ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 28

Vince Weaver

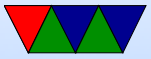
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

13 April 2026

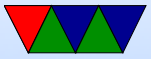
Announcements

- HW#7 not assigned yet
- Working on grading the Midterm
- Senior Project Expo April 22nd



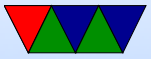
Lab9 – Breakout

- Arcade game by Atari (fairly early: 1976)
- Designed by Nolan Bushnell and Steve Bristow, initial hardware design by Steve Jobs and Steve Wozniak
- Story about Woz optimizing the number of chips down, Atari in end didn't use
- Arcade version monochrome, colored film for color
- One of top 5 grossing games in 1976
- Many clones (there were lawsuits). One famous version is Arkanoid



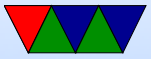
Lab9 – Breakout – Apple II Connection

- Jobs and Wozniak founded Apple on April 1st 1976
- First computer was the Apple I, boring output options
- Apple II, 1977, unlike other systems that year (PET) had color graphics, sound, and came with paddles because Woz wanted it to play breakout in software
- Came with breakout written in BASIC



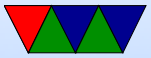
Lab9 – Copying Your Code Over

- Be sure you finish Lab#8 first
- You'll need to copy your `paddle.c` file over from Lab#8 and have it over-write `breakout.c`
- How do you do that?
- Can just copy with the GUI on windows / mac
- If you're on Linux at the command line can use `cp`
- Make sure it's not `breakout.c.c`



Lab9 – Initial Cleanups

- Clean up a few things to make things easier



Lab9 – Split up Struct Definitions

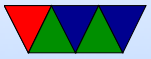
- Instead of initializing your `ball` struct like

```
struct ball_state {
        int x,y,xadd,yadd;
        short color;
} ball;
```

declare it outside of `main()`, before the function prototypes

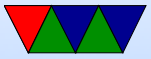
```
struct ball_state {
        int x,y,xadd,yadd;
        short color;
};
```

and in `main()` declare ball as



```
struct ball_state ball;
```

- Do the same for paddle

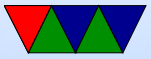


Lab9 – Make Lives Count More Robust

- Change the code so the ball doesn't over-write the Lives count

  Find the code that checks for top of screen, something like `if (ball.y<0)` and change it to `if (ball.y<16)`
- Also find your code in `update_lives()` that erases the text and change it so that it clears a block of size 120,16 instead of 160,10. This is needed so it doesn't over-write the score we add later

```
Adafruit_ST7735_fillRect(0,0,120,16,0x0000);
```



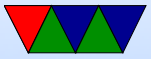
Lab9 – Adding Blocks – Block Array

- Going to have an 8x4 array of blocks
- Use defines

```
#define MAX_BLOCK_X      8
#define MAX_BLOCK_Y      4
```

- integer array, make it a global

```
int blocks[MAX_BLOCK_X][MAX_BLOCK_Y];
```

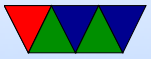


Lab9 – Initializing Blocks

- Put this in a function called `init_blocks()`
  Put prototype at top, function at bottom
- Initialize all elements to 1 meaning block is present.
  Using nested for loops probably easiest way to do this.

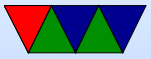
```
for(x=0;x<MAX_BLOCK_X;x++) {
    for(y=...
        blocks[x][y]=1;
    }
}
```

- Add a call to this function before the main `while()` loop with the other initialization code.



Lab9 – Adding Blocks – Drawing Initial

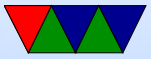
- Put this in a function called `draw_blocks()`
  Put prototype at top, function at bottom
- Draw all blocks using nested loops. 8x4 grid. Each block is 30x16, but we draw 29x15 wide to leave a gap
- What colors should they be?
- `Adafruit_ST7735_fillRect()`, remember takes `x`, `y` of upper left, then `width`,`height`, then `color`
- Where should these start at the screen? Define `BLOCK_OFFSET` set to 48 for now and use it where



appropriate

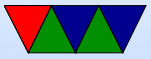
```
Adafruit_ST7735_fillRect(
    x*30,                    // every 30 pixels
    BLOCK_OFFSET+(y*16), // 16 pixels tall
    29,15,                   // leave room for edge
    0xff<<(y/2));            // blue gradient color
```

- Add a call to this function with the other init routines
- TODO: diagram?



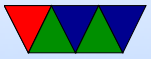
Lab9 – Adding Blocks – Erasing Block

- Put this in a function called `void erase_block(int x, int y);`
  Put prototype at top, function at bottom
- This will be used when a block is hit
- This should be easy to make if you have working `draw_blocks()` code, it should just be the inner fillRect code but with the color 0 (black)



Lab9 – Writing Score

- Define an integer named `score` to hold the score value
- Initialize this to 0 in main where you init other variables
- Create an `update_score()` function. Predeclare it at top, put code at bottom. It should take one parameter, an integer named `score` and return nothing.
- The code here is going to be very similar to that in `update_lives()`
- Create a string that contains "SCORE: NNN" where instead of N you have the score



- Use `sprintf()` for this. Something like:

```
char string[100];
sprintf(string,"SCORE: %03d",lives);
```

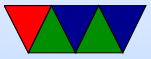
- The 03d means to print the score as 3 digits, and include leading zeros
- As with `update_lives()`, erase the old score before printing the new one

```
Adafruit_ST7735_fillRect(120,0,120,16,0x0000);
```

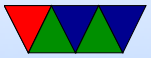
- Use the function

```
size_t graphics_drawText(char *s, uint16_t x,
                         uint16_t y);
```

to draw this string at location 120,0

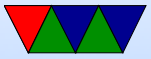


- Be sure to call `update_score()` once before the main while() loop to print lives left at start of game



Lab9 – Block Collision

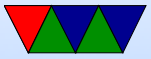
- This is complex to do properly
- We are just going to do a simple check to see if it's hitting a block from above or below. Properly we should check if hitting from the side too.
- Create a `block_collide()` function. It should return nothing, but should take a pointer to a `struct ball_type *ball` as a parameter.
- In this function, check each block in the block array in turn to see if it's colliding with the ball



- Have a nested for loops like before. Your detection code might look something like this: it first checks to see if the block is still there, then it compares to see if the block overlaps with the ball. Note we passed the ball in as a struct pointer so we need to use the -> operator.

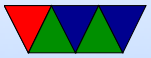
```
int block_x,block_y;
// nested x,y for loops
block_x=x*30;
block_y=BLOCK_OFFSET+y*16;

if (blocks[x][y]) {
   if ((ball->x >= block_x) &&
      ((ball->x+XSIZE)<= block_x+29) &&
      ((ball->y+YSIZE)>=block_y) &&
      (ball->y<=block_y+15)) {
```



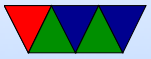
```
            // we collided
        }
    }
```

- Call this function after move ball but before move paddle



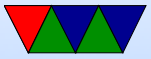
Lab9 – What to do if Block Collision

- Add 10 to the score
- Run `update_score(score);`
- Mark the block we collided with as being destroyed (so set the proper `blocks[][]` value to 0
- Call `erase_block()` with our `x` and `y` values to erase it from the screen
- Negate `ball->yadd` so the ball bounces off the block
- Call `play_tone()` to make a sound effect

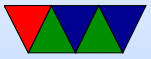


Lab9 – Detecting Blocks All Clear

- Create a function called `detect_win()` it should take no arguments, but return an integer: `1` if all the blocks are gone, `0` otherwise
- Remember to put a function prototype for this at the top with the others.
- Have a local integer called `blocks_left` and set it to 0
- Then loop `x` from 0 to `MAX_BLOCK_X` and `y` from 0 to `MAX_BLOCK_Y`, adding the value of `blocks[x][y]` to `blocks_left`

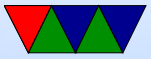


- After the loops end, check the `blocks_left` value. If it's 0 (no blocks left) `return 1`, if it's nonzero, `return 0`.



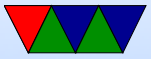
Lab9 – Handling All Clear

- Add a call to this function as the last thing in the main game `while(1)` loop.
- Check the result, and if it is 1 then do the following:
  - print "YOU WIN!"
  - print `SCORE: %03d` and pass in the score value to print your score
  - Wait 1 second using the `sleep_ms()` function
  - Run `break;` to break out of the while loop



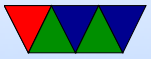
Lab9 – Play Again

- Add code so that instead of return/exit when count goes down to 0, does a break; to exit out of the infinite while loop
- After the while loop, before the return 0, add code that prints a newline, "PRESS ANY KEY", newline, then "TO PLAY AGAIN" then newline
- Then have a do/while loop that runs `ch=read\_keypad()` while `ch==-1`
- Why a do-while not a while? Because we only want it



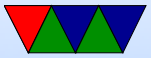
to run at least once or else ch isn't set

- After this exits, use a goto to restart level

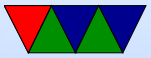


Lab9 – Restart Level

- Put the `restart_level:` before the beginning of the main while(1) loop. The following things need to be *after* `restart_level:` but before the while(1) loop
  - The code that sets the initial ball position
  - The code that sets the initial paddle position
  - Code to set lives to 3
  - Code to set score to 0
  - Code that clears the screen
  - The call to `init_blocks()`

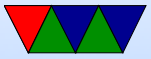


- The call to `draw_blocks()`
- The call to `update_lives()`
- The call to `update_score()`

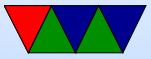


Lab9 – Something Cool

- Have the ball be a different shape
- Have more advanced ball physics (hitting off-center on the paddle change the x-speed, hitting the ball while the paddle is moving also changes x-speed)
- Collision detect the ball on the left/right side of bricks not just top/bottom
- Track the high score
- Have bonuses that change the size of the paddle or ball
- Have different sound effects depending what you hit



- Have music that plays when you finish a level or die
- Have a background graphics rather than just a plain black background
- Turn the LED bargraph on and do something with that (maybe a percentage of how many blocks are left)
- Have a button "grab" the ball, then you can move it and release it from a new location
- Have multiple levels. Block pattern can be different, speed of ball can increase.



Lab9 – Note on Code Comments

- You'll need to copy your `paddle.c` file over from Lab#8 and have it over-write `breakout.c`
- How do you do that?
- Can just copy with the GUI on windows / mac
- If you're on Linux at the command line can use `cp`