Byte-Drawer Challenge

The Problem

Assume that we've been hired by a client to develop a system that uses data values encoded in the format that we outlined in so if you were given text like

4000 0000 7f7f

...your code will be able to correctly convert it into the corresponding list of integer values:

[0, -8192, 8191]

Where each byte is represented as its converted from a hexadecimal within to a signed integer.

The data we're working with will describe a set of simple commands to build a vector-based drawing system. This system uses a pen-based model where an imaginary pen can be raised or lowered. If the pen is moved while it is down, we draw along the line of motion in the current color. If the pen is moved while it is up, no drawing is done.

The commands supported in this mini-language are:
- Clear the drawing and reset all parameters
- Raise/lower the pen
- Change the current color
- Move the pen

In this system, commands are represented in the data stream by a single (un-encoded) opcode byte that can be identified by always having its most significant bit set, followed by zero or more bytes containing encoded data values.

Any unrecognized commands encountered in an input stream should be ignored.

Generating Output
For the purposes of this challenge, we'd rather avoid the complications of actually generating graphics output. Instead, we'll just define a simple text output that shows that each command in the data stream is being interpreted correctly. The format for the output of each command type is shown here as the last row in each table below.
Commands

Clear

Command	`CLR`
Opcode	`F0`
Parameters	(none)
Output	`CLR;\n`

Cause the drawing to reset itself, including:
• setting the pen state to up
• setting the pen position back to (0,0)
• setting the current color to (0, 0, 0, 255) (black)
• clearing any output on the screen (not shown in our example output here)
Pen Up/Down

Change the state of the pen object to either up or down. When a pen is up, moving it leaves no trace on the drawing. When the pen is down and moves, it draws a line in the current color.

Command	`PEN`
Opcode	`80`
Parameters	0 = pen up any other value = pen down
Output	either `PEN UP;\n` or `PEN DOWN;\n`

Set Color

Command	`CO`
Opcode	`A0`

Parameters	`RGBA` Red, Green, Blue, Alpha values, each in the range 0..255. All four values are required.
Output	`CO {r} {g} {b} {a};\n` (where each of the r/g/b/a values are formatted as integers 0..255)

Change the current color (including alpha) of the pen. The color change takes effect the next time the pen is moved. After clearing a drawing with the CLR; command, the current color is reset to black (0,0,0,255).

Move Pen

Command	`MV`
Opcode	`C0`
Parameters	`dx0 dy0` *`[dx1 dy1 .. dx`$_n$ `dy`$_n$`]`* Any number of (dx,dy) pairs.
Output	if pen is up, move the pen to the final coordinate position as defined below If pen is down: `MV (x`$_o$`, y`$_0$`) (x`$_1$`, y`$_1$`)` *`[... (x`$_n$`, y`$_n$`)]`*`;\n`

Change the location of the pen relative to its current location. If the pen is down, draw in the current color. If multiple (x, y) points are provided as parameters, each point moved to becomes the new current location in turn.

Also note that the values used in your output should be absolute coordinates in the drawing space, not the relative coordinates used in the encoded movement commands.

For example, after clearing a drawing, the current location is the origin at (0, 0). If the pen is moved (10, 10) and then (5, -5), the final coordinate position of the pen will be at (15, 5). For the purposes of this exercise, the string output would be

MV (10, 10) (15, 5);

if the pen is down, but just

MV (15, 5);

if the pen is up. If the specified motion takes the pen outside the allowed bounds of (-8192, -8192) .. (8191, 8191), the pen should move until it crosses that boundary and then lift. When additional movement commands bring the

pen back into the valid coordinate space, the pen should be placed down at the boundary and draw to the next position in the data file.

Examples

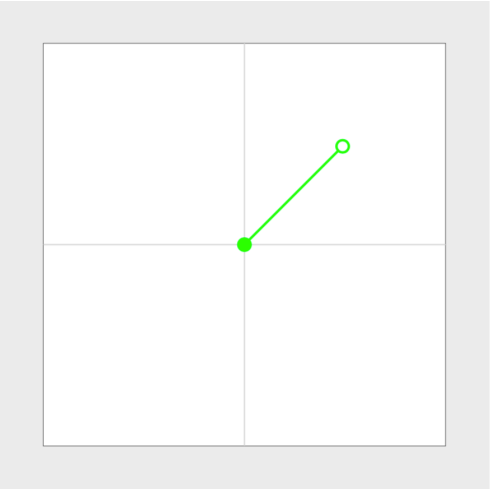
Simple Green Line

Input stream: F0A04000417F4000417FC040004000804001C05F205F20804000

Set color to green, draw a line from (0,0) to (4000, 4000). Filled circle in this diagram indicates pen down position, empty circle indicates pen up position.

Output:

CLR;
CO 0 255 0 255;
MV (0, 0);
PEN DOWN;
MV (4000, 4000);
PEN UP;

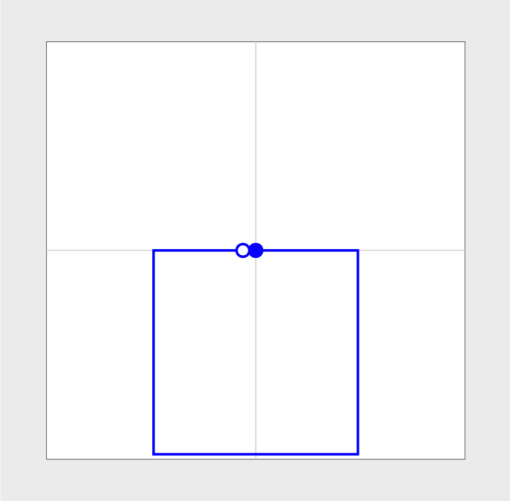


Blue Square

Input stream:
F0A040004000417F417FC040004000904000047684F5057384000804001C0
5F204000400001400140400040007E405B2C4000804000

Output:

CLR;
CO 0 0 255 255;
MV (0, 0);
PEN DOWN;
MV (4000, 0) (4000, -8000) (-4000, -8000) (-4000, 0) (-500, 0);
PEN UP;



Clipping at the Edge of the Drawing

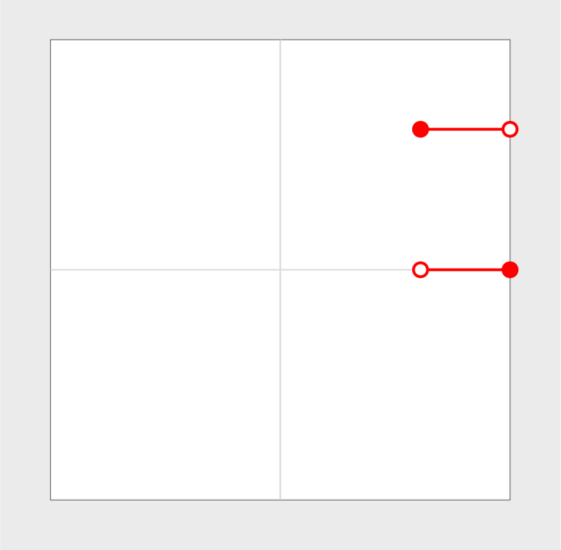
When the drawing stays inside the boundaries of the coordinate space we're working with, things are simple -- there's a 1:1 correspondence between the commands in the input and the output to be generated by your program. Since the movement commands in the input stream are each relative to the current location of the pen, it's possible for that current position to move outside of the valid coordinates, and your program will need to include some special logic to handle that situation intelligently.

Input stream:

F0A0417F40004000417FC067086708804001C06708400040001878187840008
04000

Output:

CLR;
CO 255 0 0 255;
MV (5000, 5000);
PEN DOWN;
MV (8191, 5000);
PEN UP;
MV (8191, 0);
PEN DOWN;
MV (5000, 0);
PEN UP;



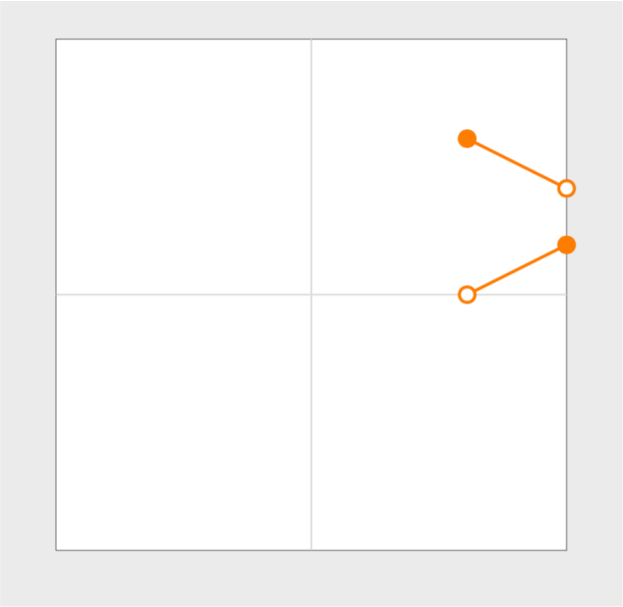
Note that when the pen moves outside of the legal coordinate space, we lift the pen where the line goes out of bounds and then move to the point where the line re-enters our space before putting the pen back down again.

Diagonal Clipping

Input stream:
F0A0417F41004000417FC067086708804001C067082C3C18782C3C804000

Output:

CLR;
CO 255 128 0 255;
MV (5000, 5000);
PEN DOWN;
MV (8191, 3405);
PEN UP;
MV (8191, 1596);
PEN DOWN;
MV (5000, 0);
PEN UP;



Your Task

Supply source code (and whatever additional collateral files may be useful or needed for us to validate and test) a working program that accepts a string of hexadecimal data encoded as outlined in this document and and converts those commands into to a block of text that follows the text format detailed above.

This program can be a command line application that reads from stdin or a file and writes to stdout or a file, or could be a web page or mobile/desktop app that uses a text edit box to accept input values by typing or pasting (or loading a file) and a button to convert that input into the desired output format.

Your goal here isn't just to write code that does the barest of minimum work to barf out a correct answer; we're looking to get a taste of what your habits and instincts are for writing production-quality code. Remember that the goal here is to help us see how awesome your code can be when you're working on a problem where none of those libraries exist yet.