Why does the compiler complain about a missing constructor when I'm just resizing my std::vector to a smaller size?

devblogs.microsoft.com/oldnewthing/20230711-00

July 11, 2023

Raymond Chen

So you've got a `std::vector` of things.

```
std::vector<Thing> things;
```

And then you learn that only the first `n` of them are any good, so you use `resize()` to throw away the extras.

```
things.resize(n); // keep only the first n
```

But this doesn't work because the compiler complains about a missing constructor for `Thing`. "Why is it trying to construct new `Thing`s when I'm just shrinking?"

Because the compiler doesn't know that you're just shrinking.

The `resize()` is used for both growing and shrinking, so the compiler has to generate code that is prepared to do either. And growing the vector requires default-constructing the new `Thing` objects.

You might choose to appease the compiler by giving it a pre-made `Thing` object to use as the fill value, even though you know it won't be used.

```
// pass dummy object to keep compiler happy
things.resize(n, dummy_thing); // keep only the first n
```

A less clunky solution is to use the `erase` method, which is used only for shrinking.

```
things.erase(things.begin() + n, things.end());
```

Mind you, this is still rather clunky, but at least it's less clunky.