Lab 10

Helpful Tips

Example of a C++ Class

```
/* image.hpp */

#include <stdint.h> // for uint8_t

class Image {

private:
  unsigned int cols;
  unsigned int rows;
  uint8_t* pixels;

public:

  /* Constructs an image of 0x0 pixels. */
  Image();

  /* Frees all memory allocated for this Image object. */
  ~Image();

  /* Changes the size of an image, allocating memory as necessary, and
     setting all pixels to "fillcolour". Returns 0 on success, or a non-zero error code.*/
  int resize( unsigned int width, unsigned int height, uint8_t fillcolour );

  /* Sets the colour of the pixel at (x,y) to "colour". Returns 0 on success, else a non-zero
     error code. If (x,y) is not a valid pixel, the call fails and the image does not change.*/
  int set_pixel( unsigned int x, unsigned int y, uint8_t colour );

  /* Gets the colour of the pixel at (x,y) and stores it at the address pointed to
     by "colourp". Returns 0 on success, else a non-zero error code. */
  int get_pixel( unsigned int x, unsigned int y, uint8_t* colourp );
};
```

Attributes
a.k.a.
data members

Methods

C struct versus C++ class

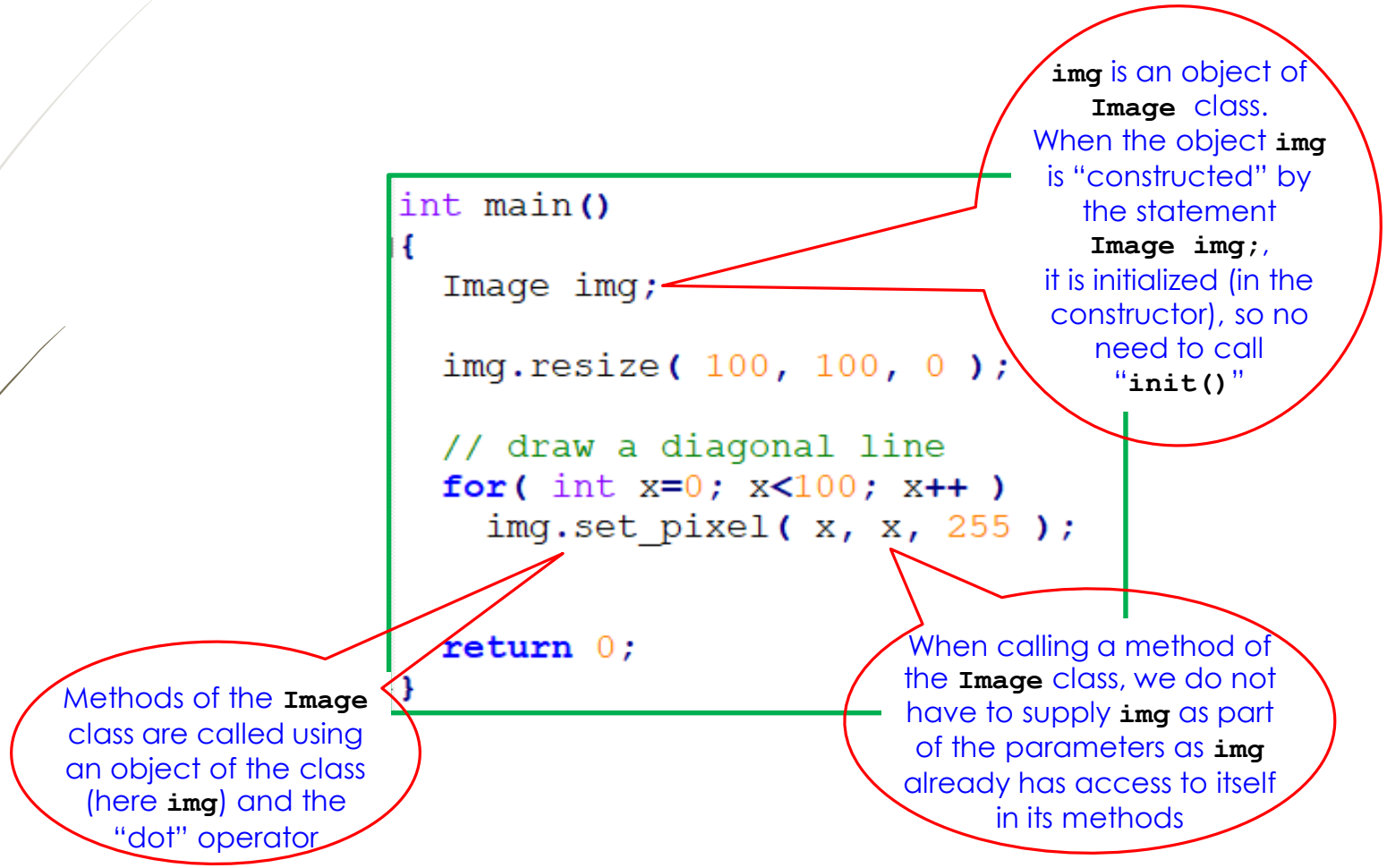
and its function (its attributes and its methods)

```c
#include <stdint.h> // for uint8_t

typedef struct image {
  unsigned int cols;
  unsigned int rows;
  uint8_t* pixels;
} img_t;

/* Returns a pointer to an allocated img_t structure containing 0x0
   pixels. */
img_t* img_create( void );

/* Frees all memory allocated for img */
void img_destroy( img_t* img );

/* Initializes an image of size 0x0 pixels */
void img_init( img_t* i );

int img_resize( img_t* img, *
        unsigned int width,
        unsigned int height,
        uint8_t fillcolour );

int img_set_pixel( img_t* img, *
           unsigned int x,
           unsigned int y,
           uint8_t colour );

int img_get_pixel( img_t* img, *
           unsigned int x,
           unsigned int y,
           uint8_t* pcolour );
```

```cpp
/* image.hpp */

#include <stdint.h> // for uint8_t

class Image {

private:
  unsigned int cols;
  unsigned int rows;
  uint8_t* pixels;

public:

  /* Constructs an image of 0x0 pixels. */
  Image();

  /* Frees all memory allocated for this Image object. */
  ~Image();

  /* Changes the size of an image, allocating memory as necessary, and
     setting all pixels to "fillcolour". Returns 0 on success, or a non
  int resize( unsigned int width, unsigned int height, uint8_t fillcolo

  /* Sets the colour of the pixel at (x,y) to "colour". Returns 0 on su
     error code. If (x,y) is not a valid pixel, the call fails and the
  int set_pixel( unsigned int x, unsigned int y, uint8_t colour );

  /* Gets the colour of the pixel at (x,y) and stores it at the address
     by "colourp". Returns 0 on success, else a non-zero error code. */
  int get_pixel( unsigned int x, unsigned int y, uint8_t* colourp );
};
```


* Function description removed due to lack of space!

How to declare and manipulate a C++ object!

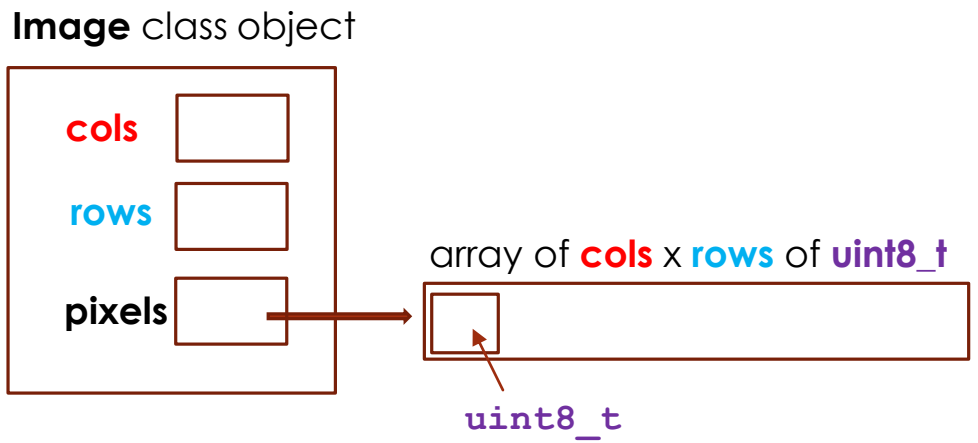


Data Structures in our Lab 10 classes

Tasks 1 and 2

From the *.hpp file:

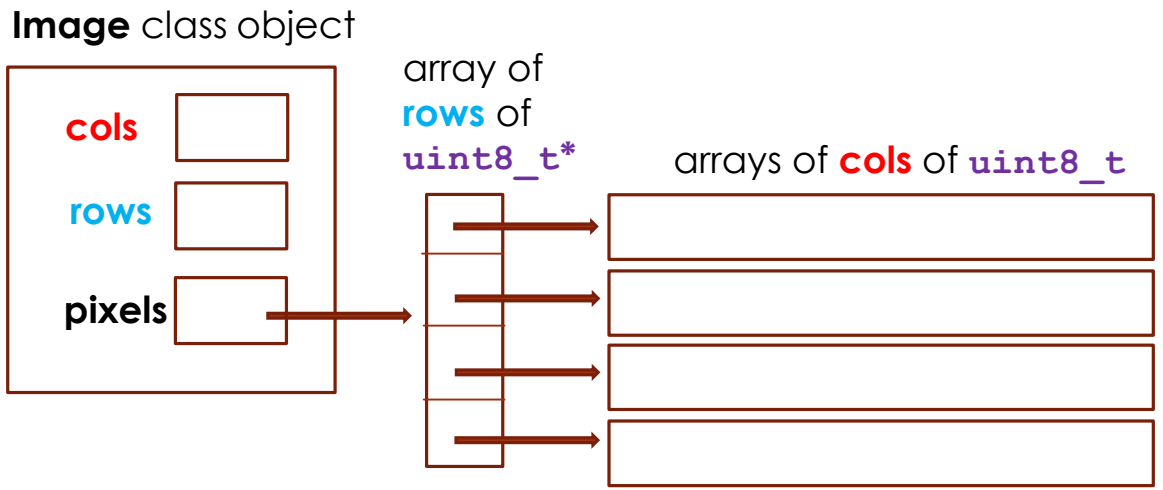
```
unsigned int cols;
unsigned int rows;
uint8_t* pixels;
```



Task 3

From the *.hpp file:

```
unsigned int cols;
unsigned int rows;
uint8_t** pixels;
```



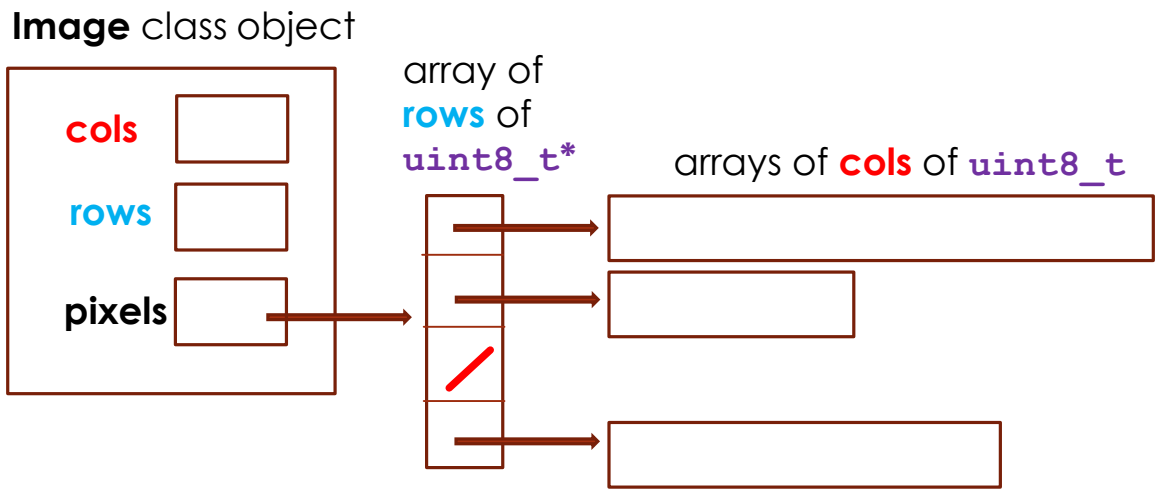
Called "ragged arrays"

Called “ragged arrays”

Task 3

From the *.hpp file:

```
unsigned int cols;
unsigned int rows;
uint8_t** pixels;
```



Called “ragged arrays”

Helpful Tips about Lab 10

- Task 1: Recycle the work you have done in Lab 3 and Lab 5
- Task 2: Recycle the work you have done in Lab 6
- Task 3: You add another dimension (layer) of arrays
  - (For all 3 tasks) Constructor: initialize `cols`, `rows` and `pixels` (no call to `new` – Why?)
  - Destructor: delete the "deepest" layer of arrays (2nd dimension) **first**
  - `set_pixel( )` and `get_pixel( )`
    - **`pixels`**`[ ][ ]`
      - you need to figure out where to put the **`x`** (**cols**) and where to put the **`y`** (**rows**)
      - Is it **`pixels`**`[x][y]` (**`pixels`**`[`**`cols`**`][`**`rows`**`]`)

      OR

      - Is it **`pixels`**`[y][x]` (**`pixels`**`[`**`rows`**`][`**`cols`**`]`) ?

Helpful Tips about Lab 10

- Continue to validite parameters
- It is OK to call

```
int Image::resize( unsigned int cols, unsigned int rows, uint8_t fillcolor )
```

this way:

- `resize( 0, 0, fillcolour );`

- To get a sense of when the constructor and the destructor are implicitly called, you may want to add a temporary cout statement in them such as
  `cout << "Constructor called and executed!" << endl;`
  - Don't forget to remove these `cout` statements before you push your code to your Git repo! ☺

C test driver (below) versus C++ test driver (next page)

```c
#include <stdio.h>

#include "image.h"

int main()
{
  printf("Declare, create and initialize an image variable by calling img_create( );\n");
  printf("Note that img_create( ) calls img_init(...))\n");
  img_t* img = img_create( );
  if ( img ) {  // if ( img != NULL )
    printf("Calling img_resize( &img, 100, 100, 0 );\n");
    if ( img_resize( img, 100, 100, 0 ) == 0 )  {  // 0 -> black

      printf("Drawing a diagonal line by calling img_set_pixel( &img, x, x, 255 );\n");
      for( int x=0; x<100; x++ )
        img_set_pixel( img, x, x, 255 ); // 255 -> white

      // Here, we would need to either display the image (in order to see the diagonal line)
      // or print the content of the img variable (in order to verify the diagonal line)
      // hence confirming that our img variable has been constructed and manipulated properly,
      // i.e., that our code works!

    }
    printf("Calling img_destroy( &img );\n");
    img_destroy( img );

    puts( "Done!" );
  }
  return 0;
}
```

This test driver is testing the C `struct` and its functions found on Slide 3

Note: This test driver is not complete since it is not yet testing `img_get_pixel(…)`!

C test driver (previous) versus C++ test driver (below)

```cpp
#include <iostream>  // for cout and endl

#include "image.hpp"

using namespace std;

int main()
{
  cout << "Declaring and constructing an img object of Image class type => Image img;" << endl;
  cout << "Since this constructs img automatically, there is no need to have a method init(...)" << endl;
  Image img;

  cout << "Calling img.resize( 100, 100, 0 );" << endl;
  img.resize( 100, 100, 0 );    // 0 -> black

  cout << "Drawing a diagonal line by calling img.set_pixel( x, x, 255 );" << endl;
  for( int x=0; x<100; x++ )
    img.set_pixel( x, x, 255 ); // 255 -> white

  // Here, we would need to either display the image (in order to see the diagonal line)
  // or print the content of the img object (in order to verify the diagonal line)
  // hence confirming that our img object has been constructed and manipulated properly,
  // i.e., that our code works!

  cout << "Since img is automatically destroyed when going out of scope, there is no need to have a method destroy(...)" << endl;

  cout << "Done!" << endl;
  return 0;
}
```

This test driver is testing the C++ class `Image` found on the Slide 3

Note: This test driver is not complete since it is not yet testing `img.get_pixel(…)`!

C test driver versus C++ test driver side by side

```c
#include <stdio.h>

#include "image.h"

int main()
{
  printf("Declare, create and initialize an image vari
  printf("Note that img_create( ) calls img_init(...))
  img_t* img = img_create( );
  if ( img ) {  // if ( img != NULL )
    printf("Calling img_resize( &img, 100, 100, 0 );\n
    if ( img_resize( img, 100, 100, 0 ) == 0 )  {  //

      printf("Drawing a diagonal line by calling img_s
      for( int x=0; x<100; x++ )
        img_set_pixel( img, x, x, 255 ); // 255 -> whi

      // Here, we would need to either display the ima
      // or print the content of the img variable (in
      // hence confirming that our img variable has be
      // i.e., that our code works!

    }
    printf("Calling img_destroy( &img );\n");
    img_destroy( img );

    puts( "Done!" );
  }
  return 0;
}
```

```cpp
#include <iostream>  // for cout and endl

#include "image.hpp"

using namespace std;

int main()
{
  cout << "Declaring and constructing an img object of Imag
  cout << "Since this constructs img automatically, there i
  Image img;

  cout << "Calling img.resize( 100, 100, 0 );" << endl;
  img.resize( 100, 100, 0 );    // 0 -> black

  cout << "Drawing a diagonal line by calling img.set_pixel
  for( int x=0; x<100; x++ )
    img.set_pixel( x, x, 255 ); // 255 -> white

  // Here, we would need to either display the image (in or
  // or print the content of the img object (in order to ve
  // hence confirming that our img object has been construc
  // i.e., that our code works!

  cout << "Since img is automatically destroyed when going

  cout << "Done!" << endl;
  return 0;
}
```

Example of a makefile for Lab 10

Before, we used the C compiler in our makefile:

```
imgC: driver.c image.c
    gcc -std=c99 -o $@ driver.c image.c

# $ make clean
#  - removes files we built using the targets above
clean:
    rm -f img* *.o
```

Now, for Lab 10, we use the C++ compiler in our makefile:

```
img1: main1.cpp image.cpp
    g++ -o $@ main1.cpp image.cpp

img2: main2.cpp image2.cpp
    g++ -o $@ main2.cpp image2.cpp

img3: main3.cpp image3.cpp
    g++ -o $@ main3.cpp image3.cpp

# $ make clean
#  - removes files we built using the targets above
clean:
    rm -f img* *.o
```

Sneak preview of CMPT 225

- In CMPT 225, we shall introduce the concept of ADT: Abstract Data Type

```
/* image.hpp */

#include <stdint.h> // for uint8_t

class Image {

private:
  unsigned int cols;
  unsigned int rows;
  uint8_t* pixels;

public:

  /* Constructs an image of 0x0 pixels. */
  Image();

  /* Frees all memory allocated for this Image object. */
  ~Image();

  /* Changes the size of an image, allocating memory as necessary, and
     setting all pixels to "fillcolour". Returns 0 on success, or a non-zero error code.*/
  int resize( unsigned int width, unsigned int height, uint8_t fillcolour );

  /* Sets the colour of the pixel at (x,y) to "colour". Returns 0 on success, else a non-zero
     error code. If (x,y) is not a valid pixel, the call fails and the image does not change.*/
  int set_pixel( unsigned int x, unsigned int y, uint8_t colour );

  /* Gets the colour of the pixel at (x,y) and stores it at the address pointed to
     by "colourp". Returns 0 on success, else a non-zero error code. */
  int get_pixel( unsigned int x, unsigned int y, uint8_t* colourp );
};
```

The attributes are "private"

The methods are "public"