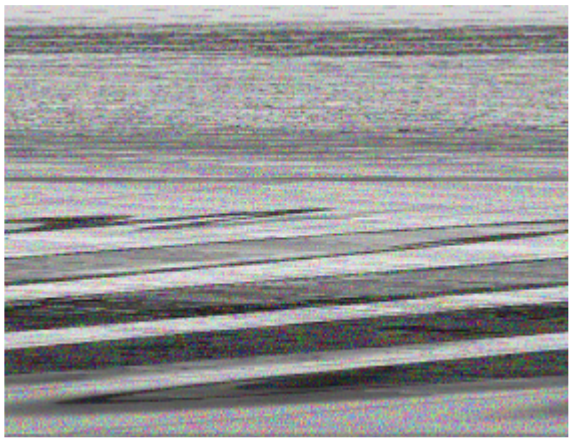
Task 4. Blended Images

Available marks: 23

You have intercepted a file sent from a suspected terrorist cell to their headquarters. It contains an image in PPM format (shown here reduced in size), which appears to be just stripy noise. It is 706 pixels wide and 540 pixels high.

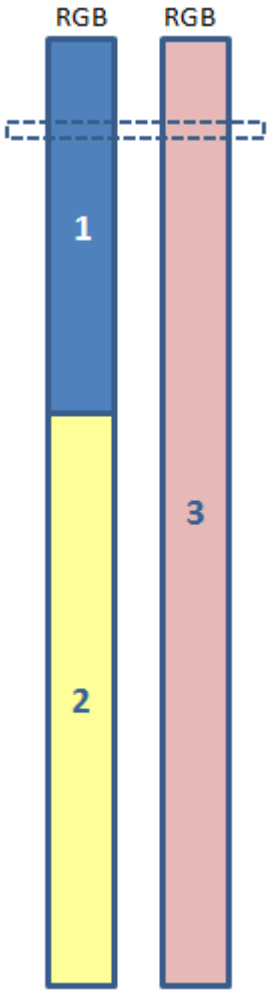


Secret intelligence has yielded some information about the file. It contains three separate images with the following characteristics:

- All images have a landscape orientation.
- All images use 8-bit colourants.
- The sum of the number of pixels in Image 1 (a x b pixels) and the number of pixels in Image 2 (c x d pixels) is equal to the number of pixels in Image 3 (e x f pixels).
- a, b, c, d and e are all 3-digit primes.
- $a < b < c < d < e$

The way the image pixel data streams have been encoded to prevent casual snooping is this:

- Image 1 and 2 data streams are concatenated, and lined up with the data stream for Image 3.
- Each adjacent pair of pixels has 6 colourants in all, R G B from the left (Image 1 or 2) and R G B from the right (Image 3).
- These 6 colourants are permuted using the pseudo-random permutation generator described in task 2 (repeated overleaf), and output as the next two pixels in the final image.



Your Task

To foil the plot you need to extract the three images from the supplied image file `task4.ppm`, store them in separate files and display them when asked by the judges. You may (and of course should) reuse the random number code from task 2. If it's even slightly different from the specification the image data will remain scrambled.

You will probably need to write a factoring and/or prime-generating program (unless you have one already) to deduce the image dimensions, which are uniquely defined by the conditions above. (Image 3's dimensions are fairly easy to work out, and you can extract that picture without knowing the dimensions of the other two).

Random number generator

The generator you must use for this task and also for Task 2 is a multiplicative linear congruential generator, defined as follows.

- S_i is the *i*-th integer seed (maximum 24 bits),
- u_i is a uniformly-distributed pseudo-random value between 0 and 1 (excluding 1), and
- $r_{i,N}$ is a uniformly-distributed pseudo-random value between 0 and *N*, excluding *N*.

$S_{i+1} = (a\ S_i) \bmod m$
where $S_0 = 1$, $a = 6423135$, $m = 16777213$.

$u_i = S_i / m$

$r_{i,N} = \text{floor}(u_i * N)$, where floor($x$) is the largest integer less than or equal to x.

You only need to remember the current seed and the parameters a and m, not the entire sequence.

```
myrand() - advances i and returns u_i
myrandint(N) - returns r_i,N, calling myrand() once
myrandperm(N) - generates a pseudo-random permution of N items
  using this algorithm (assumes zero-based indexing notation [ ]):
  for pos = 0 to N-2
      swap perm[pos] with perm[pos + myrandint(N-pos)]
```

Note 1: the very first call on `myrand` uses S_1, not S_0.

Note 2: `myrandperm` calls `myrandint` exactly *N*-1 times.

Assessment

Full marks are only available for if you display the images exactly as they were prior to encoding, not rotated or skewed or otherwise transformed.

Image 1 is worth 5 marks, Image 2 is worth 7 marks, and Image 3 is worth 11 marks.

Reference:
L'Ecuyer, P (1999). "Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure", *Mathematics of Computation* Vol **68**, No. 225, pp249-260.