1. Write a function that takes two integer arguments and returns the one whose absolute value is larger.

In C, its declaration would be

```c
int absmin(int a, int b);
```

Your implementation must adhere to the MIPS calling conventions.

```assembly
absmin:
    abs    $t0, $a0    # Compute absolute value of a
    abs    $t1, $a1    # Compute absolute value of b
    bgt    $t0, $t1, L1
    move   $v0, $a1    # abs(b) was larger; return b
    jr      $ra
L1:
    move   $v0, $a0    # abs(a) was larger; return a
    jr      $ra
```
2. Write a function that draws a horizontal line in a text “framebuffer” consisting of 32 lines of 63 characters (plus a newline). The address of a character in the framebuffer is \( \text{fb} + 64 \times y + x \).

In C, its declaration would be

\[
\text{void horiz(int x0, int x1, int y);}\\
\]

Your implementation must adhere to the MIPS calling conventions.

```
horiz:  
    \text{la \$t0, fb} 
    \text{addu \$t0, \$t0, \$a0 \# + x} 
    \text{sll \$a2, \$a2, 6 \# y \times 64} 
    \text{addu \$t0, \$t0, \$a2 \# + y \times 64} 
    \text{subu \$a1, \$a1, \$a0 \# x1 - x0} 
    \text{li \$t1, ' + '} 
    \text{b L2} 
    \text{L1: \ addiu \$t0, \$t0, 1 \# Next column} 
    \text{addiu \$a1, \$a1, -1 \# decrease count (x1-x0)} 
    \text{L2: \ sb \$t1, ($t0)} 
    \text{bne \$a1, \$0, L1} 
    \text{jr \$ra} 
```
3. Write a function that draws arbitrary lines in a text “framebuffer.” In C, its declaration would be

```c
void line(int x0, int y0, int x1, int y1);
```

Your implementation must adhere to the MIPS calling conventions.
```c
#define SWAP(a, b, c) ((c) = (a), (a) = (b), (b) = (c))
char fb[];
int abs(int x) { return x < 0 ? -x : x; }
void plot(int x, int y) { fb[x + 64 * y] = '*'; }
void line(int x0, int y0, int x1, int y1)
{
    int ystep, err;
    int dx = abs(x1 - x0); int dy = abs(y1 - y0);
    int steep = dx < dy;
    if (steep) { SWAP(x0, y0, err); SWAP(x1, y1, err); }
    if (x1 < x0) { SWAP(x0, x1, err); SWAP(y0, y1, err); }
    dx = x1 - x0; dy = abs(y1 - y0);
    ystep = y0 < y1 ? 1 : -1;
    err = dx >> 1;
    for (;;) {
        if (steep) plot(y0, x0); else plot(x0, y0);
        if (x0 == x1) break;
        x0++;
        err -= dy;
        if (err < 0) { y0 += ystep; err += dx; }
    }
}
```
# x0 and y0, starting coordinates, in $a0 and $a1
# x1 and y1, ending coordinates, in $a2 and $a3
# $t0 dx   $t1 dy   $t2 steep   $t3 ystep
# $t4 err   $t5 address   $t7 '*'

line:    subu  $t0, $a2, $a0
         abs  $t0, $t0  # dx = abs(x1 - x0)

         subu  $t1, $a3, $a1
         abs  $t1, $t1  # dy = abs(y1 - y0)

         slt  $t2, $t0, $t1  # steep = dx < dy
         beq  $t2, $0, L1  # if (steep)

         move $t4, $a0  # SWAP(x0, y0, err)
         move $a0, $a1
         move $a1, $t4
         move $t4, $a2  # SWAP(x1, y1, err)
         move $a2, $a3
         move $a3, $t4
L1:  slt  $t4, $a2, $a0  # x1 < x0
    beq  $t4, $0, L2  # if (x0 > x1)
    move  $t4, $a0  # SWAP(x0, x1, err)
    move  $a0, $a2
    move  $a2, $t4
    move  $t4, $a1  # SWAP(y0, y1, err)
    move  $a1, $a3
    move  $a3, $t4

L2:
    subu  $t0, $a2, $a0  # dx = x1 - x0
    subu  $t1, $a3, $a1  # dy = abs(y1 - y0)
    abs  $t1, $t1
    slt  $t3, $a1, $a3
    sll  $t3, $t3, 1
    addiu  $t3, $t3, -1  # ystep = (y0 < y1) * 2 - 1
    sra  $t4, $t0, 1  # err = dx >> 1
    li  $t7, '*'
    b  L3
innerloop:
    addiu $a0, $a0, 1  # x0++
    subu $t4, $t4, $t1  # err -= dy
    bgez $t4, L3
    addu $a1, $a1, $t3  # y0 += ystep
    addu $t4, $t4, $t0  # err += dx
L3:
    la $t5, fb  # fb
    beq $t2, $0, notsteep  # steep?
    addu $t5, $t5, $a1  # + y0
    sll $t6, $a0, 6  # x0 * 64
    addu $t5, $t5, $t6
    sb $t7, 0($t5)
    b next
notsteep:
    addu $t5, $t5, $a0  # + x0
    sll $t6, $a1, 6  # y0 * 64
    addu $t5, $t5, $t6
    sb $t7, 0($t5)
next:
    bne $a0, $a2, innerloop
    jr $ra