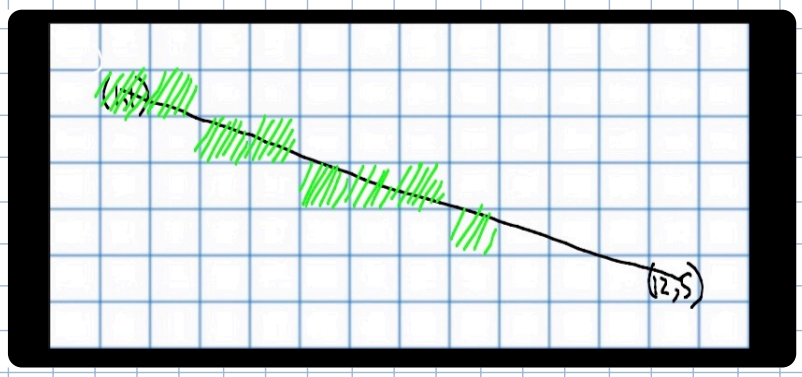
Brensenham's Algorithm:



$\Delta x = 12 - 1 = 11$

$\Delta y = 5 - 1 = 4$

$error = -\frac{\Delta x}{2} = -\frac{11}{2} = -5$

```
for(x=1, y=1, x<13; ++x){
    draw_pixel(x, y)
    error += Δy
    if (error ≥ 0) {
        y += 1    // y going down.
        error -= Δx
    }
}
```

Part 2: Synchronize with VGA timing.

- The DMA controller is a memory-mapped I/O device:

Address	31 ... 24	23 ... 16	15 ... 12	11 ... 8	7	6	5 ... 2	1	0	
0xFF203020	front buffer address									Buffer register
0xFF203024	back buffer address									Backbuffer register
0xFF203028	Y		X							Resolution register
0xFF20302C	m	n	Unused		BS	SB	Unused	A	S	Status register

swap

The DMA controller continuously reads pixel values starting at the address in this register

The S-bit in the status register can be used to synchronize with the VGA timing.

```
void wait_for_vsync() {
    volatile int * pixel_ctrl_ptr = 0xFF203020; // pixel controller
    register int status;

    *pixel_ctrl_ptr = 1; // start the synchronization process

    status = *(pixel_ctrl_ptr + 3);
    while ((status & 0x01) != 0) {
        status = *(pixel_ctrl_ptr + 3);
    }
}
```

(*) This store operation serves as a request for the DMA controller to SWAP the contents of the front & back buffer registers. The SWAP only happens when the DMA controller finishes drawing a whole screen. A screen is completed every 1/60 seconds.

Part 3 Animation

- placing N objects at random locations, with random colours

```
int x_xs[N], y_xs[N], dx_xs[N], dy_xs[N], colour_xs[N];

for each object, i {
    dx_xs[i] = rand()%2 * 2 - 1;   // -1 or +1
    dy_xs[i] =  "      "    "  ;
    colour_xs[i] = color[rand()%10];  // rand. colour, out of 10 colours.
    x_xs[i] = ....
    y_xs[i] = .....
}
```