How to delete a string object which is a member variable of a C++ union?

Asked 5 years, 6 months ago Modified 5 years, 6 months ago Viewed 410 times

2

When I was reading constructors and destructors in unions, I came across a stack overflow question Is a Union Member's Destructor Called

The accepted answer for that question is saying that we need to provide the destructor for string object explicitly. But as for the accepted answer of Deleting string object in C++, we shouldn't delete the string object explicitly and when the string goes out of scope, it's destructor will be called automatically and the memory will be freed.

These 2 are contradicting. May I know how can we delete the string even if we need to? and also for which objects we need to delete explicitly?

As per the spec If any non-static data member of a union has a non-trivial default constructor (12.1), copy constructor (12.8), move constructor (12.8), copy assignment operator (12.8), move assignment operator (12.8), or destructor (12.4), the corresponding member function of the union must be user-provided or it will be implicitly deleted (8.4.3) for the union. But what is meant by non-trivial?

c++ string language-lawyer destructor unions

Share Improve this question

Follow

edited Jul 31, 2018 at 21:16

NathanOliver

174k 28 294 409

asked Jul 31, 2018 at 21:02

kadina

5,132 6 48 88

2 Your second linked question doesn't have anything to do with unions. I fail to see how it contradicts the first. (Not my downvote, but I am confused.) – Silvio Mayolo Jul 31, 2018 at 21:06

1 Unions are special, they have special rules. One of them being you have to call the destructor for non trivial types – NathanOliver Jul 31, 2018 at 21:07

@kadina You could clarify your question a lot in making it self contained with concise code examples, rather than just straying in links to other questions. – πάντα ῥεῖ Jul 31, 2018 at 21:10

2 Answers

Sorted by: Highest score (default)

3

A trivial constructor/destructor is one that does nothing. Essentially, if your destructor would look like

```
~Type() {}
```

then it is trival. `std::string` 's destructor is not empty like that. It has to clean up any memory that the string may have allocated.

So, since a union's destructor does nothing, but `std::string` needs to do something (otherwise you get a memory leak) you have to supply a destructor for the union that calls the destructor for the string so that it gets cleaned up correctly.

Share Improve this answer Follow

answered Jul 31, 2018 at 21:12

NathanOliver
174k 28 294 409

If you define a destructor (as doing nothing), can it be trivial? – curiousguy Aug 4, 2018 at 14:55

3

In D&D, we call this situation "Specific Beats General".

In the second linked question, the advice "Don't manually delete a `std::string` " is correct, because it's general advice: Objects allocated on the stack shouldn't have their destructors manually called, because they're going to be automatically called when they fall out of scope. If you write something like this:

```
void do_a_thing() {
    std::string str = "Wheeeeeeeeee";
    str.~();
}//Double-delete == BOOM!
```

Bad things will happen as a result, because the memory allocated by `str` will be double-deleted.

But in the first linked question, you're specifically dealing with unions, and therefore the advice "you need to delete objects stored in a union" is specific advice. Unions don't automatically clean up their objects, because they can't know at compile-time which object is contained.

```
void do_a_thing() {
    union {
        int i;
        std::string str;
    } var;
    var.str = "Wheeeeeeeee";
}//Uh oh, str's destructor was never called, and we have a memory leak!
```

And, as the saying goes, "Specific beats General", meaning that you need to supercede the advice to "never manually delete a `std::string` object" with the advice to "always manually delete objects in unions" advice.

Of course, we can render this all moot. You know how? `std::variant`, introduced with C++17. `std::variant` is basically a type-safe union, that prohibits access to non-active members, and ensures proper cleanup of objects.

```
void do_a_thing() {
    std::variant<std::string, int> variant = std::string("Wheeeeeeee");
}//variant properly cleans up the memory, no hassle involved
```

Of course, you do need to learn about the Visitor Pattern in order to make concise use of `std::variant`, but with the benefit of type-safety, it's almost certainly worth it.

Share Improve this answer Follow

edited Jul 31, 2018 at 21:35

answered Jul 31, 2018 at 21:19

Xirema
20k 4 32 68

Excellent. Thanks Xirema. – kadina Jul 31, 2018 at 21:21