Copying, assignment and the Rule of 3

We know there is a difference between == and =

But there are two kinds of =:

- = in a declaration, like int a = 4; (initialization)
- = elsewhere, like a = 4; (assignment)

Complex c = {3.0, 2.0};  // = in declaration: *initialization*
Complex c2 = c;  // (same)
c = Complex(4.0, 5.0);  // = outside declaration: *assignment*
if(c2.get_real() == 3.0) {  // == is *equality testing*
  // ...
}
Image has resources managed by the constructor & destructor:

class Image {
  public:
    Image(const char *orig, int r, int c) : nrow(r), ncol(c) {
      image = new char[r*c];
      for (int i = 0; i < nrow * ncol; i++) {
        image[i] = orig[i];
      }
    }

    ~Image() { delete[] image; }

    const char *get_image() const { return image; }
    int get_nrow() const { return nrow; }
    int get_ncol() const { return ncol; }

    void set_pixel(char pix, int row, int col) {
      image[row * ncol + col] = pix;
    }

  private:
    char *image; // image data
    int nrow, ncol; // # rows and columns
};

std::ostream& operator<<(std::ostream&, const Image&);
#include <iostream>
#include "image.h"

using std::endl;
using std::ostream;

ostream& operator<<(ostream& os, const Image& image) {
    for (int i = 0; i < image.get_nrow(); i++) {
        for (int j = 0; j < image.get_ncol(); j++) {
            os << image.get_image()[i*image.get_ncol()+j] << ' ';
        }
        os << endl;
    }
    return os;
}
#include <iostream>
#include "image.h"
using std::cout; using std::endl;

int main()
{
    Image x_wins("X-O-XO--X", 3, 3);
    cout << x_wins << "** X wins! **" << endl;
    return 0;
}

$ g++ -o image_main image_main.cpp image.cpp
$ ./image_main
X - O
- X 0
- - X
** X wins! **
```cpp
#include <iostream>
#include "image.h"

using std::cout; using std::endl;

int main() {
    Image x_wins("X-O-XO--X", 3, 3);
    Image o_wins = x_wins;
    o_wins.set_pixel('O', 2, 2); // set bottom right to 'O'
    cout << x_wins << "** X wins! **" << endl << endl;
    cout << o_wins << "** O wins! **" << endl;
    return 0;
}
```
$ g++ -o image_main2 image_main2.cpp image.cpp
$ ./image_main2
X - O
- X 0
- - 0
** X wins! **

X - O
- X 0
- - 0
** 0 wins! **

Oops, both have 0 in bottom right corner

o_wins.set_pixel(...) affected both x_wins & o_wins!
image_main2.cpp

Also: destructor delete[]s the same pointer twice

$ valgrind ./image_main2 > /dev/null
==42== Memcheck, a memory error detector
==42== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==42== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==42== Command: ./image_main2
==42==
==42== Jump to the invalid address stated on the next line
==42==   at 0x0: ???
==42==   by 0x4008C9: _start (in /app/750_ruleof3/image_main2)
==42==   by 0x1FFF000C37: ???
==42==   by 0x4228F7F: ??? (in /usr/lib64/ld-2.26.so)
==42== Address 0x0 is not stack'd, malloc'd or (recently) free'd
==42==
==42== Process terminating with default action of signal 11 (SIGSEGV)
==42== Bad permissions for mapped region at address 0x0
==42==   at 0x0: ???
==42==   by 0x4008C9: _start (in /app/750_ruleof3/image_main2)
==42==   by 0x1FFF000C37: ???
==42==   by 0x4228F7F: ??? (in /usr/lib64/ld-2.26.so)
==42==
==42== HEAP SUMMARY:
==42== in use at exit: 0 bytes in 0 blocks
==42== total heap usage: 1 allocs, 1 frees, 72,704 bytes allocated
==42==
==42== All heap blocks were freed -- no leaks are possible
Initialization & assignment

Image o_wins = x_wins; does shallow copy

- Copies x_wins.image pointer directly into o_wins.image, so both are using same heap array
- Instead, we want deep copy; o_wins should be a new buffer, with contents of x_wins copied over
- Want this both for initialization and for assignment

Image x_wins("X-O-XO--X", 3, 3);
Image o_wins = x_wins;
Image is an example of a class that manages resources, and therefore has a *non-trivial destructor*

*Rule of 3*: If you have to manage how an object is destroyed, you should also manage how it’s copied

*Rule of 3 (technical version)*: If you have a non-trivial destructor, you should also define a *copy constructor* and *operator=*

Case in point: Image should be deep copied
Copy constructor initializes a class variable as a copy of another
operator= is called when one object is assigned to another

Complex c = {3.0, 2.0}; // non-default constructor
Complex c2 = c; // copy constructor
c = Complex(4.0, 5.0); // non-default ctor for right-hand side
// operator= to copy into left-hand side
Copy constructor

Copy constructor is called when:

- Initializing:
  - Image o_wins = x_wins;
  - Image o_wins(x_wins); (same meaning as above)

- Passing by value
- Returning by value
Copy constructor for Image:

```cpp
Image(const Image& o) : nrow(o.nrow), ncol(o.ncol) {
    // Do a *deep copy*, similarly to the
    // non-default constructor
    image = new char[nrow * ncol];
    for(int i = 0; i < nrow * ncol; i++) {
        image[i] = o.image[i];
    }
}
```
operator=

operator= is called when assigning one class variable to another

- Except for initialization; copy constructor handles that

```cpp
Image& operator=(const Image& o) {
    delete[] image; // deallocate previous image memory
    nrow = o.nrow;
    ncol = o.ncol;
    image = new char[nrow * ncol];
    for(int i = 0; i < nrow * ncol; i++) {
        image[i] = o.image[i];
    }
    return *this; // for chaining
}
```

It's a normal member function, not a constructor, so we can't use initializer list syntax
Rule of 3

If you don’t specify copy constructor or operator=, compiler adds *implicit* version that *shallow copies*

- Simply copies each field
- class field will have its copy constructors or operator= function called
- Pointer to heap memory will simply be copied, without the heap memory itself being copied

Another way of stating the Rule of 3: if your class has a non-trivial destructor, you probably *don’t* want shallow copying
Rule of 3

When we add the copy constructor and operator= defined above, we get the expected behavior:

```
$ g++ -o image_fixed image_fixed.cpp image.cpp
$ ./image_fixed
X - O
- X 0
-- X
** X wins! **

X - O
- X 0
-- O
** O wins! **
```
And no complaints from valgrind:

$ valgrind ./image_fixed > /dev/null
==52== Memcheck, a memory error detector
==52== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==52== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==52== Command: ./image_fixed
==52==
==52==
==52== HEAP SUMMARY:
==52== in use at exit: 0 bytes in 0 blocks
==52== total heap usage: 4 allocs, 4 frees, 76,818 bytes allocated
==52==
==52== All heap blocks were freed -- no leaks are possible
==52==
==52== For counts of detected and suppressed errors, rerun with: -v
==52== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)