An Image ADT

This is a **pair** exercise and must be competed in your **tutorial** or **lab** with your partner.

This week, we're taking some of the new data types that we've seen over the last few weeks, and asking, "can we make that an ADT?"

We've just spent some time working with bitmap images: we know that a bitmap image is a piece of data with a well-defined type that has well-defined characteristics, and we have some operations that change and retrieve values within the overall data.

That's all the building blocks we need to make an ADT!

Download `Image.h`, or copy it into your current directory on a CSE system by running

```
$ cp /web/cs1511/17s2/week09/files/Image.h .
```

Don't change *Image.h*! If you do, the fury of a thousand fish will descend on you.

(There's some extra prototypes in here, for the next few exercises. You can ignore these for now, but you might like to think about what those functions do, and how they might manipulate the data.)

Image.h defines our good friend, the `pixel` structure, and a new structure, `point`, which is just a pair of coördinates.

```
typedef struct _point {
    unsigned int x;
    unsigned int y;
} point;
```

Again, we haven't provided a stub `.c` file; you have to build it yourself. For this exercise, you only need to implement six functions:

- `Image newImage (unsigned int width, unsigned int height);`
  Given a width and a height, creates a new `Image`.

- `void destroyImage (Image i);`
  Release all resources associated with an `Image`.

- `unsigned int imageGetWidth (Image i);`
  Given an image, get that image's width.

- `unsigned int imageGetHeight (Image i);`
  Given an image, get that image's height.

- `pixel imageGetPixel (Image i, point p);`
  Given an image, and a particular point, get that point's value.

- `void imageSetPixel (Image i, point p, pixel color);`
  Given an image, and a particular point and a pixel, set that point's value to the pixel passed in.

Your `struct _image` probably should contain a two-dimensional array of pixels, and maybe also some information about the image's width and height.

We've provided an implementation of a useful helper function, `imageAsBMP`, which gives you an allocated buffer of bytes containing an image. You don't have to implement it, thankfully.

Download `imageAsBMP.c`, or copy it into your current directory on a CSE system by running

```
$ cp /web/cs1511/17s2/week09/files/imageAsBMP.c .
```

Here's the prototype:

```
int imageAsBMP (Image i, unsigned char **buf);
```

Given an image, and a reference to a buffer, `imageAsBMP` allocates an array of `unsigned char`s to fit the header and pixels in, containing a BMP image, and returns the size of the allocation.

Strictly, this definition shouldn't be in the ADT header file, as it is an ADT consumer, but we're keeping it there for convenience. You could write a somewhat better version of this with access to the ADT's innards.

To use it, you would do:

```
    unsigned char *buf;
    int size = imageAsBMP (i, &buf);
```

Now you have a full image in `buf`; you could write this to standard output, for example, by using the `fwrite` standard library function from :

```
    fwrite (buf, sizeof (unsigned char), size, stdout);
```

This writes `buf`, which contains `size` items of `unsigned char` size each, to `stdout`.

Don't forget to `free` the allocation that was made!

```
    free (buf);
```

To run some simple automated tests:

```
$ 1511 autotest imageADT
```

To run Styl-o-matic:

```
$ 1511 stylomatic Image.c
Looks good!
```

You'll get advice if you need to make changes to your code.

Submit your work with the *give* command, like so:

```
$ give cs1511 wk09_imageADT
```

Or, if you are working from home, upload the relevant file(s) to the `wk09_imageADT` activity on Give Online.