```
// we added this initially to test whether the linking stage was working
// now everything breaks if we remove it.
// I think Stephen might have done something to unbreak it.
short functionThatKeepsStuffFromBreaking(short x) {
    return 5 + 3;
}


extern short cur_pal_offset;

void pal_counter(){
    cur_pal_offset ++;
    cur_pal_offset = cur_pal_offset % 256;
}

// function that sets a pixel indicated by x and y to value, as defined in the pallette
short setPixel(short x, short y, short value) {
    // convert the x and y into a linear index in memory
    short pos = (320 * y + x);

    __asm {
        // we're going to use these registers, and I'm not sure enough
        // what we're allowed to clobber, so just save and restore everything
        // pusha doesn't work because Watcom is stupid
        push    ax
        push    di
        push    dx


        mov     ax, 0xA000
        mov     di, pos // that variable we computed above

        // don't clobber es
        push    es
        // set es=0xA000
        push    ax
        pop     es

        // this magic incantation that shows the pixel on the screen
        mov     ax, value
        stosb

        //restore everything
        pop     es

        pop     dx
        pop     di
        pop     ax
    }
    return 0;
}

// function that moves the specified block one pixel to the right, wrapping around at
the end
void moveBlock(short curPos, short yPos) {
    // black out the retreating left edge
    short x = curPos + 160;
    short y = yPos;
    for (; y < yPos + 10; y ++) {
        setPixel(x, y, 0); // black in Stephen's pallette
    }

    // white out the advancing right edge
    x = curPos + 11;
    x = (x % 160);
    x += 160;

    y = yPos;
    for (; y < yPos + 10; y ++) {
```

```
69              setPixel(x, y, 193); // white in Stephen's pallette
70          }
71      }
72
73      // hold the current horizontal positions of the blocks.
74      // block 0 never gets used
75      extern short currPos0;
76      extern short currPos1;
77      extern short currPos2;
78      // short currPos0 = 0;
79      // short currPos1 = 60;
80      // short currPos2 = 120;
81
82      void moveBlock0() {
83          moveBlock(currPos0, 50);
84          currPos0 ++;
85          currPos0 = currPos0 % 160;
86      }
87
88      // wrapper for moveBlock for block 1
89      // I never could figure out Watcom's calling convention
90      void moveBlock1() {
91          moveBlock(currPos1, 100);
92          currPos1 ++;
93          currPos1 = currPos1 % 160;
94      }
95
96      // wrapper for moveBlock for block 2
97      void moveBlock2() {
98          moveBlock(currPos2, 150);
99          currPos2 ++;
100         currPos2 = currPos2 % 160;
101     }
102
```