HERIOT WATT UNIVERSITY

F28HS Coursework 1 Tips

Heriot-Watt University Edinburgh

Coursework 1

Functionality	Question	Marks
`PPM` struct	1	2
Read PPM data	2a	2
Show PPM data	2b	2
Encode PPM data	2c	2
Decode PPM data	2d	2
Steganography program	3	8

More open ended than previous programming labs

There are no "right" or "wrong" answers

Describe your design and data structures in `Report.md`

Data Structures

Without `typedef` for `Pixel`

```
struct Pixel {
  int red; int green; int blue;
};

struct PPM {
  ...
  struct Pixel **pixels;
  ...
}

sizeof(struct Pixel)    sizeof(struct Pixel *)
(struct Pixel *) malloc(..)
```

With `typedef` for `Pixel`

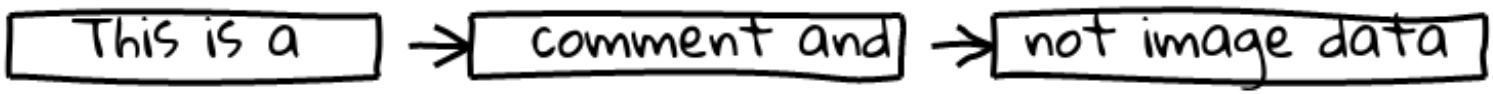
```
typedef struct Pixel {
  int red; int green; int blue;
} Pixel;

typedef struct PPM {
  ...
  Pixel **pixels;
  ...
}

sizeof(Pixel)     sizeof(Pixel *)     (Pixel *) malloc(..)
```

```c
typedef struct Comment {
  /* implement a recursive data structure */
}
```

```
# This is a
# comment and
# not image data
```



More information:

- C structs (e.g. Pixel): Lecture 4
- Recursive structs (e.g. Linked Lists): Lecture 5

Parsing PPM data

`getPPM`

1. Check file starts with `P3`
2. Initialise the comments linked list
3. If a line starts with `#`
    - consume the whole line as comment
    - add line to the linked list of comments
4. Parse `width` and `height`, add to `PPM` structure
5. Parse `max` value, add to `PPM` structure
6. Parse all the pixels, add to `PPM` structure

Parsing `width`, `height`, `pixels`

`fscanf` maybe with a suitable string formatter?

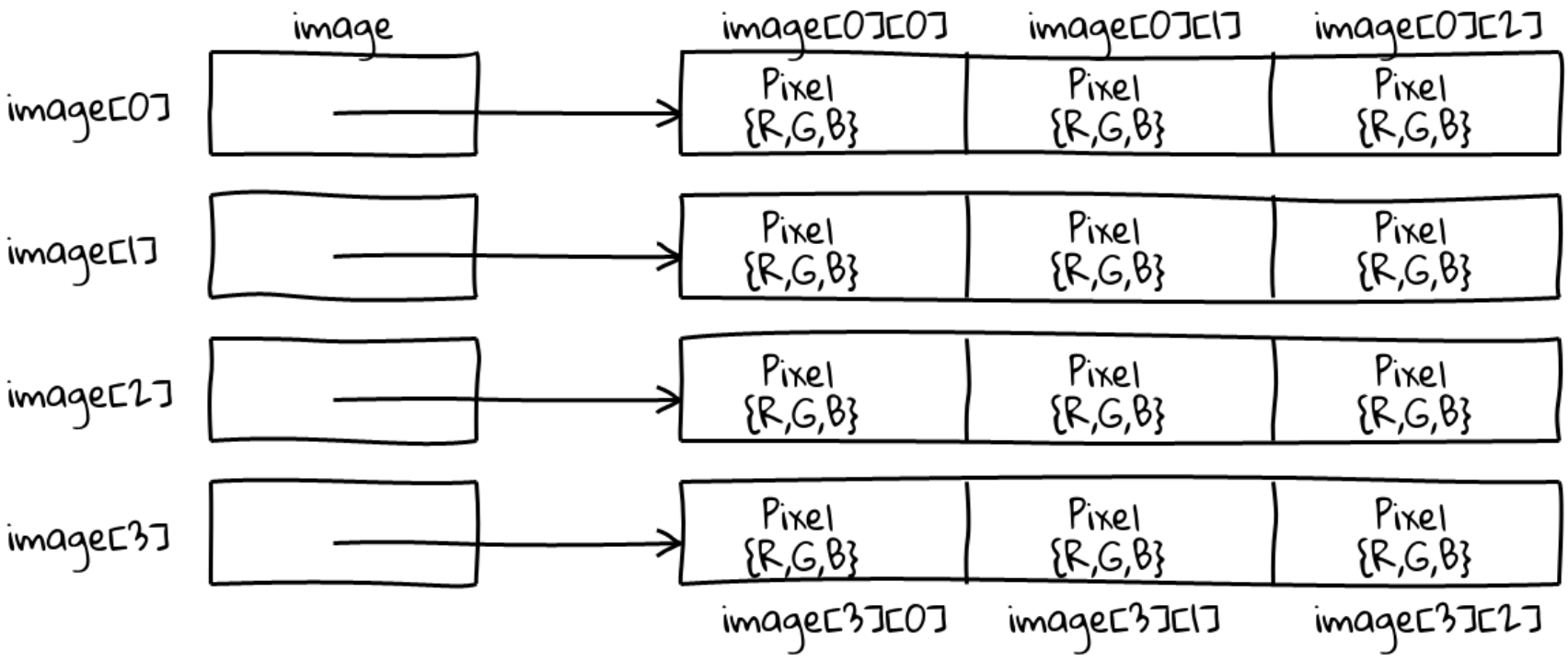
Or how about parse as sequence of `char`

e.g. `['1','4','6']` and deduce this means the integer `146`

More information:

- Lab 1: `getc`
- Tutorial 1: `fscanf`
- Lecture 2: file IO, pointers

Dynamically allocate space on the heap



allocate memory for `image`

allocate memory for `image[0]`, `image[1]` .. `image[rows-1]`

More information for allocation space for 2D PPM pixels:

- Lecture 3: memory allocation
- Lecture 4: dynamic memory allocation
- Lecture 5: multi-dimensional arrays of arbitrary size

```
P3
# CREATOR: GIMP PNM Filter Version 1.1
400 530
255
189
165
181
181
156
165
...
```

How to parse?

```
189
165
181
```

`fscanf` writes to memory *at a given address*

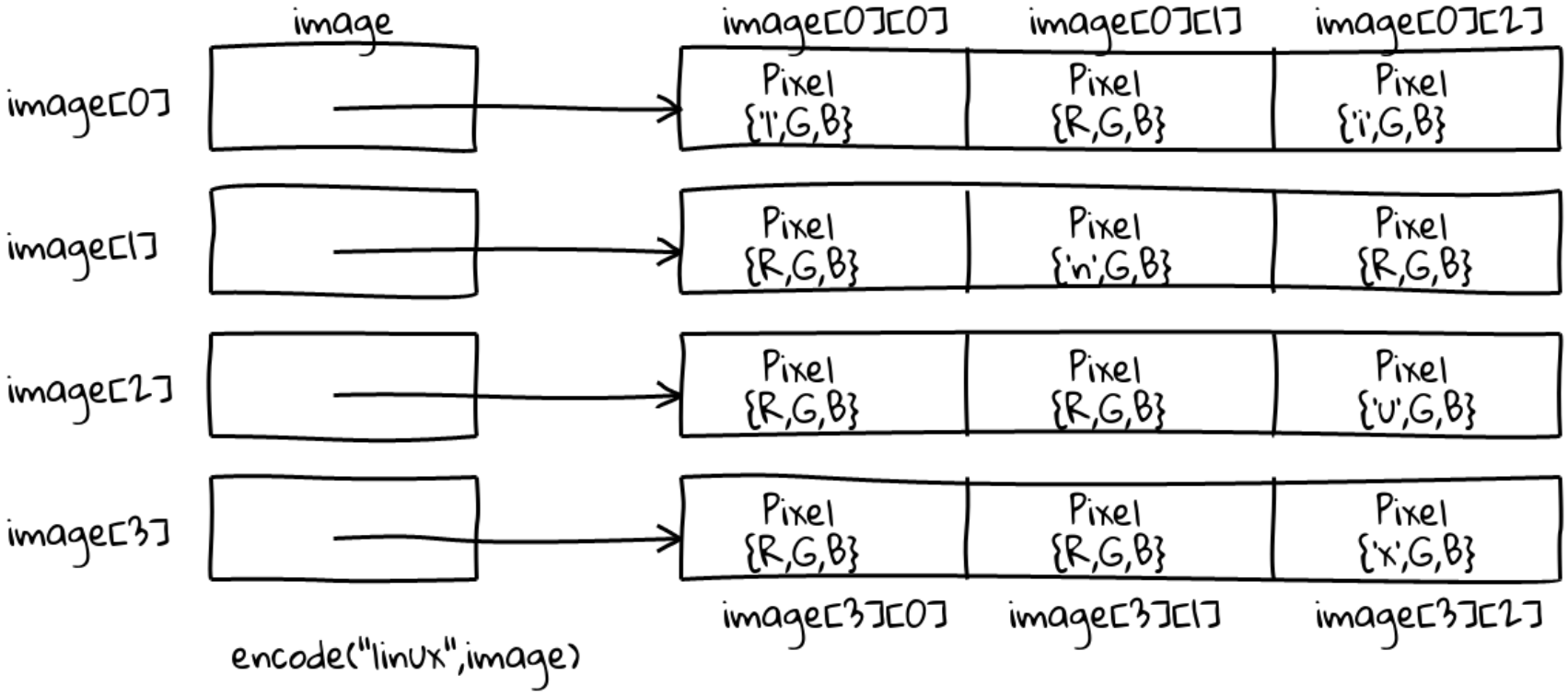
e.g. for a simple integer

```
int i;
fscanf(fd, "%d", &i);
```

`fscanf` formatters can be anything e.g. `"%d %d %d"`

Suppose we want to update `red` field for pixel on row 3, column 13. `image[2][12].red` would get its *value*. *Not quite* what `fscanf` needs if we're going to update the `red` value…

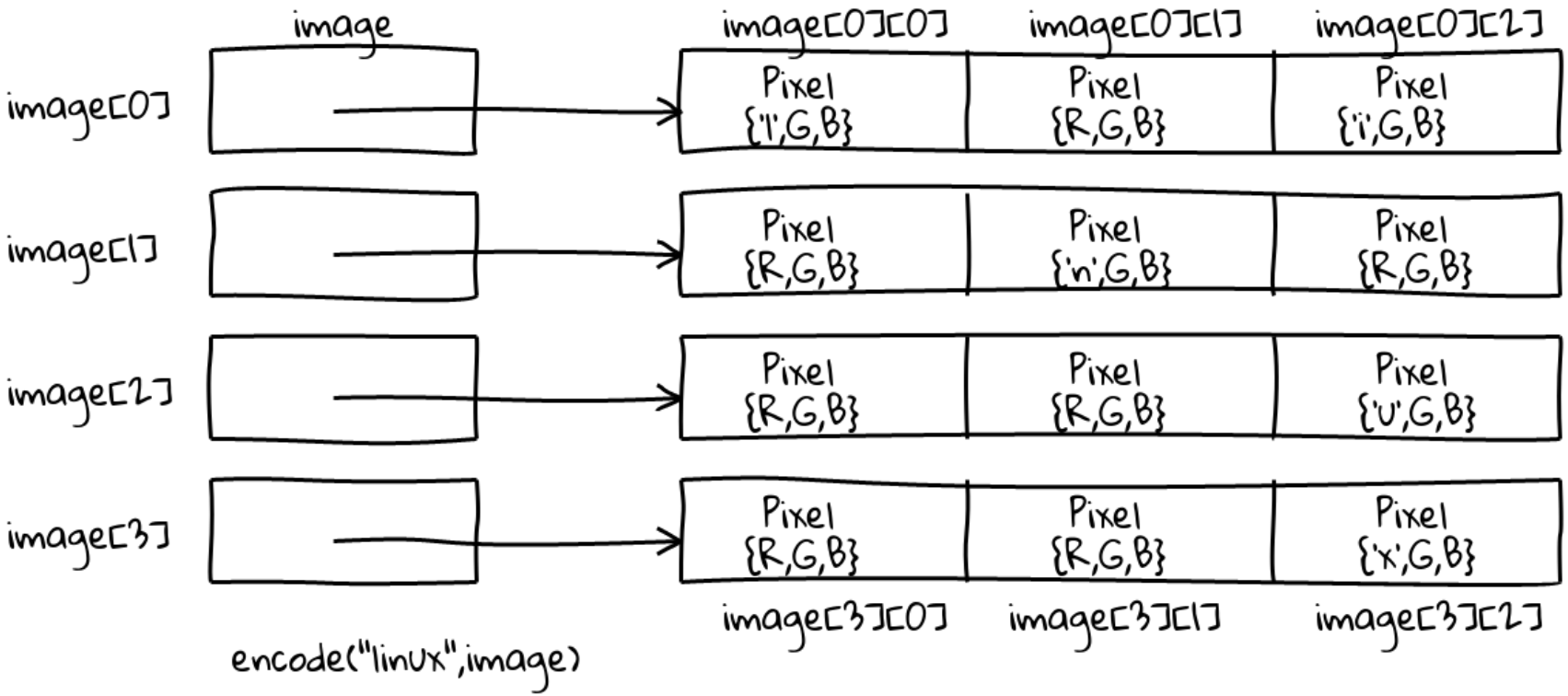
Encoding a secret



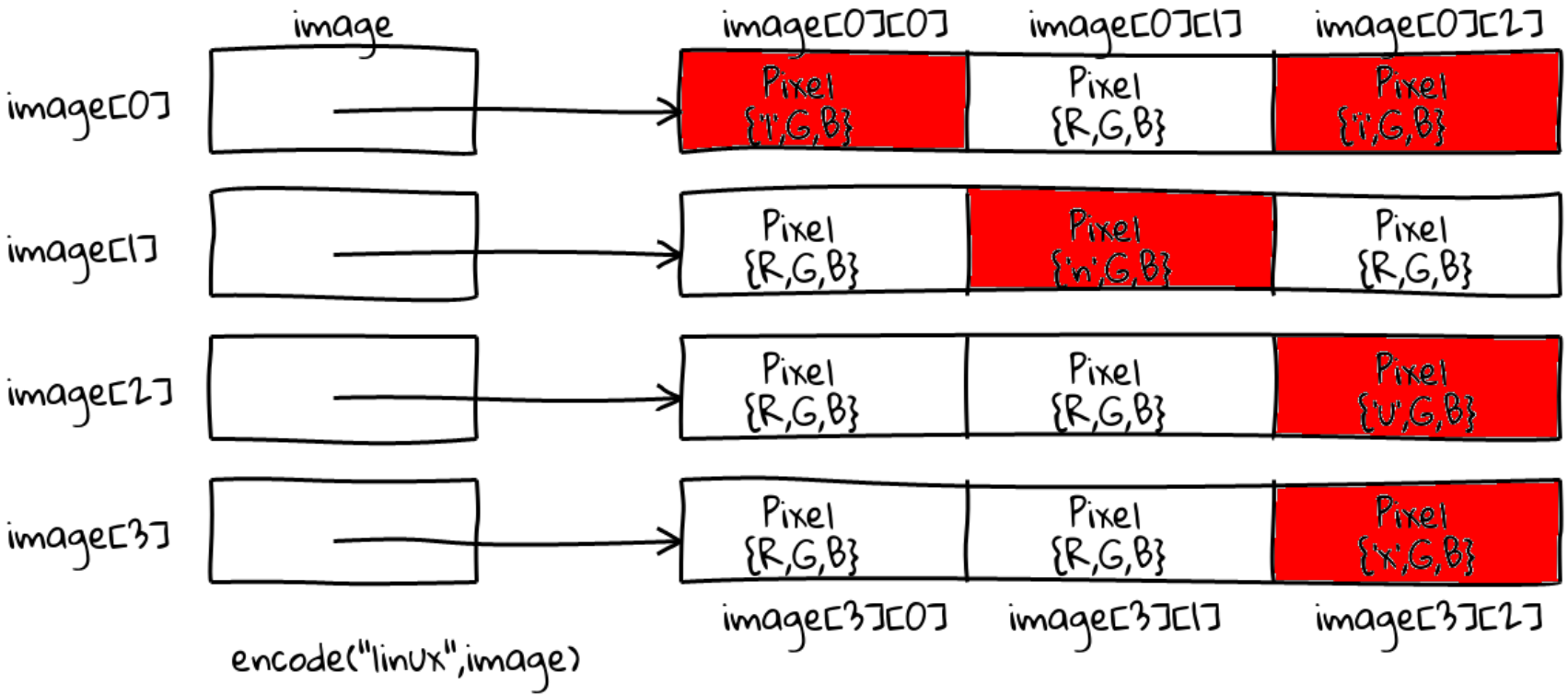
- Replace successive random pixels with letters from text
- How you target pixels completely up to you
- Requirements
    1. characters must be in sequence in row-major order
    2. all the characters must be encoded in the image

Decoding the secret

```
char *decode(PPM* i1, PPM* i2)
```



```
char *decode(PPM* i1, PPM* i2)
```



```
char *decode(PPM* i1, PPM* i2)
```

should return a char pointer containing the encoded text.

In your design you may impose some limitation, e.g. a maximum length for the encoded text.

This is fine, justify why in `Report.md`

Show PPM data

`writePPMtoFile`

1. Print `P3`
2. Print comments (perhaps a `showComments` function?)

   ```
   void showComments(Comment *comment) { .. }
   ```
3. Print the width then height of the image
4. Print the maximum value
5. For each pixel print the R G and B values

```
P3
# this is a picture
# of a cat
400 530
255
141 242 90
190 90 1
...
```

```
void writePPMtoFile(PPM* ppm){ .. }
```

Since you'll run like this:

```
./steg e ape.ppm > output.ppm
```

you could just use `printf(..)`

since Linux's redirect will write standard output to `output.ppm`

If using Windows, you can design your program differently e.g. without the ">" standard output redirect.

`stdout` versus `stderr`

These won't be redirected as input to another program

```
fprintf(stderr,"enter message> ");
fprintf(stderr,"encoding successful!");
```

whereas...

```
printf("encoding successful!");
```

would mean

```
./steg e ape.ppm > output.ppm
```

`output.ppm` includes "encoding successful!" (whoops!)

Final tips

With dynamically allocated memory you can use `[]` brackets

```
image = (Pixel **) malloc( .. )
image[0] = (Pixel *) malloc( .. )

/* value of Green at pixel position [0][2] */
image[0][2].green

/* memory address of Green at the same position */
&(image[0][2].green)
```

More information:

- Lecture 2: the `&` operator

When constructing strings, remember to append with `'\0'` as its last character to mark the end of the string.

If you want to return a char pointer (pointer to the 1st byte of a string) from a function, you could:

1. Define a char array locally of a maximum size e.g.

   ```
   char s[MAX];
   ```

2. Populate `s` up to `MAX`
3. Create a char pointer `text` and `malloc` space for the number of characters "actually" added to `s`
4. Copy chars one-by-one from `s` into `text`
5. Return `text` char pointer from the function

Useful for parsing comments (lines of different lengths)

More information about functions that "return strings":

- Lecture 4: String copy, dynamic memory allocation
- Lecture 4 GitLab project: `scopy.c`

GitLab CI

If you've already forked, and GitLab CI fails (red cross), you might want to ask GitLab to compile with C99 of C:

`.gitlab-ci.yml`

```
script:
   - gcc -std=c99 -o steg steg.c
```

Encoding:

```
./steg e ape.ppm > output.ppm
enter message> Edinburgh
Encoding successful
```

Decoding:

```
./steg d ape.ppm output.ppm
Decoded message:
Edinburgh
```

Good luck!

Ask questions on Discourse!