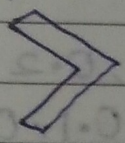
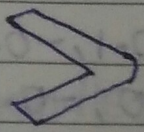
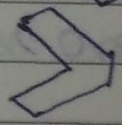
Joins:

To give good finishing on joining 2 lines

(1) Mitter join 

(2) Round join 

(3) Bevel join 

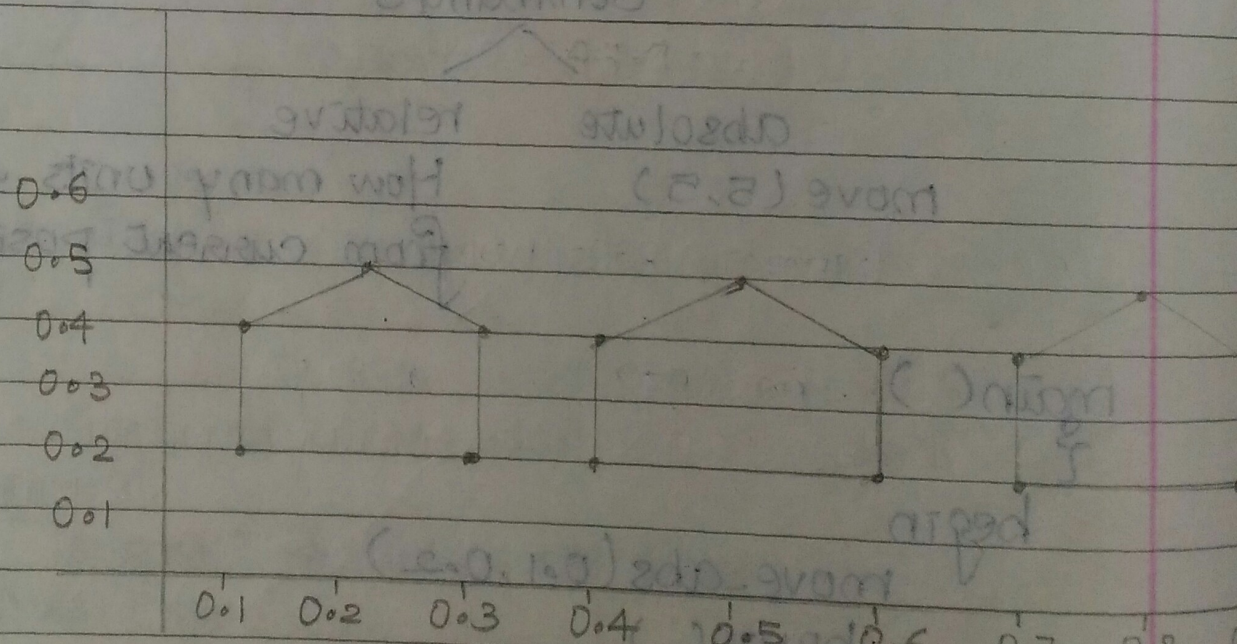
Commands

absolute — move(5,5)

relative — How many units to be moved from current position.

```
main( )
{
  begin
      move_abs(0.1,0.2)
       house( )
      move_abs(0.4,0.2)
       house( )
      move_abs(0.7,0.2)
       house( )
  end
}
```

```
Subprogram house( )
{
    begin
        line_rel(0, 0.2)
        line_rel(0.1, 0.1)
        line_rel(0.1, -0.1)
        line_rel(0, -0.2)
        line_rel(-0.2, 0)
    end
}
```



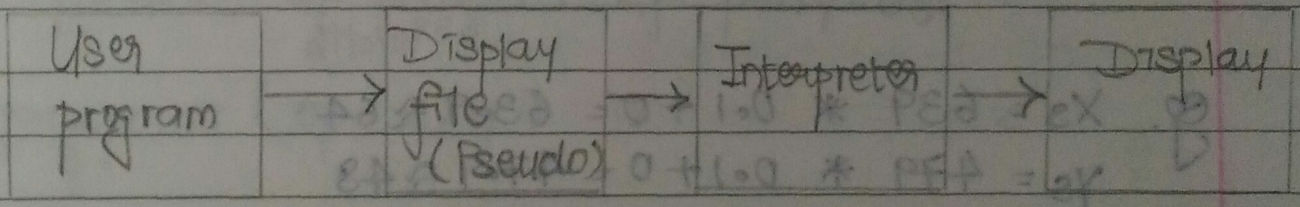
Starting position : (0.1, 0.2), (0.4, 0.2), (0.7, 0.2)

0.1, 0.4	0.4, 0.4	0.7, 0.4
0.2, 0.5	0.5, 0.5	0.8, 0.5
0.3, 0.4	0.6, 0.4	0.9, 0.4
0.3, 0.2	0.6, 0.2	0.9, 0.2
0.1, 0.2	0.4, 0.2	0.7, 0.2

Display file	Frame Buffer.
→ Commands	Pixel wise
→ Editing is easy	Hard
→ Less space	More space.

Display file:

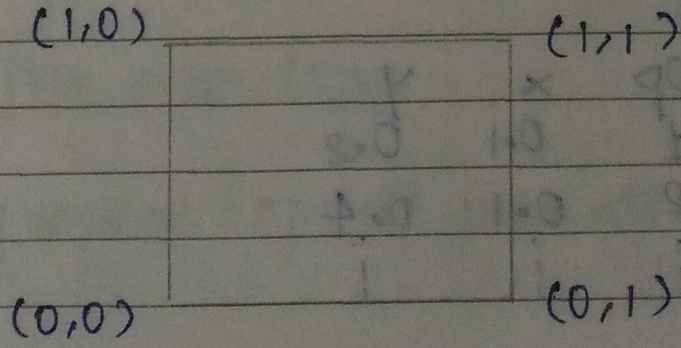
Virtual file handled by the user. It acts as an interpreter so called display file interpreter.



Interpreter:

Interface between user program & display. It takes instruction from display file & convert into machine understandable form, gives O/p to display.

Normalized co-ordinates: (Xn, Yn)

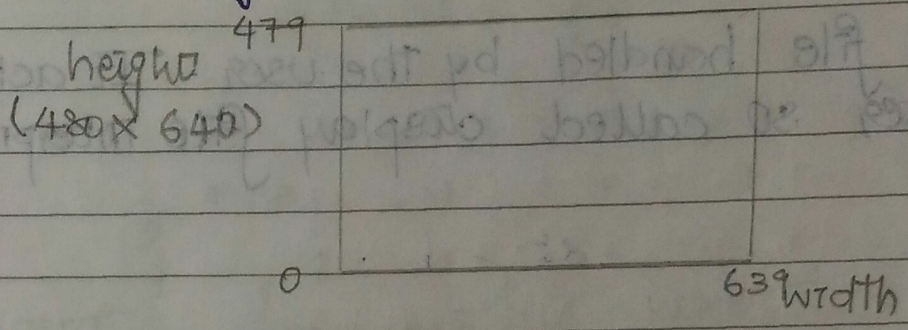


Screen Co-ordinates:

To convert normalized to screen co-ordinates.

x_s = width * x_n + widthstart

y_s = height * y_n + heightstart



eg. $x_s = 639 * 0.1 + 0 = 63.9 \approx 64$

$y_s = 479 * 0.1 + 0 = 47.9 \approx 48.$

Display file / command

opcode — what actions you are doing.

operands — (x, y) value

Display file:

Op	x	y
1	0.1	0.2
2	0.1	0.4
⋮	⋮	⋮

→ free - indicates which position in array is empty.
→ df-penx, df-peny - tells the position of penptr or penpositions of x & y resp/current pen position.
→ display-file-enter - cmd is ready to be entered in display file.

```
const df_size = 50;
df_op[df_size], df_x[df_size], df_y[df_size];
free=1, df_penx=0, df_peny=0;

Putpoint (op, x, y)
{
    If free > df_size then
        Error "Display file full"

    df_op[free] = op
    df_x[free] = x
    df_y[free] = y
        free++
}

display-file-enter (op)
{
    putpoint (op, df-penx, df-peny)
}
```

```
move_abs(x,y)
{
    df_penx = x
    df_peny = y
    display_file_enter(1)
}

move_rel(dx,dy)
{
    df_penx += dx
    df_peny += dy
    dis_file_enter(1)
}

line_abs(x,y)
{
    df_penx = x
    df_peny = y
    dis_file_enter(2)
}

line_rel(dx,dy)
{
    df_penx += dx
    df_peny += dy
    dis_file_enter(2)
}
```

```
interpret(start, count)
{
    for(n= start to count)
    {
        getpoint(n, op, x, y);
        if(op==1)
            domove(x, y);
        else if(op==2)
            doline(x, y);
        else
            Error "Wrong opcode"
    }
}

getpoint(n, op, x, y)
{
    op= df_op[n];
    x= df_x[n];
    y=df_y[n];
}

domove(x, y)
{
Global variables: frame_penx, frame_peny;
    frame_penx = max(ws, min(we, x * w + ws))
    frame_peny = max(hs, min(he, y * h + hs))
}

where, ws, we - width start & end
       hs, he - height start & end
```

→ frame_penx, frame_peny are the penpositions of screen co-ordinates

```
doline(x,y)
 {
    x1 = frame_penx
    y1 = frame_peny
frame_penx = max( ws, min( we, x * w+ws))
frame_peny = max( hs, min(he, y * h + hs));
    bresenham( int (x1+0.5), int (y1+0.5), int
frame_penx+0.5), int (frame_peny +0.5));
 }

new_frame( )
 {
Global: erase_flag = 1
 }

make_picture_current( )
    {
        if( erase_flag == = 1 ) then
                erase( );
        erase_flag = 0;
          if( free > 1 ) then
            {
                interpret (1, (free-1))
            }
          free = 1
    }
```

```
erase( )
 {
   for(i=1; i<640; i++)
       {
          for(j=1; j<480; j++)
              pp(i,j,0);
       }
 }

initialize( )
 {
   free=1, df_penx=0, df_peny=0
   frame_penx=0, frame_peny=0, hs=0, he=479
    h=he-hs, ws=0, we=639, w=we-ws.
 }

main( )
 {
    initgraph( );
    initialize( );
    new_frame( );
     move_abs( );
     house( );
       make_picture_current( );
 }
```