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. Consider the following example:

```cpp
auto g() {
    auto [a, b, c] = f(); // May generate a QoI diagnostic about b being unused
    return a + c;
}
```

It would be useful for the programmer to have a way to specify the `[[maybe_unused]]` attribute on the introduced declaration of `b` to silence compiler diagnostics. Additional motivating examples come from other standard attributes (like alignment specifiers) and vendor-specific attributes.

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

```cpp
auto g() {
    auto [a, b, [maybe_unused], 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:

```cpp
... attributed-identifier-list:
    identifier attribute-specifier-seqopt
    attributed-identifier-list , identifier attribute-specifier-seqopt

simple-declaration:
    decl-specifier-seq init-declarator-listopt ;
    attribute-specifier-seq decl-specifier-seq init-declarator-list ;
    attribute-specifier-seqopt decl-specifier-seq ref-qualifieropt [ attributed-identifier-list ] initializer ;
...```
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-qualifiers. 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.

A structured binding declaration introduces the identifiers 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-qualifiers 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 ]

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. ...

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. ...

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.