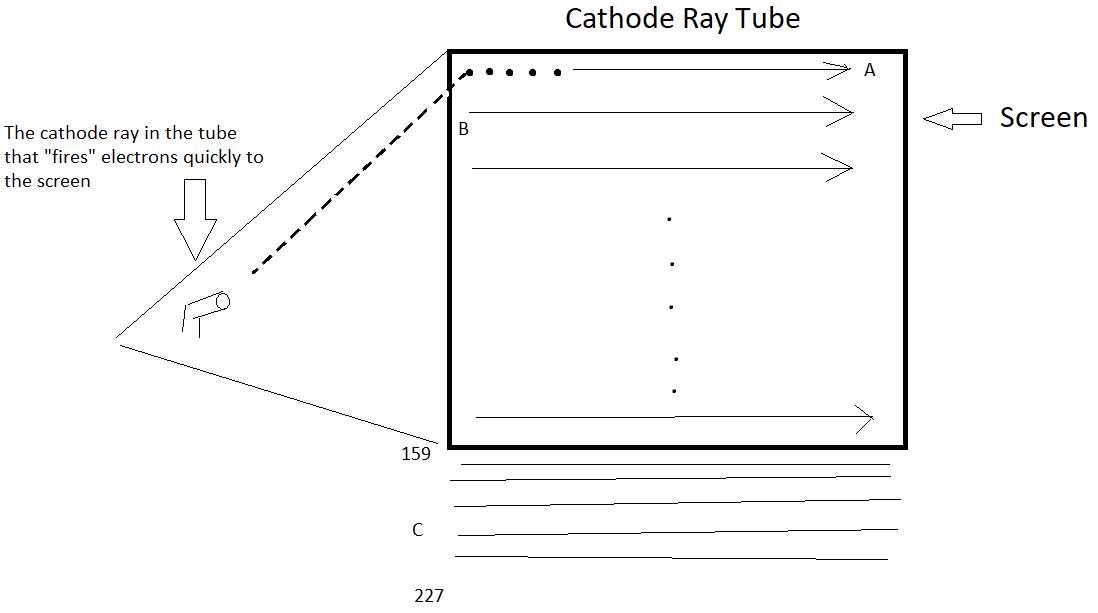
int main () {

drawRect()

}

* + - * But this is fine:
        + int a; (variable declarations OK)
        + macros
        + function prototypes
    - For myLib.c we create myLib.h and copy stuff in
      * Remove this stuff from myLib.c after copying
    - At the top of the myLib.c file, put #include “myLib.h”
      * #include is like the same as importing in Java
      * Preprocessor sees this line and copy pastes all of myLib.h and pastes it into myLIb.c
      * Also put the #include “myLib.h” in the main as well
      * DON’T #INCLUDE A .C FILE
      * Anything you want main.c file, you stick in the myLb.h file
      * MakeFile doesn’t need header files
  + If you are including a .h file from a C library, use <> instead of “”
    - #include <stdlib.h>
      * Tells compiler where to look for this header files
    - Use quotes for any header files you wrote
      * #include “mylib.h”
* Editing header files
  + If you do these:
    - Compile
    - Edit a header file
    - Don’t edit any file that #includes headerFile.h
    - Compile
  + Changes in your header file have no effect
  + The reason is because compiler saves time by not recompiling any files that haven’t been changed, so new header changes won’t show in the .c files because the .c files haven’t been changed themselves
  + Solution
    - make clean before you make build
      * Advantages: always works
      * Disadvantages: takes a while when you have many files
    - Edit the .c files that include it, then undo, save, and compile
      * Advantages: way faster
      * Disadvantages: difficult to find and edit all the files including the header file

|  |  |  |
| --- | --- | --- |
| myLib.h | myLib.c | main.c |
| void setPixel(int, int) | #include “myLib.h”  void setPixel(int a, int b) {  video….(definition)  } | #include “myLib.h”  Void setPixel… |

* + Shouldn’t ever need a main.h file, because its only purpose would be to reference main in other files which u shouldn’t do
* How Display Works
  + The processor sets a location in memory
    - A wire to the memory location sets the value
  + The screen gets its info from memory
    - A wire to the memory location accesses the value
  + These two things cannot happen simultaneously
    - If the screen refreshes ~60fps, and you write constantly, they will conflict often
    - Causes the flicker
  + If they conflict, who wins? Write to memory
    - The processor wins each time
    - After the processor is finished, you display
  + You are writing code for the processor when you write in C, and you can’t really control the screen the way you can by writing to the processor
  + The processor will have to wait, because you have control over the processor.
  + Cathode Ray Tube (CRT) – imagine this is how your GBA screen works, even though it’s not accurate
  + The little cathode ray on the left fires the electrons to the screen from the left to the right of the row, and the time it takes to move from the end of one row to the start of another (from A to B) is called the horizontal blank
  + After reaching the last row (159), it keeps going down till row 227. At the end of row 227, it starts back up again at the top
    - The time here is called the vertical blank
  + Now, the horizontal blank is too short a time to do anything, but the vertical blank is long enough to tell the processor to set locations in memory, because during this time the screen isn’t accessing the memory. This avoids the flicker caused by the conflict of processor setting location in memory and screen accessing the info in the memory.