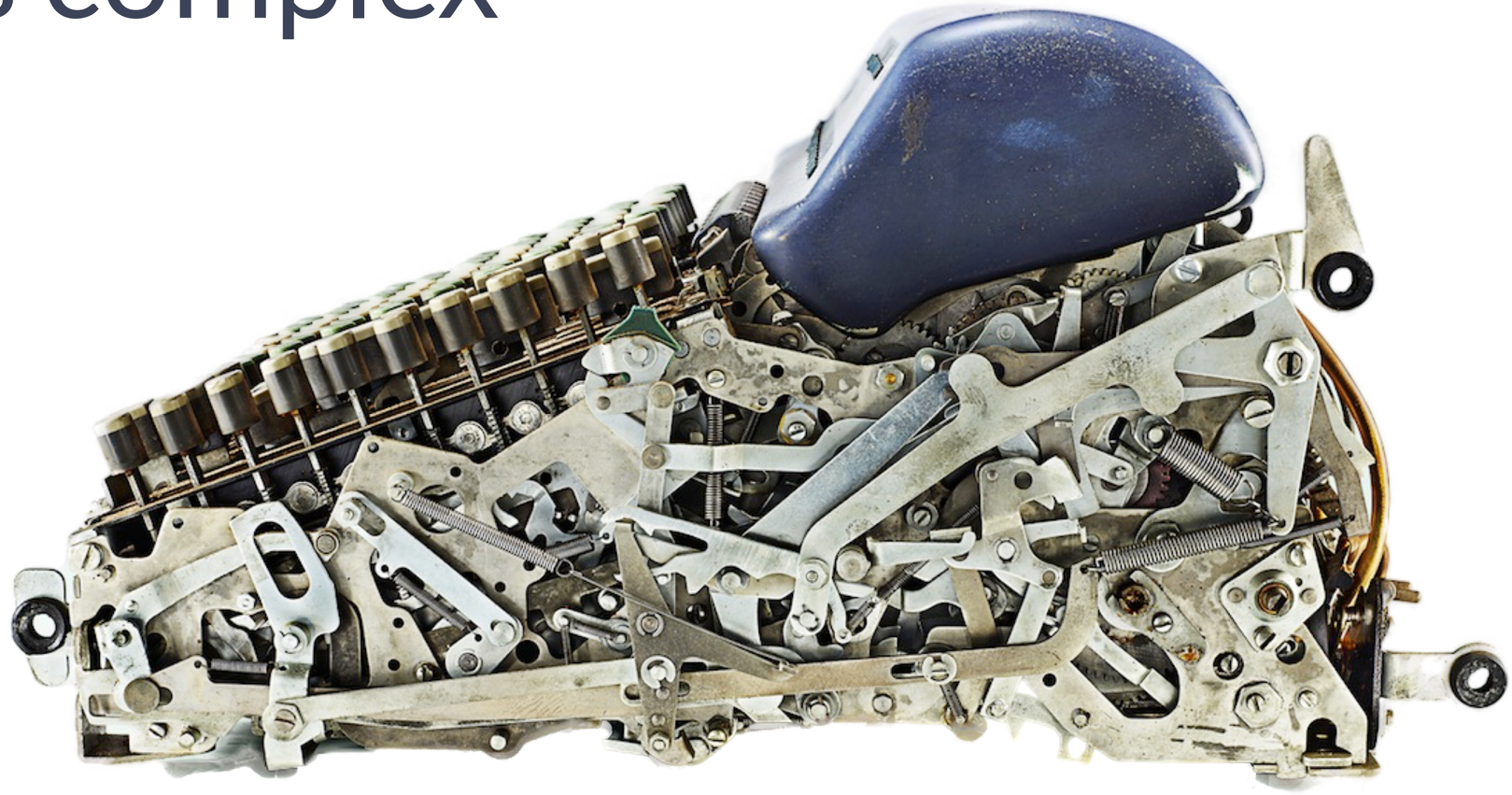
Back To Basics
Lifetime Management

PHIL NASH

C++ is complex



mostly for historical reasons

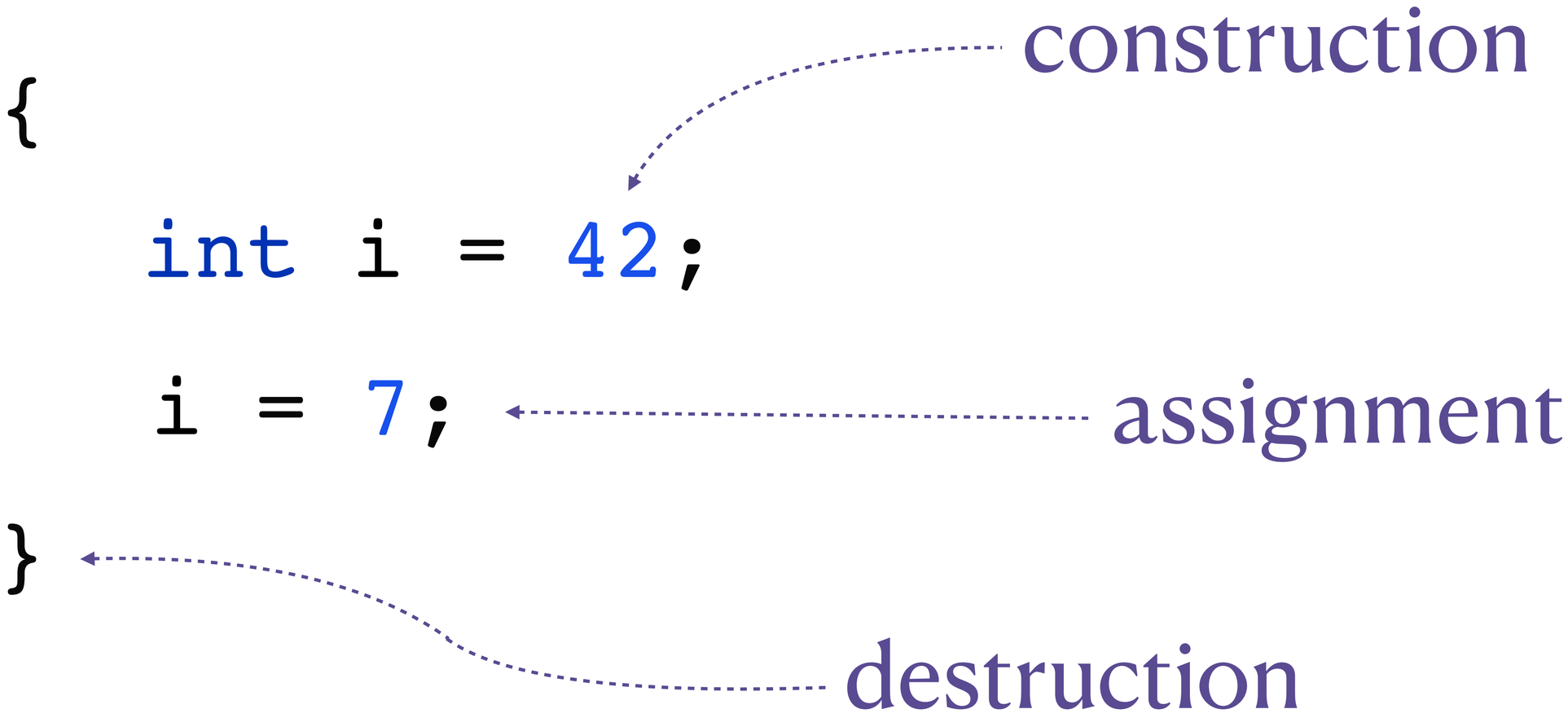
BUT...

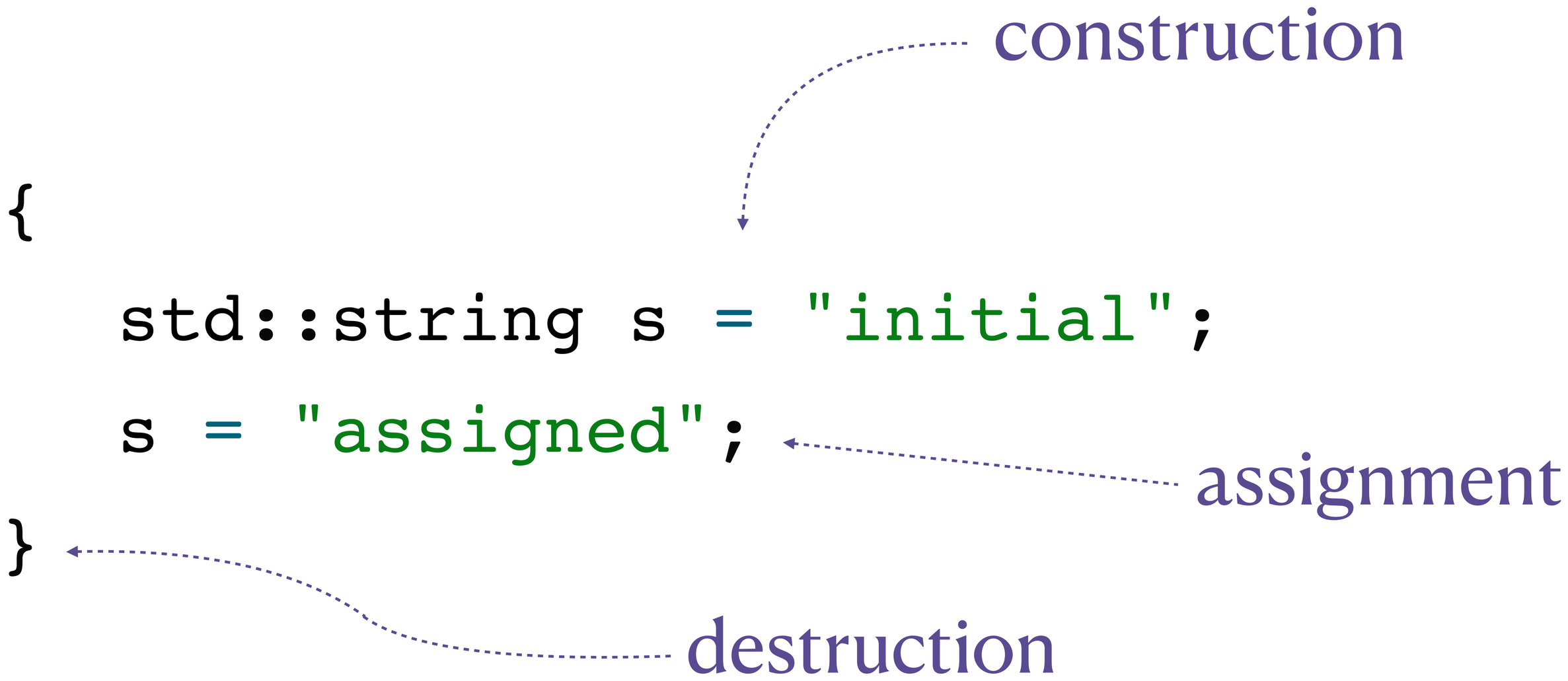
C++ is a value-based language by default

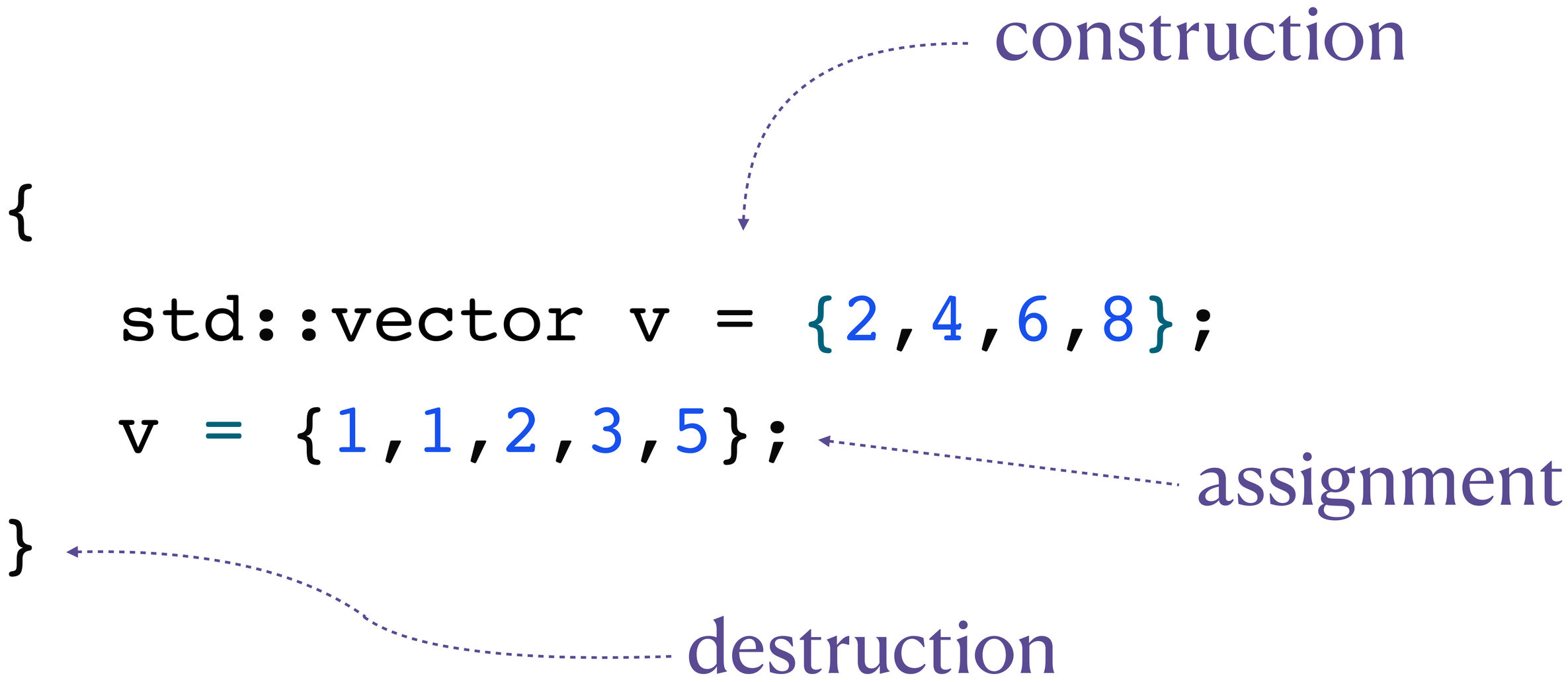
```
int i = 42;
```

```
int i = 42;
i = 7;
```

```
{
    int i = 42;
    i = 7;
}
```


{
int i = 42;
i = 7;
}
construction
assignment
destruction


construction
{
std::string s = "initial";
s = "assigned";
assignment
}
destruction


construction
{
std::vector v = {2,4,6,8};
v = {1,1,2,3,5};
assignment
}
destruction

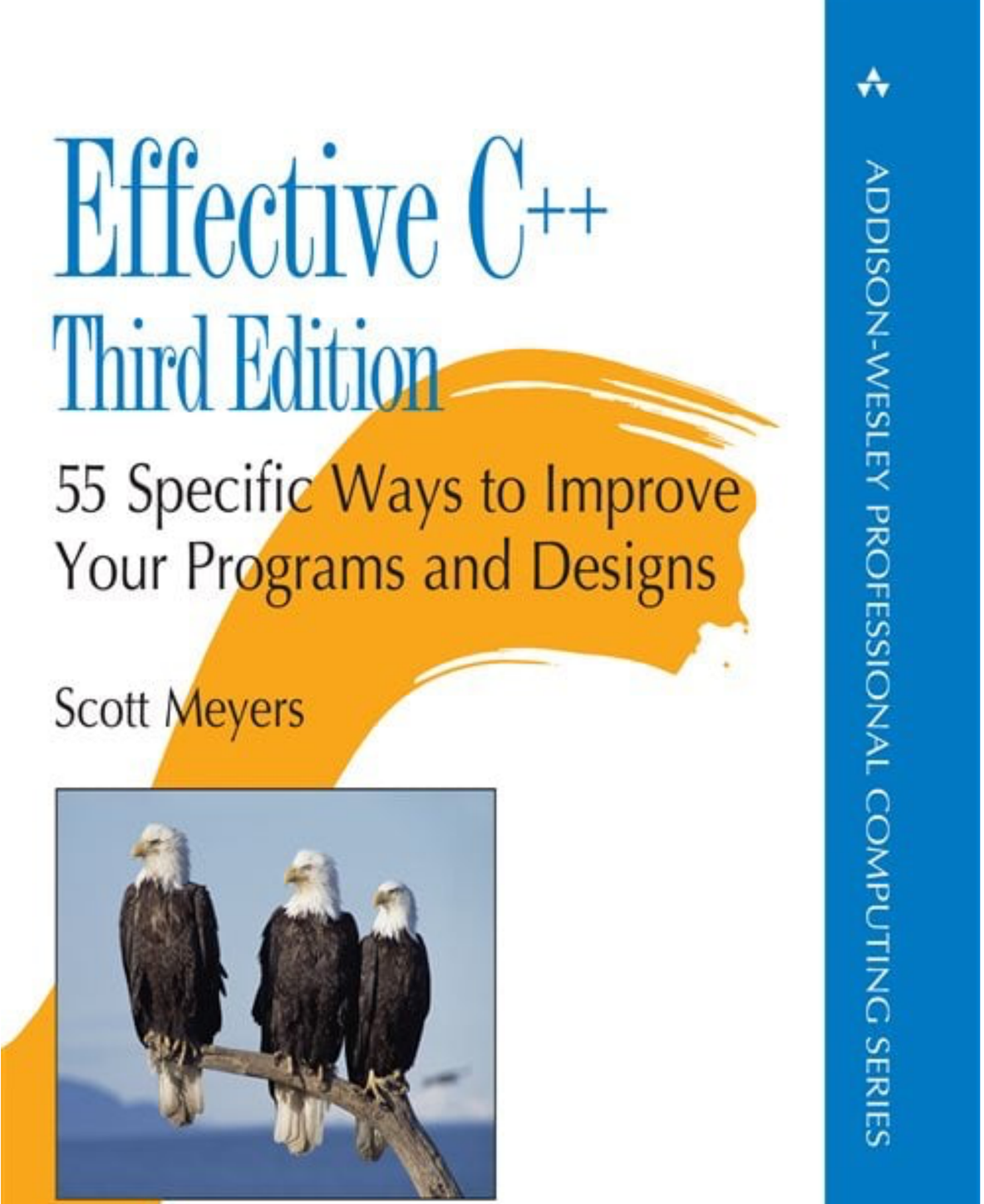
Effective C++

Third Edition

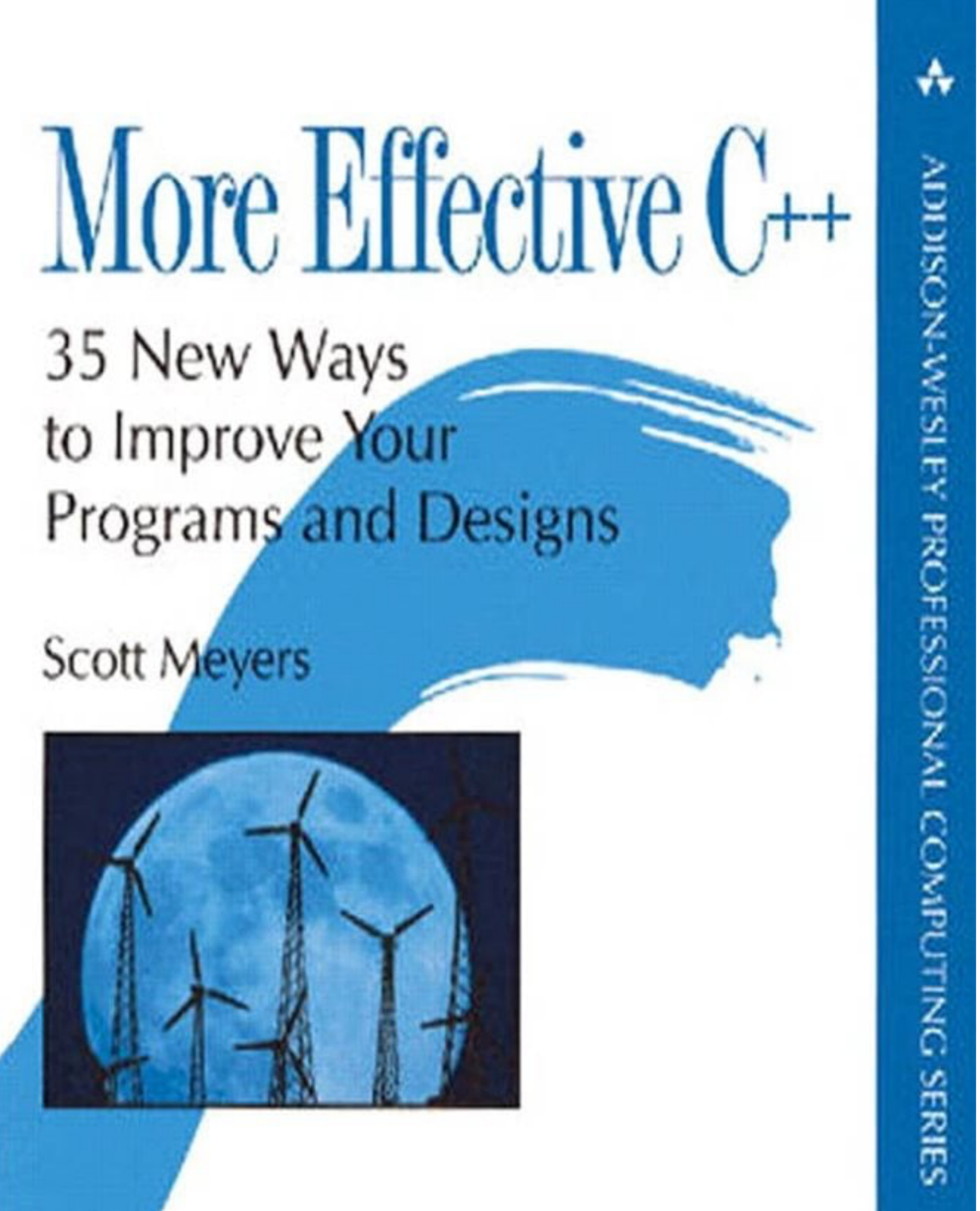
55 Specific Ways to Improve Your Programs and Designs

Scott Meyers

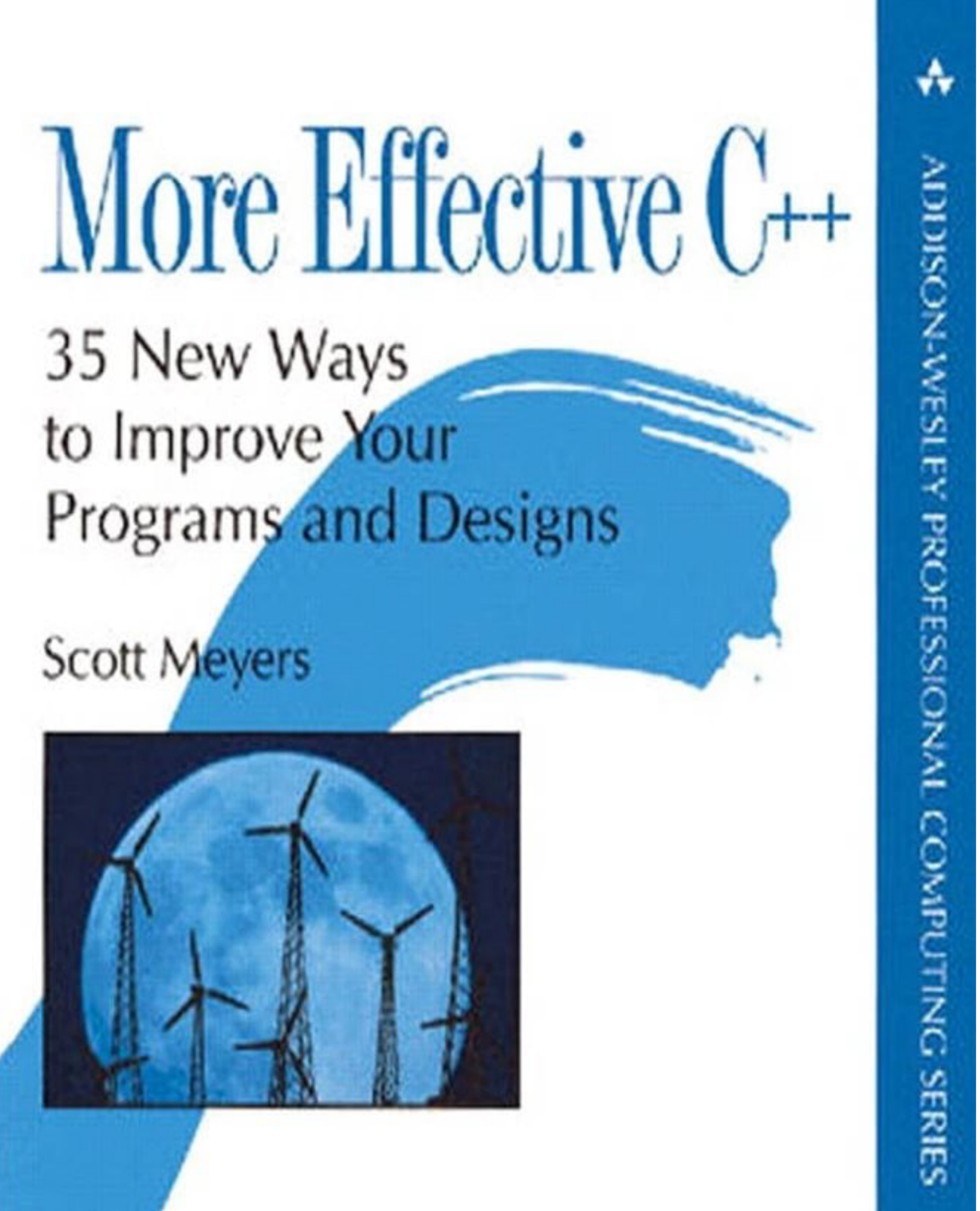
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



“Do as the ints do”

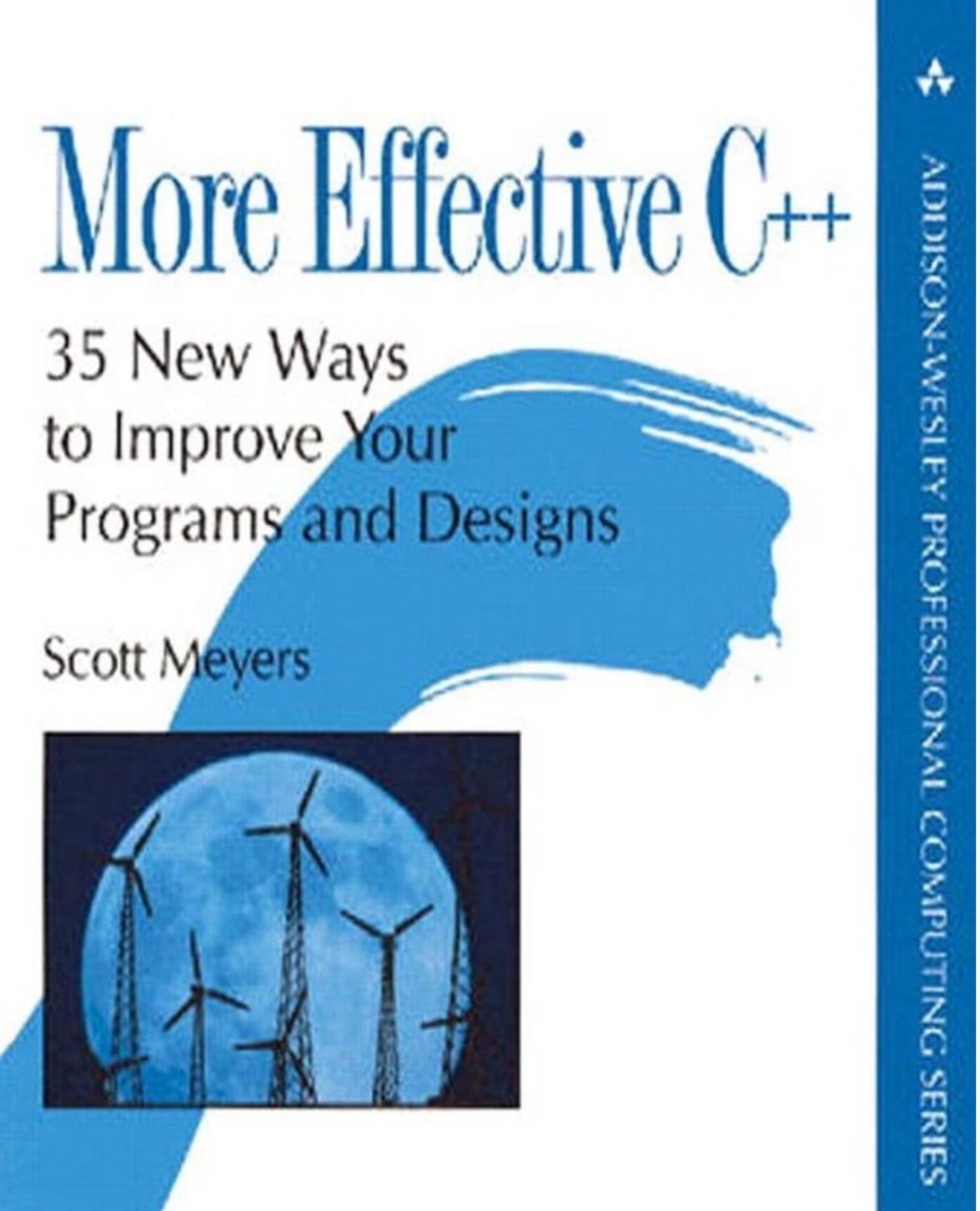


"Do as the ints do"



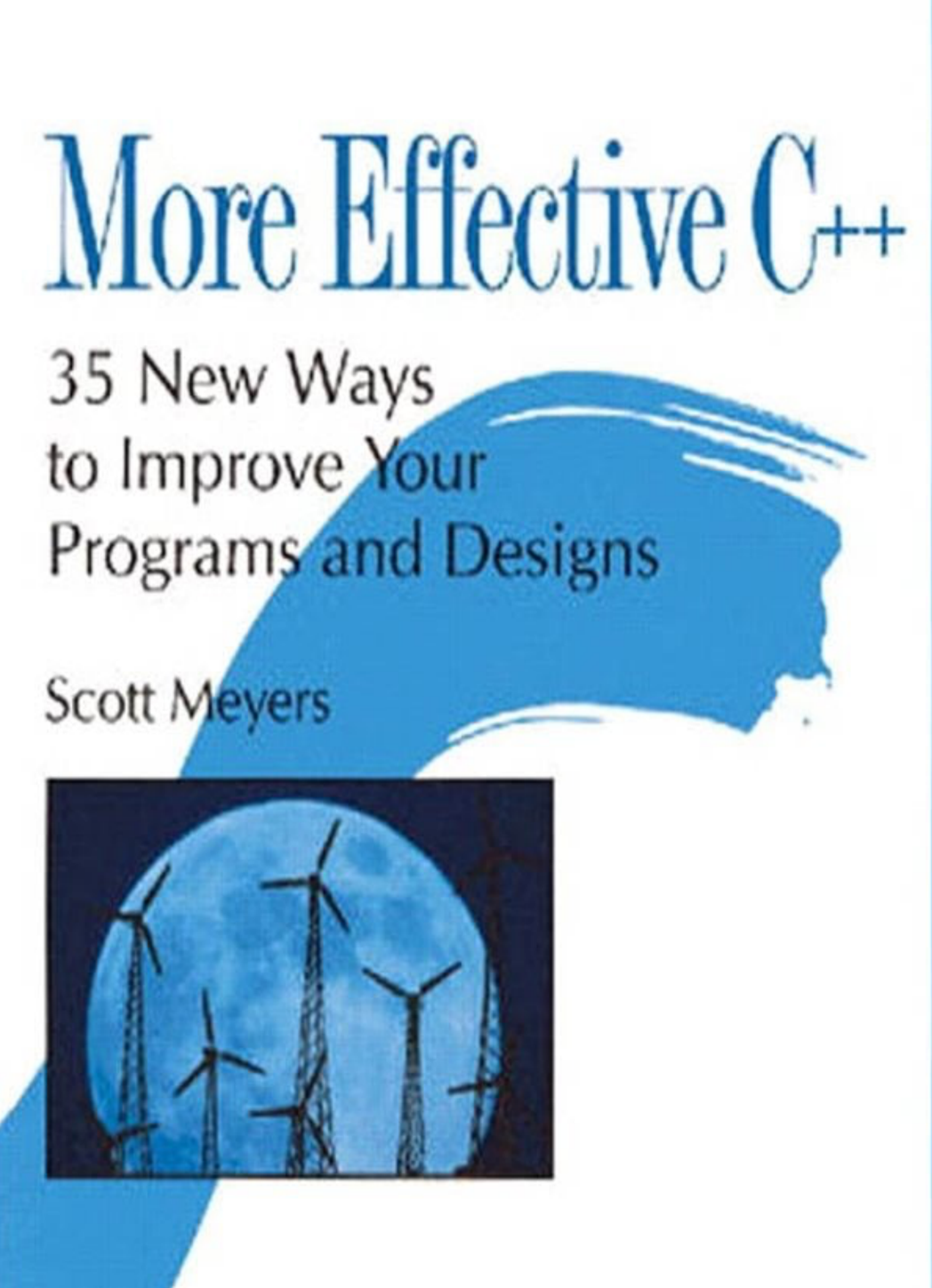
“Do as the ints do”

“Adhere to the principle of least astonishment”



“Do as the ints do”

“Recognize that anything somebody *can* do, they *will* do. They'll throw exceptions, they'll assign objects to themselves, they'll use objects before giving them values”



“Do as the ints do”

“As a result, make your classes easy to use correctly and hard to use incorrectly.”

```
Gadget gadget1;
Gadget gadget2;

{
    Gadget* gadget = &gadget1;
    gadget = &gadget2;
}
```

construction

assignment

destruction

```
{
    Gadget* gadget = &gadget1;
    gadget = &gadget2;
}
```

construction

assignment

destruction

even pointers are just value types

construction	default ctor `T()`	custom ctor `T(A, B, C…)`
	copy ctor `T(T const&)`	move ctor `T(T&&)`
assignment	copy-assign `operator=(T const&)`	move-assign `operator=(T&&)`
destruction	destructor `~T()`	

```
class Gadget {};
```

```
class Gadget {};
```

```
Gadget gadget;
```

```
class Gadget {
public:
    int i;
};
```

```
Gadget gadget;
```

```
class Gadget {
public:
    int i;
};
```

```
Gadget gadget;
std::cout << gadget.i << "\n";
```

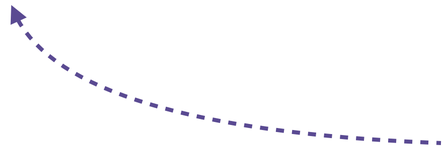
```
-2132860828
```

```
struct Gadget {
    int i;
};
```

```
Gadget gadget;
std::cout << gadget.i << "\n";
```

```
-2132860828
```

```
struct Gadget {
    int i;
    Gadget() : i(0) {}
};
```



default constructor

```
Gadget gadget;
std::cout << gadget.i << "\n";
```

```
0
```

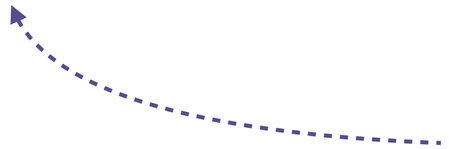
```
struct Gadget {
    int i = 0;
};
```

default member initializer

```
Gadget gadget;
std::cout << gadget.i << "\n";
```

```
0
```

```
struct Gadget {
    int i;
    Gadget(int i) : i(i) {}
};
```



custom constructor

```
Gadget gadget(42);
std::cout << gadget.i << "\n";
```

```
42
```

```
struct Gadget {
    int i;
    Gadget(int i) : i(i) {}
};
```

```
Gadget gadget;
```

```
basic.cpp:56:16: error: no matching function for call to 'Gadget::Gadget()'
   56 |         Gadget gadget;
      |                ^~~~~~
```

```
struct Gadget {
    int i = 0;
    Gadget(int i) : i(i) {}
};
```

```
Gadget gadget;
```

```
basic.cpp:56:16: error: no matching function for call to 'Gadget::Gadget()'
   56 |         Gadget gadget;
      |                ^~~~~~
```

the Special Member Functions

```
T()
T(T const&)
T(T&&)
operator=(T const&)
operator=(T&&)
~T()
```

1. The compiler will generate Special Member Functions (SMFs) for you

2. Defining some SMFs may delete or disable some others

```cpp
struct Gadget {
    int i = 0;
    Gadget(int i) : i(i) {}
};
```

```cpp
Gadget gadget;
std::cout << gadget.i << "\n";
```

```
basic.cpp:56:16: error: no matching function for call to 'Gadget::Gadget()'
   56 |         Gadget gadget;
      |                ^~~~~~
```

```
struct Gadget {
    int i = 0;
    Gadget() {}
    Gadget(int i) : i(i) {}
};
```

explicitly implemented
default constructor

```
Gadget gadget;
std::cout << gadget.i << "\n";
```

```
0
```

```
struct Gadget {
    int i = 0;
    Gadget() = default;
    Gadget(int i) : i(i) {}
};
```

compiler generated
default constructor

```
Gadget gadget;
std::cout << gadget.i << "\n";
```

```
0
```

```
struct Gadget {
    int i = 0;
    Gadget() = default;
    Gadget(int i) : i(i) {}
};
```

```
Gadget gadget;
gadget = Gadget(42);
std::cout << gadget.i << "\n";
```

```
42
```

```
int gadget_number = 0;
struct Gadget {
    int i;
    Gadget() : i(++gadget_number) {
        std::cout << "gadget ctor: #" << i << "\n";
    }
    ~Gadget() {
        std::cout << "gadget dtor: #" << i << "\n";
    }
};
```

Explicitly implemented destructor

```
Gadget gadget;

std::cout << gadget.i << "\n";
```

```
gadget ctor: #1
1
gadget dtor: #1
```

```
struct Widget {
    std::string name;
    int age;
    Gadget* gadget;
};
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";
```

```
1835594944
0x8f38800180df2064
```

```
struct Widget {
    std::string name;
    int age;
    Gadget* gadget;
    Widget() : age(0), gadget(nullptr) {}
};
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";
```

```
0
0
```

```
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget() : age(42), gadget(new Gadget()) {}
};
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x6000025bc030
```

```
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
};
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x6000025bc030
```

```
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    ~Widget() {
        delete gadget;
    }
};
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x600003a90030
gadget dtor: #1
```

```
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    ~Widget() {
        delete gadget;
    }
};
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2 = widget;
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x600003488000
widget
42
0x600003488000
gadget dtor: #1
gadget dtor: #-655720448
malloc: *** error for object 0x600003488000: pointer being freed was not allocated
malloc: *** set a breakpoint in malloc_error_break to debug
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2 = widget;
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n";
```

```cpp
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    Widget(Widget const& other)
    : name(other.name),
      age(other.age),
      gadget(/* .. */) {}
    ~Widget() {
        delete gadget;
    }
};
```

Explicitly implemented

copy constructor

```cpp
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2 = widget;
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n";
```

```cpp
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    Widget(Widget const& other)
    : name(other.name),
      age(other.age),
      gadget(new Gadget(*other.gadget)) {}
    ~Widget() {
        delete gadget;
    }
};
```

```cpp
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2 = widget;
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x6000027d0030
widget
42
0x6000027d0040
gadget dtor: #1
gadget dtor: #1
```

```cpp
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    Widget(Widget const& other)
    : name(other.name),
      age(other.age),
      gadget(new Gadget(*other.gadget)) {}
    ~Widget() {
        delete gadget;
    }
};
```

```cpp
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2 = widget;
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x600003240030
gadget copy ctor: #2
widget
42
0x600003240040
gadget dtor: #2
gadget dtor: #1
```

```cpp
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    Widget(Widget const& other)
    : name(other.name),
      age(other.age),
      gadget(new Gadget(*other.gadget)) {}
    ~Widget() {
        delete gadget;
    }
};
```

```cpp
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

widget = Widget();
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x6000009c4030
gadget ctor: #2
gadget dtor: #2
widget
42
0x6000009c4040
gadget dtor: #-1386594240
malloc: *** error for object 0x6000009c4040: pointer being fre
malloc: *** set a breakpoint in malloc_error_break to debug
```

```
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    Widget(Widget const& other)
    : name(other.name),
      age(other.age),
      gadget(new Gadget(*other.gadget)) {}
    Widget& operator=(Widget const& other) {
      name = other.name;
      age = other.age;
      gadget = new Gadget(*other.gadget);
      return *this;
    }
    ~Widget() {
        delete gadget;
    }
};
```

Explicitly implemented

copy-assignment operator

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

widget = Widget();
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x600001c44030
gadget ctor: #2
gadget copy ctor: #3
gadget dtor: #2
widget
42
0x600001c44050
gadget dtor: #3
```

```
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    Widget(Widget const& other)
    : name(other.name),
      age(other.age),
      gadget(new Gadget(*other.gadget)) {}
    Widget& operator=(Widget const& other) {
        name = other.name;
        age = other.age;
        delete gadget;
        gadget = new Gadget(*other.gadget);
        return *this;
    }
    ~Widget() {
        delete gadget;
    }
};
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

widget = Widget();
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x60000006c030
gadget ctor: #2
gadget dtor: #1
gadget copy ctor: #3
gadget dtor: #2
widget
42
0x60000006c030
gadget dtor: #3
```

```cpp
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    Widget(Widget const& other)
    : name(other.name),
      age(other.age),
      gadget(new Gadget(*other.gadget)) {}
    Widget& operator=(Widget const& other) {
        name = other.name;
        age = other.age;
        Gadget* temp = new Gadget(*other.gadget);
        delete gadget;
        gadget = temp;
        return *this;
    }
    ~Widget() {
        delete gadget;
    }
};
```

```cpp
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

widget = Widget();
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x600003b3c030
gadget ctor: #2
gadget copy ctor: #3
gadget dtor: #1
gadget dtor: #2
widget
42
0x600003b3c050
gadget dtor: #3
```

```cpp
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    Widget(Widget const& other)
    : name(other.name),
      age(other.age),
      gadget(new Gadget(*other.gadget)) {}
    Widget& operator=(Widget const& other) {
        name = other.name;
        age = other.age;
        Gadget* temp = new Gadget(*other.gadget);
        delete gadget;
        gadget = temp;
        return *this;
    }
    ~Widget() {
        delete gadget;
    }
};
```

```cpp
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

widget = widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x600003300030
gadget copy ctor: #2
gadget dtor: #1
widget
42
0x600003300040
gadget dtor: #2
```

```
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    Widget(Widget const& other)
    : name(other.name),
      age(other.age),
      gadget(new Gadget(*other.gadget)) {}
    Widget& operator=(Widget const& other) {
        if(this != &other) {
            name = other.name;
            age = other.age;
            Gadget* temp = new Gadget(*other.gadget);
            delete gadget;
            gadget = temp;
        }
        return *this;
    }
    ~Widget() {
        delete gadget;
    }
};
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

widget = widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";
```

```
gadget ctor: #1
widget
42
0x600001dc0030
widget
42
0x600001dc0030
gadget dtor: #1
```

the Rule of Three

```
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    Widget(Widget const& other)
    : name(other.name),
      age(other.age),
      gadget(new Gadget(*other.gadget)) {}
    Widget& operator=(Widget const& other) {
        if(this != &other) {
            name = other.name;
            age = other.age;
            Gadget* temp = new Gadget(*other.gadget);
            delete gadget;
            gadget = temp;
        }
        return *this;
    }
    ~Widget() {
        delete gadget;
    }
};
```

copy constructor

copy-assignment operator

destructor

```
Widget(Widget const& other)
: name(other.name),
  age(other.age),
  gadget(new Gadget(*other.gadget)) {}

Widget& operator=(Widget const& other) {
   if(this != &other) {
      name = other.name;
      age = other.age;
      Gadget* temp = new Gadget(*other.gadget);
      delete gadget;
      gadget = temp;
   }
   return *this;
}
```

```
Widget(Widget const& other)
: name(other.name),
  age(other.age),
  gadget(new Gadget(*other.gadget)) {}

Widget& operator=(Widget const& other) {
    Widget temp(other);
    using std::swap;
    swap(name, temp.name);
    swap(age, temp.age);
    swap(gadget, temp.gadget);
    return *this;
}
```

```cpp
struct Widget {
    std::string name = "widget";
    int age;
    Gadget* gadget;
    Widget()
    : age(42),
      gadget(new Gadget()) {}
    Widget(Widget const& other)
    : name(other.name),
      age(other.age),
      gadget(new Gadget(*other.gadget)) {}
    Widget& operator=(Widget const& other) {
        Widget temp(other);
        using std::swap;
        swap(name, temp.name);
        swap(age, temp.age);
        swap(gadget, temp.gadget);
        return *this;
    }
    ~Widget() {
        delete gadget;
    }
};
```

```cpp
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2 = std::move(widget);
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n"
```

```
gadget ctor: #1
widget
42
0x600000708000
gadget copy ctor: #2
widget
42
0x600000708010
gadget dtor: #2
gadget dtor: #1
```

```
Widget(Widget&& other)
: name(std::move(other.name)),
  age(std::move(other.age)),
  gadget(std::move(other.gadget)) {}
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2 = std::move(widget);
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n"
```

```
gadget ctor: #1
widget
42
0x6000037b4030
widget
42
0x6000037b4030
gadget dtor: #1
gadget dtor: #1157382192
malloc: *** error for object 0x6000037b4030: pointer being freed was not al
malloc: *** set a breakpoint in malloc_error_break to debug
```

```
Widget(Widget&& other)
: name(std::move(other.name)),
  age(other.age),
  gadget(other.gadget) {
    other.gadget = nullptr;
}
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2 = std::move(widget);
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n"
```

```
gadget ctor: #1
widget
42
0x600000760000
widget
42
0x600000760000
gadget dtor: #1
```

```
Widget(Widget&& other)
: name(std::move(other.name)),
  age(other.age),
  gadget(
  std::exchange(other.gadget, nullptr))
  {}
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2 = std::move(widget);
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n"
```

```
gadget ctor: #1
widget
42
0x600000760000
widget
42
0x600000760000
gadget dtor: #1
```

```
Widget(Widget&& other) noexcept
: name(std::move(other.name)),
  age(other.age),
  gadget(
  std::exchange(other.gadget, nullptr))
  {}
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2 = std::move(widget);
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n"
```

```
gadget ctor: #1
widget
42
0x600000760000
widget
42
0x600000760000
gadget dtor: #1
```

```
Widget(Widget&& other) noexcept
: name(std::move(other.name)),
  age(other.age),
  gadget(
  std::exchange(other.gadget, nullptr))
  {}
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2;
widget2 = std::move(widget);
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n"
```

```
gadget ctor: #1
widget
42
0x600001f0c030
gadget ctor: #2
gadget copy ctor: #3
gadget dtor: #2
widget
42
0x600001f0c050
gadget dtor: #3
gadget dtor: #1
```

```
Widget& operator=(Widget&& other) noexcept
{
    name = std::move(other.name);
    age = other.age;
    gadget =
     std::exchange(other.gadget, nullptr);
    return *this;
}
```

```
Widget widget;
std::cout << widget.name << "\n";
std::cout << widget.age << "\n";
std::cout << widget.gadget << "\n";

Widget widget2;
widget2 = std::move(widget);
std::cout << widget2.name << "\n";
std::cout << widget2.age << "\n";
std::cout << widget2.gadget << "\n"
```

```
gadget ctor: #1
widget
42
0x600001198000
gadget ctor: #2
widget
42
0x600001198000
gadget dtor: #1
```

the

Rule of Five

```
struct Widget {
    Widget();
    Widget(Widget const& other);
    Widget(Widget&& other) noexcept;
    Widget& operator=(Widget const& other);
    Widget& operator=(Widget&& other) noexcept;
    ~Widget();
};
```

copy constructor

move constructor

copy-assignment operator

move-assignment operator

destructor

```cpp
struct Widget {
    std::string name = "widget";
    int age = 42;
};
```

```cpp
void print(Widget const& widget) {
    std::cout << widget.name << "\n";
    std::cout << widget.age << "\n";
}
```

```cpp
Widget widget2 = widget;
print(widget2);
Widget widget3 = std::move(widget);
print(widget3);
print(widget); // moved from

widget2 = widget3;
print(widget2);
widget3 = std::move(widget2);
print(widget3);
print(widget2); // moved from
```

```
widget
42
widget
42
widget
42

42
widget
42
widget
42

42
```

```
struct Widget {
    std::string name = "widget";
    int age = 42;
    std::unique_ptr<Gadget> gadget = std::make_unique<Gadget>();
};
```

```
Widget widget;
print(widget);

Widget widget2 = std::move(widget);
print(widget2);
print(widget);

widget = std::move(widget2);
print(widget2);
print(widget);
```

```
gadget ctor: #1
widget
42
widget
42

42

42
widget
42
gadget dtor: #1
```

the

Rule of Zero

```
struct Widget {
    // only value types or managers
};
```

Always strive to have the compiler generated Special Member Functions do the right thing

Category	When to use	Rule(s)	Special members
Value types	Simple direct values	Rule of Zero	
Views	Non-owning managers		
Polymorphic base classes	Classic OO hierarchies	Rule of Five with disabled copy & move ("DesDeMovA")	`virtual ~T() = default` `operator=(T&&)`
Scoped Managers	Short-lived, on the stack		`~T()` `operator=(T&&)`

Unique Managers	Single ownership - can be stored. Expensive or nonsensical to copy	Rule of Five with disabled copy	`~T()`
			`T(T&&)`
			`operator=(T&&)`

General Managers	Impart value semantics to managed resources - with full copying of independent objects.	Rule of Five	`~T()`
			`T(T const&)`
			`T(T&&)`
			`operator=(T const&)`
			`operator=(T&&)`

https://www.sonarsource.com/blog/beyond-the-rules-of-three-five-and-zero/

Back To Basics
Lifetime Management

PHIL NASH