# Can shrinking a std::string throw an exception?

devblogs.microsoft.com/oldnewthing/20200120-00

January 20, 2020

Raymond Chen

I had a C++ string that I wanted to truncate. Say, something like this:

```
void remove_extension(std::string& s)
{
 auto pos = s.rfind('.');
 if (pos != std::string::npos) {
  s.resize(pos);
 }
}
```

The question is whether this function can throw an exception. Can the call to `resize` throw an exception when used to make a string smaller?

And the answer appears to be *yes*, at least in C++17.

The specification of the `resize(n)` method in C++17 says that in the case where `n <= size()`, "the function replaces the string designated by `*this` with a string of length `n` whose elements are a copy of the initial elements of the original string designated by `*this`."

In other words, the `resize(n)` method, when shrinking a string (or leaving the size unchanged), behaves as if a new string is created, which replaces the current string. And creating a new string may throw `bad_alloc`.

Of course, implementations may use the *as-if* rule and resize the string in place, but the standard does not require them to do so.

But wait, all is not lost. Because another way to shrink a string is to use the `erase(n)` method.

- **[basic.string]**: `basic_string` is a contiguous container.
- **[container.requirements.general]** (11): Unless otherwise specified…, all container types defined in this Clause meet the following additional requirements:
- **[container.requirements.general]** (11.3): No `erase()` … function throws an exception.

- **[string.erase]**: *Throws*: `length_error` if `n > max_size()`.

There are a few things referenced in the "…" portion of **[container.requirements.general]** (11), but they do not apply to `basic_string`.

Hooray, we can use the `erase` method to shrink the string and avoid an exception.

```
void remove_extension(std::string& s)
{
 auto pos = s.rfind('.');
 if (pos != std::string::npos) {
  s.erase(pos);
 }
}
```

**Bonus chatter**: It appears that the issue of `resize()` throwing an exception when trimming was brought up[1] by Stephan T. Lavavej and fixed by Tim Song in P1148R0: Starting in C++20, if you call the `resize()` method to shrink the string (or keep it the same size), the behavior is defined in terms of erasure and therefore does not throw an exception.

[1] I could have written "raised" but I didn't.[2]

[2] Except that I just did.

Raymond Chen

**Follow**