

TDDD38 - Extra lecture

A rant on the auto keyword and
range-based for-loops

Eric Elfving

Department of Computer and Information Science
Linköping University

C++11's range-based for-loops are awesome: they're less verbose than traditional for-loops, they handle both arrays and containers in a generic and extensible manner, and they allow users to focus on the elements they're interested in instead of iterators / pointers / indices.

Stephan T. Lavavej (STL) in N3853

This is a summary of standard documents N3853 and N3994, both concerning introduction of an improved syntax for range-based for-loops.

```
for ( elem : container )
```

Even if we are using `auto`, with the current syntax it's easy to do the wrong declaration of `elem`!

auto

```
for ( auto elem : c )
```

Semantically the same as

```
for ( auto it = begin(c); it != end(c); ++it ) {  
    auto elem = *it;  
}
```

- Each element is copied! (probably not efficient and will give compile errors for uncopyable types)
- Alterations of elem will only change the copy.

auto &

```
for (auto & elem : c )
```

Better

- + will modify the actual elements.
- + will not make copies.
- Will not work for proxy objects!
vector<bool> is a classic example. Stores each bool value as one bit. One bool is one byte so accessing one element will give a temporary proxy object instead of the actual value. A reference can't bind temporaries.

const auto &

```
for ( const auto & elem : c )
```

- + Will bind to temporary values.
- Obviously will not alter elements.

auto &&

```
for ( auto && elem : c )
```

Looks like a rvalue-reference, but auto will work together with reference collapsing to work with any expression category!

Called universal reference by Scott Meyers¹, now standardized as forwarding references.

⇒ This should be the default choice with the new syntax according to STL.

¹<https://isocpp.org/blog/2012/11/universal-references-in-c11-scott-meyers>

Why a new syntax?

- Easier for novices (no index / iterator or explicit automatic type deduction).
- Easier to read (less visual noise).
- Easier to do the right thing (no problem with type of elem)

What about constness?

This [constness] is controlled by the range's type. If the range is `vector<int>`, `elem` will be modifiable. If the range is `const vector<int>`, `elem` will be `const`. If the range is a braced-init-list, `elem` will be `const` (because `initializer_list<E>::begin()` returns `const E *`).

N3853

What about constness?

The old syntax will (of course) still be available if you want to limit yourself to a const reference!

STL also proposes² some alternatives for syntax if EWG (evolution working group) wants it in the language:

```
for (const elem : range)
for (elem : const range)
for const (elem : range)
```

²in N3994

www.liu.se