

601.220 Intermediate Programming

Rule of 3

An Image class

// image.h

```
1  #ifndef IMAGE_H__
2  #define IMAGE_H__
3  #include <iostream>
4  class Image {
5  public:
6      Image(const char *orig, int r, int c) : nrow(r), ncol(c) {
7          image = new char[r*c];
8          for(int i = 0; i < nrow * ncol; i++) image[i] = orig[i];
9      }
10     ~Image() { delete[] image; }
11
12     void set_pixel(char pix, int row, int col) {
13         image[row * ncol + col] = pix;
14     }
15     friend std::ostream& operator<<(std::ostream& os, const Image& image) {
16         for(int i = 0; i < image.nrow; i++) {
17             for(int j = 0; j < image.ncol; j++)
18                 os << image.image[i*image.ncol+j] << ' ';
19             os << std::endl;
20         }
21         return os;
22     }
23 private:
24     char *image;    // image data
25     int nrow, ncol; // # rows and columns
26 };
27 #endif // IMAGE_H__
```

image_main.cpp

```
// image_main.cpp

1  #include "image.h"
2
3  int main() {
4      Image x_wins("X-O-XO--X", 3, 3);
5      std::cout << x_wins << "** X wins! **" << std::endl;
6      return 0;
7  }

$ g++ -o image_main image_main.cpp -std=c++11 -pedantic -Wall -Wextra
$ ./image_main

X - O
- X O
- - X
** X wins! **
```

image_main2.cpp

```
// image_main2.cpp

1  #include "image.h"
2
3  int main() {
4      Image x_wins("X-O-XO--X", 3, 3);
5      Image o_wins = x_wins;
6      o_wins.set_pixel('O', 2, 2); // set bottom right to 'O'
7      std::cout << x_wins << "** X wins! **" << std::endl <<
8      std::cout << o_wins << "** O wins! **" << std::endl;
9      return 0;
10 }
```

image_main2.cpp

```
$ g++ -o image_main2 image_main2.cpp -std=c++11 -pedantic -Wall -Wextra
```

```
$ ./image_main2
```

```
X - 0
```

```
- X 0
```

```
- - 0
```

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

```
X - 0
```

```
- X 0
```

```
- - 0
```

```
** 0 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

```
$ valgrind --leak-check=full ./image_main2

==3651== Memcheck, a memory error detector
==3651== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==3651== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==3651== Command: ./image_main2
==3651==
==3651== error calling PR_SET_PTRACER, vgdb might block
==3651== Invalid free() / delete / delete[] / realloc()
==3651==    at 0x4C2F74B: operator delete[](void*) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3651==    by 0x400C9C: Image::~Image() (in /d/Study/PhDCS(JHU)/JHU2020/2020-09-12(Fall) - Intermediate Programming)
==3651==    by 0x400B4F: main (in /d/Study/PhDCS(JHU)/JHU2020/2020-09-12(Fall) - Intermediate Programming)
==3651== Address 0x5b19c80 is 0 bytes inside a block of size 9 free'd
==3651==    at 0x4C2F74B: operator delete[](void*) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3651==    by 0x400C9C: Image::~Image() (in /d/Study/PhDCS(JHU)/JHU2020/2020-09-12(Fall) - Intermediate Programming)
==3651==    by 0x400B43: main (in /d/Study/PhDCS(JHU)/JHU2020/2020-09-12(Fall) - Intermediate Programming)
==3651== Block was alloc'd at
==3651==    at 0x4C2E80F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3651==    by 0x400C22: Image::Image(char const*, int, int) (in /d/Study/PhDCS(JHU)/JHU2020/2020-09-12(Fall) - Intermediate Programming)
==3651==    by 0x400AA4: main (in /d/Study/PhDCS(JHU)/JHU2020/2020-09-12(Fall) - Intermediate Programming)
==3651==
==3651== HEAP SUMMARY:
==3651==    in use at exit: 72,704 bytes in 1 blocks
==3651==    total heap usage: 3 allocs, 3 frees, 76,809 bytes allocated
==3651==
==3651== LEAK SUMMARY:
==3651==    definitely lost: 0 bytes in 0 blocks
==3651==    indirectly lost: 0 bytes in 0 blocks
==3651==    possibly lost: 0 bytes in 0 blocks
==3651==    still reachable: 72,704 bytes in 1 blocks
==3651==    suppressed: 0 bytes in 0 blocks
```

Shallow copy

```
...  
Image x_wins("X-O-XO--X", 3, 3);  
Image o_wins = x_wins;  
...
```

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

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 **assignment operator**

Case in point: Image should be deep copied

Rule of 3

Copy constructor (ClassName(**const** ClassName&)) initializes a **class** variable as a copy of another

Assignment operator (**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 for a deep copy:

```
Image(const Image& o) : nrow(o.nrow), ncol(o.ncol) {  
    // Do a *deep copy*  
    image = new char[nrow * ncol];  
    for(int i = 0; i < nrow * ncol; i++) {  
        image[i] = o.image[i];  
    }  
}
```

Assignment 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 assignment operator, compiler adds **implicit** version that **shallow copies**

- Simply the contents of the fields
- class fields will have their corresponding copy constructors or assignment 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 assignment operator in previous slides for the deep copy, we get the expected behavior:

```
// image_main_fixed.cpp
```

```
1  #include "image_fixed.h"
2
3  int main() {
4      Image x_wins("X-O-XO--X", 3, 3);
5      Image o_wins = x_wins;
6      o_wins.set_pixel('O', 2, 2); // set bottom right to 'O'
7      std::cout << x_wins << "** X wins! **" << std::endl << std::endl;
8      std::cout << o_wins << "** O wins! **" << std::endl;
9      return 0;
10 }
```

```
$ g++ -o image_main_fixed image_main_fixed.cpp -std=c++11 -pedantic -Wall -Wextra
```

```
$ ./image_main_fixed
```

```
X - O
- X O
- - X
** X wins! **
```

```
X - O
- X O
- - O
** O wins! **
```

Rule of 3

And no complaints from valgrind:

```
$ valgrind --leak-check=full ./image_main_fixed
```

```
==3663== Memcheck, a memory error detector
==3663== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==3663== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==3663== Command: ./image_main_fixed
==3663==
==3663== error calling PR_SET_PTRACER, vgdb might block
==3663==
==3663== HEAP SUMMARY:
==3663==     in use at exit: 72,704 bytes in 1 blocks
==3663==   total heap usage: 4 allocs, 3 frees, 76,818 bytes allocated
==3663==
==3663== LEAK SUMMARY:
==3663==    definitely lost: 0 bytes in 0 blocks
==3663==    indirectly lost: 0 bytes in 0 blocks
==3663==    possibly lost: 0 bytes in 0 blocks
==3663==    still reachable: 72,704 bytes in 1 blocks
==3663==         suppressed: 0 bytes in 0 blocks
==3663== Reachable blocks (those to which a pointer was found) are not shown.
==3663== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==3663==
==3663== For counts of detected and suppressed errors, rerun with: -v
==3663== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```