Array size deduction in new-expressions

Timur Doumler (papers@timur.audio)

Abstract

In this paper we propose to fix a particular inconsistency in the initialization rules of C++ by allowing array size deduction in new-expressions. This aligns their behaviour with that of initialization everywhere else in the language.

1 Motivation

Last year, Bjarne Stroustrup pointed out the following inconsistency in the C++ language:

```cpp
double a[]{1,2,3}; // this declaration is OK, ...
double* p = new double[]{1,2,3}; // ...but this one is ill-formed!
```

Jens Maurer promptly provided the explanation: For a new-expression, the expression inside the square brackets is currently mandatory according to the C++ grammar. When uniform initialization was introduced for C++11, the rule about deducing the size of the array from the number of initializers was never extended to the new-expression case. Presumably this was simply overlooked. There is no fundamental reason why we cannot make this work.

Admittedly, deducing the array size in a new-expression is code that probably only very few people would actually write. One could therefore argue that this is a problem not worth fixing.

However, when teaching C++ initialization rules, we observe the following. When people learn about uniform initialization, and then realize that you can (and perhaps should) use it also in new-expressions, they ask:

"Does uniform initialization in a new-expression follow the same rules as everywhere else?"

And the answer is, of course,

"Well, most of the time, except..."

These things are exactly the reason why C++ initialization rules are so notorious for being complicated, and why most C++ developers struggle with them. There are just too many non-obvious inconsistencies. We therefore propose to remove this particular one—not because this is a problem that people would frequently run into (they don’t), but because fixing it is straightforward, the fix is a pure extension that does not impact any other part of the standard, and it would make initialization rules in C++ simpler, more uniform, and easier to teach.
2 Proposed wording

The changes are relative to the C++ working paper [Smith2018].

Modify [expr.new] paragraph 1 as follows:

\[
\text{noptr-new-declarator :}
\begin{align*}
&[ \text{expression_opt} ] \text{attribute-specifier-seq_opt} \\
&\text{noptr-new-declarator [ constant-expression ] attribute-specifier-seq_opt}
\end{align*}
\]

Modify [expr.new] paragraph 6 as follows:

Every constant-expression in a noptr-new-declarator shall be a converted constant expression of type \texttt{std::size_t} and shall evaluate to a strictly positive value. \textit{If the expression in a noptr-new-declarator is present, it is implicitly converted to \texttt{std::size_t}.} [Example: Given the definition \texttt{int n = 42, new float[n][5]} is well-formed (because \texttt{n} is the expression of a noptr-new-declarator), but \texttt{new float[5][n]} is ill-formed (because \texttt{n} is not a constant expression). — end example] If the expression in a noptr-new-declarator is omitted, a new-initializer shall be provided and shall be a braced-init-list. In this case the number of array elements is determined by the number of initial elements as described in [dcl.init.aggr] for initializing an array with a braced-init-list.

Acknowledgements

Many thanks to Richard Smith for his help with the wording.

References