Back To Basics
The Rule of Five

ANDRE KOSTUR

Cppcon
The C++ Conference
2023
October 01 - 06

About Me

Agenda

- Background
- Core Guidelines
- What is the Rule of Five
- Compiler provided functions
- What is the Rule of Zero

C++ is a Value-Based Language

C++ has a lot to think about

C++ Core Guidelines

Guideline C.21 is “The Rule of Five”

Simple String

Simple String - Constructor

```
struct SString {
    SString(char const * cp) : data_(new char[strlen(cp) + 1]) {
        strcpy(data_, cp);
    }
    // … All of the other uninteresting member functions …
    char * data_;
};
```

What if?

```
{
    SString s{“User Constructed”};
    // …
}
```

Simple String - Destructor

```
struct SString {
    // …
    ~SString() { delete[] data_; }
    // …
    char * data_;
};
```

What if?

```
extern void somefn(SString val);


{
    SString s{“Pass to a function by value”};
    // …
    somefn(s);
    // …
}
```

Simple String - Copy Constructor

```
struct SString {
    // …
    SString(SString const & rhs)
            : data_(new char[strlen(rhs.data_) + 1]) {
        strcpy(data_, rhs.data_);
    }
    // …
    char * data_;
};
```

What if?

```
{
    SString src{“I’m going to be copied”};
    SString dst{“I have a value”};
    // …
    dst = src;
    // …
}
```

Simple String - Copy Assignment Operator

```
struct SString {
    // …
    SString & operator=(SString const & rhs) {
        char * newdata = new char[strlen(rhs.data_) + 1];
        strcpy(newdata, rhs.data_);
        std::swap(newdata, data_);
        delete[] newdata;
        return *this;
    }
    // …
    char * data_;
};
```

What if?

```
extern void somefn(SString val);

{
    SString s{“I’m going to be moved!”};
    // …
    somefn(std::move(s));
    // …
};
```

Simple String - Move Constructor

```
struct SString {
    // …
    SString (SString && rhs) noexcept : data_(rhs.data_) {
        rhs.data_ = nullptr;
    }
    // …
    char * data_;
};
```

What if?

```
extern void somefn(SString val);

{
    SString src{“I’m going to be moved!”};
    SString dst{“I have a value”);
    // …
    dst = std::move(src);
    // …
};
```

Simple String - Move Assignment Operator

```
struct SString {
    // …
    SString & operator=(SString && rhs) {
        delete[] data_;
        data_ = rhs.data_;
        rhs.data_ = nullptr;
        return *this;
    }
    // …
    char * data_;
};
```

The Rule of Five

C.21: “If you define or =delete any copy, move, or destructor function, define or =delete all of them.”

1. Destructor
2. Copy Constructor
3. Copy Assignment Operator
4. Move Constructor
5. Move Assignment Operator

But not normal constructors

Compiler Generated Functions

If you don’t provide them, the compiler will try to generate them for you.

Declare any of them, and the other ones either are no longer declared, or are deleted.

Special Members

compiler implicitly declares

user declares	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Howard Hinnant's Special Members chart

Special Members

compiler implicitly declares

user declares	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Howard Hinnant's Special Members chart

Special Members

compiler implicitly declares

user declares	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Howard Hinnant's Special Members chart

Leveraging the Compiler-Generated Functions

- Copy Constructor
  - Copy constructs each member variable in order
- Copy Assignment Operator
  - Copy assigns each member variable in order
- Move Constructor
  - Move constructs each member variable in order
- Move Assignment Operator
  - Move-assigns each member variable in order
- Destructor
  - Destroys each member variable in reverse order

Simple String - First RAII pass

```
struct SString {
    SString(char const * cp) : data_(new char[strlen(cp) + 1]) {
        strcpy(data_.get(), cp);
    }
    std::shared_ptr<char[]> data_;
};
```

Simple String - Second RAII pass

```
struct SString {
    SString(char const * cp) : data_(new char[strlen(cp) + 1]) {
        strcpy(data_.get(), cp);
    }
    std::unique_ptr<char[]> data_;
};
```

Simple String - Second RAII Copy Constructor

```
struct SString {
    SString(SString const & rhs)
        : data_(new char[strlen(rhs.data_) + 1]) {
        strcpy(data_.get(), rhs.data_.get());
    }
    std::unique_ptr<char[]> data_;
};
```

Simple String - Second RAII Copy Assignment

```
struct SString {
    SString & operator=(SString const & rhs) {
        data_.reset(new char[strlen(rhs.data_) + 1]);
        strcpy(data_.get(), rhs.data_.get());
        return *this;
    }
    std::unique_ptr<char[]> data_;
};
```

Special Members

compiler implicitly declares

user declares	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Howard Hinnant's Special Members chart

Special Members

compiler implicitly declares

user declares	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Howard Hinnant's Special Members chart

Simple String - Second RAII Move Constructor

```
struct SString {
    SString(SString && rhs) noexcept = default;
    std::unique_ptr<char[]> data_;
};
```

Simple String - Second RAII Move Assignment

```
struct SString {
    SString & operator=(SString && rhs) = default;
    std::unique_ptr<char[]> data_;
};
```

Simple String - Second RAII Destructor

```
struct SString {
    ~SString() = default;
    std::unique_ptr<char[]> data_;
};
```

Simple String - All of them

```
struct SString {
    SString(char const * cp) : /* … */ }
    SString(SString const & rhs) : /* … */ }
    SString(SString && rhs) noexcept = default;
    SString & operator=(SString const & rhs) { /* … */ }
    SString & operator=(SString && rhs) = default;
    ~SString() = default;
    std::unique_ptr<char[]> data_;
};
```

Guideline C.20 is “The Rule of Zero”

Extended String - Rule of Zero

```
struct EString {
    EString(char const * cp) : data_(cp) {
    }
    std::string data_;
};
```

Special Members

compiler implicitly declares

user declares	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Howard Hinnant's Special Members chart

Special Members

compiler implicitly declares

user declares	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Howard Hinnant's Special Members chart

Code isn’t just for the compiler

Q&A

Rule of Five: If you declare any of the destructor, copy constructor, copy assignment operator, move constructor, or move assignment operator: you should declare all of them.

Rule of Zero: Strive to use appropriate types for your member variables so that you do not need to write any of the special member functions.

Andre Kostur
andre@kostur.net
@AndreKostur