Document Number: P0662R0
Date: 2017-06-13
Reply to: Casey Carter
casey@carter.net
Reply to: Eric Niebler
eric.niebler@gmail.com

Wording for Ranges TS Issue 345 / US-2: Update ranged-for-loop wording

1 Description of Instructions [intro]

[stmt.ranged] in the Ranges TS is written as a set of editorial instructions relative to C++14 that update the wording for the range-based for loop. This paper proposes alterations to those editorial instructions in the form of — you guessed it — even more editorial instructions. In an attempt to provide clarity of presentation, this paper uses *five* distinct formatting styles to represent text with different properties:

— Text that is the same in C++ 14 and in the Ranges TS is presented in a plain style without adornment.

— Text which the TS strikes from C++ 14 is red in color.

— Text which the TS adds to C++ 14 is cyan in color.

— ~~Text which *this* paper proposes to strike from the *TS* is purple and struck-through.~~

— Text which this paper proposes to add to the TS is gold and underlined.

5.1.1 The range-based `for` statement [stmt.ranged]

[Editor's note: Modify [stmt.ranged] to use a formulation similar to the C++17 FDIS:]

1 ~~For a~~The range-based `for` statement ~~of the form~~

`for (` *for-range-declaration* `:` ~~*expression*~~*for-range-initializer* `)` *statement*

~~let *range-init* be equivalent to the expression surrounded by parentheses~~

~~`(` *expression* `)`~~

~~and for a range-based for statement of the form~~

~~`for (` *for-range-declaration* `:` *braced-init-list* `)` *statement*~~

~~let *range-init* be equivalent to the *braced-init-list*.~~ ~~In each case, a range-based for statement~~ is equivalent to

```
{
    auto &&__range = range-init ;
    for ( auto __begin = begin-expr ,
               __end = end-expr ;
          __begin != __end;
          ++__begin ) {
        for-range-declaration = *__begin;
        statement
    }
}
{
    auto &&__range = range-init for-range-initializer ;
    auto __begin = begin-expr ;
    auto __end = end-expr ;
    for ( ; __begin != __end; ++__begin ) {
        for-range-declaration = *__begin;
        statement
    }
}
```

where

(1.1) — if the *for-range-initializer* is an *expression*, it is regarded as if it were surrounded by parentheses (so that a comma operator cannot be reinterpreted as delimiting two *init-declarator*s);

(1.2) — `__range`, `__begin`, and `__end` are variables defined for exposition only; and ~~`__RangeT` is the type of the expression, and *begin-expr* and *end-expr* are determined as follows:~~

(1.3) — *begin-expr* and *end-expr* are determined as follows:

(1.3.1) — if ~~`__RangeT`~~the *for-range-initializer* is an an expression of array type `R`, *begin-expr* and *end-expr* are `__range` and `__range + __bound`, respectively, where `__bound` is the array bound. If ~~`__RangeT`~~`R` is an array of unknown ~~size~~bound or an array of incomplete type, the program is ill-formed;

(1.3.2) — if ~~`__RangeT`~~the *for-range-initializer* is ~~a~~an expression of class type `C`, the *unqualified-id*s `begin` and `end` are looked up in the scope of ~~class `__RangeT`~~`C` as if by class member access lookup (3.4.5), and if either (or both) finds at least one declaration, *begin-expr* and *end-expr* are `__range.begin()` and `__range.end()`, respectively;

(1.3.3) — otherwise, *begin-expr* and *end-expr* are `begin(__range)` and `end(__range)`, respectively, where `begin` and `end` are looked up in the associated namespaces (3.4.2). [*Note:* Ordinary unqualified lookup (3.4.1) is not performed. — *end note*]

[*Example:*

```
int array[5] = { 1, 2, 3, 4, 5 };
for (int& x : array)
  x *= 2;
```

— *end example*]

2 In the *decl-specifier-seq* of a *for-range-declaration*, each *decl-specifier* shall be either a *type-specifier* or `constexpr`. The *decl-specifier-seq* shall not define a class or enumeration.