Anydesk Hackersprint

Optimization, multi-threading and SIMD

I hope this message reaches you safely!

There is some important intel that I must share with you, but for the sake of security I cannot send them over to you.

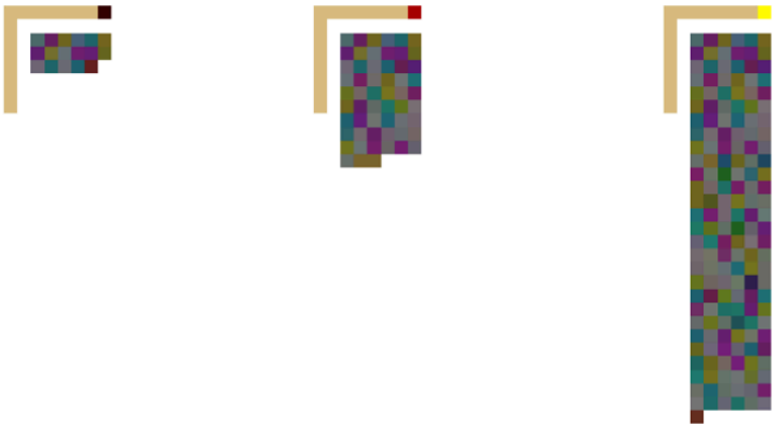
I have obfuscated my messages by embedding my secret messages in the pictures that I have sent you.

Since you are good at programming, I supposed you can make a decoder application that scans and decodes these pictures for these messages.

I will outline below how these codes are encoded so you can write the decoder.

Code structure:

Here are some examples of these codes that have been scaled up so that you can see them. It's normally hard to see these codes in an image, but your scanner should be able to pick these up easily.

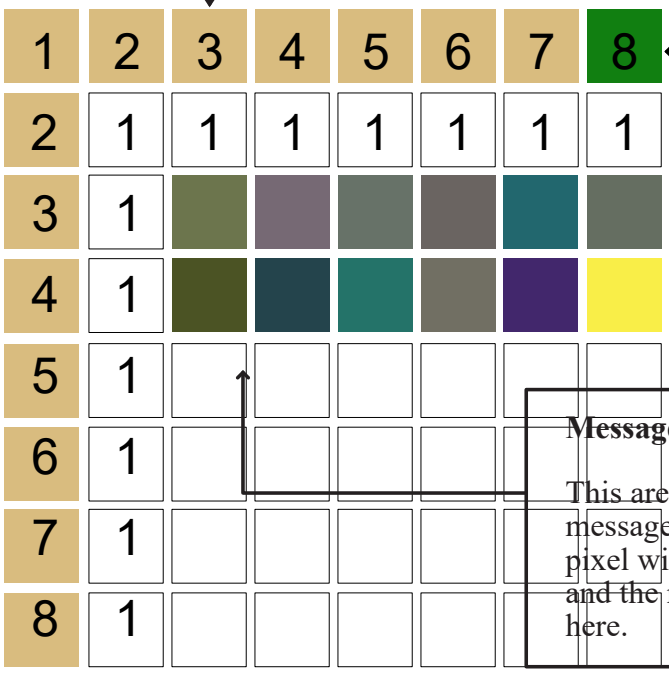


Each of these small rectangles are 1 pixel by 1 pixel. Each of these pixels contains some data that when you put together, you'll get my entire message.

This code is composed of 3 main parts:

Header

This L-shaped group of pixels is the message header. It always have 8 pixels wide and 8 pixels tall. It tells you that this is the beginning of the message and the length of the message.



Message Length

This pixel defines the length of the encoded message. This is part of the header.

Message Data

This area contains the actual message data. There is always a 1 pixel wide gap between the header and the message data, as shown here.

Each pixel of the image is 4 bytes, it stores 4 colour channels in the following order, the blue channel, the green channel, the red channel and an unused channel. In other words, the image is encoded as BGR.

Header:

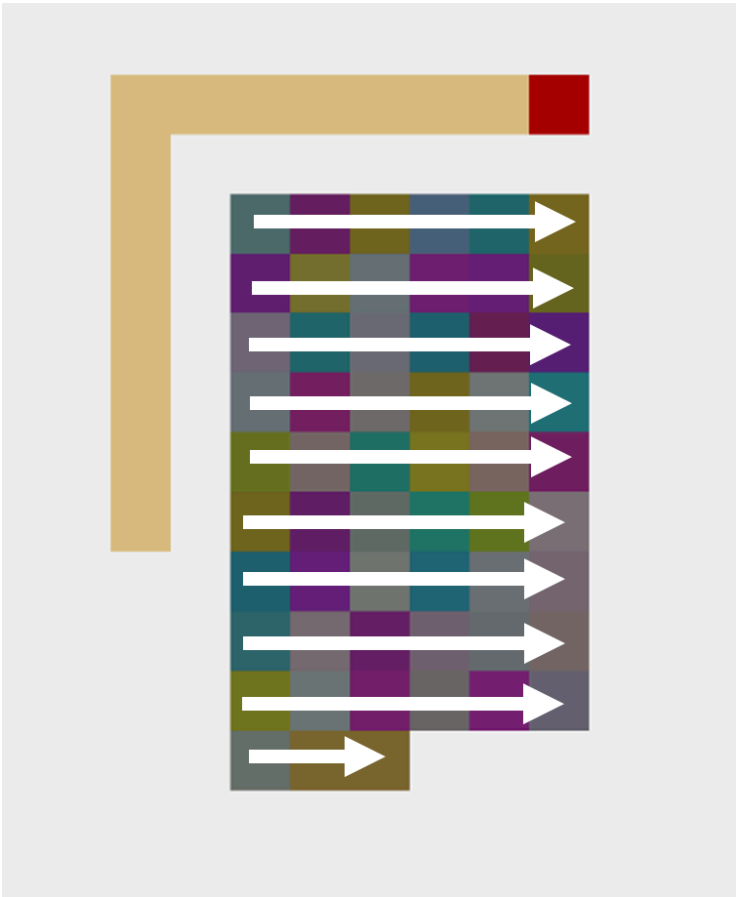
The header is the L-shaped group of pixels. It indicates the beginning of the message. The shape, size, and colour of the header is fixed as displayed in the previous page. The colour of the header pixels is always (127, 188, 217) in BGR.

Message Length:

The rightmost pixel of the header covey encodes the total length of the message. It always has a different colour than the header colour. E.g. message length of 42 means there are 42 characters in the message. The length is encoded by storing an integer in the red and blue channels of the pixel. To get the total length, you must add the value of the red and blue channels together. Since each channel has a range of 0 to 255, the maximum length of the message is 510.

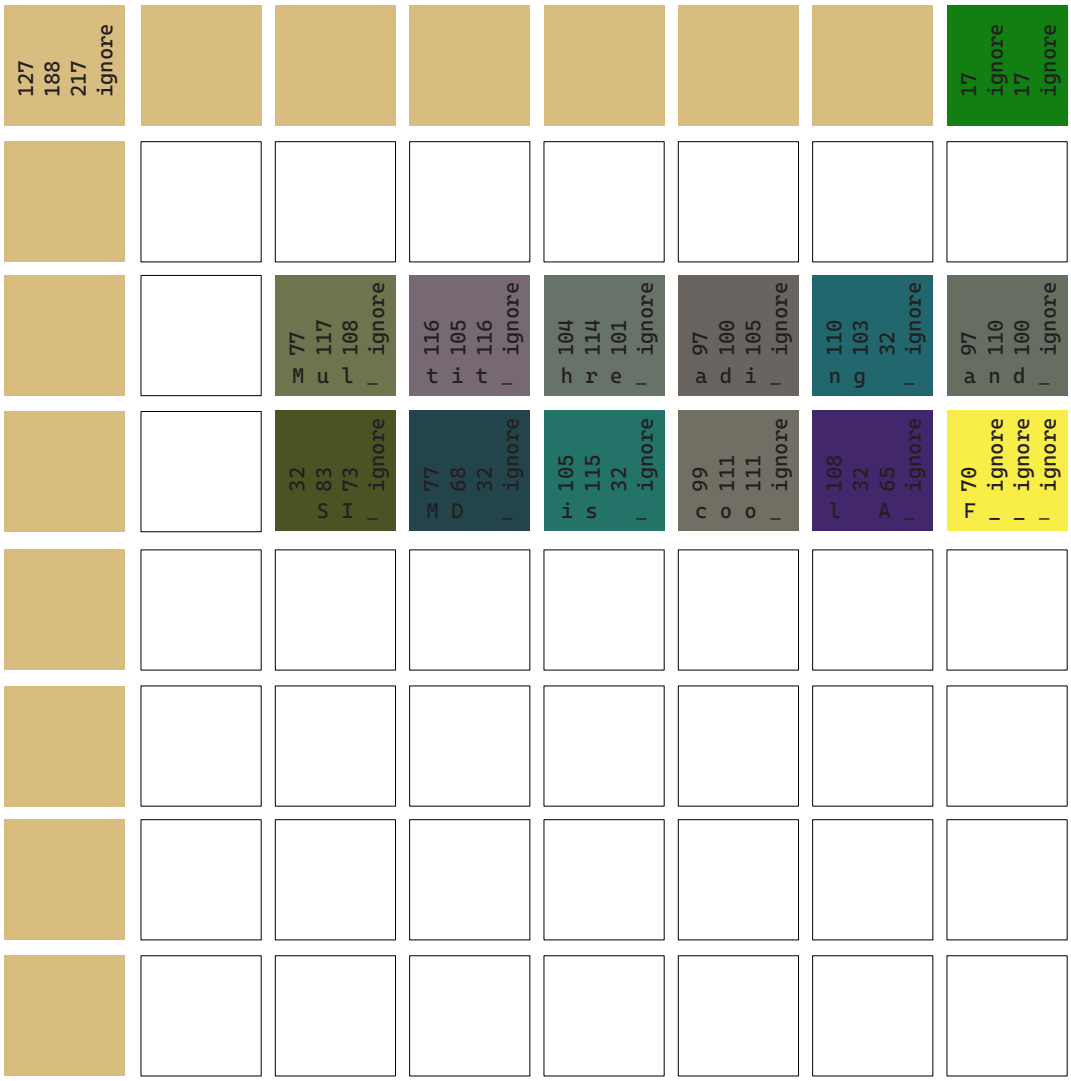
Message Data:

In the message data section of the code, each pixel stores 3 ASCII characters. The characters are stored in the blue, green, and red channels of the pixels. The order of reading is BGR. You must read these pixels from left to right, top to bottom.



Example:

This is an example of a code. There are also markings of how the bytes are layout on each pixels. “ignore” means that byte is not important and can be anything. You can also see that the message hidden in this code is “Multithreading and SIMD is cool AF”.



Windows BMP format:

I want to use an easy-to-understand photo format, so I thought bitmap (BMP format) might be a good idea. In case you are not familiar with it, please read this specification: here. But anyway I have written some small memos here for the images that you need to decode:

- To keep things simple, each pixels are 4 bytes, storing the BGR channels in this order. The last byte is unused. (Believe me, you need to do more work if pixels are stored as 3 bytes)
- BMP file has 2 parts: header and image data. Header contains information about the image, such as width, height etc. Image data are the actual pixels. EXTRA IMPORTANT: pixels are stored starting from the lower left, and ends at the top right of the image.
- Oh btw the last programmer I hired gave up in the mid way decoding my message because of personal reasons, so I can send you his half-assed incomplete decoder source code, probably it can help you!

Requirements:

Hand in file: main.c

Allowed functions: Everything in the allowed headers

Allowed headers:

- pthread.h
- immintrin.h (Please refer to the Intel Intrinsic guide: here)
- sys/stat.h
- sys/mmap.h
- fcntl.h
- stdint.h
- stdio.h
- stdlib.h
- string.h
- unistd.h

IMPORTANT: You can only call malloc once

Your main.c will be compiled like this:

gcc -O0 -Wall -Wextra -Werror main.c -o decoder -lpthread