WIKIPEDIA

Rule of three (C++ programming)

The **rule of three** and **rule of five** are rules of thumb in C++ for the building of exception-safe code and for formalizing rules on resource management. It accomplishes this by prescribing how the default members of a class should be used to accomplish this task in a systematic manner.

Contents

Rule of Three

The **rule of three** (also known as the Law of The Big Three or The Big Three) is a rule of thumb in C++ (prior to C++11) that claims that if a class defines one (or more) of the following it should probably explicitly define all three:[1]

- destructor
- copy constructor
- copy assignment operator

These three functions are special member functions. If one of these functions is used without first being declared by the programmer it will be implicitly implemented by the compiler with the following default semantics:

- **Destructor** – Call the destructors of all the object's class-type members
- **Copy constructor** – Construct all the object's members from the corresponding members of the copy constructor's argument, calling the copy constructors of the object's class-type members, and doing a plain assignment of all non-class type (e.g., *int* or pointer) data members
- **Copy assignment operator** – Assign all the object's members from the corresponding members of the assignment operator's argument, calling the copy assignment operators of the object's class-type members, and doing a plain assignment of all non-class type (e.g. *int* or pointer) data members.

The Rule of Three claims that if one of these had to be defined by the programmer, it means that the compiler-generated version does not fit the needs of the class in one case and it will probably not fit in the other cases either. The term "Rule of three" was coined by Marshall Cline in 1991.[2]

An amendment to this rule is that if the class is designed in such a way that Resource Acquisition Is Initialization (RAII) is used for all its (nontrivial) members, the destructor may be left undefined (also known as The Law of The Big Two[3]). A ready-to-go example of this approach is the use of smart pointers instead of plain ones. [3]

Because implicitly-generated constructors and assignment operators simply copy all class data members ("shallow copy"),[4] one should define explicit copy constructors and copy assignment operators for classes that encapsulate complex data structures or have external references such as pointers, if you need to copy the objects pointed to by the class members. If the default behavior ("shallow copy") is actually the intended one, then an explicit definition, although redundant, will be a "self-documenting code" indicating that it was an intention rather than an oversight.

Rule of Five

With the advent of C++11 the rule of three can be broadened to *the rule of five* as C++11 implements *move semantics*,[5] allowing destination objects to *grab* (or *steal*) data from temporary objects. The following example also shows the new moving members: move constructor and move assignment operator. Consequently, for *the rule of five* we have the following *special members*:

- destructor
- copy constructor
- move constructor
- copy assignment operator
- move assignment operator

Situations exist where classes may need destructors, but cannot sensibly implement copy and move constructors and copy and move assignment operators. This happens, for example, when the base class does not support these latter *Big Four* members, but the derived class's constructor allocates memory for its own use. In C++11, this can be simplified by explicitly specifying the five members as default.[6]

Example in C++

```
#include <cstring>
#include <iostream>

class Foo
{
public:
    /** Default constructor */
    Foo() :
        data (new char[14])
    {
        std::strcpy(data, "Hello, World!");
    }

    /** Copy constructor */
    Foo (const Foo& other) :
        data (new char[std::strlen (other.data) + 1])
    {
        std::strcpy(data, other.data);
    }

    /** Move constructor */
    Foo (Foo&& other) noexcept : /* noexcept needed to enable optimizations in containers */
        data(other.data)
    {
        other.data = nullptr;
    }

    /** Destructor */
    ~Foo() noexcept /* explicitly specified destructors should be annotated noexcept as best-practice */
    {
        delete[] data;
    }

    /** Copy assignment operator */
    Foo& operator= (const Foo& other)
    {
        Foo tmp(other);         // re-use copy-constructor
        *this = std::move(tmp); // re-use move-assignment
        return *this;
    }
    /** Move assignment operator */
    Foo& operator= (Foo&& other) noexcept
    {
        if (this == &other)
        {
            // take precautions against `foo = std::move(foo)`
            return *this;
        }
        delete[] data;
        data = other.data;
        other.data = nullptr;
        return *this;
    }

private:
    friend std::ostream& operator<< (std::ostream& os, const Foo& foo)
    {
        os << foo.data;
        return os;
    }

    char* data;
};

int main()
{
    const Foo foo;
    std::cout << foo << std::endl;

    return 0;
}
```

See also

- C++ classes
- Class (computer programming)

References

1. Stroustrup, Bjarne (2000). *The C++ Programming Language* (3 ed.). Addison-Wesley. pp. 283–4. ISBN 978-0-201-70073-2.
2. Koenig, Andrew; Barbara E. Moo (2001-06-01). "C++ Made Easier: The Rule of Three" (http://www.ddj.com/cpp/184401400). *Dr. Dobb's Journal*. Retrieved 2009-09-08.
3. Karlsson, Bjorn; Wilson, Matthew (2004-10-01). "The Law of the Big Two" (http://www.artima.com/cppsource/bigtwo.html). *The C++ Source*. Artima. Retrieved 2008-01-22.
4. *The C++ Programming Language*. p. 271.
5. Stroustrup, Bjarne (2013-04-07). "C++11 - the new ISO C++ standard" (http://www.stroustrup.com/C++11FAQ.html). Retrieved 2013-05-10.
6. "The rule of three/five/zero" (http://en.cppreference.com/w/cpp/language/rule_of_three). *cppreference.com*. Retrieved 15 February 2015.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Rule_of_three_(C%2B%2B_programming)&oldid=816096142"