Document Number: P0609R1
Date: 2017-09-14
Author: Aaron Ballman <aaron@aaronballman.com>
Audience: Evolution Working Group

Attributes for Structured Bindings

Motivation

We added the ability to write structured binding declarations in C++17. The optional *attribute-specifier-seq* in such a declaration appertains to the hidden variable declared by the structured binding declaration. Despite the variable being hidden, this is still useful functionality (for instance, it allows the programmer to specify the alignment of the structured binding declaration itself, which may allow for useful compiler optimizations when loading from an array).

However, there is no way to specify attributes that appertain to the individual structured bindings. It is desirable to allow vendor-specific attributes to appertain to these bindings for attributes that would otherwise appertain to variables to enable better diagnostics, especially through static analysis. For instance, some implementations support thread-safety attributes (`guarded_by`, et al) that denote a variable requires a particular locking primitive to be held before accessing the variable. Other implementations support an annotation which denotes an object with an array of `char` or pointer to `char` type does not necessarily contain a terminating null character (`nonstring`). Given the prevalence of vendor-specific attributes, it is likely that other motivating use cases currently exist.

I propose to allow optional attributes for each of the introduced structured bindings, as in this example:

```
auto g() {
  auto [a, b [[vendor::attribute]], c] = f();
  return a + c;
}
```

While this may generate an overabundance of square brackets in a declaration, the syntax is consistent with our other treatments of attributes in declarations.

Proposed Wording

Modify [dcl.dcl]p1:

> ...
> *attributed-identifier-list:*
> *identifier attribute-specifier-seq*$_{opt}$
> *attributed-identifier-list* , *identifier attribute-specifier-seq*$_{opt}$
>
> *simple-declaration:*
> *decl-specifier-seq init-declarator-list*$_{opt}$;
> *attribute-specifier-seq decl-specifier-seq init-declarator-list* ;
> *attribute-specifier-seq*$_{opt}$ *decl-specifier-seq ref-qualifier*$_{opt}$ [*attributed-identifier-list*] *initializer* ;
> ...

Modify [dcl.dcl]p8:

A *simple-declaration* with an *attributed-identifier-list* is called a *structured binding declaration* (11.5). The *decl-specifier-seq* shall contain only the *type-specifier* auto (10.1.7.4) and *cv-qualifier*s. The *initializer* shall be of the form "= *assignment-expression*", of the form "{ *assignment-expression* }", or of the form "(*assignment-expression*)", where the *assignment-expression* is of array or non-union class type.

Modify [dcl.struct.bind]p1:

A structured binding declaration introduces the *identifier*s v0, v1, v2, ... of the *attributed-identifier-list* as names (6.3.1), called *structured bindings*. The optional *attribute-specifier-seq* of an *attributed-identifier* from the *attributed-identifier-list* appertains to the introduced structured binding. Let *cv* denote the *cv-qualifier*s in the *decl-specifier-seq*. First, a variable with a unique name e is introduced. If the *assignment-expression* in the *initializer* has array type A and no *ref-qualifier* is present, e has type *cv* A and each element is copy-initialized or direct-initialized from the corresponding element of the *assignment-expression* as specified by the form of the *initializer*. Otherwise, e is defined as-if by

attribute-specifier-seq$_{opt}$ *decl-specifier-seq ref-qualifier*$_{opt}$ e *initializer* ;

where the declaration is never interpreted as a function declaration and the parts of the declaration other than the *declarator-id* are taken from the corresponding structured binding declaration. The type of the *id-expression* e is called E. [*Note:* E is never a reference type (Clause 5). — *end note*]

Modify p2:

If E is an array type with element type T, the number of elements in the *attributed-identifier-list* shall be equal to the number of elements of E. ...

Modify p3:

Otherwise, if the *qualified-id* std::tuple_size<E> names a complete type, the expression std::tuple_size<E>::value shall be a well-formed integral constant expression and the number of elements in the *attributed-identifier-list* shall be equal to the value of that expression. ...

Modify p4:

Otherwise, all of E's non-static data members shall be public direct members of E or of the same unambiguous public base class of E, E shall not have an anonymous union member, and the number of elements in the *attributed-identifier-list* shall be equal to the number of non-static data members of E.

Acknowledgements

Thanks to Richard Smith for reviewing this paper.