ECOAR lab – MIPS project summer 2018/2019

The black and white image is encoded using one bit per pixel. The highest bit in the byte corresponds to the leftmost pixel in the image. The image line is aligned in the file to the nearest multiple of 4 bytes. In the black and white image we have following dependencies:
Number of bytes containing image pixels = (image width + 7)/8
Number of bytes in line (in file) = (((image width + 7)/8 + 3)/4) * 4

Information about image is held in the imgInfo structure:

```
struct {
      int w, h;   // width and height of image
      unsigned char* pImg; // pointer to the image buffer
} imgInfo;
```

(Note that you can add additional fields in this structure).

Implement function:

```
void setPattern(imgInfo* pImg, int xy, int size, int mode);
```

where

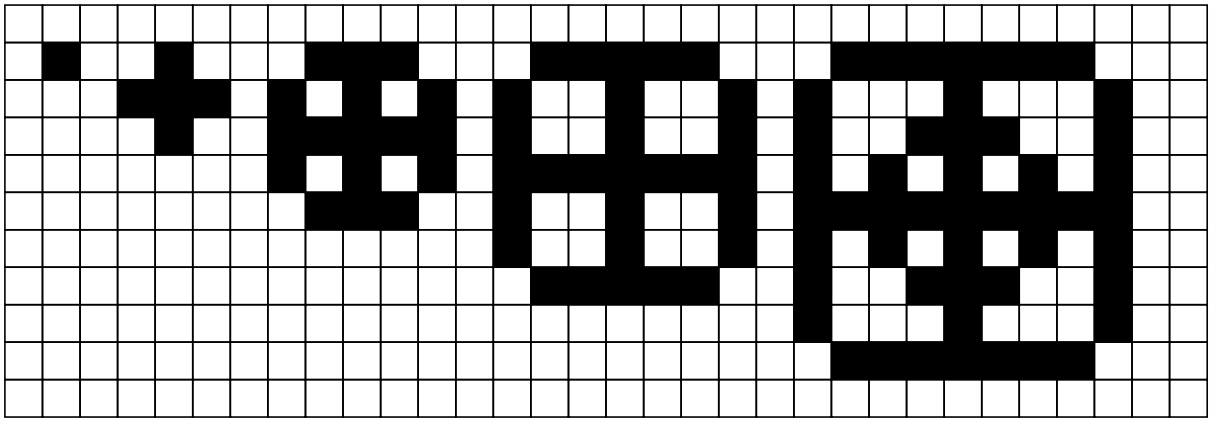
pImg is a pointer to the imgInfo structure with image descriptor

xy are coordinates of upper left corner of pattern to be written to the image (x is stored in upper 16 bits and y in lower 16 bits of xy argument)

size is the size of the pattern to be drawn

mode is (almost) colour of the patter (0 – black, 1 – white, 3 – “inverting” pattern)

The patterns for several first sizes are depicted below:



This should be effect of the following sequence of calls:

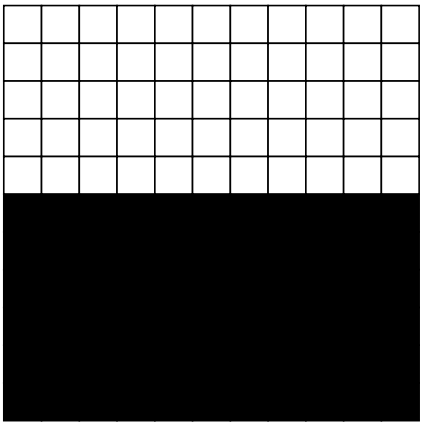
```
setPattern(pImg, 0x10001, 1, 0);
setPattern(pImg, 0x30001, 2, 0);
setPattern(pImg, 0x70001, 3, 0);
setPattern(pImg, 0xD0001, 4, 0);
setPattern(pImg, 0x150001, 5, 0);
```

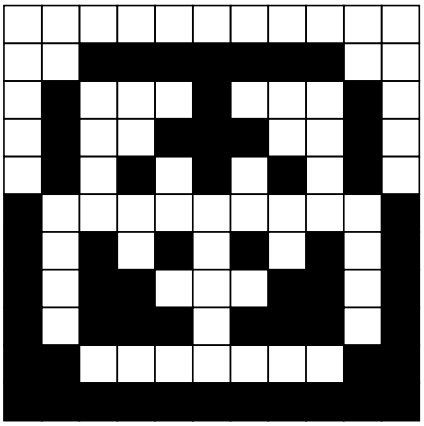
under assumption that pImg contains pointer to 32 x 11 pixels black and white image.
In case of the black image of the same size we should call function with the last parameter set to 1.
The final image should be inverted version of the above image (white patterns on black background).

Inverting is means that “active” pattern pixels (these displayed in black above) cause the inversion of the original image pixel. If the original image is 11 x 11 pixels with upper part white and lower black (on the left below), applying function:

```
setPattern(pImg, 0x10001, 5, 3);
```

Should result in image shown on the right below.





Your program should read image from file (filename can be fixed in the code) draw several patterns in the image and save the resulting image (in exactly the same format) under different filename (also fixed). The resulting image should convince me that your function operates as intended.
Tip: it makes sense to implement additional function which displays image in the console (each pixel as individual character). You’ll be able to check quickly the result of your function.

During grading I will take into account optimization of memory accesses. You should minimize (or at least try to) reading from and writing to memory. This means that you should aggregate pixels that need to be changed within a byte.

On April the 16th I plan small workshop on reading, writing, modifying and displaying images.