

601.220 Intermediate Programming

Rule of 3

An Image class

Image has resources managed by the constructor & destructor:

```
// image.h:
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&);
```

image.cpp

```
// image.cpp:
#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;
}
```

image_main.cpp

```
// image_main.cpp:
#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 O
- - X
** X wins! **
```

image_main2.cpp

```
// image_main.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;
}
```

image_main2.cpp

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

X - 0
- X 0
- - 0
** O wins! **
Aborted (core dumped)
```

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_main > /dev/null
==3492== Memcheck, a memory error detector
==3492== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==3492== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==3492== Command: ./image_main
==3492==
==3492== error calling PR_SET_PTRACER, vgdb might block
==3492== Invalid free() / delete / delete[] / realloc()
==3492==   at 0x4C2F74B: operator delete[](void*) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3492==   by 0x400CEE: Image::~Image() (in /mnt/c/CS220/slides/source/day30_rule_of_3_files/image_main)
==3492==   by 0x400BA3: main (in /mnt/c/CS220/slides/source/day30_rule_of_3_files/image_main)
==3492== Address 0x5b19c80 is 0 bytes inside a block of size 9 free'd
==3492==   at 0x4C2F74B: operator delete[](void*) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3492==   by 0x400CEE: Image::~Image() (in /mnt/c/CS220/slides/source/day30_rule_of_3_files/image_main)
==3492==   by 0x400B97: main (in /mnt/c/CS220/slides/source/day30_rule_of_3_files/image_main)
==3492== Block was alloc'd at
==3492==   at 0x4C2E80F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3492==   by 0x400C76: Image::Image(char const*, int, int) (in /mnt/c/CS220/slides/source/day30_rule_of_3_files/image_main)
==3492==   by 0x400AF8: main (in /mnt/c/CS220/slides/source/day30_rule_of_3_files/image_main)
==3492==
==3492== HEAP SUMMARY:
==3492==   in use at exit: 72,704 bytes in 1 blocks
==3492==   total heap usage: 3 allocs, 3 frees, 76,809 bytes allocated
==3492==
==3492== LEAK SUMMARY:
==3492==   definitely lost: 0 bytes in 0 blocks
==3492==   indirectly lost: 0 bytes in 0 blocks
==3492==   possibly lost: 0 bytes in 0 blocks
==3492==   still reachable: 72,704 bytes in 1 blocks
==3492==   suppressed: 0 bytes in 0 blocks
```

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;
```


Rule of 3

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

Rule of 3

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

Copy constructor for Image:

```
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

```
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 the contents of the fields
- `class` fields will have their corresponding copy constructors or `operator=` functions called
- Pointers 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:

Rule of 3

```
// image_fixed.cpp:
#include <iostream>
#include "image_fixed.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_fixed image_fixed.cpp image.cpp
```

```
$ ./image_fixed
```

```
X - O
```

```
- X O
```

```
- - X
```

```
*** X wins! ***
```

```
X - O
```

```
- X O
```

```
- - O
```

```
*** O wins! ***
```


Rule of 3

And no complaints from valgrind:

```
$ valgrind ./image_fixed > /dev/null
==3506== Memcheck, a memory error detector
==3506== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==3506== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==3506== Command: ./image_fixed
==3506==
==3506== error calling PR_SET_PTRACER, vgdb might block
==3506==
==3506== HEAP SUMMARY:
==3506==     in use at exit: 72,704 bytes in 1 blocks
==3506==   total heap usage: 4 allocs, 3 frees, 76,818 bytes allocated
==3506==
==3506== LEAK SUMMARY:
==3506==    definitely lost: 0 bytes in 0 blocks
==3506==    indirectly lost: 0 bytes in 0 blocks
==3506==    possibly lost: 0 bytes in 0 blocks
==3506==    still reachable: 72,704 bytes in 1 blocks
==3506==         suppressed: 0 bytes in 0 blocks
==3506== Rerun with --leak-check=full to see details of leaked memory
==3506==
==3506== For counts of detected and suppressed errors, rerun with: -v
==3506== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```